TÉCNICO LISBOA

A Recommender System for Automation Rules in the Internet of Things

Diogo Alexandre Inácio Franco

Thesis to obtain the Master of Science Degree in

Information Systems and Computer Engineering

Supervisors

Dr. Fernando Mira da Silva
Dr. Luís Miguel Martins Nunes

Instituto Superior Técnico

May 2017

He who wonders discovers that this in itself is wonder.

M. C. Escher

Acknowledgments

I want to thank my parents and my brother for their unconditional support throughout my academic path.

I also want to thank everyone at Muzzley for their invaluable help in this work. My suggestion for the topic of this thesis was kindly and quickly accepted, and even in the face of changing priorities, as is typical of a fast moving startup, everyone made amazing efforts to see this work completed. In particular I would like to thank:

Pedro Figueiredo, for motivating me to take on this course and for all the help and time he generously invested in discussing the ideas in this work.

Domingos Bruges, for offering me the time to work on this thesis at Muzzley and enforcing it regardless of any other pressing concerns, as well as for his agility in accommodating my many requests in short notice.

I would also like to thank my supervisors:

Fernando Mira da Silva for his invaluable insights in our meetings, and for finding the time for them in his busy schedule.

Luís Nunes, for the precious guidance, tireless reviewing, and helpful advice he consistently provided me. I would not have completed this work without his periodic requests for updates on our discussions.

Lastly, I would like to give the biggest thanks to Marisa, for her continuous help and motivation, unlimited patience while my focus was on this thesis, and for all the other reasons that would not fit in these pages.

Abstract

More and more devices featuring internet connectivity are being created every day. The value of such devices is in their ability to interact with one another, as part of automation workflows created by users to improve their home experience. Various platforms offer systems that allow for the creation of automation rules between different connected devices, but formulating and defining such automations can be quite complex.

This work explores the automated, personalized recommendation of automation rules for users of connected devices. The application of recommender systems' techniques to this domain is novel, so that exploring the applicability of the recommendation approaches discussed in the literature is one of the main contributions of this work. The hypothesis that certain groups of automations provide synergies was explored, and a system that exploits this by providing recommendations based on learning association rules was developed. Various strategies to make association rule mining feasible on an automation dataset were devised, such as generalizing automation rules into recommendable items, and applying a similarity operator to all items, making automations identifiable across users.

The developed system showed very promising results, under an evaluation methodology consisting of gathering precision, recall and coverage metrics for the most important hyperparameters, as well as when comparing it against a naive recommender developed as a baseline. Finally, it was discussed how the techniques developed in this work are generalizable to domains outside of automation rule territory.

Keywords

Recommender Systems, Automation Rules, Internet of Things, Personalization, Association Rules.

Resumo

Cada vez mais dispositivos suportando conectividade à internet são criados diariamente. O valor destes dispositivos reside na possibilidade de interagirem entre eles, como parte de regras de automação criadas por utilizadores por forma a melhorarem a sua experiência em casa. Várias plataformas oferecem sistemas que permitem a criação de regras de automação entre diferentes dispositivos, mas definir estas automações pode ser bastante complexo.

Esta tese explora a recomendação de regras de automação de forma automática e personalizada, a utilizadores de dispositivos ligados à internet. A aplicação das técnicas de sistemas de recomendação a este domínio é nova, pelo que a exploração da aplicabilidade das estratégias discutidas na literatura é uma das principais contribuições deste trabalho. Exploramos a hipótese de que determinados grupos de automações oferecem sinergias, e desenvolvemos um sistema baseado na aprendizagem de regras de associação. Várias estratégias para tornar possível esta aprendizagem com recurso a dados de automações foram estudadas, tal como a generalização de regras de automação para produtos recomendáveis, ou a aplicação de um operador de semelhança a todos os produtos, tornando as regras de associação inidentificáveis entre utilizadores.

O sistema desenvolvido mostrou resultados muito promissores, com uma metodologia de avaliação consistindo em obter métricas como precisão, sensibilidade e cobertura, bem como comparando-o com um sistema de recomendação mais simples, desenvolvido como referência. Finalmente, foi discutido como as técnicas desenvolvidas neste trabalho são generalizáveis para domínios que não o de regras de automação.

Palavras Chave

Sistemas de Recomendação, Regras de Automação, Internet das Coisas, Personalização, Regras de Associação.

Contents

List of Figures

List of Tables

Abbreviations

IoT Internet of Things

API Application Program Interface

RS Recommender System

CF Collaborative Filtering

CB Content-based Filtering

KB Knowledge-based

CL Common Lisp

IR Information Retrieval

1

Introduction

Contents

1.1 Background

The Internet of Things (IoT) is the network of devices featuring sensors and/or actuators as well as internet connectivity, allowing them to both interact with the environment and exchange data over the internet. Some estimations predict that this network will grow to encompass 50 billion connected objects by 2020 [1].

Because IoT devices are remotely addressable using the existing internet infrastructure, manufacturers often provide both software and Application Program Interfaces (APIs) to allow consumers and developers alike to interact with their own "things". This, however, has caused some amount of fragmentation in the interoperability that is possible between devices, which is a critical concern for the IoT sector [2]. For this reason, some companies are integrating many heterogeneous objects into one unified protocol, typically using the manufacturers' APIs, in order to abstract away each device specification and allow users to take advantage of synergies between their objects. Examples of consumer-facing companies tackling this problem are IFTTT[1] and Muzzley[2], which takes part in the present work by providing the data and infrastructure used.

The aforementioned companies provide software that allows users to create automation rules between connected devices (and other web services), which typically consist of programs of the form if-then, or slightly more complex rules. To illustrate, a user can create an automation rule of the form "When I turn on the TV, change the brightness of my light bulb to 40%", or "When my smoke detector fires, turn on all of my light bulbs". While these rules are usually defined explicitly by the IoT user, this work explores their automatic recommendation, in a personalized manner.

The software tools and techniques used to provide suggestions of items that might be of interest to users belong to the topic of Recommender Systems (RSs). RSs emerged as an independent research topic in the mid-90's [3] and have enjoyed an increase in popularity in recent years [4], likely due to the observed growth in every aspect of the online world, which has made the problem of information discoverability and selection increasingly relevant. RSs have been used successfully in a wide variety of domains, the most prominent and commonly cited being the Netflix[3] movie recommender [5], the Amazon[4] RS for generic products [6], the music recommenders built by companies like Spotify[5] [7], among many others, like recommenders for restaurants, news or people (for instance, in online dating). The main reasons as to why service providers implement RSs are to increase sales and/or increase user satisfaction and fidelity, by helping them cope with information overload [3].

1.2 Motivation

Companies such as Muzzley provide value to their customers by allowing them to interact with differently branded connected devices in a unified manner, as well as providing interoperability between

[1] www.ifttt.com
[2] www.muzzley.com
[3] www.netflix.com
[4] www.amazon.com
[5] www.spotify.com

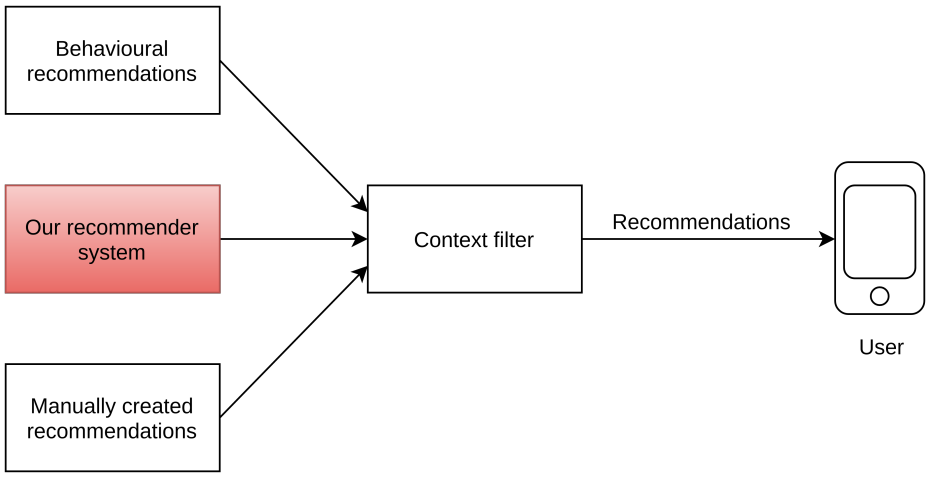

Figure 1.1: The Muzzley rule recommendation architecture. The highlighted system is the focus of this work.

them, via the definition of rules. However, explicitly defining automation rules between connected devices requires mental effort and some amount of creativity on the part of the user. Thus, we propose that the automated and personalized suggestion of if-then rules provides value to the end user and to the companies who provide the best suggestions in their ecosystems.

The development side of this work was performed at Muzzley, using their data. Muzzley has a few developed systems related to the recommendation of rules: A system that allows human staff to manually create rules and recommend them to all users, as well as one that recommends rules based on the actions that users do more often — the behavioural system. There is also another piece of software that filters the recommendations provided by the previous two systems and dispatches them to users based on their current context, such as time of day, or the amount of recommendations they have received lately. Figure 1.1 depicts this architecture.

The motivation for this work from Muzzley's perspective was to start taking advantage of information that is not currently exploited by existing systems, such as the rules that users create explicitly. Manually created recommendations allow Muzzley to educate users on interesting automations that can be defined, but have drawbacks such as not being personalized, or requiring human effort to create. The behavioural system, on the other hand, can only learn from actions realized often by the user. One drawback of this is that rules such as "When my smoke detector fires, turn on all of my bulbs", cannot be easily learned, since such situations should (hopefully) happen infrequently. However, if many users are creating this automation, Muzzley would like to be able to automatically suggest it to whom it might make sense. Thus, a RS that automatically recommends personalized automation rules stands to be beneficial to the company.

This work focuses on answering the question of how to effectively recommend automation rules in the internet of things. In this context, a recommendation consists of a suggestion of a program where the user is faced with a binary choice: to accept the recommendation, further programming his connected devices, or to decline it. An example of a suggestion is: "Do you want your lights to turn off automatically when you leave home?".

The main goals of this work are to discover if and which state-of-the-art recommender system's

techniques can successfully be applied to the suggestion of programs in the IoT domain, to understand how these programs can be abstracted into recommendable items, and to implement and evaluate a recommender engine that performs useful automation rule recommendations.

1.3 Contributions

Because the space of possible rules between connected "things" is quite large, requiring effort on the part of the user to select and construct appropriate automations, it is likely that this domain will benefit from the use of automated suggestions. Specifically, this work aims to contribute to an understanding of:

- The characteristics of IoT rules in the context of their recommendation, i.e., when viewed as *items* in a RS.
- Which data can be successfully exploited to produce useful recommendations.
- Which RS approaches and algorithms are suitable or can be adapted to the specificities of the IoT automation domain.

More generally, given the growing importance of the research and practical applications of recommender systems, as well as the growing rate of adoption of IoT devices by end-users, this thesis's topic stands to be of increasing importance for both researchers and companies in this space.

1.4 Outline

This section provides a brief description of the chapters present in this document.

In Chapter 2, the fundaments of recommender systems and its challenges are introduced, and an overview of the related work and state-of-the-art in the various types of recommender engines is presented.

Chapter 3 discusses the formalisms of automation rules, as well as the dataset used in this work and the characteristics of the automation rules domain.

The solution developed in this work is described in Chapter 4, where we discuss the approach taken, from generalizing automation rules into templates, to computing the similarity between them, mining association rules, and recommending automations.

The developed system is evaluated in Chapter 5, where the evaluation methodology is presented and the results discussed.

The main conclusions of this work are drawn in Chapter 6.

2

Related Work

Contents

RSs exist with the goal of generating useful personalized recommendations of items to users, where *item* is the generic term used to denote what the system recommends. To produce recommendations to a user, typically called the *active user* [8], recommenders use information about the users and their preferences, the items, domain knowledge, or combinations of these. The output is usually either a list of the top recommendations for the active user, or a probability representing a prediction for how much the user will like a given item.

The three most popular approaches to building RSs are Collaborative Filtering (CF), where the user is recommended items that people with similar tastes liked in the past, Content-based Filtering (CB), where the user is recommended items similar to the ones that he himself liked in the past, and Knowledge-based (KB) systems, that use domain knowledge about which item features meet the users' needs [4]. These classes of recommenders are detailed, respectively, in sections 2.1, 2.2 and 2.3 of this document. However, different taxonomies have been proposed. Some authors like [9] or [10] classify RSs into CF, CB and *Hybrid* systems, which combine methods of the previous two. This classification scheme sees knowledge-based approaches as techniques to augment Hybrid systems. Burke [11] proposes yet another taxonomy, where besides the CF, CB and KB approaches, he also classifies RSs into *Utility-based* and *Demographic*. Because Utility-based systems use a utility function over the space of items that describes the active user's preferences, they can be seen as a variant of KB systems, since the utility function is a specific encoding of domain knowledge. In a similar manner, Demographic recommendations can be seen as a specific CF approach that exploits additional demographic user information to determine similar users [4].

2.1 Collaborative Recommendation

To date, collaborative filtering is the most popular and successful approach for providing recommendations to users [8, 12]. The main assumption behind it is that if two users had similar tastes in the past — for instance, they listened to the same songs, or bought the same albums — they are likely to keep exhibiting similar behaviour in the future. Thus, if historically the opinions of two particular users A and B overlap substantially, and user A expresses a new positive opinion on an item, it makes sense to suggest that item to user B.

A user opinion on an item is typically denoted by *rating*. Ratings can be gathered explicitly, by asking the user directly, or measured implicitly by, for example, analyzing which items the user has bought or which product pages he has visited. When asked explicitly, 5 or 7 point Likert scales [1] are often used, although some previous work [13, 14] discusses the impact of the chosen scale on the ratings provided by users, arguing for more granular options so as not to lose precision on the opinions.

