Chapter 2

Computing Recommendations With Collaborative Filtering

Recommender systems, based on Collaborative Filtering (CF) algorithms, generate personalised content for their users by relying on a simple assumption: those who have had similar opinions in the past will continue to share similar tastes in the future. This chapter serves as a review of the state of the art in collaborative filtering. We first introduce the rating data, detailing how the ratings are collected (Section 2.1). We then introduce the various approaches that have been adopted when designing CF algorithms. In particular, we differentiate between *data mining* (Section 2.2) and *user modelling* (Section 2.3) approaches: the focus of the former is on designing algorithms that augment the predictive power of CF when applied to a set of ratings; the latter, on the other hand, builds methods based on preconceived models of trust between system users. We then explore how these systems are evaluated (Section 2.4), including the methodology and metrics related to CF evaluation, and enumerate a number of open problems (Section 2.5) that we will address in this thesis.

2.1 Ratings And User Profiles

The focal point of recommender systems is the *user-rating data* upon which the system operates. Before introducing the underlying algorithms of recommender systems, we define the terms related to this data that will be used throughout this thesis.

- **User:** the end user of the system, or the person we wish to provide with recommendations. The entire set of users is referred to as the community.
- **Item:** a generic term used to denote the system's content, which may be a song, movie, product, web page, etc.
- **Rating:** a numerical representation of a user's preference for an item; these are considered in more depth in Section 2.1.1.
- **Profile:** the set of ratings that a particular user has provided to the system. Equivalently, the profile of an *item* is the set of ratings that have been input for that item by the users.
- **User-Item Matrix:** The ratings input by all users are often represented as a matrix, where columns are individual items, and rows are users. Each matrix entry $r_{u,i}$ is the rating (an numerical value

on a given range, e.g. 1-5 stars) input by user u for item i.

Recommender systems revolve around the set of user profiles; by containing a collection of ratings of the available content, this set is the focal source of information used when providing each user with recommendations. Ratings can be collected either *explicitly* or *implicitly*; in the following section we describe and compare each of these methods.

2.1.1 Implicit and Explicit Ratings

Ratings (human opinions of the system's content) can come from two separate sources. On the one hand, the opinions could be in the form of *explicit* ratings: users may be asked to rate items on a N-point (e.g., five-star) Likert scale. In this case, the rating is a numeric value that is input directly by the user. On the other hand, opinions can be extracted from the user's *implicit* behaviour. These include time spent reading a web page, number of times a particular song or artist was listened to, or the items viewed or purchased when browsing an online catalogue; logging implicit behaviour is an attempt to capture taste by reasoning on how users interact with the content.

The most significant difference between explicit and implicit ratings is that, in the latter scenario, users cannot tell the system that they *dislike* content: while an explicit scale allows users to input a low score (or negative feedback), implicit ratings can only infer positive behaviour. Hu *et al* [HKV08] further this notion by describing how, in the implicit case, the system does not determine a user's *preference* for an item, but rather reasons on the *confidence* it has in a user's affinity to the item, based on measured behaviour. A consequence of this is the possibility of making noisy measurements. For example, Hu *et al* [HKV08] mention that it is impossible to differentiate between a user who watches a television program for a long period of time or is asleep in front of the television; this is a problem where simple thresholding is not a sufficient solution. However, Amatriain *et al* [APO09] show that explicit ratings are also prone to noise: users may be careless and irregular when they manually input ratings.

The art of collecting ratings thus seems to be context-specific: furthermore, Herlocker *et al* identified that the act of rating itself is motivated by different reasons, including self-expression and helping or influencing others' decisions [HKTR04]. For example, movie-recommender systems often prefer to let users explicitly rate movies, since users may dislike a particular movie they have watched. Music recommender systems tend to construct user profiles based on listening habits, by collecting metadata of the songs each user has listened to; these systems favour implicit ratings by assuming that users will only listen to music they like. Implicit ratings can be converted to a numeric value with an appropriate transpose function (the algorithms we describe next are hence equally applicable to both types of data).

Both implicit and explicit ratings share a common characteristic: the set of available judgments for each user, compared to the total number of items that can be rated, will be very small. In particular, it is impossible to determine whether an item remains *unrated* because it is disliked or because the user has simply not yet encountered the content to date. The lack of information is known as the problem of *data sparsity*, and has a strong effect on the efficacy of any algorithms that base their recommendations on this data. However, rating sparsity is a natural consequence of the problem that recommender systems address: if it were feasible for users to find and rate all the content they were interested in, recommender

systems would no longer be needed.

Rating data is related to the broader category of relevance feedback from the information retrieval community [RL03, FO95]; in fact, recommender systems can be broadly considered to be query-less retrieval systems, that operate solely using user feedback, in order to both find and rank content. The main goal of recommender systems is to filter content in order to provide relevant and useful suggestions to each user of the system. There are two methods to do so: the *content-based* and *collaborative* approaches. Content-based algorithms formulate recommendations by matching descriptions of the system's items to those items in each user profile [PB07]. These systems, however, are not appropriately or readily applied to the entire range of scenarios where users may benefit from recommendations: they require content that can be described in terms of its attributes, which may not always be the case. Furthermore, simply matching on attributes disregards what users think about the content, as the preference data remains untouched. In the following section, we review the alternative approach: collaborative filtering algorithms.

2.2 Collaborative Filtering Algorithms

The task of a recommender system algorithm is to take as input a set of user ratings and output personalised recommendations for each user. To do so, recommender systems use a collaborative filtering (CF) algorithm, which mines patterns within the ratings in order to forecast each user's preference for unrated items. At the broadest level, the recommendations are generated by:

- **Collecting Ratings**. The system collects ratings from each user.
- **Predicting Missing Values**. Collected ratings are input to a CF algorithm: the algorithm is trained with the available ratings, and asked to *predict* the values of the missing ratings.
- **Ranking and Recommending**. The predictions are used to create a personalised ranking of unrated items for each user; this tailored list is served to each user as a ranked list of recommendations.

Users can continue rating items, and the process of rate-predict-recommend continues. This process highlights a number of features of CF. The founding assumption is that *like-mindedness is persistent*: if users have shown similar interests in the past, they are likely to enjoy similar items in the future. In other words, the ratings contain useful information to train learning algorithms. CF algorithms tend to disregard any descriptive attributes of the items (or what the items actually are) in favour of the ratings, and focus on generating recommendations based solely on the opinions that have been input. Embedded in this is a fine-grained notion of similarity between items: two items are similar if they are enjoyed by the same users, regardless of what they actually are.

The problem of generating recommendations, and the use of the data that is available to tackle this task, has been approached from a very wide range of perspectives: in this section, we elaborate on a number of widely-used algorithms. Each perspective applies different heuristics and methodologies in order to create recommendations. Historically, the two broadest categories of collaborative filters were the memory and model based approaches. In the following section, we define these categories and

discuss whether this grouping of approaches reflects state of the art research.

2.2.1 Grouping the Algorithms

A range of literature partitions the CF algorithms into two groups: memory based and model based approaches. Memory-based CF has often been referred to as the dominant method of generating recommendations due to both its clear structure and successful results, making it an easy choice for system developers. In essence, this group includes the k-Nearest Neighbour (kNN) algorithm and the range of variations it has been subject to (we explore these further in Section 2.2.3). Model based approaches to CF, instead, aim to apply any of a number of other classifiers developed in the field of machine learning, including the use of singular value decomposition, neural net classifiers, Bayesian networks, support vector machines, perceptrons, induction rule learning, and latent semantic analysis [BHK98, YST+04, CS01] to the problem of information filtering. A complete introduction to all available machine learning algorithms is beyond the scope of this thesis, and we point the reader to appropriate introductory texts [Alp04]. Each differs in the method applied to learn how to generate recommendations, but they all share a similar high-level solution: they are based on inferring rules and patterns from the available rating data.

