RecRules: Recommending IF-THEN Rules for End-User Development

RecRules: Recommending IF-THEN Rules for End-User Development

FULVIO CORNO, Politecnico di Torino
LUIGI DE RUSSIS, Politecnico di Torino
ALBERTO MONGE ROFFARELLO, Politecnico di Torino

Nowadays, end users can personalize their smart devices and web applications by defining or reusing IF-THEN rules through dedicated End-User Development (EUD) tools. Despite apparent simplicity, such tools present their own set of issues. The emerging and increasing complexity of the Internet of Things, for example, is barely taken into account, and the number of possible combinations between triggers and actions of different smart devices and web applications is continuously growing. Such a large design space makes end-user personalization a complex task for non-programmers, and motivates the need of assisting users in easily discovering and managing rules and functionality, e.g., through recommendation techniques. In this paper, we tackle the emerging problem of recommending IF-THEN rules to end users by presenting *RecRules*, a hybrid and semantic recommendation system. Through a mixed content and collaborative approach, the goal of *RecRules* is to *recommend by functionality*: it suggests rules based on their final purposes, thus overcoming details like manufacturers and brands. The algorithm uses a semantic reasoning process to enrich rules with semantic information, with the aim of uncovering hidden connections between rules in terms of shared functionality. Then, it builds a collaborative semantic graph, and it exploits different types of path-based features to train a learning to rank algorithm and compute *top-N* recommendations. We evaluate *RecRules* through different experiments on real user data extracted from IFTTT, one of the most popular EUD tool. Results are promising: they show the effectiveness of our approach with respect to other state-of-the-art algorithms, and open the way for a new class of recommender systems for EUD that take into account the actual functionality needed by end users.

CCS Concepts: • **Information systems → Retrieval models and ranking**; **Recommender systems**; • **Human-centered computing → Collaborative filtering**; • **Theory of computation → Semantics and reasoning**; • **Computing methodologies → Learning to rank**; *Knowledge representation and reasoning*;

Additional Key Words and Phrases: End-User Development; Trigger-Action Programming; Internet of Things; Hybrid Recommender System; Top-N Recommendations

ACM Reference Format:
Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. 2010. RecRules: Recommending IF-THEN Rules for End-User Development. *ACM Trans. Intell. Syst. Technol.* 9, 4, Article 39 (March 2010), 27 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

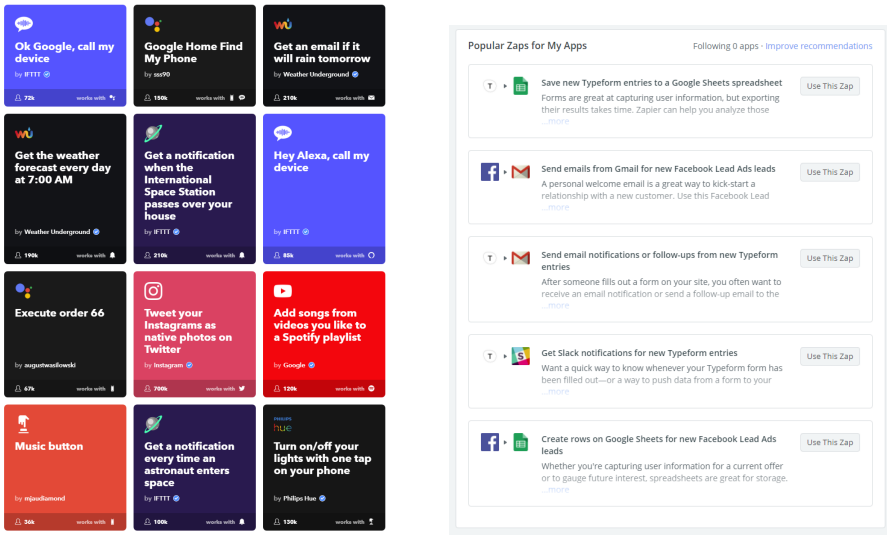
The advent of interconnected objects, devices, and sensors, commonly referred to as the Internet of Things (IoT), already helps society in many different ways, through applications ranging in scope

Authors' addresses: Fulvio Corno, Politecnico di Torino, Corso Duca degli Abruzzi, 24, Torino, Italy, 10129, fulvio.corno@polito.it; Luigi De Russis, Politecnico di Torino, Corso Duca degli Abruzzi, 24, Torino, Italy, 10129, luigi.derussis@polito.it; Alberto Monge Roffarello, Politecnico di Torino, Corso Duca degli Abruzzi, 24, Torino, Italy, 10129, alberto.monge@polito.it.