CF algorithms manage a list of users $U = \{u_1, u_2, \ldots, u_n\}$ and a list of items $I = \{i_1, i_2, \ldots, i_m\}$, both finite. Associated to each user there is also a list of the items that he expressed his opinion on. Pure CF approaches use only this information to produce recommendations, usually in the form of a

[1] A well researched approach to scale responses in surveys, usually ranging from "Strongly dislike" to "Strongly like".

matrix $U \times I$ containing the user's ratings. Thus, these systems do not need to explicitly model each item, allowing them to recommend arbitrarily complex products.

The next sections discuss the different types of CF algorithms and the related research. The neighborhood-based approaches detailed in sections 2.1.1 and 2.1.2 are also called *memory-based* [15] algorithms, in opposition to the *model-based* strategies described in 2.1.3.

2.1.1 User-based Nearest Neighbor

User-based nearest neighbor methods are one of the earliest approaches to collaborative recommendations, but still enjoy wide popularity due to its simplicity, efficiency and accuracy [16]. These methods produce recommendations directly from the user-item ratings matrix stored in the system, predicting the opinion of the active user on an item by using the ratings of users with similar rating patterns, called *neighbors*.

Finding the nearest neighbors of the active user is done by applying a similarity measure to his ratings and those of every other user. The most popular are based on the Pearson's correlation coefficient [17, 18] and the *cosine similarity* [19], although many different measures of similarity have been applied to RSs and more are emerging, such as *fuzzy distance* [20]. Below, the most often used measures to discover user similarity [15, 16, 21] are described.

Pearson correlation Pearson correlation measures — in the context of user similarity in RSs — the extent to which two vectors of ratings are linearly correlated [12, 21]. The similarity between users a and b is defined in equation 2.1, where I is the set of items that both users have rated, $r_{a,i}$ is the rating given by user a to item i and $\overline{r_a}$ is the average of the ratings provided by user a.

$$sim(a,b) = \frac{\sum_{i \in I}(r_{a,i} - \overline{r_a})(r_{b,i} - \overline{r_b})}{\sqrt{\sum_{i \in I}(r_{a,i} - \overline{r_a})^2}\sqrt{\sum_{i \in I}(r_{b,i} - \overline{r_b})^2}}. \tag{2.1}$$

$sim(a,b)$ takes values between -1 and 1, which correspond to a strong negative and positive correlation, respectively. One reason for the popularity of this method within user-based CF is that the calculation factors out the averages of the user's ratings [4]. This means that if, in general, user a scores items much higher than user b, they may still have a strong similarity value as long as their ratings are linearly correlated.

The Pearson correlation coefficient was used as the basis for the user similarity calculations in the GroupLens project [22], which is the earliest work on neighborhood-based collaborative filtering in the published literature [15]. In [23], the authors concluded empirically that the Pearson coefficient outperformed other measures for user-based CF, although predictions are poor if based on the ratings of users that have very few items in common with the active user.

Cosine similarity Cosine similarity (also known as *Vector similarity*) is a popular measure in the field of information retrieval, where documents are represented as word frequency vectors and the similarity is the cosine of the angle formed by the two frequency vectors [12, 15]. This formalism

was adapted to CF, where the computation of the cosine is performed on the user rating vectors. Using the previous nomenclature, the calculation is:

$$sim(a,b) = \cos(\vec{r_a}, \vec{r_b}) = \frac{\vec{r_a} \cdot \vec{r_b}}{\|\vec{r_a}\| \times \|\vec{r_b}\|} = \frac{\sum_{i \in I} r_{a,i} r_{b,i}}{\sqrt{\sum_{k \in I_a} r_{a,k}^2} \sqrt{\sum_{k \in I_b} r_{b,k}^2}}, \tag{2.2}$$

where $\vec{r_a}$ is the vector with user a's ratings and I_a is the set of items for which user a has provided a rating.

While the Pearson correlation is the most widely used measure for user-based nearest neighbor algorithms, cosine similarity is the most popular for item-based nearest neighbor [4, 8, 21] which is discussed in Section 2.1.2. One drawback of the vector similarity is that, unlike the Pearson correlation, it does not automatically correct for the fact that different users may have different rating patterns. To offset this, each user's rating average is often subtracted from the corresponding term [8]. The resulting measure is called the *Adjusted cosine similarity*, and has the same formula as equation 2.1. Thus, the Pearson correlation coefficient actually performs cosine similarity with a normalization of the user's ratings [12].

Mean squared difference This measure evaluates the similarity between two users (or items) by computing the inverse of the average squared differences between the ratings of users a and b on the same items [17]:

$$sim(a,b) = \frac{|I|}{\sum_{i \in I} (r_{a,i} - r_{b,i})^2}. \tag{2.3}$$

The mean squared difference cannot capture negative correlations between user's ratings, making it less popular than the previously discussed measures [16], since negative correlations have been shown to improve the quality of predictions [24].

Other measures have also been proposed, such as the *Spearman's rank correlation* [16], which is similar to the Pearson correlation but uses the rank of the ratings instead of the raw rating values.

In order to find the nearest neighbors of the active user after computing the similarities, the typical techniques are to either choose a fixed number k and take only the k nearest neighbors, or to specify a minimum threshold of user similarity. In [23], these two options are discussed: If too many neighbors are considered, users with low similarity may introduce noise into the predictions. On the other hand, if too few neighbors are chosen, the quality of the predictions is also shown to be affected [24], due to the loss of information. The optimal value for k is often described in the literature to be between 20 and 50 [16].

Knowing the set of nearest neighbors, the computation of a prediction for how much the active user a likes an item i is often performed by averaging all the neighbors ratings on i, weighted with the similarity score:

$$P_{a,i} = \overline{r_a} + \frac{\sum_{u \in U} (r_{u,i} - \overline{r_u}) \times sim(a,u)}{\sum_{u \in U} |sim(a,u)|}, \tag{2.4}$$

where $\overline{r_a}$ and $\overline{r_u}$ are the users a and u average ratings for the other items.

While the literature and the successful practical examples of user-based collaborative filtering methods provide evidence for the value of this approach, its challenges and shortcomings are also well researched:

Sparsity In many applications, recommender systems are used to recommend very large item sets, whereas each user has provided feedback in a very small subset of the items. This results in a very sparse User $\times$ Item matrix, which can cause systems based on the nearest-neighbor strategy to not be able to find good recommendations for certain users. One option to deal with the data sparsity problem is to use additional information about the users in order to find the neighborhood, such as demographic data (or any available information that might help in classifying users) [9, 12]. The extension of CF algorithms with demographic information is often called *demographic filtering* [25]. Other options to deal with this problem have been proposed, such as combining CF with personal agents to improve recommendations [26], or use dimensionality reduction techniques [27].

Cold start A specific challenge related to the sparsity problem is that some systems cannot produce recommendations while there is no information on a new user or new item. In the literature, the *cold start* problem is sometimes referred to as *new user* and *new item* problems [9]. Like with the sparsity issue, this is often mitigated by using additional information beyond the user's ratings.

Scalability The computational complexity of nearest-neighbor algorithms grows with the number of users as well as the number of items, which becomes a problem in systems handling millions of users and items. Strategies to minimize this issue exist, like clustering the users before hand and only looking for neighbors inside each cluster, at the cost of increasing the possibility of separating real neighbors during the clustering process [12].

2.1.2 Item-based Nearest Neighbor

The main idea behind item-based CF is to calculate recommendations using the similarity between items instead of users. Intuitively, the assumption is that if item i received similar ratings to items that user a previously liked (given by the same group of users), it makes sense to recommend item i to user a.

Item-based CF approaches are popular in domains with many users and items that need to compute predictions in real time, such as large e-commerce websites [4]. In [6], the authors discuss how this approach was used at Amazon.com. The reason for the popularity of this approach in large scale domains is that while the algorithmic complexity is the same as in user-based strategies, these methods allow for more offline precomputation, specifically of the similarity between items. While, in principle, similarities between users in user-based CF could also be precomputed, a 2001 paper [8] discusses why that does not work as well as with item-based CF: The user similarity shows a lot of variability when new ratings are provided (due to the typically low number of overlapping ratings

for any two users), while the item similarities are much more stable, so that offline precomputation doesn't degrade the quality of recommendations as much.

The similarity measures discussed in Section 2.1.1 can also be applied to item-based CF, with the main difference being that in user-based CF, the similarity is computed along the rows of the User $\times$ Item matrix, while in item-based CF it is computed along the columns. In other words, the rating vectors used for user similarity contain only ratings given by the same user, while the vectors used for item similarity contain only ratings provided by different users. In item-based CF, the *adjusted cosine similarity* is established as the standard metric [4]. The formula for finding the similarity between items i and j is shown in equation 2.5, where U is the set of users that rated both i and j, $r_{u,i}$ is the rating given by user u to i, and $\overline{r_u}$ is the average of u's ratings.

$$sim(i,j) = \frac{\sum_{u \in U}(r_{u,i} - \overline{r_u})(r_{u,j} - \overline{r_u})}{\sqrt{\sum_{u \in U}(r_{u,i} - \overline{r_u})^2}\sqrt{\sum_{u \in U}(r_{u,j} - \overline{r_u})^2}}. \tag{2.5}$$

Once the neighborhood for item i is found, the prediction for how much a user u likes i, $P_{u,i}$, is most often computed as the sum of the ratings given by u to the items in the neighborhood (N) of i, weighted by the corresponding similarity value [8]:

$$P_{u,i} = \frac{\sum_{j \in N} sim(i,j) \times r_{u,j}}{\sum_{j \in N} |sim(i,j)|}. \tag{2.6}$$

In [8], empirical evidence that item-based algorithms provide better performance than user-based methods while providing comparable recommendation quality is presented. However, problems like data sparsity, mentioned in Section 2.1.1 are still challenges in item-based CF.

2.1.3 Model-based approaches

Model-based algorithms for collaborative filtering are those that preprocess the ratings matrix offline, in opposition to those that process the ratings database in real time, called *memory-based* [4]. In their pure form, the user-based and item-based strategies mentioned in Sections 2.1.1 and 2.1.2 are memory-based. However, some methods that rely on offline preprocessing were already mentioned, such as clustering users before calculating their similarities, or preprocessing the item similarities entirely, which are actually model-based approaches.

Model-based CF strategies try to address the sparsity and scalability challenges of the neighborhood approaches, at the cost of increased complexity and sometimes expensive model building [12]. One such strategy is *Association Rule Mining*, widely used in contexts such as market basket analysis. The goal of this approach in the context of collaborative filtering is to detect rules such as "If users like $item_1$, they also like $item_2$ 85% of the time". [28] describes an architecture to discover association rules offline and use them to efficiently compute recommendations at run time. The two most popular algorithms to perform association rule mining are the *Apriori* and *FP-growth*. In [29], the authors systematize these and other algorithms by their strategy to traverse the search space.

Some other examples of the many different model-based methods that have been proposed in the literature are:

Dimensionality reduction Dimensionality reduction methods address the sparsity problem by decomposing the user-item rating matrix (or the similarity matrix) into a reduced latent space, that captures only the most important features [19, 27]. This space is denser than the ratings space, making this methodology less sensitive to sparse data [16].

Graph-based methods Graph-based approaches construct a graph where nodes can be users, items or both, and edges represent ratings or the similarity between users/items. This representation allows for the propagation of information through the graph, discovering relations between nodes that are not directly connected. In [30], the authors use the distance between users as similarity values for a user-based neighborhood recommendation. In [31], the distance of a user to an item is used directly as a prediction for the recommendation.

Clustering Forming clusters of similar users is another model-based technique that is sometimes used in CF to improve run time performance. [32] compares different clustering techniques, applied to CF. In [33], the authors propose a clustering algorithm that partitions users into groups with high similarity, allowing the calculation of the prediction at run time to simply consist of calculating the active user neighborhood within the members of his cluster.

2.2 Content-based Recommendation

Collaborative strategies, as discussed in Section 2.1, do not use specific information about the items (e.g. the genre or publisher of a movie), beyond the ratings given to them by users. This avoids the expensive acquisition and maintenance of item descriptions, but has the drawback of not being able to compute recommendations based on this information [4]. The approach consisting of describing each item — typically as a vector $X = (x_1, x_2, \ldots, x_n)$ where the dimensions represent item features — and matching these descriptions to a user profile to produce recommendations, is called *Content-based recommendation* [34].

The user profile in content-based filtering is often a description of the user's interests in terms of the item features. Thus the recommendation process consists of matching the attributes of the user profile against those of the item to be recommended, resulting in a prediction of the user's interest in the item. Article [34] presents an overview of content-based approaches, and introduces the following high level architecture:

Content analyzer The part of the system responsible for extracting features and structure from items. The output from the *content analyzer* is the input for the next components.

Profile learner The *profile learner* tries to generalize the user preferences from the items that they liked/disliked in the past, with the goal of constructing the user profiles.

Filtering component Matches the constructed user profiles with the items to be recommended. The result can be a binary choice (recommend/don't recommend), a probability that the active user will like each item, or a ranked list of the top items that the user might like.

As with other classes of recommender systems, the user feedback used to construct a profile can be gathered explicitly, by asking the user, or implicitly, by assigning a score to certain user actions on the item (such as bookmarking or buying it). While the latter strategy has the advantage of not needing user involvement, [34] mentions that this might lead to biased feedback in some cases. The model for the user profile is often built by learning a classifier, where the training set consists of the item descriptions labeled with the ratings provided by the user. In [35], the authors discuss the various supervised learning algorithms typically applied to this task, such as decision trees, probabilistic methods like *Naive Bayes*, and linear classifiers.