The kNN algorithm differentiates itself from members of the model based group by being a *lazy* or *instance-based learning algorithm*; it operates on a subset of ratings (or instances) when computing predicted values. As we explore below, the kNN training phase consists of forming neighbourhoods for each user (or item). In other words, this phase entails selecting a *subset* of the user-item matrix for each of the predictions that the algorithm will subsequently be asked to make. On the other hand, model based approaches adopt *eager learning* characteristics, and need not return to the training data when computing predictions. Instead, they formulate a *model*, or higher level abstraction, of the underlying relationships between user preferences. For example, matrix factorisation (Section 2.2.4) approaches decompose the user-item matrix into a set of user and item factors and compute recommendations with these, rather than the ratings themselves.

These differences have motivated the grouping of CF algorithms into memory and model based approaches, and lead to studies comparing the relative performance of each group in different CF domains [GFM05]. However, this partitioning is not clear-cut. For example, the kNN algorithm *models* user preferences with user (or item) neighbourhoods; a user's taste is represented by a sample of peers who have exhibited similar preferences. Similarly, model based approaches learn from the available instances (or *memory*) of user ratings; their efficacy is also strictly related to the training data.

Furthermore, grouping algorithms in this way fails to acknowledge the diversity of research approaches used when designing and evaluating CF algorithms. Broadly speaking, these approaches tend to be based on *data mining*, where the focus is on extrapolating predictive power from the available rating data, or *user modelling*, that designs algorithms using preconceived notions of how the system's users will interact with (or, for example, trust) each other. Should a user-modelling perspective, which uses a kNN algorithm, be a memory or model based approach? Is trust a memory or model based characteristic? Due to these shortcomings, in the following sections we review a set of CF algorithms (baseline, k-Nearest Neighbour, matrix factorisation, and hybrid methods) that reflect the contributions of both the

mining and modelling approaches without strictly adhering to the memory/model dichotomy.

2.2.2 Baselines

The first set of approaches we examine are *baselines*. These are the simplest predictions that can be made, and include the user mean (or mean of items i rated by user u). Given a user u, with profile R_u, predictions $\hat{r}_{u,t}$ for each target item t are assigned the same value:

$$\hat{r}_{u,t} = \frac{1}{|R_u|} \times \sum_{i \in R_u} r_{u,i} \tag{2.1}$$

Similarly, given a target item t with profile R_i, item mean values are computed as:

$$\hat{r}_{u,t} = \frac{1}{|R_i|} \times \sum_{u \in R_i} r_{u,i} \tag{2.2}$$

In general, the item-mean method is preferred to the user-mean, since the latter is not conducive to ranking any of the content (since all predictions for a user will have the same value). Similarly, if the item-mean method is coupled with the *frequency* of ratings for an item, this method corresponds to recommending a non-personalised popularity-sorted ranking of items.

The *bias model*, as described by Potter [Pot08], builds on the above by predicting user u's rating for an item i using u's mean rating (bias, b_u) and the average deviation from each user's mean for item i (preference, p_i). Overall, this method is highly dependent on each user's mean to predict the ratings; we elected to investigate it since user means will change over time. For a set of users U and items I, the biases and preferences are initialised as zeroes and then computed by iterating over the following:

$$\forall i \in I : p_i = \frac{1}{|R_u|} \times \sum_{u \in R_i} (r_{u,i} - b_u) \tag{2.3}$$

$$\forall u \in U : b_u = \frac{1}{|R_i|} \times \sum_{i \in R_u} (r_{u,i} - p_i) \tag{2.4}$$

The iteration continues until the difference in the Root Mean Squared Error (RMSE) achieved on the training set is less than a pre-defined value δ. The predicted rating $\hat{r}_{u,i}$ for a user-item pair is computed as:

$$\hat{r}_{u,i} = b_u + p_i$$

The main role of these and similar [BK07] baseline methods (sometimes referred to as "global effects") has been to normalise data prior to applying other classifiers. Further amendment to the method scales each rating by the user's variance, if that variance is non-zero [Pot08].

2.2.3 k-Nearest Neighbours

The k-Nearest Neighbour (kNN) algorithm has enjoyed enourmous popularity in the domain of recommender systems; it appeared in early research efforts [HKBR99] and continues to be applied in state of the art solutions [BK07]. Two flavours of the algorithm exist: the user-based [HKBR99] and item-based [SKKR01, LSY03] approaches. The two approaches differ in how they view the underlying rating data: the user-based approach views the data as a collection of *users* who have rated items, while the item-based approach views the same data as a collection of *items* that have been rated by users. It is important

to note, however, that the techniques described here can be equally applied to both user or item profiles. In the interest of consistency, we adopt the terminology of the user-based approach throughout this section.

In general, it is impossible to claim with any authority that one method will always outperform the other. However, the item-based approach is often preferred since available CF datasets tend to have many more users than items. In this section, we elaborate on how the algorithm works by decomposing it into two stages: neighbourhood formation and opinion aggregation.

Neighbourhood Formation

This first step aims to find a unique subset of the community for each user by identifying others with similar interests to act as recommenders. To do so, every pair of user profiles is compared, in order to measure the degree of similarity $w_{a,b}$ shared between all user pairs a and b. In general, similarity values range from 1 (perfect similarity) to -1 (perfect dissimilarity), although different measures may only return values on a limited range. If a pair of users have no profile overlap, there is no means of comparing how similar they are, and thus the similarity is set to zero.

Similarity can be measured in a number of ways, but the main goal of this measure remains that of modelling the potential relationship between users with a numeric value. The simplest means of measuring the strength of this relationship is to count the proportion of co-rated items, or Jaccard similarity, shared by the pair of users [Cha02]:

$$w_{a,b} = \frac{|R_{a,i} \cap_i R_{b,i}|}{|R_{a,i} \cup_i R_{b,i}|} \tag{2.5}$$

This similarity measure disregards the values of the ratings input by each user, and instead only considers what each user has rated; it is the size of the intersection of the two users' profiles over the size of the union. The underlying assumption is that two users who continuously rate the same items share a common characteristic: their choice to rate those items.

The most cited method of measuring similarity is the Pearson Correlation Coefficient (PCC), which aims at measuring the degree of linearity that exists on the intersection of the pair of users' profiles [BHK98, HKBR99]:

$$w_{a,b} = \frac{\Sigma_{i=1}^{N}(r_{a,i} - \bar{r}_a)(r_{b,i} - \bar{r}_b)}{\sqrt{\Sigma_{i=1}^{N}(r_{a,i} - \bar{r}_a)^2 \Sigma_{i=1}^{N}(r_{b,i} - \bar{r}_b)^2}} \tag{2.6}$$

Each rating above is normalised by subtracting the user's mean rating (e.g., $\bar{r}_a$), computed using Equation 2.3. The PCC similarity measure has been subject to a number of improvements. For example, if the intersection between the pair of user's profiles is very small, the resulting similarity measure is highly unreliable, as it may indicate a very strong relationship between the two users (who have only co-rated very few items). To address this, Herlocker *et al* [HKBR99] introduced significance weighting: if the number of co-rated items n is less than a threshold value x, the similarity measure is multiplied by $\frac{n}{x}$. This modification reflects the fact that similarity measures become more reliable as the number of co-rated items increases, and has positive effects on the predictive power of the filtering algorithm. The same researchers also cite the constrained Pearson correlation coefficient, which replaces the user means in the above equation with the rating scale midpoint.