2157-6904/2010/3-ART39 $15.00
https://doi.org/0000001.0000001

from the individual to the planet [7]. As reported in recent studies [1, 22], the IoT ecosystem can be defined from a wide perspective, by including not only physical devices, but also web applications such as messaging services and social networks. The result is a complex network of connected entities, either physical or virtual, that are able to interact and communicate with each other, with humans, and with the environment. In this context, the End-User Development (EUD) vision aims at putting customization mechanisms in the hands of end users, i.e., the subjects who are most familiar with the actual needs to be met. Different EUD tools such as IFTTT[1] and Zapier[2] empower users to customize the joint behaviors of their web applications and physical devices through the *trigger-action programming* paradigm, i.e., by defining IF-THEN rules in the form of "if something happens, then perform an action." By exploiting wizard-based procedures, typically, users can reuse or directly define rules such as *"if I am tagged in a Facebook photo, then send me a Telegram message"*, or *"if my smart car enters the home geographical area, then set a given temperature on my Nest thermostat."*

To *reuse* a rule on such EUD tools, users can browse popular rules already created and shared by other people (Figure 1). To *define* a rule, instead, users have to select the involved technologies, i.e., specific devices or web applications, and then specify the desired trigger and action (Figure 2).

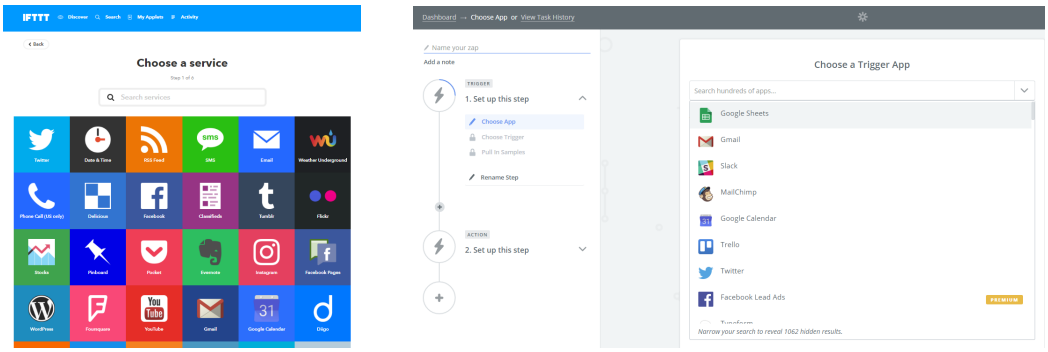


(a) Rule reuse in IFTTT (b) Rule reuse in Zapier

Fig. 1. Some IF-THEN rules ordered by their popularity, shared and reusable on IFTTT (a) and Zapier (b). Due to the growth of new supported devices and web applications, the number of shared rules is continuously growing.

Despite the trigger-action programming paradigm can express most of the behaviors desired by potential users [54], and it is adopted by the most common EUD tools [12], it presents its own set of issues. The emerging and increasing complexity of the IoT ecosystem, in particular, is barely taken into account by contemporary EUD tools, which mainly model devices and web applications on the basis of the underlying brand or manufacturer [8]. With such a low-level of abstraction, the

[1] https://ifttt.com/, last visited on February 12, 2019
[2] https://zapier.com/, last visited on February 12, 2019



(a) Rule definition in IFTTT (b) Rule definition in Zapier

Fig. 2. Rule composition in IFTTT (a) and Zapier (b). To define a trigger (or an action), users have to select a device or web application by searching in large menus of supported products. With the spread of new supported technologies, the amount of information may become too high, thus making the rule definition process difficult.

number of possible combinations among triggers and actions of different technologies is high, and the number of shared rules is growing. Zapier, for example, supports more than 1,000 devices and web applications, each one with its own triggers and actions, while the number of publicly available rules on IFTTT already exceeded 200,000 back on September, 2016 [55]. The user experience with contemporary EUD tools is therefore put to a hard test: users are forced to know in advance any involved technological detail, and they have to define several rules to program their IoT ecosystem, i.e., every web application and physical device needs to be managed separately. For instance, a user who wants to customize the behavior of her smart home through IFTTT has many possibilities: she can define a temperature to be set on the *Nest* thermostat whenever her *BMW* smart car is approaching the home area, or she can make the *Philips Hue* lamp in the kitchen turn on whenever the *Arlo* security camera detects some movements. Even in such a limited scenario, the 4 mentioned technologies offer 15 triggers and 19 actions on IFTTT, thus generating 285 candidate rules. This number is even larger if we consider specific details of each trigger and action, such as the temperature threshold for the thermostat or the light intensity of the lamp.

As a consequence, trigger-action programming is often a complex task for non-programmers [30], and the increasing number of supported technologies suggests the need of providing users with more support for discovering and managing rules and related functionality [55]. We claim that this particular type of information overload could be addressed through recommender systems, and that recommendation techniques could improve both the *reuse* and the *definition* of trigger-action rules, thus helping users who do not have technological and programming skills to easily customize their smart devices and web applications. When browsing rules already created and shared by other users, in fact, a recommender system could suggest relevant rules to be reused. When defining a new rule, instead, a recommender system could help users to complete their rules, e.g., by dynamically suggesting relevant rules on the basis of what the user has already defined.