In opposition to collaborative RSs, content-based systems do not suffer from the *new item problem*. This is because items with no ratings can still be recommended, as long as their features can be extracted. One other advantage of content-based systems is the increased transparency of the recommendations: The characteristics of the recommended item can be used to explain the actions of the RS. Finally, content-based approaches do not need as many users in the system as collaborative ones, since a user profile is built using only the active user own ratings. However, various shortcomings of content-based filtering also exist [9, 34]:

New user This is the same problem as with collaborative filtering. Because it is not possible to learn a user profile for a user who hasn't provided feedback, no recommendations can be made to a new user.

Limited content analysis The quality of a content-based RS is limited by the features that represent an item. In some domains, such as when recommending jokes or poems, it is hard to extract features from the text that capture the relevant information that makes a user like or dislike an item.

Serendipity The *serendipity problem*, or *over-specialization problem* [34] is the inability of content-based systems to provide novel recommendations. Because these systems suggest items who score highly when matched against the user profile, they can only recommend items similar to the ones that the active user likes.

Because these challenges are relevant in many domains, pure content-based systems are rarely found in commercial systems today [4]. One example found in the literature is the NewsRec system [36], which recommends textual documents by using a Support Vector Machine (SVM) to build the user profile from extracted textual features. Recent research focuses more on using these techniques in the context of hybrid systems, which are discussed in Section 2.4 of this document.

2.3 Knowledge-based Recommendation

KBs RSs use information about the domain and the users needs in order to arrive at the recommendations, instead of relying on ratings. For this reason, problems like the *cold start* mentioned previously do not exist in KB systems, since user's ratings are not needed. On the other hand, a typi-

cal challenge in KB systems is the knowledge acquisition, since the information possessed by domain experts must be encoded in the system [4, 37].

Systems of this type are sometimes referred to as *conversational systems* [4], because the user needs to interact with it to provide the requirements. A typical interaction is:

1. The user specifies his or her initial preferences.
2. When the system has collected enough information about the user needs, it computes and presents a set of matching items.
3. Optionally, explanations might be provided.
4. The user may iterate on his requirements, returning to 2.

KB recommenders are classified into two basic types: *constraint-based* [38] and *case-based* [39, 40]. In both types of systems the user specifies his requirements and the system tries to find a solution, presenting it and — usually — explaining it. However, contraint-based recommenders look for a solution by exploiting a knowledge base of rules with information on which item features are needed to solve the requirements, while case-based recommenders look for items that are similar to the specified requirements using similarity measures [37].

The interaction between the user and the KB conversation system has been subject of some research. [41] discusses different approaches to the selection of interesting questions. An approach called *critiquing* is often cited, where the user is presented with items and guides the system by telling it what it is that he does not like about the current items. Entree [42] is a well-known system that recommends restaurants using a critique-based approach, where the user is presented with restaurants and provides feedback such as “too expensive”, in order to insert his requirements.

2.4 Hybridization

The types of RSs discussed in the previous sections exploit different sources of information to produce recommendations: For example, CF often uses the ratings given to items by the user community, CB makes use of product features, and KB exploits domain knowledge. However, in their pure forms, no one approach can take advantage of all this information at once. Consequently, research on how to combine different methods (i.e. *Hybrid* RSs) has been increasing in recent years, with the goals of improving recommendation quality and mitigate the issues of each specific approach. Table 2.1 shows an overview of the tradeoffs between the RS classes discussed in Sections 2.1, 2.2 and 2.3 and was adapted from [11], where it is presented with a more granular taxonomy and a slightly different nomenclature.

It is possible to find a few different classification schemes in the literature, regarding how different methods are combined to produce a Hybrid RS. For example, [9] discusses four strategies:

- Combining separate recommenders.

Table 2.1: Tradeoffs between recommendation techniques.

Technique	Advantages	Drawbacks
Collaborative	A. Ability to produce novel recommendations. B. No domain knowledge needed. C. Quality improves over time. D. Implicit feedback can be sufficient.	I. New user problem. J. New item problem. K. Does not work well for users different than everyone else ("gray sheep" problem). L. Needs large quantities of data. M. Difficult to adapt to a user's changing preferences.
Content-based	B, C, D	I, L, M
Knowledge--based	E. No cold start problems. F. Sensitive to user preference changes. G. Can include non-product related features. H. Can find products by mapping directly to user needs.	N. Quality does not improve over time. O. Knowledge engineering required.

- Adding content-based characteristics to collaborative models.
- Adding collaborative characteristics to content-based models.
- Developing a single unifying recommendation model.

A different, highly cited taxonomy is the one proposed in [11] and further discussed in articles such as [43], which consists of seven different hybridization designs:

Weighted In a weighted hybrid RS, the recommendation scores of independent systems are combined — for example, via a linear combination — to produce the final score.

Switching Switching hybrids use different techniques depending on some criteria (e.g., using CF but switching to CB to recommend new items).

Mixed A mixed strategy is one in which recommendations from different systems are simply shown together.

Feature combination This approach combines various data sources into a single algorithm. An example would be to use a content-based RS while treating collaborative data (ratings from other users) as another feature of the items.

Cascade In cascade hybrids, one recommender refines the results of the other. The previously mentioned Entree restaurant recommender [42] is an example of this approach.

Feature augmentation Feature augmentation consists of using the output from one recommender as input to another. For example, one system can use its output to populate the ratings matrix, acting as an artificial user, which a second system takes advantage of to compute recommendations.

Meta-level In the meta-level technique, one recommender uses the model generated by another to compute recommendations. Burke [11] presents some examples of meta-level hybrids and notes that this differs from feature augmentation because instead of using a model to generate input for the second RS, the entire model becomes the input.

While there is no hybridization technique that is applicable in every domain and circumstance, researchers agree that pure RSs can very often be improved by these approaches [3, 4, 10, 12]. For example, in the Netflix Prize competition [5], a challenge much publicized in the RS field, the winner system was a weighted hybrid that linearly combined many different recommendation approaches, where the weights of the combination were determined by regression analysis. In [44], the winner team details their solution to the challenge.

2.5 Emerging Topics

Research in recommendation systems is evolving in many directions, in part fueled by the constant change in the way users interact with software. This section discusses related work in emerging topics such as *Context-aware* RSs, which are particularly relevant in ubiquitous environments and *Trust-aware* RSs, which are applicable in domains where users can not only rate items, but also each others trustworthiness [45, 46].

2.5.1 Context-aware Recommendations

Context is generally defined in the RS literature as the information about the circumstances of a user that might be relevant to the recommendation process [9, 47]. Most recommendation systems compute the relevant items to users without taking into account contextual information, such as time, location, or whether the user is alone. However, Adomavicius et al. argue in [47] that this information matters in RSs and thus it should be taken into account. The same article describes three paradigms to incorporate context information in the recommendation process:

Contextual pre-filtering In contextual pre-filtering, the current user context is used to inform the data selection process, where the result can then be processed by any 2D (Users $\times$ Items) recommender system. The context is essentially used as a query to filter the data. An example would be: when suggesting movies to a user on a Saturday, use only the movies that he has previously watched on Saturdays.

Contextual post-filtering In this strategy, contextual information is initially ignored and the predictions are performed on the entire dataset. Afterwards, the resulting recommendations are contextualized for each user. The context information can be used to either filter out recommendations based on the current user context, or to adjust the ranking of the recommendation list.

Context modeling Context modeling is the contextualization of the recommendation function itself, where the context is used directly in the modeling algorithm. One way to accomplish this is to

extend the typical 2D recommenders that work on a Users $\times$ Items matrix into multi-dimensional recommenders, e.g., working with 3D input such as Users $\times$ Items $\times$ Time.

Context-aware recommenders represent a relatively new and underexplored research area [4, 9, 47]. A description of a restaurant recommender system that incorporates additional contextual dimensions into the model — such as time and weather — can be found in [48].

2.5.2 Trust-aware Recommendations

In some modern online platforms, users are not only given the ability to rate items, but also to declare other users as trustworthy. Amazon.com, for instance, allows users to rate each others reviews, as well as many other e-commerce websites. Systems that take advantage of this information to produce recommendations are called *trust-aware* RSs [46].

The highest cited works in this topic are by Massa and Avesani [45, 46]. They propose the use of a graph to represent a *trust* network that encodes the information related to inter-user ratings. Using this representation, it becomes possible to propagate trust through the graph, so that more users are assigned a trust weight than if a similarity weight was computed. Thus, by computing predictions using trust weights instead of the standard similarity weights (mentioned in Section 2.1 of this document), they report a mitigation of the new user problem, causing an increase in the amount of items that are recommended (coverage) while maintaining equivalent recommendation quality.

A key assumption when propagating trust is that trust relationships are transitive i.e., if A trusts B and B trusts C, then A trusts C to some extent. A 2004 article on this topic [49] introduces various trust propagation schemes. A typical option is to propagate trust values in multiplicative fashion, combined with a maximum propagation distance and a minimum trust threshold, where these parameters can be determined in an empirical manner [4].

"Trust and Recommendations" [50], the chapter 20 of "Recommender systems handbook" provides a thorough overview of the topic, where some architectures for trust-enhanced RSs are compared in the few public datasets that contain both rating and trust data, with no clear winner.

2.6 Explaining Recommendations

A RS capable of providing reasons for recommendations can be valuable: It might, for example, help a seller promote certain products, or help a buyer make the right decision. Thus, the explanation of recommendations has been subject of some research in the field. [51] discusses the possible aims of explanations in the context of RSs, where among others, the authors mention the aims of transparency (to explain how the system works), persuasiveness (to convince users of something), satisfaction (to increase the usability of the system) or trustworthiness (to increase users' confidence in the system).

Knowledge-based systems, due to their conversational nature, are the ones where explanations have been the most applied. One usual concern is the fact that user requirements are sometimes not met by any items. Chapter 6 of [4] mentions that a common technique to improve usability when there

are no results is to recommend items that are close to the requirements, explaining which constraints were violated by the user query.

In a highly cited 2000 article [52], the authors perform an empirical study on the effectiveness of explanations in a CF system, where twenty-one different types of explanations were shown in their movie recommender "MovieLens" and users were then asked how likely they were to go see the suggested movie. The study concluded that explicitly telling users that their neighbors liked the recommended item — in the form of a histogram with the user's neighbors and ratings — works best. More generally, it also showed that users appreciate explanations, since the study included a base case where no explanations were provided.

2.7 Evaluating Recommender Systems

Assessing the performance of RSs is a crucial challenge in the field, since many different techniques claim to improve recommendation accuracy over the others. Furthermore, accuracy alone has been shown to be insufficient for the usefulness of a recommender system in many domains. A discussion of relevant metrics is present in [53], where the authors mention the degree to which the recommendations cover all items, the degree to which recommendations are not obvious and the ability of the system to explain the recommendations.

A popular metric to evaluate accuracy is the mean absolute error (MAE), which computes the average deviation between rating predictions $p_{u,i}$ and actual known rating values $r_{u,i}$ for all users u and items i in the test set T [4]:

$$MAE = \sqrt{\frac{\sum_{(u,i)\in T} |p_{u,i} - r_{u,i}|}{|T|}}. \tag{2.7}$$

Ratings $r_{u,i}$ are usually known due to offline user experiments. A similar metric, also often mentioned in the literature [4, 53] is the root mean squared error (RMSE), which disproportionately penalizes large errors:

$$RMSE = \sqrt{\frac{\sum_{(u,i)\in T} \left(p_{u,i} - r_{u,i}\right)^2}{|T|}}. \tag{2.8}$$

Also commonly mentioned metrics are user coverage and item coverage, which simply measure the percentage of users or items for whom recommendations can be generated. These can be useful to learn about how the system behaves regarding the new user and new item problems.

Other relevant metrics beyond accuracy and coverage that are described in the Herlocker et al. article on evaluating RSs [53] are:

Learning rate The degree to which an algorithm needs many data points in order to generate good recommendations. A typical way to express this is the per-user learning rate, which consists of the quality of the recommendations as a function of the amount of information gathered on the user.

Serendipity The ability of the system to recommend unexpected items.

Confidence How much the recommender trusts a recommendation. It is sometimes represented as a probability, but it is often difficult to arrive at this value.

User evaluation Measuring directly what the user thinks of the recommendations, either by asking him or her directly, or implicitly by observing the user interaction with the system in user studies.

The same article also discusses some approaches and challenges to the accurate evaluation of RSs: Using offline datasets and analyzing metrics on the results (such as the ones discussed above) can be less expensive than performing live user experiments, since it is possible to quickly run many experiments, with different datasets and algorithms. However, offline analysis cannot evaluate less objective criteria about the system, like the quality of the interface that interacts with the user. The authors also mention that offline evaluation is sometimes performed with synthetized datasets, when real data is not available. While generating synthetic datasets can be a useful strategy to discover obvious flaws in the algorithms, it is risky to use the results of such analysis, since some bias is often introduced by having the data fit some algorithms "too well".

2.8 Security in Recommender Systems

RSs can be vulnerable to malicious user input that deteriorates the quality of recommendations. This is especially concerning in the case of collaborative systems, since users' ratings influence the suggestions provided to their neighbors. The most obvious motivation for an attacker to interfere with a RS is to promote some items, making the system recommend them more frequently. However, it is also possible to attack a system so that he recommends some items less often (or not at all).

Classifications of various attack strategies can be found in [54] and [55]. All of the attacks are based on the creation of many fictitious user profiles followed by promotion (*push attack*) or demotion (*nuke attack*) of a given target item, by providing it with maximum or minimum ratings. These attacks also include rating other items, called *filler items* so that the fictitious profiles can find neighbors to affect. Below is a description of the most common attack types on collaborative RSs, as characterized by [55]:

Random attack In the random attack, the ratings given to the filler items are random values that form a normal distribution, centered at the mean of all ratings in the system. The goal of such strategy is to have the fictitious user profiles look typical, increasing the number of users that they are neighbors to. The mean of the ratings in the system can often be found empirically, for example, by observing users rating items.