Similarity measures are also often coupled with other heuristics that aim at improving the reliability and power of the derived measures. For example, Yu *et al* [YWXE01] introduced variance weighting; when comparing user profiles, items that have been rated by the community with greater variance receive a higher weight. The aim here is to capture the content that, being measurably more "controversial" (and eliciting greater disagreement amongst community members) is a better descriptor of taste. Measuring similarity, however, remains an open issue; to date, there is little that can be done other than comparing prediction accuracy in order to demonstrate that one similarity measure outperforms another on a particular dataset.

There are a number of other ways of measuring similarity that have been applied in the past. These include the Spearman Rank correlation, the Vector Similarity (or cosine angle between the two user profiles), Euclidean and Manhattan distance, and other methods aimed at capturing the proportion of agreement between users, such as those explored by Agresti and Winner [AW97]. Each method differs in the operations it applies in order to derive similarity, and may have a strong effect on the power the algorithm has to generate predicted ratings.

Opinion Aggregation

Once comparisons between the user and the rest of the community of recommenders (regardless of the method applied) are complete, predicted ratings of unrated content can be computed. As above, there are a number of means of computing these predictions. Here we present two [HKBR99, BK07]. Both equations share a common characteristic: a predicted rating $p_{a,i}$ of item i for user a is computed as a weighted average of neighbour ratings $r_{b,i}$. The first is a weighted average of neighbour ratings:

$$p_{a,i} = \frac{\Sigma_b r_{b,i} \times w_{a,b}}{\Sigma w_{a,b}} \tag{2.7}$$

The second subtracts each recommender's mean from the relative rating; the aim of this normalisation step is to minimize the differences between different recommenders' rating styles, by considering how much ratings deviate from each recommender's mean rather than the rating itself.

$$p_{a,i} = \bar{r}_a + \frac{\Sigma_b (r_{b,i} - \bar{r}_b) \times w_{a,b}}{\Sigma w_{a,b}} \tag{2.8}$$

The weights $w_{a,b}$ may come from one of two sources. They may be the similarity measures we found in the first step; neighbours who are more similar will have greater influence on the prediction. On the other hand, recent work [BK07] uses similarity weights to *select* neighbours, and then re-weights the selected group simultaneously via a linear set of equations in order to maximise the accuracy of the selected group on the user's profile.

The natural question to ask at this step is: which recommender ratings are chosen to contribute to the predicted rating? A variety of choices is once again available, and has a direct impact on the performance that can be achieved. In some cases, only the top-k most similar neighbours are allowed to contribute ratings, thus guaranteeing that only the closest ratings create the prediction. However, it is often the case that none of the top-k neighbours have rated the item in question, and thus the prediction coverage, or the number of items that can be successfully predicted, is negatively impacted. A straightforward

alternative, therefore, is to consider the top-k recommenders who can give rating information about the item in question. This method guarantees that all predictions will be made; on the other hand, predictions may now be made according to ratings provided by only modestly-similar users, and may thus be less accurate. A last alternative is to only select users above a pre-determined similarity threshold. Given that different similarity measures will produce different similarity values, generating predictions this way may also prevent predictions from being covered. All methods, however, share a common decision: what should the threshold value, or value of k, be? This question remains unanswered and dependent on the available dataset; research in the area tends to publish results for a wide range of values.

In general, while the process of selecting neighbours is a focal point of kNN approaches, selecting the best neighbours is an inexact science. Rafter *et al* [ROHS09] examined the effect that neighbours (selected with the methods described above) have on prediction accuracy, and found that neighbours are often *detrimental* in this process. Instead, [LAP09] reports on the potential massive gains in accuracy if an optimal set of neighbours is selected. The problem of neighbour selection, and approaches designed according to user models, will be further addressed in the Section 2.3.

Up to this point, we have considered the process of generating recommendations strictly from the so-called memory based, nearest-neighbour approach. In the following section, we review another of the most prominent CF algorithms, which is based on matrix factorisation.

2.2.4 Matrix Factorisation

Recommender systems connect *large* communities of users to *large* repositories of content; the rating data that they contain is thus invariably sparse. A powerful technique to address this problem is that of matrix factorisation, based on Principle Component Analysis (PCA) or Singular Value Decomposition (SVD) [Pat06, MKL07, KBC07]; in this section we focus on SVD.

As detailed by Amatriain *et al* [AJOP09], the core function of an SVD is to use the sparse rating data in order to generate two *descriptive* matrices that can be used to approximate the original matrix. In other words, given a matrix of user-item ratings R, with n users and m items, the task of an SVD is to compute matrices U and V such that:

$$R = U\lambda V^T \tag{2.9}$$

U is an $(n \times r)$ matrix, and V is an $(r \times m)$ matrix, for a given number of required features r, and λ is a diagonal matrix containing the singular values. Each matrix contains important information. For example, V describes the system content in terms of affinity to each feature: in fact, this information can be used to describe the implicit relations among the system's content by showing how each item relates to others in the computed feature space.

Once the decomposed matrices have been computed, the rating for a user-item pair is approximated as the dot product between the user's feature vector (from U) and the item's feature vector (from V). In other words, for a user u and item i, the predicted rating $\hat{r}_{u,i}$ is:

$$\hat{r}_{u,i} = \sum_{f=0}^{r} U_{u,f} \times V_{f,i} \tag{2.10}$$

Based on the above, the U and V matrices themselves can be approximated iteratively: after initialising each matrix, the features can be updated in order to minimise the squared error between the computed predictions and ratings for the available data instances. This process is bounded by using an appropriate learning rate (to avoid overfitting) and a number of rounds that should be dedicated to each feature [Pia07].

Although factorisation addresses data sparsity, it does not overcome it: computing the SVD itself is challenging when data is missing. Sarwar *et al* [SKKR00] address this by replacing the missing values with item mean ratings. Factorisation serves many purposes: it may be used as both a preprocessing or prediction mechanism; in the next section, we explore how it has been used in the context of *hybrid* algorithms.

2.2.5 Hybrid Algorithms

As we have seen above, there is a wide range of algorithms suitable for the CF context. Each algorithm offers a different method of reasoning on the rating data, produces different results, and has different shortcomings. This range of differences between these methods motivates using more than one recommendation algorithm in unison, in order to reap the benefits of each method (and, hopefully, overcome the limitation that each one suffers when used alone). In [Bur02], Burke provides a comprehensive review of hybrid algorithms: in this section, we review popular approaches, which we decompose into two groups: *preprocessing* and *ensemble methods*. Each group need not be implemented alone. In fact, there is no limit as to how hybrid algorithms may be designed; it is simply the case that CF methods become *hybrid* when they are designed using more than a single classifier.

Preprocessing

The purpose of preprocessing is either to *modify* or *partition* the raw data, in order to apply one of the above classification algorithms. Two widely used approaches are *normalising* user ratings (as discussed in Section 2.2.2), and *clustering* the data into smaller groups. The former tends to transform the integer ratings into a set of residuals, by subtracting user biases [Pot08, BK07]; these account for the different ways that users interpret the rating scale (e.g., some users consistently rate higher than others). There is a variety of techniques available for the latter purpose, including k-means, hierarchical, and density based clustering [AJOP09, JMF99]. For example, Rashid *et al* [RLKR06] proposed a filtering algorithm suitable for extremely large datasets that combines a clustering algorithm with the kNN prediction method. The aim was to cluster similar users together first, in order to overcome the costly operation of measuring the similarity between all user pairs in the system, and then apply a nearest-neighbour technique to make predictions. Much like the work presented by Li and Kim [LK03], clustering methods can be implemented to replace the "neighbourhood formation" step of the kNN approach. The Yoda system, designed by Shahabi *et al* [SBKCM01], is an example of a system that performs similar functions: clustering is implemented to address the scalability issues that arise as the community of users and available items grows.