Average attack This is slightly more sophisticated than the random attack, in that the mean of the random ratings is determined by item, instead of globally. Often the system itself gives out this information publicly, to inform users on how an item is usually rated (for instance, Amazon.com provides this information in the form of "stars" for each product).

Bandwagon attack In the *bandwagon* strategy, the attacker uses external domain information, namely which items are popular among the real users, in order to use those as filler items. The filler

ratings to those items are typically high, so that the fictitious profiles can find as many neighbors as possible.

Segment attack The idea in the segment attack is to provide high ratings not only to the target item, but also to items that might be similar, in order to find neighborhoods that are likely to buy the promoted item. It also avoids promoting the target item to audiences where it would not make sense (e.g., recommending a science fiction book to someone who never bought or rated any books of this genre), making the attack more difficult to detect.

The random and average attacks were first introduced in [56], and are the simplest classes of attacks. Even more specific strategies are mentioned in [55], including some that only work for nuke attacks such as the *Love/hate attack* or attacks tailored to systems that collect user feedback implicitly, instead of via explicit ratings.

In chapter 9 of [4], an overview of possible defenses to attacks on RSs is presented. One common strategy is simply to make it expensive for the attacker to generate many fictitious accounts (e.g., by the use of bot detection techniques during registration). Also, the use of model-based techniques makes the RS less vulnerable, since computing predictions through direct manipulation of the ratings matrix is what makes these attacks effective. [57] discusses another defense framework based on automatic detection of fictitious profiles, introducing a metric called *deviation from mean agreement*. Using such a metric, it is possible to probabilistically classify profiles into *real* or *fake*, allowing the system to discount ratings given by fictitious profiles.

3

Automation Rules

Contents

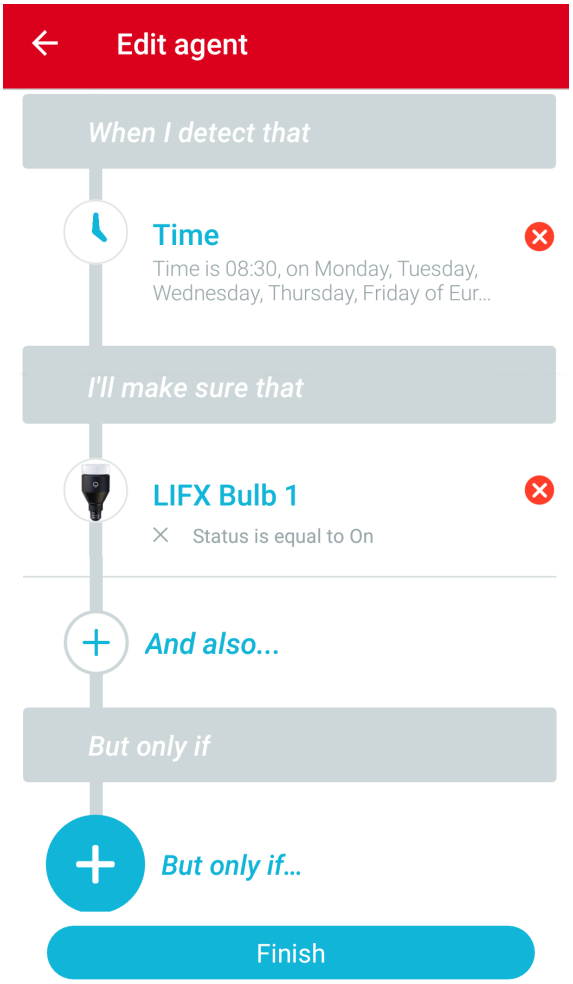

Figure 3.1: Creating an automation rule on the Muzzley application. This rule will turn on a light bulb every weekday at 08:30.

3.1 Rule Specification

In this section some formalisms about automation rules at Muzzley are described, as these concepts are referenced throughout this work. A rule is composed of one trigger (the "if" part), one or more actions (the "do" part), and zero or more state checks (the "but only if" part). The trigger describes the logic for when the actions should execute, the actions describe the changes that are applied to the world when a rule executes, and the states describe what else must be true in the world when the trigger fires, in order for the actions to execute. Figure 3.1 shows an example of an automation rule in the Muzzley mobile application.

Each of the mentioned rule components — trigger, action and state — has the following constituents:

Device The device that is being referred to. This is usually an identifier of a physical IoT device, but may also be time, location, or other concepts that are considered "virtual devices" for the purposes of automation rules.

Property Each device has one or more properties, which represent the different measures that

are supported. For example, a thermostat might have the properties *temperature*, *status* and *battery-health*. Each property has a value at each point in time, conforming to a well defined schema: *status* may be a boolean value, representing whether the thermostat is on or off, while *battery-health* may be a float between 0 and 1. Some properties, such as *battery-health*, are not actionable, meaning that they cannot be used in the actions of an automation rule.

Value Each component of the rule must also have a value, conforming to the property schema. In a trigger or state check, this value is used to test if the rule should be executed. In an action, it is the new value for the property of the device.

Operator The operator describes the logic used for this component of the automation rule. Without the operator, a trigger with a specific thermostat as the device, "temperature" as the property, and a value of 25º, still would not describe whether the automation rule should execute if the temperature rises above 25º, drops below 25º, or becomes equal to 25º. A set of supported, well defined operators exists at Muzzley, some examples being *greater-than-or-equal*, *less-than-or-equal* or *equals*.

3.2 Dataset

This work makes heavy use of the Muzzley automation rules dataset, which is summarized in the present section. The dataset evolves as users create and delete automation rules on the Muzzley application, but to facilitate the evaluation of different models and parameters on the same underlying data, this work analyzes a *snapshot* of this dataset, frozen in time. The snapshot used throughout this work is composed of 2545 automation rules, which were created by 1322 distinct users. Figure 3.2 shows the distribution of users per number of rules possessed, where it is possible to see that most users have few rules — in fact, slightly more than half of the users considered have only one automation rule.

Although users tend to create simpler rules, these numbers are not too extreme: 506 (20%) automation rules contain state components, and 1099 rules (43%) contain only one action, meaning that while the most common number of actions is 1, more than half the rules have more than 1 action.

Of the 2545 rules, only 2448 are actually used to learn recommendations, as 97 rules are discarded due to being "corrupted". This is related to users having removed from their accounts devices that were used in the rules.

3.3 Domain

Various recommendation approaches were discussed in Chapter 2, each being useful in different situations. Thus, it is worth laying out some characteristics of the automation rule domain and requirements for this work within Muzzley, before discussing the solution in the next chapter.

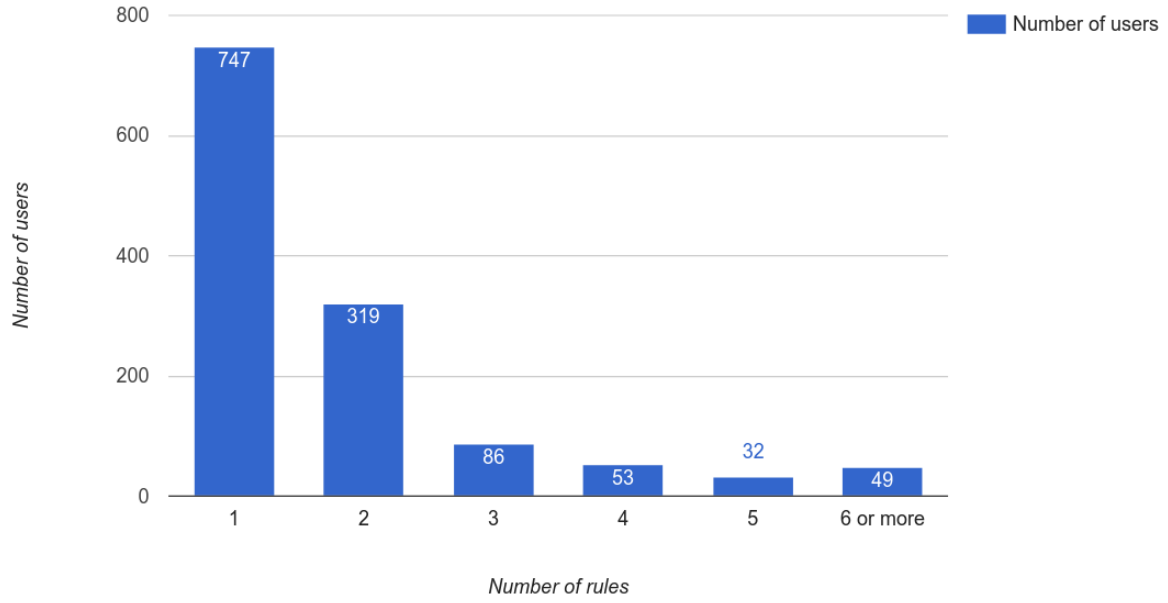

Figure 3.2: User distribution per number of rules.

3.3.1 Arbitrary items

The recommended item in our domain — the automation rule — is an object that can be composed arbitrarily by the user. Due to all the possible combinations of operators, devices and properties, and the fact that there is no limit to the number of actions that might exist in a rule, the items in our RS are actually infinite. Some recommendation strategies require a categorization of the items, and it is worth noting that while, for example, movies are usually categorized by its genre (*comedy*, *horror*), it is not trivial to manually come up with natural categories for automation rules.

3.3.2 The identity problem

The identity of an automation rule is not as well defined as that of most items discussed in the previous chapter. For example, it is possible for one user to define a rule such as “when I arrive within 100m of my home, open the garage door”, and for another to declare “when I arrive within 101m of my home, open the garage door”. While these are objectively different rules, if they are accounted as such, the item space becomes extremely large, and the data prohibitively sparse. Identifying them as the same item will allow us to exploit the information that two users created rules to the same effect. For this reason, it is necessary to define a higher level abstraction of what constitutes a recommendable rule, ideally matching as much as possible the human semantics.

3.3.3 Rating-based approaches

A user may create an automation rule and later delete it because it stopped being useful, but this does not say much about how good he believes the rule to be. Because the use case for a user to keep an active automation is its utility, we argue that rating-based RSs are not the most suitable for

our domain, since how much a user “likes” an automation rule is not as natural a question as, for example, how much a user “likes” a movie.

3.3.4 Recommendation frequency

In opposition to some domains where recommendations must always be available, or generated in real-time, Muzzley automation suggestions can be shown to a user only when the system is confident in the prediction. Recommendations are also post-filtered through a contextualizer component (as shown in Figure 1.1) so that the developed RS needs only to concern itself with finding good recommendations and assigning them a confidence score, while having some freedom to empirically determine any thresholds that affect the frequency — and consequently, the quality — of recommendations.

3.3.5 Recommendation typology

The different types of RSs discussed in Chapter 2, differ not only with regards to the input data, implementation and computational properties, but also in the types of recommendations that are generated. For example, pure CF provides a given recommendation because “users that rated items similarly to the active user also like the recommended item”, while CB provides it because “the active user liked items similar to the recommended item”. Association rule mining approaches (discussed in Section 2.1.3) have a notion of items that work well together now, providing recommendations of the sort “this item is often bought together with the items you are buying”. Used in the automation rule domain, such approach allows the recommendation of rules with the explanation “this rule is often used together with the ones you currently have”. Since automation rules affect the state of devices in the real world, it is desirable to perform recommendations of automations that may have synergies with the other rules that the active user already has. Thus, this work explores the application of association rule mining techniques to the recommendation of automation rules, and the developed RS is described in detail in the following chapter.

4

Solution

Contents

In light of the characteristics and trade-offs between the different RSs presented in Chapter 2 and the discussion about the automation rules domain in Chapter 3, the developed system exploits association rules learned from the automation dataset to provide recommendations. In such a RS, suggestions are generated on the basis of groups of rules "working well" together, which is a desirable characteristic of this approach. It also does not require that user ratings are gathered (explicitly or implicitly). However, given the complexity of automation rules and the identity problem discussed in Section 3.3, it is not trivial to learn association rules directly from the data.

To clarify the problem with an example, consider that two users have the same automation rule: "When I leave home, turn off all my light bulbs". Because these users live in different homes, the coordinates in the trigger will be different values, so that a naive comparison of the data structures would see these rules as different, defeating the purpose of learning association rules. The reason this problem is not discussed in the related literature might be that, in traditional applications of association rules for recommendation such as when cashing out in a supermarket, items are not created dynamically by users — they are well known to the RS and can therefore be trivially compared, for example based on an product identifier.

The solution explored in this thesis is to generalize each rule into a recommendable item, which we call *templates*. A similarity function between templates is developed and used to decide whether different automation rules can be considered the same for the purposes of mining association rules. The learned association rules are then used to recommend automation rule templates, along with a conviction value. These strategies are discussed in more detail in the following sections.

4.1 Rules Generalization

While it is desirable to produce recommendations that users can accept with the least amount of work possible, it is useful that some customization is possible when the user is accepting a suggestion. As an example, the RS might ask "Do you want your lights to be turned off automatically when you leave home?". For this recommendation to become a proper automation rule, it requires not only that the user accepts it, but also that he configures two things:

- The geographical coordinates of his home.
- Which of his light bulbs are to be used with this automation rule.

Because the user configures parts of the automation rule when accepting it, the recommendation is actually of an automation "template", which is instantiated by the user, giving rise to an automation rule. If the system were to recommend an already instantiated rule, it would always have to guess which devices the user might like to include in it, or values such as the exact color of a light bulb. Given the above discussion, it is worth noting that the items in our RS are the automation rule templates. This work explores an approach in which the rule instances in the dataset are first generalized into these templates, where the model is then learned, so that the recommendations produced are also of templates. Thus, the rule generalization step described in this section strongly influences the type of output of the system.

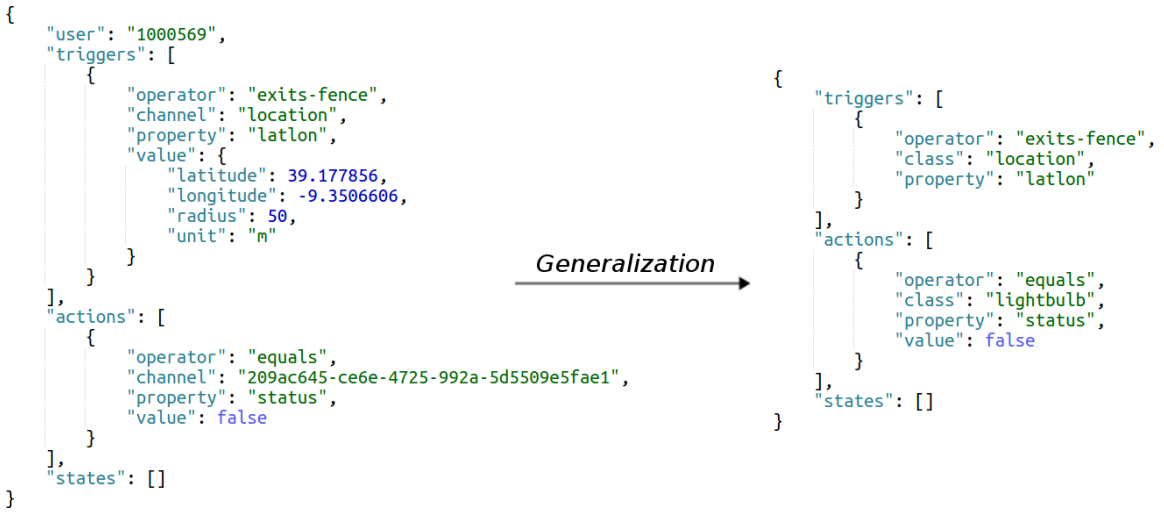

Figure 4.1: Generalizing from an automation instance to a template. *Channel* is the device identifier.

In the example above, the system is recommending a template in which the action is to turn off "lights". This is possible because the main step of generalizing a rule instance into a template is to replace the specific devices (identifiers for real world IoT devices) by the *class* of the devices, which are strings such as "lightbulb", or "camera". All specific IoT devices integrated in the Muzzley ecosystem already had classes, prior to this work.

The above example also shows that the value of the trigger — the geographical coordinates — is left for the user to fill, yet the value of the action — turning the devices "off" instead of "on" — is already instantiated, as opposed to asking the user to choose the status for the light bulbs. To accomplish such flexibility in the developed RS, the generalization function uses information from a configuration file that details whether the "value" part of each rule component should be discarded, based on the operator and property. If the behaviour above is desired, the configuration file encodes the information that the value should be discarded when the operator is *exits-fence* (the operator used for triggers that fire when a user leaves a known location).

Figure 4.1 shows the data structures for an automation rule in the dataset that conforms to the running example in the present section, as well as its generalization into a template. For the evaluation of this work (Chapter 5), a configuration file for rule generalization was used with reasonable defaults: For properties such as *status*, which take on boolean values, templates maintain the value, meaning that the recommendation will also include them. For properties such as location, values are discarded during generalization, so that recommendations will not encode such values — the user chooses them.

4.2 Similarity

To arrive at a dataset that is suitable for the mining of association rules, the system must be able to identify which templates from different users should be considered equal. Strictly comparing the JSON objects that describe the templates would be a possible solution, but very similar rules that differ

only slightly — for instance, two rules with the same triggers and states as well as many equal actions except for one — would be seen as different. Thus, a more robust equality operator was developed between templates, which returns a similarity score between 0 and 1 (0 being strictly different and 1 being strictly equal). Because it still is necessary to decide whether to consider two templates the same for the association rule mining purposes, a hyperparameter called *similarity-threshold* was introduced.

The developed equality operator takes two automation templates and compares the triggers components, obtaining a triggers similarity score. The actions and states similarity scores are obtained in the same way and the 3 results averaged. When comparing two components, the JSON objects are strictly compared, which works well because the rule generalization step already took care of removing device identifiers or values that we would not like to be meaningful for such comparison.

In order to find the user's items in the RS, we go through every generalized rule from the users sequentially, applying the equality operator, in the following manner:

1. The first generalized rule/template is given the ID "1".
2. The next template is compared against all the previously analyzed templates.
   (a) If, given the *similarity-threshold* parameter, the template being analyzed is not considered equal to any of the existing IDs, attribute it a new ID.
   (b) Otherwise, attribute the ID of the highest similarity score to the template being analyzed.
3. Repeat step 2 until no more templates are left.

This strategy allows us to arrive at a dataset that is suitable to learn association rules, since different users now have templates with the same IDs, meaning that these users have the same item for the purposes of our RS. As templates are analyzed, they are compared against more and more "already analyzed" templates, making it more likely that they will be considered equal to one of them (2b), instead of generating a new ID (2a). Thus, with a big enough dataset, the ID growth curve should be logarithmic, as more templates are analyzed. Figure 4.2 shows that while growth does slow down in our dataset, the number of IDs generated (534) is quite high in comparison with the number or automation rules considered (2545), and is still growing healthily as more rules are analyzed.

With 534 distinct items out of 2545, there are less association rules to learn than with a more dense dataset. This sparsity comes from the fact that users do create many different automation rules, and the generalization strategies described map each rule to a given template in a one to one relationship. But one realization is worth exploring: It is possible to increase the granularity of items by generalizing some rules to multiple, simpler templates, without loss of correctness. For example, let's say user "1000569" has only one automation rule — "Whenever my garage door opens, turn on the TV and set the thermostat to 23ºC, but only if time is between 18:00 and 20:00 PM.". Instead of generalizing this rule to a single template (which might occur rarely in other users), the system can equivalently consider that user "1000569" has 2 rules:

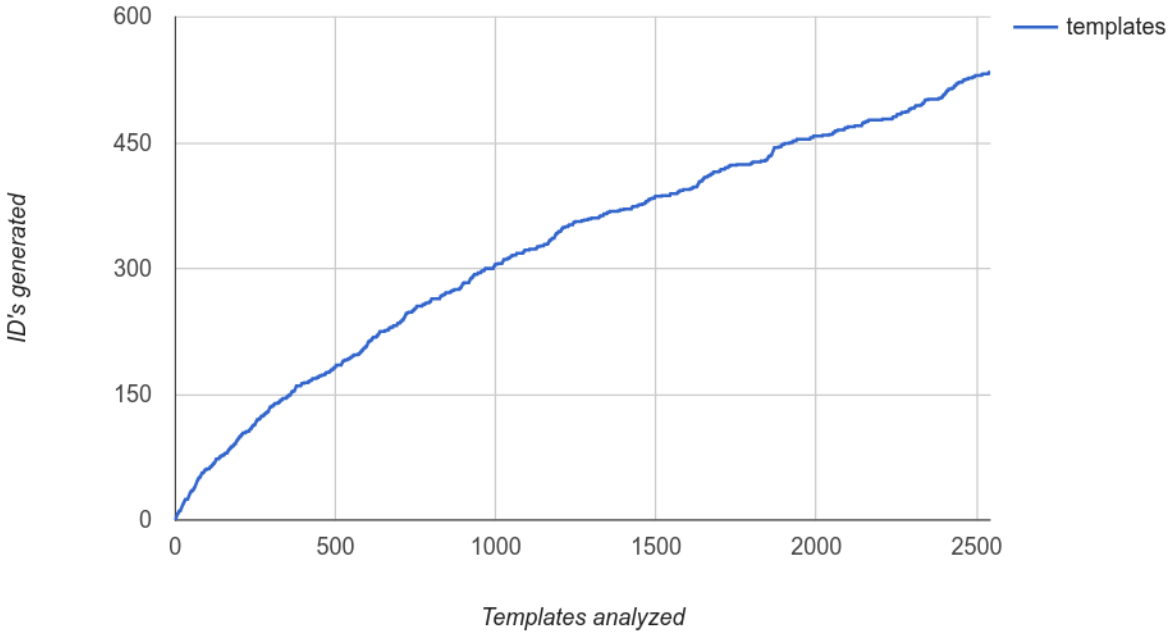

Figure 4.2: Template ID growth with the number of templates analyzed.

1. “Whenever my garage door opens, turn on the TV, but only if time is between 18:00 and 20:00 PM.”

2. “Whenever my garage door opens, set the thermostat to 23ºC, but only if time is between 18:00 and 20:00 PM.”

Together, these two rules have the same behaviour than the one actually created by the user. The formalism for accomplishing this division while maintaining the rule/s semantics is simply to create as many templates as there are actions in the original rule, each with one action, keeping the triggers and states equal in all of them. This approach has the advantage that the system has a more granular dataset to learn from, so that instead of trying to learn which items work well together from complex and often rare items, it does so from simpler ones. It is worth noting that the system can still perform recommendations of complex templates, because recommendations for the same user can be merged together using the inverse of the strategy described above — merging templates that have the same trigger and states into a single one, uniting all the actions.

Figure 4.3 shows the same measure as 4.2, but using the latter approach. There are now 4384 items in our dataset, instead of 2545, due to the division of rules. The number of IDs generated is not only lower as a proportion of the amount of templates analyzed, it is actually lower in absolute terms — 510 vs 534, which means that the dataset used to mine association rules is less sparse. The curve also has a more logarithmic shape, hinting at the fact that if the dataset was bigger, the number of distinct templates would grow slower and slower.

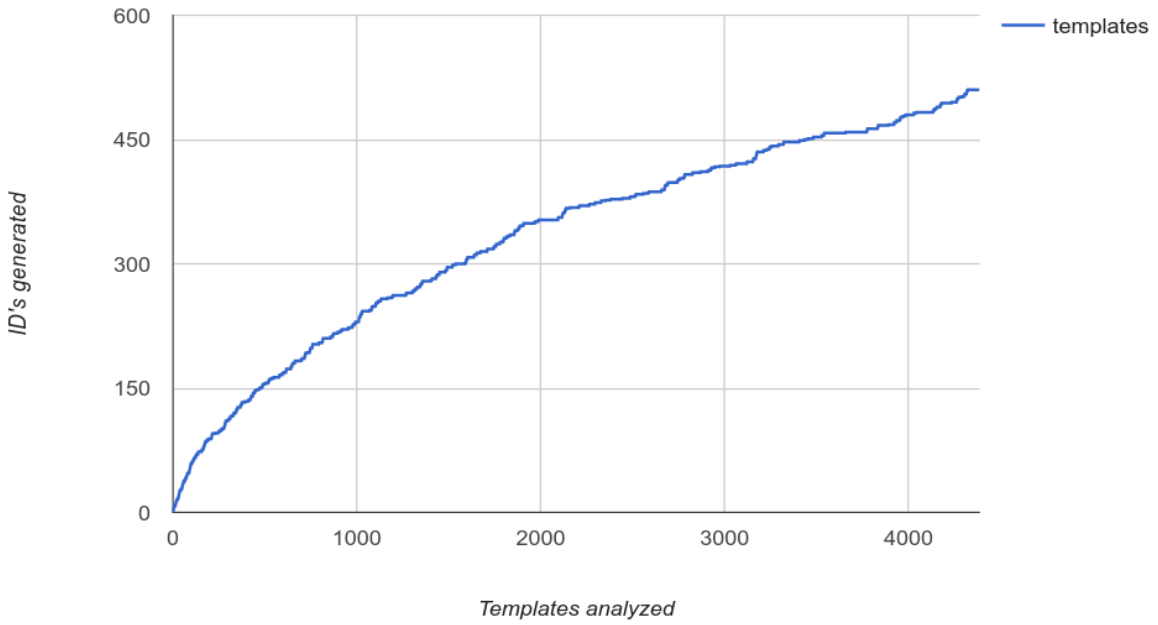

Figure 4.3: Template ID growth with the number of simplified templates analyzed. This approach shows a more logarithmic shape than the naive strategy.

Table 4.1: Excerpt of the users' items dataset. The numbers on the right are the template IDs.

Users	Templates
User 102341	43
User 101822	59
User 103202	247, 248, 249, 250, 251, 252
User 102366	326, 327
User 103320	100, 73, 457, 458, 12
User 102277	2
User 102475	43, 73
User 102393	43
User 102989	10, 11, 12
User 100741	2, 16
User 103059	10, 12

4.3 Association Rule Mining

From the steps described in Sections 4.1 and 4.2, the system arrives at a dataset of the form shown in Table 4.1. Using association rule mining terminology, each row in this dataset can be seen as a transaction, where templates are, naturally, the transaction items. In this small excerpt, templates 12 and 43 are the most common, appearing 3 times. For context, Table 4.2 describes the 5 most common templates in the dataset, in natural language.

Two important parameters to association rule mining algorithms are the *minimum support* and *minimum confidence*. If these variables take on values that are too high, few or no association rules might be found, as the bar for what constitutes a relevant association is set too high. On the other hand, if these values are too low, many weak association rules might be found, which in the context of RSs may lower the precision of recommendations. These parameters end up heavily affecting the

Table 4.2: The 5 most common templates in the dataset, described in natural language.

ID	Count	Template
2	393	“At a specified time of day, turn on my light bulbs”
16	318	“At a specified time of day, turn off my light bulbs”
1	280	“At a specified time of day, change the brightness of my light bulbs”
12	185	“When I arrive at some place, turn on my light bulbs, but only if time is between a certain interval”
43	157	“When I leave some place, turn off my light bulbs”

Table 4.3: Sample of 3 related association rules mined from the dataset.