Similarly, a sequence of classification algorithms can be designed, where the output of each method is the input to the next; each algorithm preprocesses the data for the subsequent method. For example,

the rating matrix can be factorised first and each user's k-nearest neighbours can then be computed using the feature matrix, rather than the raw data itself [Kor08]. The aim of this is to compute neighbours using the *dense* feature matrix, rather than the extremely sparse rating data—-the hope being that more reliable neighbours can be found this way.

Ensemble Methods

The second set of hybrid techniques we review differentiate themselves from the above by focusing on combining output rather than refining it over sequential steps [Die00, Pol06]. In other words, classification algorithms are run independently of one another, and the output of each is then collapsed into a unified set of predictions and recommendations. There are two options here: hybrid-*switching* or *weighting*. Switching entails selecting individual predictions from each classifier, based on the assumption that some classifiers will be more accurate on a subset of instances than others [LAP09]. Weighting, on the other hand, combines the predictions of each classifier as a set of linear equations [ZWSP08]; weights are typically found by means of regression. First, the training data is further split into *training* and *probe* subsets. The algorithms are then given the training subset and queried with the probe instances.

Given a set $P = [a, b, ..., z]$ of predictors weighted by a vector $w = [w_a ... w_z]$, a vector of probe ratings $r \in [r_1 ... r_n]$, and predictions $\hat{r}_{x,n}$ by classifier x for item n, predictions of each probe rating can be formulated as a linear combination of each classifier's prediction:

$$\hat{r}_n = \sum_{c \in P} (\hat{r}_{c,n} \times w_c) \tag{2.11}$$

If the set of predictions from each classifier is combined into a single matrix X, the problem of finding a classifier weight vector can be expressed in matrix form:

$$\begin{bmatrix} \hat{r}_{a,1} & \hat{r}_{b,1} & ... & \hat{r}_{z,1} \\ \hat{r}_{a,2} & \hat{r}_{b,2} & ... & \hat{r}_{z,2} \\ ... & ... & ... & ... \\ \hat{r}_{a,n} & \hat{r}_{b,n} & ... & \hat{r}_{z,n} \end{bmatrix} \begin{bmatrix} w_a \\ w_b \\ ... \\ w_z \end{bmatrix} = \begin{bmatrix} r_1 \\ r_2 \\ ... \\ r_n \end{bmatrix} \tag{2.12}$$

The idea is to minimise the error between the weighted predictions matrix X and the vector of ratings r by simultaneously solving a set of linear equations. To solve for the weights w, each side of the equation must be multiplied by the inverse X^T of X:

$$\left(X^T X\right) w = \left(X^T r\right) \tag{2.13}$$

The same weights that have been learned are then applied to the full training set in order to predict any unrated items.

There are other techniques available for blending, based on gradient boosted decision trees [Kor09b], neural networks [PC09], and kernel ridge regression [TJB09]; it is beyond the scope of this thesis to cover all of these in detail. In the context of the Netflix prize, hundreds of predictors were blended together to produce the final solution. However, this is where the border between using CF to solve a *prediction* problem or build a recommender system lies. In fact, the chief product officer at Netflix stated[1] in an interview that:

[1] $http://www.appscout.com/2009/09/netflix\_1m\_prize\_winners\_inclu.php$

> "There are several hundred algorithms that contribute to the overall 10 percent improvement - all blended together," Hunt said."In order to make the computation feasible to generate the kinds of volumes of predictions that we needed for a real system we've selected just a small number, two or three, of those algorithms for direct implementation."

2.2.6 Online Algorithms

All of the above algorithms share a common trait: they are all trained and queried offline; thus reinforcing why system administrators need to iteratively retrain their system in order to include the latest ratings. However, there is a separate class of techniques—online algorithms—that dynamically update as ratings are introduced into the system. Online algorithms are usually decomposed into four steps: (a) receiving an instance to predict (e.g., a user-movie pair), (b) predicting the value (rating) of that instance, (c) receiving the true rating input by the user, and finally (d) updating itself according to some predetermined loss function. There are a number of examples where online algorithms have been applied to collaborative filtering contexts. These range from slope one approaches [LM05] to online matrix factorisation feature update [RST08].

One of the main challenges facing online algorithms is that the way they work is not a perfect match with how recommender systems work. First, algorithms are not given a *single* instance to predict at any time; they are given a (large) *set* of instances—all the items that a particular user has not rated. Predictions of this set will then be used to generate a ranked list of recommendations. They will also only receive feedback on a subset of these predictions; there are no guarantees governing if and when they will receive the true rating for an item, and how the update will be biased by the predictions that it has made itself. Lastly, since items need to be ranked, the algorithm will need to be queried prior to the user interacting with the system; it may also be excessive to re-predict all instances when the user inputs a single rating. In fact, at this point it becomes difficult to distinguish between how online algorithms will be used differently to the offline ones explored above.

2.2.7 From Prediction to Recommendation

Once predicted ratings have been generated for the items, and sorted according to predicted value, the top-N items can be proposed to the end user as recommendations. This step completes the process followed by recommender systems, which can now elicit feedback from the user. User profiles will grow, and the recommender system can begin cycling through the process again: re-computing user similarity measures, predicting ratings, and offering recommendations.

It is important to note that the user interface of the system plays a vital role in this last step. The interface not only determines the ability the system has to present generated recommendations to the end user in a clear, transparent way, but will also have an effect on the response that the user gives to received recommendations. Wu and Huberman [WH07b] conducted a study investigating the temporal evolution of opinions of products posted on the web. They concluded that if the aggregate rating of an item is visible to users and the cost of expressing opinions for users is low (e.g. one click of a mouse), users will tend to express either neutral ratings or reinforce the view set by previous ratings. On the other hand, if the cost is high (such as requiring users to write a full review), users tended to offer opinions when they

felt they could offset the current trend. Changing the visibility of information and the cost imposed on users to express their opinions, both determined by the interface provided to end users, will thus change the rating trend of the content, and the data that feeds into the filtering algorithm.

2.3 Trust and User Modelling

The previous section highlighted the data mining approaches to CF; what follows is a review of state of the art research that aims to augment these techniques by incorporating *user models* into collaborative filtering. In particular, we focus on the use of *trust* in recommender systems. However, before we proceed, we explore trust itself: what is trust? How has it been formalised as a computational concept?

A wide range of research [ARH98, JIB07, AH08] stems from sociologist Gambetta's definition of trust [Gam90]. Gambetta states:

> "trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent will perform a particular action"

Trust is described as the level of belief established between two entities in a given context. Discussing trust as a probability paved the way for computational models of trust to be developed, as first explored by Marsh [Mar94] and subsequently by a wide range of researchers [Gol08]. The underlying assumption of trust models is that users' (or agents', peers', etc) historical behaviour is representative of how they will act in the future: much like CF, the common theme is one of *learning*. The differences between the two emerges from the stance they adopt toward their target scenarios; unlike CF, trust models are often adopted as a control mechanism (by, for example, rewarding good behaviour in commerce sites with reputation credit) and are user-centred techniques that are both aware and responsive to the particular characteristics desired of the system (such as, in the previous example, reliable online trade).