Association Rule	Support	Confidence
(2) => (1 124)	0.054517135	0.28225806
(1) => (2 124)	0.054517135	0.443038
(124) => (2 1)	0.054517135	0.92105263

quality of our system’s recommendations, and are for that reason evaluated empirically in Section 5.3, where we test metrics such as precision and recall for various values of minimum support and confidence.

The association rule mining algorithm used in this work is the Apriori, mentioned in Subsection 2.1.3. A Common Lisp (CL) version of Apriori was implemented in the context of this thesis and the source code was published openly as a CL library named *cl-association-rules* [58].

Apriori outputs association rules, as well as the support and confidence for each. A sample of 3 related association rules mined from the dataset is shown in Table 4.3. Support is 0.054517135 for the rules shown, which means that the 3 automations involved in each association rule are found together in around 5.5% of transactions/users. The confidence values represent how often a user has the automations in the body of the rule, given that he has the head, e.g., 92% of users that created automation 124 also have automations 1 and 2. Section 4.4 describes how this is used to perform recommendations at Muzzley.

4.4 Recommending

As previously discussed (Section 3.3), automation recommendations at Muzzley are post-filtered by an existing *contextualizer* component, which takes care of dispatching the suggestion to users at the appropriate times and frequencies. For that reason, the developed system must only output the recommendation itself and a *conviction*[1] value, used by the contextualizer to decide the priorities of inputs from different systems.

Recalling that mining an association rule such as (73) => (43), with support of 5.8% and confidence of 62%, means that 5.8% of all users have both automations 73 and 43, and 62% of users that have automation 73 also have 43, RSs typically operate by recommending the body of the association rule (43) to users that have the head (73) but not the body. For this reason, each association rule mined can produce many recommendations across different users.

[1]The term *conviction* is used in this work instead of *confidence* (the actual name used within Muzzley) so as not to be confused with the association rules confidence value.

In this solution, every mined association rule is used to produce recommendations. Instead of filtering association rules that have low support and confidence values after they are learned, the Apriori parameters (minimum support and minimum confidence) are simply increased so that those associations are never mined at all. This improves the system's performance and simplifies analysis, since the Apriori parameters already exist and must be studied, and no more parameters are added to the overall model. This, however, makes these parameters very influential in the quality and quantity of recommendations. Section 5.3 evaluates the recommendations under different minimum support and minimum confidence values.

In order to compute a conviction value for a recommendation, it is natural to take into account both the support and confidence of the association rule that originated it. However, although both values lie between 0 and 1, they differ in magnitude — Support values are usually 10% or lower in our dataset, while confidence values vary greatly between the minimum confidence and 1. For this reason, our approach consists on standardizing the support and confidence values before averging them, by subtracting each value to the mean of the sample of association rules discovered, and dividing by the standard deviation, as shown in equation 4.1.

$$Zscore = \frac{x - \mu}{s} \tag{4.1}$$

Thus, the conviction value sent to the contextualizer alongside recommendations is the average of the standardized confidence and support from the corresponding association rules.

This chapter has discussed the approach taken to build the RS and described how the system learns to perform recommendations from the original automations dataset. However, production data at Muzzley is continuously changing, as users update their automations, so that the system must keep processing the dataset as it evolves. Given that learning association rules is a model-based offline process, and given also the presence of the post-filtering component, it was sufficient to implement the following logic:

1. Load the most up-to-date automation data.
2. Perform a recommendation iteration, consisting of the steps described in this chapter:
   (a) Generalizing all association rules into recommendable items, i.e., templates.
   (b) Applying the similarity operator to templates, obtaining a dataset of transactions.
   (c) Mining association rules using an implementation of Apriori.
   (d) Computing recommendations to users along with a conviction value.
3. Filter recommendations.
4. Send user recommendations to the contextualizer system.
5. Wait a (configurable) amount of time and return to step 1.

Step 3 above consists on filtering recommendations in two ways:

- Filter out those recommendations that were already performed on behalf of a given user, so as not to keep suggesting the same automations to the same users. To accomplish this, the system keeps a persistent record of all user/recommendation pairs ever performed, and checks against it on this step.
- Filter out those recommendations that contain classes of devices not held by the user. This is necessary so as not to suggest an automation that uses, for example, a thermostat, when the user doesn't have a thermostat integrated in his Muzzley account.

The wait time between iterations is a configuration variable of the system. A low wait time between iterations makes the system react quicker with new recommendations when users modify their automations, with the trade-off being higher waste of computing resources, since the system will likely be computing many of the same suggestions over and over again, just to have them filtered out.

5

Evaluation

Contents

5.1 Methodology

In Section 2.7, the techniques and challenges of evaluating RSs were discussed. The applicability of the described metrics and strategies is highly dependent on the type of recommender and the available data. With rating-based CF systems, it is common to compute *accuracy*, which is a measure of how well the model's rating predictions match the real ratings. There is no concept of ratings in our *RS*, but since the goal is to find automations that work well together, it is possible to validate the model using the transactions dataset.

One approach to evaluate our model against a given (unseen) user in the data is to select a random subset of her rules, perform recommendations, and compare against the rest. The more recommendations match with the non-selected rules, the better the system is performing. Intuitively, this approach is measuring the degree to which the system would recommend the automation rules that were in fact created by users, had he only created a subset of them. Given that our dataset is not huge, we try in this work to reduce the variance in the approach described by computing every possible combination of rules. For example, if a user's automation list is (3 7 41), we first perform recommendations assuming he only has automation (3), and compare them against the remaining rules that he does have, (7 41). Then we do the same assuming he only has automation (7), then (41), followed by (3 7), (3 41) and (7 41). The metrics gathered when comparing recommendations with the ground truth are described in Subsection 5.2.

The overall strategy is to perform 5-fold cross-validation, where users are randomly split into 5 groups. One group is the validation set used to test the model, and the remaining 4 groups are used to learn the association rules. The cross-validation process is then repeated 5 times (the folds), with each of the 5 subsamples used exactly once as the validation data. The results from the folds are then averaged to produce a single estimation. It is worth noting that the entire automations dataset is generalized to templates, so that the validation set in each fold consists of user templates that can easily be compared by ID against recommendations. Of course, automation rules are only learnt on the 4 subsamples in each fold that represent the training data.

Metrics gathered using the approach described above are useful to compare different systems or different versions of the system, but their absolute values should be negatively affected by a few factors:

- Given a subset of a user's automation rules, a recommendation may be good, but not be present in his remaining automations, counting as a miss (in fact, any recommendation made by the system that would be accepted by the user is considered a miss in this strategy, since it is necessarily an automation not already present).
- A user in the validation dataset might not have created an automation that works well with his other rules for reasons such as him not having the necessary IoT devices. If our recommendation model produces that suggestion, it is also counted as a miss.

In this work, the problem of a user not meeting the requirements to accept an automation rule is

seen as an implementation detail outside of the model, where recommendations are still generated but simply post-filtered later. Thus, all tests in the present section were taken to the raw recommendations, before any filtering is applied, so that no bias is inadvertently introduced in the results.

5.2 Metrics

When comparing the system's recommendations for a user with the ground truth in the transactions dataset, a few metrics typically used in the field of Information Retrieval (IR) were gathered:

Precision Precision is the fraction of recommended items that are relevant, i.e., that can be found in the user's non-selected automations. Equation 5.1 describes this formally:

$$precision = \frac{|recommendations \cap relevant\ automations|}{|recommendations|}. \tag{5.1}$$

A high precision value means that most recommendations were found in the validation user, and were thus "good". However, a system that optimizes solely for precision might end up performing very few recommendations, so as not to degrade this value.

Recall Recall is the fraction of relevant automations that were successfully recommended. Equation 5.2 shows how it is computed:

$$recall = \frac{|recommendations \cap relevant\ automations|}{|relevant\ automations|}. \tag{5.2}$$

A high recall value means that most automations that could have been found for a user, were indeed recommended. While this is desirable, this value can be inflated as a consequence of the RS performing many recommendations, with no regard for precision.

F1-score Because of the mentioned drawbacks of looking solely to each of the two previous metrics and the fact that they tend to vary in opposite directions, the F1-score is often used to combine precision and recall. It is also referred to simply as *F-measure*, and it is the harmonic mean of precision and recall, as equation 5.3 describes:

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}. \tag{5.3}$$

Coverage Coverage is the fraction of all items that can be recommended by the RS that were in fact recommended. Equation 5.4 shows how it is computed:

$$coverage = \frac{|distinct\ recommendations\ performed|}{|possible\ recommendations|} \tag{5.4}$$

In this work, the possible recommendations are the number of templates generated.

Table 5.1: Precision, recall, F1-score and coverage with varying min-support. Min-confidence is fixed at 20%.

Minimum Support	Precision	Recall	F1 Score	Coverage
10.00%	52.45%	11.60%	19.00%	0.50%
8.00%	62.00%	18.64%	28.66%	0.81%
6.00%	57.87%	23.67%	33.60%	1.20%
4.00%	52.62%	32.03%	39.82%	2.01%
3.00%	46.54%	37.09%	41.28%	2.34%
2.00%	43.96%	43.81%	43.88%	3.05%
1.50%	42.81%	45.47%	44.10%	3.44%
1%	41.36%	47.85%	44.37%	4.64%
0.50%	41.22%	54.48%	46.93%	7.41%
0.25%	38.12%	58.59%	46.19%	13.20%

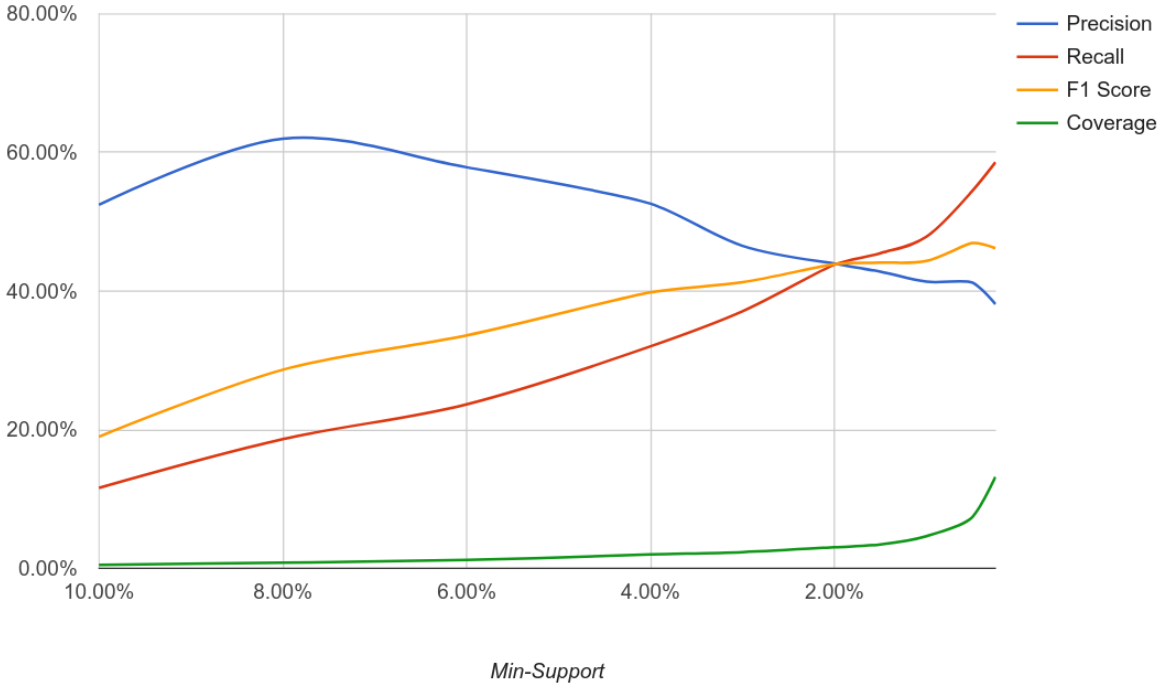

Figure 5.1: Precision, recall, F1-score and coverage with varying min-support. Min-confidence is fixed at 20%.

5.3 Parameter evaluation

We've discussed that the Apriori parameters *min-support* and *min-confidence* strongly influence the quantity and quality of recommendations. Thus, we use the methodology and metrics described in the present chapter to study them empirically, allowing us to find the best parameters while evaluating the system as whole.

Table 5.1 shows the results for the 5-fold cross-validation, obtained by averaging each measure across the 5 folds, when fixing *min-confidence* at 20% and varying the *min-support* values. Figure 5.1 is a visualization of these results. The tables and charts in the present section display the variable of interest in decreasing order — This is by design, as we are looking at what happens as the number of recommendations increase, which occurs when *min-support* and *min-confidence* decrease.

These results show that as *min-support* decreases (making the system learn association rules more liberally), metrics generally behave in the expected directions:

Table 5.2: Precision, recall, F1-score and coverage with varying min-confidence. Min-support is fixed at 3%.

Minimum Confidence	Precision	Recall	F1 Score	Coverage
90.00%	95.90%	9.76%	17.71%	1.16%
75.00%	90.56%	14.18%	24.52%	1.48%
60.00%	82.67%	17.81%	29.30%	1.63%
40.00%	57.48%	31.93%	41.06%	2.29%
20.00%	46.98%	38.18%	42.13%	2.44%
10.00%	47.24%	38.31%	42.31%	2.57%

- Precision decreases, since we are recommending automations with less conviction. This shows that recommendations based on association rules with high support values do have higher predictive value than with lower support values.
- Recall increases, since more of the validation user's automations are recommended by the system.
- Coverage increases, showing more distinct automation rules being recommended.