Trust models have been applied to a wide range of contexts, ranging from online reputation systems (e.g. eBay.com) to dynamic networks [CNS03] and mobile environments [QHC06]; a survey of trust in online service provision can be found in [JIB07]. Due to its widespread use, trust modelling may draw strong criticism with regards to its name: it is arguable that, in many of these contexts, "trust" is a vague synonym of "reliability," "competence," "predictability," or "security." However, encapsulating these scenarios under the guise of trust emphasises the common themes that flow between them; namely, that researchers are developing mechanisms for users to operate in computational environments that mimic the way humans interact with each other outside of the realm of information technology.

2.3.1 Motivating Trust in Recommender Systems

The motivations for using a notion of trust in collaborative filtering can be grouped into three categories:

1. In order to accommodate the **explainability** required by system users. Both Tintarev [TM07] and Herlocker *et al* [HKR00] discuss the effect explainability has on users' perceptions of the recommendations that they receive, especially those recommendations that are significantly irrelevant or disliked by the users. Chen and Pu [PC06] further investigate this issue by building explanation interfaces that are linked to, and based on, a formal model of trust. Although a major compo-

nent of these works revolve around presenting information to the end users, they recognise that building an explainable algorithm is a key component of transparency: it converts a "black-box" recommendation engine into something to which users can relate.

2. To address **data sparsity**. The volume of missing data has a two-fold implication. First, new users cannot be recommended items until the system has elicited preferences from them [RAC+02]. Even when ratings are present, each of a pair of users who may actually share common *interests* will never be cited in each other's neighbourhood unless they share *ratings* for items: information cannot be propagated beyond each user's neighbourhood. Second, computed similarity will be incomplete, uncertain, and potentially unreliable.
3. To **improve the robustness** of CF to malicious attacks. Since recommender systems are often deployed in an e-commerce environment, there are many parties who may be interested in trying to exploit the system for their benefit, using what are known as *shilling* attacks [MBW07]. From the point of view of the ratings themselves, it is difficult to differentiate between what was input by honest users and the ratings that have been added in order to perform an attack. Trust models come to the rescue: by augmenting traditional collaborative filtering with a notion of how users interact, the robustness of recommender systems can be improved.

A consequence of incorporating trust models into CF is also often a measurable benefit in terms of prediction accuracy; however, state of the art algorithms that are *only* tuned for accuracy [BK07] do not mention trust models at all.

2.3.2 Using Trust For Neighbour Selection

One of the central roles that trust modelling has served in CF is to address the problem of *neighbour selection*, by replacing the kNN *neighbourhood formation* step above. Traditional approaches to CF are based on populating users' kNN neighbourhood with others who share the highest measurable amount of similarity with them [HKBR99]. However, these methods do not guarantee that the right neighbours will be selected; the aim of using trust is thus to capture information that resides outside of each user's local *similarity* neighbourhood in a transparent, robust and accurate way.

Two main approaches have been adopted: *implicit* methods, which aim to infer trust values between users based on item ratings, and *explicit* methods, that draw trust values from pre-established (or manually input) social links between users. Both methods share a common vision: the underlying relationships (whether inferred or pre-existing) can be described and reasoned upon in a web of trust, a graph where users are nodes and the links are weighted according to the extent that users trust each other.

Computing Implicit Trust

The first perspective of trust in CF considers values that can be inferred from the rating data: a web of trust between users is built from how each user has rated the system's content. In these cases, trust is used to denote *predictability* and to allow the different ways that users interact with the recommender system; in fact, many of these measures build upon error measures, such as the mean absolute error.

For example, Pitsilis and Marshall focus on deriving trust by measuring the uncertainty that sim-

ilarity computations include [PM04, PM05]. To do so, they quantify the uncertainty $u(a, b)$ between users a and b, which is computed as the average absolute difference of the ratings in the intersection of the two user's profiles. The authors scale each difference by dividing it by the maximum possible rating, $max(r)$:

$$u(a,b) = \frac{1}{|R_a \cap R_b|} \sum_{i \in (R_a \cap R_b)} \left(\frac{|r_{a,i} - r_{b,i}|}{max(r)} \right) \tag{2.14}$$

The authors then use this uncertainty measure in conjunction with the Pearson correlation coefficient to quantify how much a user should *believe* another. In other words, trust is used to scale similarity, rather than replace it. Similarly, O'Donovan and Smyth define a trust metric based on the recommendation error generated if a single user were to predict the ratings of another [OS05, OS06]. The authors first define a rating's correctness as a binary function. A rating $r_{b,i}$ is *correct* relative to a target user's rating $r_{a,i}$ if the absolute difference between the two falls below a threshold ϵ:

$$correct(r_{a,i}, r_{b,i}) \iff |r_{a,i} - r_{b,i}| \leq \epsilon \tag{2.15}$$

The notion of correctness has two applications. The first is at the *profile* level, $Trust^P$: the amount of trust that user a bestows on another user b is equivalent to the proportion of times that b generates correct recommendations. Formally, if $RecSet(b)$ represents the set of b's ratings used to generate recommendations, and $CorrectSet(b)$ is the number of those ratings that are *correct*, then profile-level trust is computed as:

$$Trust^P(b) = \frac{|CorrectSet(b)|}{|RecSet(b)|} \tag{2.16}$$

The second application of Equation 2.15 is item-level trust, $Trust^I$; this maps to the reputation a user carries as being a good predictor for item i, and is a finer-grained form of Equation 2.16, as discussed in [OS05]. Both applications rely on an appropriate value of ϵ: setting it too low hinders the formation of trust, while setting it too high will give the same amount of trust to neighbours who co-rate items with the current user, regardless of how the items are rated (since *correct* is a binary function). Similar to Pitsilis and Marshall, this metric also operates on the intersection of user profiles, and does not consider what has not been rated when computing trust.

In [LHC08c], the authors approach trust inference from a similar perspective, but extend it from a binary to continuous scale and include ratings that fall outside of the profile intersection of a user pair. Rather than quantifying the correctness of a neighbour's rating, the *value* that b's rating of item i would have provided to a's prediction, based on a's rating:

$$value(a, b, i) = 1 - \rho |r_{a,i} - r_{b,i}| \tag{2.17}$$

This equation returns 1 if the two ratings are the same, and 0 if user b has not rated item i; otherwise, its value depends on the *penalising* factor $\rho \in [0, 1]$. The role of the penalising factor is to moderate the extent to which large differences between input ratings are punished; even though the two ratings may diverge, they share the common feature of having been input to the system, which is nevertheless relevant in sparse environments such as CF. A low penalising factor will therefore have the effect of populating neighbourhoods with profiles that are very similar in terms of what was rated, whereas a high

penalising factor places the emphasis on how items are rated. In [LHC08c], the authors use $\rho = \frac{1}{5}$. The trust between two users is computed as the average value b's ratings provide to a:

$$trust(a,b) = \frac{1}{|R_a|}\left(\sum_{i \in R_a} value(a,b,i)\right) \tag{2.18}$$

This trust metric differs from that of O'Donovan and Smyth by being a pairwise measure, focusing on the value that user b gives to user a. Unlike the measures explored above, the value sum is divided by the size of the target user's profile, $|R_a|$, which is greater than or equal to the size of the pair's profile intersection, $|R_a \cap R_b|$, depending on whether a has rated more or fewer items than b. This affects the trust that can be awarded to those who have the sparsest profiles: it becomes impossible for a user who rates a lot of content to trust highly those who do not, while not preventing the inverse from happening.

The three methods we have presented here are not the only proposals for trust inference between users in CF contexts. For example, Weng *et al* [WMG06] liken the CF web of trust structure to a distributed peer-to-peer network overlay and describe a model that updates trust accordingly. Hwang and Chen [HC07] proposed another model that again marries trust and similarity values, taking advantage of both trust propagation and local similarity neighbourhoods. Papagelis *et al* [PPK05] do not differentiate between similarity and trust, by defining the trust between a pair of users as the correlation their profiles share; they then apply a propagation scheme in order to extend user neighbourhoods.

Many of the problems of computed trust values are akin to those of similarity; for example, it is difficult to set a neighbourhood for a new user who has not rated any items [RAC+02]. However, the characteristics of trust modelling allow for solutions that would not emerge from similarity-centric CF. For example, [LHC08c] includes a constant *bootstrapping* value β for users who have rated no items, which translates to initial recommendations that are based on popularity, and would become more personalised as the user inputs ratings.

All of the methods we have explored share the common theme of using error between profiles as an indication of trust. Similarly, there is a broad literature on similarity estimation that does not adopt the language of trust modelling, such as the "horting" approach by Aggarwal *et al* [AWWY99] and the probabilistic approach by Blanzieri and Ricci [BR99]. In all of the above, each user pair is evaluated independently; the significant differences appear in how each method reflects an underlying user model of trust.

Selecting Neighbours Explicitly

The alternative to computing trust values between users is to transfer pre-existing social ties to the recommender system. There are two approaches that have been followed here: on the one hand, users may be asked explicitly to select trustworthy neighbours. On the other hand, social ties may be drawn from online social networks where it is possible to identify each user's friends.

Massa and Avesani describe trust-*aware* recommender systems [MB04, MA07]. In this scenario, users are asked to rate both items and *other users*. Doing so paves the way to the construction of an explicit web of trust between the system users. Since users cannot rate a significant portion of the other users, the problem of sparsity remains. However, assuming that user-input trust ratings for other users are

more reliable than computed values, trust can then be propagated to broaden each user's neighbourhood.

Trust propagation is a highly *explainable* process: if a trusts b, and b trusts c, then it is likely that a will trust c. However, this transparency is obscured as the propagation extends beyond a two-hop relationship. The validity of propagation rests on the assumption that trust is transitive, an assumption that can be challenged once the propagation extends beyond "reasonable" limits. In small-world scenarios (such as social networks), this limit is likely to be less than the famed six-degrees of separation, since it is apparent that people do not trust everyone else in an entire social network. Much like similarity and computed trust, the efficiency of trust propagation is therefore dependent on the method used and the characteristics of the underlying data.

A range of other works centre their focus on the social aspect of recommendations. For example, Bonhard and Sasse [Bon04, PBH07] perform a series of experiments that analyse users' perception of recommendations: they conclude that participants overwhelmingly prefer recommendations from familiar (as opposed to similar) recommenders. The experiments reflect the ongoing asymmetry between algorithmic approaches to CF, which tend to focus on predictive accuracy, and user studies that mainly consider recommender system interfaces. It is difficult to evaluate one independently of the other, and Bonhard's motivations for the use of social networks echo those used to motivate the use of trust models in Section 2.3.1: in order to reconcile the end users' mental model of the system and the system's model of the users.

Golbeck explored the power of social networking in the FilmTrust system [Gol06], showing that these systems produce comparable accuracy to similarity-based CF. The application of social networks can also be beneficial to CF since relationships in the web of trust can be augmented from simple weighted links to annotated, contextual relationships (i.e., b is my sister, c is my friend). Context-aware recommender systems is a nascent research area; Amodavicius *et al* [ASST05] provide a first view into this subject by looking at multi dimensional rating models. Full coverage of this falls beyond the scope of this chapter; however, it is apparent how network ties can be fed into mechanisms that include who and where the users are before providing recommendations.

The main criticism of many of these approaches is that they require additional explicit input from the end user; in effect, they move against the fully automated view of recommender systems that original collaborative filtering proposed. However, social networks are on the rise, and users proactively dedicate a significant portion of time to social networking. The implementation of these methods therefore aims to harness the information that users input in order to serve them better.

It is important to note that both the computed and explicit methods of finding trustworthy neighbours are not in conflict; in fact, they can be implemented side by side. Both require users to be rating items in order to provide recommendations, while the latter also requires social structure. Popular social networking sites, such as Facebook[2] include a plethora of applications for which users are requested to rate items, making the conjunction of the two methods ever easier.

[2] http://www.facebook.com/apps/

2.3.3 Trust-Based Collaborative Filtering

Once neighbours have been chosen, content can be filtered. However, there are a range of choices available to do so; in this section we outline the methods implemented by the researchers we discussed in the previous section. The approaches revolve around rating aggregations; in other words, taking a set of neighbour ratings for an item and predicting the user's rating using Equation 2.8. The difference between each method is (a) what neighbours are selected, and (b) how the ratings from each neighbour are weighted. We split the methods into three strategies, trust-based *filtering*, *weighting*, and social filtering.

1. **Trust-Based Filtering**. In this case, neighbours are selected (filtered) using computed trust values. The ratings they contribute are then weighted according to how similar they are with the target user.
2. **Trust-Based Weighting** departs fully from similarity-based CF: neighbours are both selected and their contributions weighted according to the trust they share with the target user.
3. **Social Filtering**. Neighbours are selected based on the social ties they share with the target user. Ratings can then be weighted according to either their shared similarity or trust with the target user.

All of these methods assume that users will be using the rating scales symmetrically, i.e. if two users predict each other perfectly, then the difference $(r_{a,i} - \bar{r}_a)$ will be the same as $(r_{b,i} - \bar{r}_b)$, regardless of what each user's mean rating actually is. In practice, this is not always the case: predictions often need to be changed to fit the rating scale, since users each use this scale differently. This notion was first explored in the aforementioned work by Aggarwal *et al* [AWWY99], who aimed to find a linear mapping between different users' ratings. However, [LHC08c] extends this notion to encompass what they refer to as *semantic distance*, by learning a non-linear mapping between user profiles based on the rating contingency table between the two profiles. The results offer accuracy benefits in the MovieLens dataset, but do not hold in all cases: translating from one rating scheme to another is thus another research area that has yet to be fully explored.

The above work further assumes that the underlying classifier is a kNN algorithm. Recent work, however, has been moving away from kNN-based recommender systems. In fact, the data derived from users telling the system whom they trust can also be input into other algorithms, such as matrix factorisation techniques [MYLK08, MKL09]. In these works, Ma *et al* describe matrix factorisation models that account for both what users rate (their preferences) and to whom they explicitly connect (who they trust). While certainly beneficial to cold-start users, introducing trust data into factorisation models reignites the problem of *transparency*: how will users understand how their input trust values contribute to their recommendations? A potential avenue for research lies in the effect that hybrid trust models have on users. For example, Koren describes how a neighbourhood and factorisation model can be combined [Kor08], and this work may begin to bridge the chasm between the factorisation-based and kNN-based use of trust in recommender systems.

2.4 Evaluating Recommendations

In order to design and evaluate CF algorithms, researchers require three components: (a) a *dataset* of user ratings to which to apply the algorithm, (b) a *methodology* to conduct repeatable experiments, and (c) appropriate *metrics* to measure system performance. In this section, we review each of these components.