One exception is the increase in precision as *min-support* changes from 10% to 8%, showing that 10% would not be a good value for this parameter — There is no trade-off in decreasing it, since even precision improves, along with recall and coverage. It is worth noting that for such high values of *min-support*, few automations are recommended, and given that precision varies in the expected direction afterwards, this increase might be attributed to some form of variance.

As we move along the horizontal axis in Figure 5.1, F1-score improves except when *min-support* drops below 0.5%. Since recall keeps rising, this means that we start to lose too much precision with so many recommendations. Coverage strictly grows as more recommendations are made, of course. At the lowest *min-support* value studied (0.25%) coverage rises as high as 13.2%, which amounts to a total of around 67 different automation rules recommended, given the existence of 510 templates/items.

The *min-support* chosen for our *RS* in production at Muzzley was 0.5%, but this choice is entirely dependent on the business logic. For certain recommendation use cases, for instance when recommendations must be made with a certain (high) frequency, it may be sensible to accept further losses in precision, in order for the system to obtain higher coverage and/or recall. More generally, we see that even though the absolute values of the precision and recall metrics are negatively affected by the factors discussed in Subsection 5.1, they seem very promising. For instance, the fact that the model can get precision values above 50% hints that the recommendation approach of mining association rules has merit in our domain. Subsection 5.4 discusses this more precisely, as we compare our solution with a baseline strategy of recommending the top automation rules.

Table 5.2 shows the results of a similar test, this time varying the *min-confidence* parameter and fixing *min-support* at 3%. Figure 5.2 is the equivalent visualization.

The most interesting aspect of these results is how high precision can get when we only learn association rules that have very high confidence. Recalling what confidence in association rules means — when the head of the rule is present in the training set transactions/users, how often the

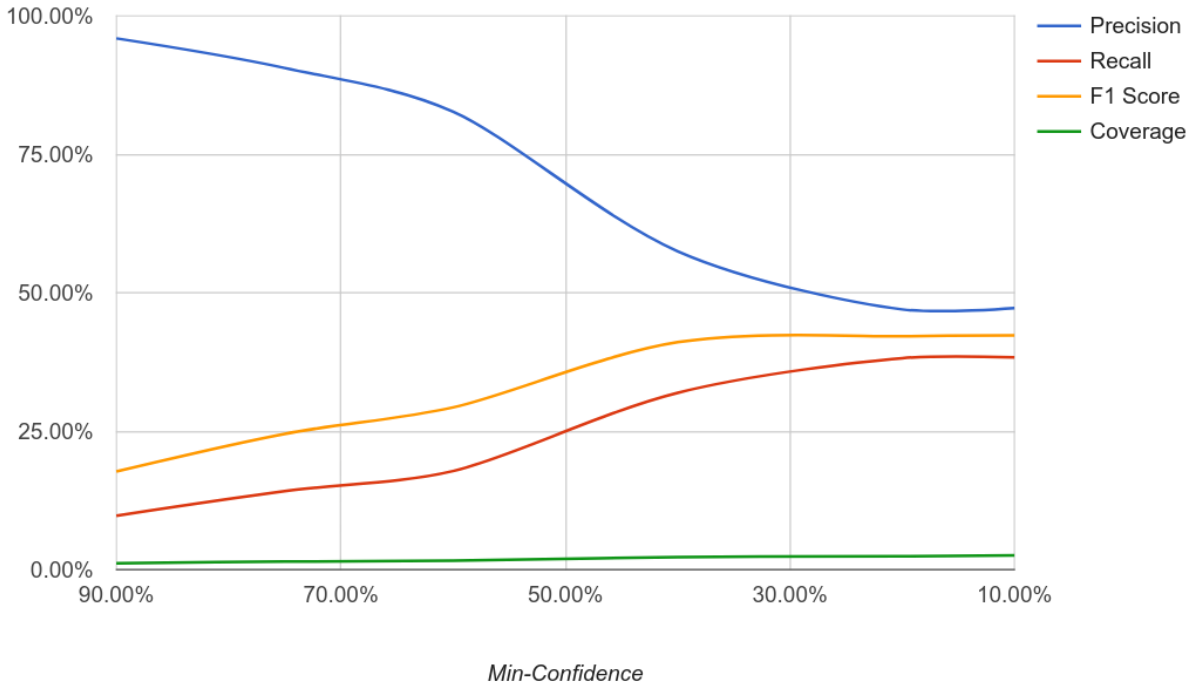

Figure 5.2: Precision, recall, F1-score and coverage with varying min-confidence. Min-support is fixed at 3%.

body also is present — and the fact that the precision calculation is performed against users not yet seen by the model, this also seems to validate our hypothesis that there are indeed automations that are often used together. However, the high precisions obtained for high *min-confidence* values come at the cost of low coverage. While a low number of distinct automations suggested may still result in many recommendations for different users, we have seen that a RS is more valuable if it can provide many different and unexpected suggestions.

Precision and recall tend to stabilize at around 47% and 38%, respectively, as *min-confidence* decreases, which is slightly unexpected — It would make sense for precision to keep dropping as the bar lowers for association rules to be mined. This is caused by the fact that in our dataset, few more association rules are mined as *min-confidence* decreases past 20% (with *min-support* fixed at 3%). In other words, almost all of the associations that can be found between items at this support, have a confidence value over 20%. Because the set of associations rules learned almost stabilizes, the number of recommendations also does so, as well as the resulting precision and recall measures.

Another noteworthy point about these results is that coverage increases very slowly, as we decrease *min-confidence*. This happens because *min-support* is set at 3%, which is a limiting factor regarding the variety of recommendations, as Table 5.1 shows. For the Muzzley production system, we settled on a *min-confidence* of 40%, where precision is still above 50%.

5.4 Comparison with baseline strategy

Blindly recommending the most popular items to every user is the simplest approach to building a RS. While recommendations in such a rudimentary system are not personalized, previous work

Table 5.3: Top-n popular items recommendation results.

Top-n most popular	Precision	Recall	F1 Score	Coverage
1	25.71%	8.65%	12.94%	0.19%
2	19.86%	15.91%	17.67%	0.39%
3	16.81%	20.80%	18.59%	0.58%
4	12.91%	26.37%	17.33%	0.77%
5	11.34%	30.59%	16.54%	0.97%
6	10.48%	32.99%	15.90%	1.16%
7	9.87%	37.41%	15.62%	1.35%
8	9.44%	41.06%	15.34%	1.55%
9	8.96%	44.08%	14.89%	1.74%
10	8.33%	45.53%	14.09%	1.93%

on RSs has shown that this simple strategy performs impressively well, sometimes outperforming complex models [4]. This is why even companies that provide personalized recommendations successfully, often still inform the user about the most popular items, like Spotify with the “Top tracks world-wide” feature, or Amazon with the “Amazon best sellers” web page.

In order to evaluate the quality of our recommendations more decisively, a RS that blindly suggests the most popular automations to every user was developed and the metrics previously analyzed in this chapter were determined. Such naive recommender is interesting to study as a baseline — The system developed in this work would be of little value were it to be outperformed by a much simpler solution. To make sure no bias was introduced, the tests were conducted in the same dataset and following the same methodology, except that no cross-validation is necessary, as instead of learning a model, the top-n items recommender simply:

1. Generalizes all association rules into recommendable items, i.e., templates.
2. Applies the similarity operator to templates, obtaining a dataset where each item is a template ID.
3. Counts the occurrence of templates in the entire dataset and generates a sorted list with the n most popular automations.
4. Recommends the automation rules in the sorted list.

Table 5.3 shows the results obtained, and Figure 5.3 is the corresponding visualization. Predictably the coverage metric grows exactly linearly as the number of recommendations increases, as it is simply the number of distinct recommendations made (either 1, 2, ... 10, in this example) divided by the little more than 500 total items that the system can recommend. Precision and recall also vary in the expected manner.

The best version of this system by the F1-Score is the top-3 automations recommender which, of course, suggests the 3 most popular templates to every user. However, these results heavily underperform the association rules based system. Table 5.4 shows the best results from top-n RS as well as the results from the model-based RS with the production parameters.

The model-based recommender outperforms the top-3 system in every metric. Coverage is much higher because the system recommends 32 distinct automation rules instead of just 3. It can be

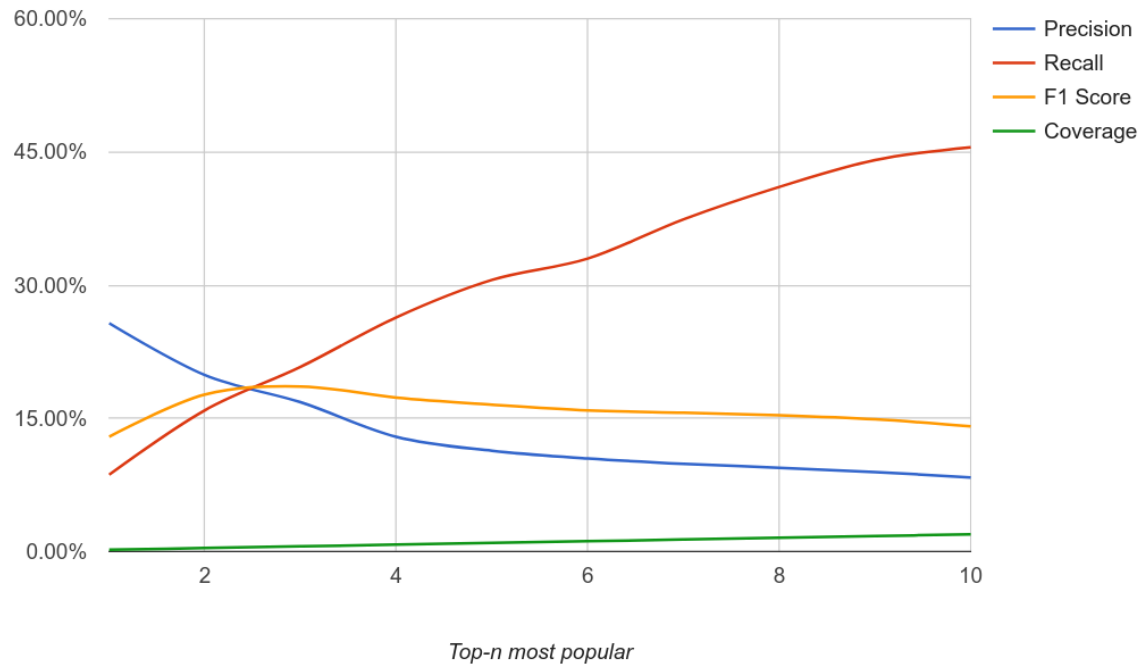

Figure 5.3: Top-n popular items recommendation results.

Table 5.4: Top-n items recommender vs association rule based recommender.

Top-n most popular	Precision	Recall	F1 Score	Coverage
Top-3 items recommender	16.81%	20.8%	18.59%	0.58%
Model-based recommender min-support: 0.5% min-confidence: 40%	56.41%	43.12%	48.9%	6.25%

argued that this is also the reason why recall is better — The more recommendations that are made, the more likely the system is to “guess” the validation users’ templates. However, while the higher number of recommendations should make precision lower, the model-based system has 56.41% in the precision metric, versus the 16.81% of the top-n RS. This is still more than twice the precision of even the top-1 recommender, which has even worse recall and coverage.

These results show that the hypothesis that groups of automation rules can often be found together due to synergies between them is likely true, since the approach that exploits this by recommending automations based on mining association rules from existing data showed strong results.

6

Conclusions and Future Work

Contents

6.1 Summary

The trend of appliances featuring internet connection is a growing one, as the diversity of IoT devices is increasing at a staggering pace. Superficially, such devices might look similar to their older counterparts, featuring only a remote control that can be a computer or mobile phone. However, the value in connected "things" seems to reside in their ability to be automated and work together to the benefit of the user. Many platforms referenced throughout this work, including Muzzley, are building systems that allow for the creation of such automations and interactions between IoT devices. This thesis explored one natural next step to bring the smart home closer to a reality, which is the automatic and personalized recommendation of automation rules, easing the still cumbersome process of coming up with, and defining, automations.

This work was laid out in a way that resembles the path that enabled us to build such a system. We began by exploring the related work in RSs as thoroughly as possible, focusing on the advantages and disadvantages of each type of solution, as well as defining the domains where the different approaches are suitable. This is because there is no previous work that we could find discussing IoT automation rules in the context of recommendations, as this is a novel applicability of RSs. After this extensive exploration, we defined the characteristics of our domain, as well as some general requirements for the system to be developed at Muzzley. This led us to the hypothesis that generating recommendations by learning association rules from the automations dataset would be the most suitable approach, given the likelihood that certain automation rules work well together, and such groups can be found in the current Muzzley data. While this type of recommendation is desirable, we also laid out some challenges that arise when trying to apply such strategy, such as what we named the *identity problem*, and the fact that existing automation rules are part of an infinite set.

We go on to describe our solution to make mining association rules feasible in the automations dataset. This consists of transforming the specific rule instances into more generic items, that we named *templates* and which when recommended, allow the user to input certain values when desirable. Using a comparison operator between them to arrive at a new dataset where it is possible to identify the same automation in different users, it became viable to learn associations between items, with an implementation of Apriori that we developed and open-sourced. Other techniques used to achieve better results were discussed, such as pre-processing the dataset by simplifying complex rules in an equivalent manner.