2.4.1 Rating Datasets

The rise of Web 2.0 sites equipped with developer application programming interfaces (APIs) to access user data is improving both the *ease of access* and *wealth of data* available to CF research. Dell'Amico and Capra [DC08] are but one example of researchers who have collected and experimented with a dataset of music preferences from Last.fm[3]. Amatriain *et al* [ALP+09] gathered movie rating datasets from Rotten Tomatoes[4] and Flixster[5]. However, requiring researchers to each crawl the web for a dataset is not only an unnecessary overhead, but does not allow for results to be compared between different researchers' experiments if the dataset is not shared. There is also a number of pre-packaged rating datasets. In this thesis we focus on two; the largest and most widely studied datasets of ratings:

- **Netflix Prize Dataset**: The largest of the available datasets, this set of movie ratings consists of 100 million ratings, by $480,189$ users who have rated one or more of the $17,770$ movies on a $1-5$ star scale.
- **MovieLens Dataset**: There are currently three movie-rating datasets available from the GroupLens website[6]. The first includes $100,000$ ratings of $1,682$ movies by 943 users; the second has 1 million ratings for $3,900$ movies by $6,040$ users; the last includes 10 million ratings and $100,000$ tags for $10,681$ movies by $71,567$ users. As with the Netflix dataset, items are rated on a $1-5$ star scale. In this thesis, we have used the first two datasets since the third dataset was only released in 2009.

Other available datasets include the *Jester* joke-rating dataset, and the *Book-Crossing* dataset of book ratings. This is by no means an exhaustive list of datasets; for example, Github[7] (an online collection of source code repositories) released a dataset of the repositories "watched" by each of its members, and ran a competition that also aimed at designing an algorithm to recommend repositories to its users.

2.4.2 Methodology

The aim of an experiment with user data is to manipulate the ratings in a way that (a) reflects assumptions of how the users will interact with the end system, and (b) produces measurable results that mirror the users' experience.

The assumptions that researchers make with regards to the former goal are twofold: first, that users will have already provided *at least* one rating to the system. Given that only the ratings are being

[3] http://www.last.fm
[4] http://www.rottentomatoes.com
[5] http://www.flixster.com
[6] http://www.grouplens.org
[7] http://contest.github.com

used to generate recommendations (demographic or item-attribute data is unavailable), then lack of any rating data does not allow CF algorithms to produce personalised results. Second, if a user's preferences were already known, then ranking items according to the user's ratings would provide the most useful recommendations. Therefore, the main focus of CF evaluation has historically been that of *predicting* the ratings of items that users have not rated.

To evaluate how well an algorithm is accomplishing the task of providing recommendations, researchers use one of the available rating datasets. The dataset is first partitioned into two subsets; the first acts as a *training* set that will be available for the algorithm to learn from. The second subset is the *test* set, with rating values that remain hidden to the algorithm. An evaluation will query the algorithm to make predictions on all the items in the test set. Results are then cross-validated by repeating experiments on multiple partitionings of the data. Hidden in these steps is an assumption that the performance of a *deployed* algorithm (with an online recommender system) will be comparable to that of the selected training and test sets.

However, one of the fundamental problems with this methodology is related to the second goal: the extent to which each user's *qualitative* experience with the system can be translated into measurable, *quantitative* results is questionable. In effect, researchers reduce the problem of generating interesting, serendipitous, and useful recommendations into one of accurate preference prediction. In the following section, we review the metrics used for this task and discuss the controversy surrounding their use.

2.4.3 Metrics

The most widely adopted metrics used to evaluate the efficacy of CF algorithms deal with prediction accuracy. Available measures of statistical accuracy include the mean absolute error (MAE) and the root mean squared error (RMSE):

$$MAE = \frac{\sum_N |r_{a,i} - \hat{r}_{a,i}|}{N} \tag{2.19}$$

$$RMSE = \sqrt{\frac{\sum_N (r_{a,i} - \hat{r}_{a,i})^2}{N}} \tag{2.20}$$

Both of the above measures focus on the difference between a rating of item i by user a, $r_{a,i}$, and the prediction for the same user and item, $\hat{r}_{a,i}$. In general, both metrics measure the same thing and will thus behave similarly; the difference lies in the degree to which different mistakes are penalised. There are a number of modified mean error metrics that aim to introduce fairer representations of the system users. For example, Massa uses the mean absolute *user* error in order to compensate for the fact that more predictions are often made for some users rather than others [MA07] and O'Donovan and Smyth compare algorithm performance by looking at how often one is more accurate than another [OS05].

Coverage metrics complement accuracy results: they aim to quantify the breadth of predictions that are possible using a given method. They compare the proportion of the dataset that is uncovered to the size of the test set, in order to measure the extent that predictions were made possible using the current algorithm and parameters.

While prediction accuracy continues to receive significant attention, due to it being the metric of choice of the Netflix prize, the focus on accuracy in recommender research continues to be disputed.

While we already described the difficulty of translating qualitative experience into quantitative values, there are a number of further reasons for this:

- **Accuracy Metrics Focus on Predictions** that have been produced, regardless of whether the prediction was at all possible or not. Massa and Avesani [MA07] show that prediction accuracy continues to perform well when pre-defined values are returned. For example, if each prediction simply returned the current user mean (thus not allowing content to be ranked and converted into recommendations), accuracy metrics would still not reflect such poor behaviour.
- **Test Set Items Have Already Been Rated**. An evaluation method that only makes predictions on items in the test set (items that the user has rated) may tend to show good performance, especially if there is low variance in the way users rate. Real systems, that have to provide recommendations based on making predictions on *all* unrated items, may have much worse performance.
- **Predictions vs. Recommendations**. McLaughlin and Herlocker [MH04] argue that striving for low mean errors biases recommender systems towards good *predictors* rather than *recommenders*. In other words, an error in a prediction affects the mean error the same way, regardless of whether the prediction enabled the entry to qualify as a recommendation or not.

Mean errors will therefore not tend to reflect the end user experience. Concerns over accuracy-centric research continues; McNee *et al* [MRK06] even argued that striving for accuracy is detrimental to recommender system research, and propose that evaluations should revert to user-centric methods. Accuracy metrics persist, however, due to the need for empirical evaluations of filtering algorithms that can compare the relative performance of different techniques without including the subjective views of a limited (and, more often than not, inaccessible) group of test subjects. Furthermore, while it remains difficult to understand exactly how accurate predictions translate into usefully ranked recommendations, accuracy is a useful metric for understanding the extent to which an algorithm is approximating the preference values input by the system users.

Examining an algorithm from the point of view of top-N recommendations provides an alternative means of evaluation; rather than considering the predictions themselves, information retrieval metrics (i.e., precision and recall) are used on a list of items sorted using the predictions. However, sorting a list of recommendations often relies on more than predicted ratings: for example, how should one sort two items that both have the highest possible rating predicted? How large should the list size N be? These kinds of design decisions affect the items that appear in top-N lists, and has motivated some to change from deterministic recommendation lists to looking at whether items are "recommendable" (i.e., their prediction falls over a predefined threshold) or not [ALP+09].

Other error measures have been applied when analyzing the accuracy of a filtering algorithm, including receiver-operating characteristic (ROC) sensitivity [HKBR99]. This metric aims at measuring how effectively predicted ratings helped a user select high-quality items. Recommendations are therefore reduced to a binary decision: either the user "consumed" the content (i.e., watched the movie, listened to the song, read the article) and rated it, or did not. By comparing the number of false-positives, or items that should have been recommended that were not, and false-negatives, or not recommending an

Source	Problems	Example Resources
Data	Sparsity	[MKL07]
	Noise	[APO09, HKV08]
	Cold-Start	[NDB07, PPM+06, RAC+02]
	Privacy	[LHC07, Can02, BEKR07]
	Popularity Bias	[CC08]
Algorithm	Accuracy	[BK07]
	Latency	[GRGP00]
	Scalability	[BK07]
System	Robustness	[MBW07, LR04]
	Distributed	[Zie05]
User	Explainability	[HKR00, PC06]
	Context	[ASST05]
	Implicit Trust	[MA07]
	Explicit Trust	[Bon04, Gol06]
Evaluation	Metrics	[HKTR04, BHK98]
	Accuracy	[MRK06]
	User Experience	[MH04]

Table 2.1: A Sample of Open Problems in Recommender Systems

item that should have been, this metric aims at measuring the extent to which the recommender system is helping users making good decisions. However, this method relies on a prediction score threshold that determines whether the item was recommended or not, which often does not translate to the way that users are presented with recommendations. A comprehensive review of metrics used when evaluating CF can be found in [HKTR04].

2.5 Open Problems

There is a wide variety of problems that state of the art recommender systems face. These range from *user-centric issues* (poor recommendation explainability, users distrusting recommendations), to *data-related problems* (namely, the adverse consequences of data sparsity), *algorithm-related challenges* (latency, scalability, accuracy), *system-wide weaknesses* (vulnerability to attack), and *evaluation-related issues* (or, how to best evaluate CF). A summary of these open issues, along with pointers to relevant research in each direction, is provided in Table 2.1. Each category that we have enumerated, however, is intertwined with the others: for example, *data* cold-start issues are related to *user* preference elicitation, and affect the *algorithm*'s accuracy. This section enumerates the open research problems pertaining to recommender systems that we addressed in this thesis.

2.5.1 Ratings: Changing Over Time

Given the number of users and items in a typical recommender system, the rating matrix tends to be extremely sparse. The problem of data sparsity highlights the dependence that filtering algorithms have on the altruism of the community of users making use of the recommender system; if users do not rate

items then the cyclical process of generating recommendations cannot be completed. Both current state of the art algorithms and evaluation techniques are aware of this shortcoming; however, they both also view the matrix as a *static* data collection. In doing so, they assume that (a) changes in the data over time are not relevant when predicting user preferences, and (b) the current data is sufficient for computing appropriate recommendations (e.g., to find similar kNN neighbourhoods).

The relationship between the assumption of persisting like-mindedness and the reality of neighbourhood patterns over time in recommender systems remains largely unexplored. We thus explore ratings over time: delving into how datasets change over time and the different patterns of behaviour that emerge. We show that observing a snapshot of similarity between all the system users is both subjective to how the similarity is computed, unreliable, and difficult to differentiate from a random process. By modelling neighbourhood-based CF as a graph that is iteratively updated, we highlight a significant break between the assumptions and operation of CF. Analysis of the ratings over time lays the foundation for our proposed methodology that modifies traditional CF evaluation by including a notion of temporal updates.

2.5.2 Methodology & Evaluation

In Section 2.4.2, we recounted how CF algorithms are evaluated: datasets are split and algorithms are queried about a test set, after learning from the training set. Repeating this process provides a cross-validated snapshot of the expected performance of an algorithm when trained with a dataset with those particular characteristics. However, deployed recommender systems do not handle unchanging datasets: they are iteratively updated on *growing* sets of user ratings. Deployed systems need to cope with a growing set of both users and items; unfortunately, though, they are not evaluated as such. In Chapter 4 we propose a novel method to perform CF experiments that includes the notion of temporal updates. We extend current accuracy-based metrics onto the temporal dimension, and evaluate the performance of a series of CF algorithms over time. Lastly, we show that introducing an awareness of temporal updates into the algorithm's operation can offer improved temporal accuracy.

While accuracy remains a crucial focal point of recommender system evaluation, the controversy that surrounds its use reflects the fact that it only highlights a single dimension of performance: how close the predictions are to the user input values. There are a wide range of qualities that may be desired of a recommender system, such as diversity. The problem of diversity has already been explored by Smyth and McLave [SM01]. If a user rates an item (for example, rating an album by The Beatles), loading the user's recommendations with extremely similar items (i.e., all of the other albums by The Beatles) is often not helpful at all; the user has not been pointed towards new information, and is only inundated with recommendations towards content that is probably known already. The question therefore becomes: to what extent do filtering algorithms generate diverse recommendations? The temporal dimension to this problem considers the sequence of recommendations with which users are provided as they interact with the system: to what extent does the system compute the *same* recommendations over and over? In Chapter 5, we explore the temporal diversity of CF algorithms and the relationship they share with temporal accuracy. We then design and evaluate a hybrid algorithm that increases temporal diversity.

2.5.3 System Robustness

The last problem we consider here relates to *system vulnerabilities*. Attackers may aim to modify the recommendations that are output by a system for any number of selfish reasons. They may wish artificially to promote a piece of content, to demote content (perhaps since it competes with the attacker's content), or to target a specific audience of users. These attacks are aided by the near-anonymity of users participating in the recommender system. In some cases, signing up to an e-service that uses recommender system technology only requires an email address. Creating a number of fake accounts is thus not beyond the realms of possibility; furthermore, if each of these fake accounts is treated as an honest user, it becomes possible to change the output of recommender systems at will.

Research in the field of recommender system vulnerabilities can be divided into two categories. On the one hand, system administrators require a means of identifying attacks by being able to recognise both when an attack is occurring and which users are malicious. On the other hand, the vulnerabilities themselves are addressed; how can these attacks be prevented? How can the cost or effect be minimise? A comprehensive review of the vulnerabilities of collaborative recommender systems and their robustness to attack can be found in Mobasher *et al* [MBW07].

The growing body of literature addressing CF attacks takes a binary view of the rating matrix: either it has been attacked, or it has not. The former is created using the latter along with an appropriate attack model, and the objective of attack detection algorithms is to separate the honest and dishonest profiles. However, as recommender systems are updated over time, this scenario is unlikely to appear in deployed systems: sybils' profiles may be built *over time*, and may be able to inflict damage prior to being in the detectable state that they are when evaluated by researchers. In Chapter 6, we address this problem by designing and evaluating a series of monitors that alleviate the impact of these attacks by detecting behavioural shifts in the system that indicate ongoing anomalous activity.

2.6 Summary

In this chapter, we have introduced the data, algorithms, and open problems related to CF recommender systems. CF automates the process of generating recommendations by drawing from its assumption of like-mindedness between its users. In other words, people who have displayed a degree of similarity in the past will continue sharing the same tastes in the future. The model of users held by these systems therefore focuses on the set of preferences that each individual has expressed, and interactions between users can be determined according to values derived by operating on the information available in users' profiles.

The approaches themselves, however, originate from a wide variety of backgrounds, and include content-based methods, which infer recommendations from item attributes and machine-learning inspired solutions; we discussed baseline, kNN, matrix factorisation, and hybrid approaches. Nearest neighbour algorithms follow a three-stage process: finding a set of recommenders for each user based on a pre-defined measure of similarity, computing predicted ratings based on the input of these recommenders, and serving recommendations to the user, hoping that they will be accurate and useful

suggestions. The choice of what method to implement relies on a fine balance between accuracy and performance, and is also dependent on the specific context that recommendations need to be made for. Each method has its own strengths and weaknesses, and hybrid methods attempt to reap the best of both worlds by combining a variety of methods.

The general problems faced by recommender systems remain the same, regardless of the approach used to build the filtering algorithm. These problems were grouped into a set of categories: problems originating from the data, algorithm, system, users, and evaluation. In this thesis, we tackle a fundamental issue underlying all approaches that have been proposed before: that is, how to model and evaluate a system that will be deployed over time.