The developed system was then evaluated to demonstrate the hypothesis, showing promising results. A methodology to analyze the system offline was devised and metrics such as precision, recall and coverage were gathered with various combinations of the model's hyper-parameters, namely the *min-support* and *min-confidence* Apriori input variables. A baseline, non-personalized RS that recommends the most popular automations was also developed so that we could analyze it in a similar fashion and compare the results. These showed that our hypothesis that recommending based on the idea that certain groups of automations make sense together is likely correct, as our recommender beat the simpler one by decisive margins in all metrics studied. Generally, the fact that the system

could achieve very high precision values even with more relaxed parameters so that more recommendations were produced, seems to us to be a strong indicator that the approach taken has merit for this domain.

All in all, we consider the proposed goals of this work to have been accomplished, as we were able to develop a working RS for automation rules that showed promising results with the evaluation methodology described, hinting that the devised solution is a promising one for this domain.

6.2 Contribution

We believe that a strong contribution of this thesis is the exploration on the applicability of different RSs to the automation rules domain. The main contribution however, should be the demonstration of an approach to perform automation recommendations that can easily be applied outside of the Muzzley dataset, as most of the hurdles overcame should be applicable to every platform dealing with IoT automation.

More generally, the techniques discovered in this work such as item generalization, similarity comparison between items to arrive at a well defined identity, or preprocessing the data to achieve higher granularity of items, should be applicable to recommendation problems outside of the automation rule realm. This might be the case whenever some of the following attributes are true of the domain in question:

- The item set is infinite.
- Items to be recommended are virtual, such as computational data structures/programs.
- Items are not well known apriori to the RS, e.g. when they are created dynamically by users.
- The objects that the system can learn from are specific instances of the more general recommendations that are desirable.
- Items do not have a well defined category.

6.3 Further Work

This subsection discusses some potentially promising ways to improve the approach described in this work.

Currently, a configuration file allows us to define which values should be recommended to the user, and which should be left for the user to select when accepting an automation. Instead of this being a binary option, it would be interesting for the system to recognize a third option, where it would compute a running average for that value as it finds similar templates, allowing a recommendation to contain a specific value for continuous properties (e.g. brightness). This would have to be used with care, as an average wouldn't necessarily represent the typical values. Thus, different metrics such as *mode* could be explored, which could even be applied to properties with discrete values (e.g. color).

Another improvement to tackle the new-user problem of our system (no recommendations are performed to users without automation rules) could be to use it as part of a switching hybrid system (Section 2.4), where a different RS would be used for such users. Even using a naive system such as the top-n recommender developed as a baseline might be better than providing no recommendations. For use cases with higher demands on the frequency of recommendations than ours, a mixed hybrid could also be explored.

Bibliography

[1] D. Evans, “The internet of things: How the next evolution of the internet is changing everything,” *CISCO white paper*, vol. 1, pp. 1–11, 2011.

[2] D. Bandyopadhyay and J. Sen, “Internet of things: Applications and challenges in technology and standardization,” *Wireless Personal Communications*, vol. 58, no. 1, pp. 49–69, 2011.

[3] F. Ricci, L. Rokach, and B. Shapira, *Introduction to recommender systems handbook*. Springer, 2011.

[4] D. Jannach, M. Zanker, A. Felfernig, and G. Friedrich, *Recommender systems: an introduction*. Cambridge University Press, 2010.

[5] J. Bennett and S. Lanning, “The netflix prize,” in *Proceedings of KDD cup and workshop*, vol. 2007, 2007, p. 35.

[6] G. Linden, B. Smith, and J. York, “Amazon. com recommendations: Item-to-item collaborative filtering,” *Internet Computing, IEEE*, vol. 7, no. 1, pp. 76–80, 2003.

[7] Y. Song, S. Dixon, and M. Pearce, “A survey of music recommendation systems and future perspectives,” in *9th International Symposium on Computer Music Modeling and Retrieval*, 2012.

[8] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, “Item-based collaborative filtering recommendation algorithms,” in *Proceedings of the 10th international conference on World Wide Web*. ACM, 2001, pp. 285–295.

[9] G. Adomavicius and A. Tuzhilin, “Towards the next generation of recommender systems: A survey of the state-of-the-art and possible extensions,” *Knowledge and Data Engineering, IEEE Transactions on*, vol. 17, no. 6, pp. 734–749, 2005.

[10] M. Balabanović and Y. Shoham, “Fab: content-based, collaborative recommendation,” *Communications of the ACM*, vol. 40, no. 3, pp. 66–72, 1997.

[11] R. Burke, “Hybrid recommender systems: Survey and experiments,” *User modeling and user-adapted interaction*, vol. 12, no. 4, pp. 331–370, 2002.

[12] X. Su and T. M. Khoshgoftaar, “A survey of collaborative filtering techniques,” *Advances in artificial intelligence*, vol. 2009, p. 4, 2009.

[13] D. Cosley, S. K. Lam, I. Albert, J. A. Konstan, and J. Riedl, “Is seeing believing?: how recommender system interfaces affect users’ opinions,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2003, pp. 585–592.

[14] K. Goldberg, T. Roeder, D. Gupta, and C. Perkins, “Eigentaste: A constant time collaborative filtering algorithm,” *Information Retrieval*, vol. 4, no. 2, pp. 133–151, 2001.

[15] J. S. Breese, D. Heckerman, and C. Kadie, “Empirical analysis of predictive algorithms for collaborative filtering,” in *Proceedings of the Fourteenth conference on Uncertainty in artificial intelligence*. Morgan Kaufmann Publishers Inc., 1998, pp. 43–52.

[16] C. Desrosiers and G. Karypis, “A comprehensive survey of neighborhood-based recommendation methods,” in *Recommender systems handbook*. Springer, 2011, pp. 107–144.

[17] U. Shardanand and P. Maes, “Social information filtering: algorithms for automating “word of mouth”,” in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM Press/Addison-Wesley Publishing Co., 1995, pp. 210–217.

[18] R. Jin, L. Si, and C. Zhai, “Preference-based graphic models for collaborative filtering,” in *Proceedings of the Nineteenth conference on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 2002, pp. 329–336.

[19] R. Bell, Y. Koren, and C. Volinsky, “Modeling relationships at multiple scales to improve accuracy of large recommender systems,” in *Proceedings of the 13th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2007, pp. 95–104.

[20] M. Y. H. Al-Shamri and K. K. Bharadwaj, “Fuzzy-genetic approach to recommender systems based on a novel hybrid user model,” *Expert systems with applications*, vol. 35, no. 3, pp. 1386–1399, 2008.

[21] H. J. Ahn, “A new similarity measure for collaborative filtering to alleviate the new user cold-starting problem,” *Information Sciences*, vol. 178, no. 1, pp. 37–51, 2008.

[22] P. Resnick, N. Iacovou, M. Suchak, P. Bergstrom, and J. Riedl, “Grouplens: an open architecture for collaborative filtering of netnews,” in *Proceedings of the 1994 ACM conference on Computer supported cooperative work*. ACM, 1994, pp. 175–186.

[23] J. L. Herlocker, J. A. Konstan, A. Borchers, and J. Riedl, “An algorithmic framework for performing collaborative filtering,” in *Proceedings of the 22nd annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 1999, pp. 230–237.

[24] J. Herlocker, J. A. Konstan, and J. Riedl, “An empirical analysis of design choices in neighborhood-based collaborative filtering algorithms,” *Information retrieval*, vol. 5, no. 4, pp. 287–310, 2002.

[25] M. J. Pazzani, “A framework for collaborative, content-based and demographic filtering,” *Artificial Intelligence Review*, vol. 13, no. 5-6, pp. 393–408, 1999.

[26] N. Good, J. B. Schafer, J. A. Konstan, A. Borchers, B. Sarwar, J. Herlocker, and J. Riedl, "Combining collaborative filtering with personal agents for better recommendations," in *AAAI/IAAI*, 1999, pp. 439–446.

[27] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Application of dimensionality reduction in recommender system-a case study," DTIC Document, Tech. Rep., 2000.

[28] ——, "Analysis of recommendation algorithms for e-commerce," in *Proceedings of the 2nd ACM conference on Electronic commerce*. ACM, 2000, pp. 158–167.

[29] J. Hipp, U. Güntzer, and G. Nakhaeizadeh, "Algorithms for association rule mining—a general survey and comparison," *ACM sigkdd explorations newsletter*, vol. 2, no. 1, pp. 58–64, 2000.

[30] H. Luo, C. Niu, R. Shen, and C. Ullrich, "A collaborative filtering framework based on both local user similarity and global user similarity," *Machine Learning*, vol. 72, no. 3, pp. 231–245, 2008.

[31] Z. Huang, H. Chen, and D. Zeng, "Applying associative retrieval techniques to alleviate the sparsity problem in collaborative filtering," *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 116–142, 2004.

[32] L. H. Ungar and D. P. Foster, "Clustering methods for collaborative filtering," in *AAAI workshop on recommendation systems*, vol. 1, 1998, pp. 114–129.

[33] S. H. S. Chee, J. Han, and K. Wang, "Rectree: An efficient collaborative filtering method," in *Data Warehousing and Knowledge Discovery*. Springer, 2001, pp. 141–151.

[34] P. Lops, M. De Gemmis, and G. Semeraro, "Content-based recommender systems: State of the art and trends," in *Recommender systems handbook*. Springer, 2011, pp. 73–105.

[35] M. J. Pazzani and D. Billsus, "Content-based recommendation systems," in *The adaptive web*. Springer, 2007, pp. 325–341.

[36] C. Bomhardt, "Newsrec, a svm-driven personal recommendation system for news websites," in *Proceedings of the 2004 IEEE/WIC/ACM International Conference on Web Intelligence*. IEEE Computer Society, 2004, pp. 545–548.

[37] A. Felfernig, G. Friedrich, D. Jannach, and M. Zanker, "Developing constraint-based recommenders." *Recommender systems handbook*, vol. 1, p. 187, 2011.

[38] A. Felfernig and R. Burke, "Constraint-based recommender systems: technologies and research issues," in *Proceedings of the 10th international conference on Electronic commerce*. ACM, 2008, p. 3.

[39] D. Bridge, M. H. Göker, L. McGinty, and B. Smyth, "Case-based recommender systems," *The Knowledge Engineering Review*, vol. 20, no. 03, pp. 315–320, 2005.

[40] F. Lorenzi and F. Ricci, "Case-based recommender systems: a unifying view," in *Intelligent Techniques for Web Personalization*. Springer, 2005, pp. 89–113.

[41] T. Mahmood and F. Ricci, “Learning and adaptivity in interactive recommender systems,” in *Proceedings of the ninth international conference on Electronic commerce*. ACM, 2007, pp. 75–84.

[42] S. Trewin, “Knowledge-based recommender systems,” *Encyclopedia of library and information science*, vol. 69, no. Supplement 32, p. 180, 2000.

[43] R. Burke, “Hybrid web recommender systems,” in *The adaptive web*. Springer, 2007, pp. 377–408.

[44] A. Töscher, M. Jahrer, and R. M. Bell, “The bigchaos solution to the netflix grand prize,” *Netflix prize documentation*, pp. 1–52, 2009.

[45] P. Massa and P. Avesani, “Trust-aware collaborative filtering for recommender systems,” in *On the Move to Meaningful Internet Systems 2004: CoopIS, DOA, and ODBASE*. Springer, 2004, pp. 492–508.

[46] ——, “Trust-aware recommender systems,” in *Proceedings of the 2007 ACM conference on Recommender systems*. ACM, 2007, pp. 17–24.

[47] G. Adomavicius and A. Tuzhilin, “Context-aware recommender systems,” in *Recommender systems handbook*. Springer, 2011, pp. 217–253.

[48] K. Oku, S. Nakajima, J. Miyazaki, and S. Uemura, “Context-aware svm for context-dependent information recommendation,” in *Proceedings of the 7th international Conference on Mobile Data Management*. IEEE Computer Society, 2006, p. 109.

[49] R. Guha, R. Kumar, P. Raghavan, and A. Tomkins, “Propagation of trust and distrust,” in *Proceedings of the 13th international conference on World Wide Web*. ACM, 2004, pp. 403–412.

[50] P. Victor, M. De Cock, and C. Cornelis, “Trust and recommendations,” in *Recommender systems handbook*. Springer, 2011, pp. 645–675.

[51] N. Tintarev and J. Masthoff, “Effective explanations of recommendations: user-centered design,” in *Proceedings of the 2007 ACM conference on Recommender systems*. ACM, 2007, pp. 153–156.

[52] J. L. Herlocker, J. A. Konstan, and J. Riedl, “Explaining collaborative filtering recommendations,” in *Proceedings of the 2000 ACM conference on Computer supported cooperative work*. ACM, 2000, pp. 241–250.

[53] J. L. Herlocker, J. A. Konstan, L. G. Terveen, and J. T. Riedl, “Evaluating collaborative filtering recommender systems,” *ACM Transactions on Information Systems (TOIS)*, vol. 22, no. 1, pp. 5–53, 2004.

[54] R. Burke, M. P. O’Mahony, and N. J. Hurley, “Robust collaborative recommendation,” in *Recommender systems handbook*. Springer, 2011, pp. 805–835.

[55] B. Mobasher, R. Burke, R. Bhaumik, and C. Williams, “Toward trustworthy recommender systems: An analysis of attack models and algorithm robustness,” *ACM Transactions on Internet Technology (TOIT)*, vol. 7, no. 4, p. 23, 2007.

[56] S. K. Lam and J. Riedl, “Shilling recommender systems for fun and profit,” in *Proceedings of the 13th international conference on World Wide Web*. ACM, 2004, pp. 393–402.

[57] P.-A. Chirita, W. Nejdl, and C. Zamfir, “Preventing shilling attacks in online recommender systems,” in *Proceedings of the 7th annual ACM international workshop on Web information and data management*. ACM, 2005, pp. 67–74.

[58] D. Franco, “cl-association-rules,” https://github.com/diogoalexandrefranco/cl-association-rules, 2017.