In this paper, we tackle the emerging problem of recommending IF-THEN rules to end users by presenting *RecRules*, a hybrid and semantic recommendation system. Through a mixed content and collaborative approach, the goal of *RecRules* is to *recommend by functionality*: it suggests rules on the basis of the final behaviors users would like to define, thus abstracting details such as brands or manufactures. *RecRules* is designed as a *top-N* recommendation algorithm and it addresses

OWL[3] content-based information and collaborative user preferences in a graph-based setting to train *learning to rank* techniques. By leveraging a high-level ontological model of trigger-action programming [9], the algorithm firstly uses a semantic reasoning process to enrich IF-THEN rules with semantic information. Such a process allows *RecRules* to uncover hidden connections between rules, e.g., in terms of shared functionality. A rule for turning on a *Philips Hue* lamp, for example, is *functionally* similar to a rule for opening the *Hunter Douglas* blinds, because they share a common final goal, i.e., to light up a place. The semantically enriched rules are then combined with collaborative information in a graph-based setting, on which *RecRules* extracts three different types of path-based features, i.e., based on collaborative, technology, and functionality paths. Such features are finally used to train a learning to rank algorithm and compute *top-N* recommendations.

We evaluate the proposed approach through different experiments by exploiting real data extracted from IFTTT [55]. In particular, *(i)* we discuss, implement, and experimentally compare 3 different learning to rank algorithms, *(ii)* we compare *RecRules* with state-of-the-art recommendation algorithms, and *(iii)* we analyze the main characteristic of *RecRules*, i.e., recommending by functionality, by exploring how the different types of path-based features influence the recommendation process. Results show that the recommendation accuracy increases by taking into account the similarity between IF-THEN rules in terms of shared functionality, and demonstrate the benefits of going beyond the information on the involved technologies, brands or manufactures. Furthermore, *RecRules* outperforms state-of-the-art recommendation algorithms, and open the way for a new class of recommender systems in EUD that takes into account the actual functionality needed by end users.

To summarize, the main contributions of our work are:

- We tackle the emerging problem of IF-THEN rules recommendations: to the best of our knowledge, this is the first work in the literature that proposes a recommendation algorithm with the aim of helping users define trigger-action rules for their Internet-enabled entities, i.e., smart devices and web applications.
- We present, evaluate, and validate *RecRules*, a hybrid and semantic recommendation system of IF-THEN rules able to combine semantic data extracted through a reasoning process and user feedback in a graph-based setting.
- We describe and discuss the different path-based features used to train *RecRules*, by showing their effectiveness with respect to recommendation accuracy, diversity, coverage, and serendipity.
- We discuss the results of our work by showing that recommendation systems in the End-User Development field should move towards *recommending by functionality*, rather than suggesting rules on the basis of their popularity or involved technologies.

The remainder of this paper is structured as follows. In Section 2 we discuss related works. In Section 3 we present the main characteristics of our approach, i.e., the recommendation by functionality, by describing the semantic information used in our work. In Section 4 we present *RecRules*, our recommendation algorithm. Different evaluations of the approach are described in Section 5, while results and limitations are discussed in Section 6. Eventually, Section 7 concludes the paper and presents future works.

2 BACKGROUND AND RELATED WORKS

In this section, we first contextualize our research by describing the application domain of *RecRules*, i.e., End-User Development in the IoT. Then, we provide an overview of the state of the art about recommendation techniques in EUD. Finally, since *RecRules* is a *top-N* recommender system that

[3] https://www.w3.org/OWL/, last visited on February 12, 2019

exploits an ontological representation of trigger-action programming, we also review previous works about *top-N* and semantic recommendation algorithms.

2.1 End-User Development in the Internet of Things

End-User Development (EUD) has been defined by Lieberman et al. [35] as *"a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artifact."* Starting from iCAP [13], a visual rule-based system for PC to create context-aware applications, there has been a long story of interest in EUD. One of the most popular paradigm to empower end users in directly programming their devices and web applications is trigger-action programming [13, 54]. By defining trigger-action rules, users can connect a pair of devices or web applications in such a way that, when an event (the *trigger*) is detected on one of them, an *action* is automatically executed on the latter.

Several interfaces and platforms for trigger-action programming, such as IFTTT and Zapier, are available as off-the-shelf products. Among such solutions, IFTTT is one of the most popular [29]. Despite their apparent simplicity, interoperability and scalability challenges remain [8], while the expressiveness and understandability of IFTTT have been criticized since it is rather limited [29, 54, 55]. To solve these issues, numerous recent works explore new approaches to empower end users in programming devices and web applications. Danado and Paternò developed Puzzle [11], a mobile framework which allows end users without IT background to create, modify, and execute applications. Brich et al. [5] report on the comparison of two different notations, i.e., rule-based and process-oriented, in the smart home context, showing that trigger-action rules are generally sufficient to express simple automation tasks, while processes fit well with more complex tasks. More recently, EUD has gained interest in the IoT field as well. Akiki et al. [1] present ViSiT, an approach that allows end users to specify transformations on IoT objects that are automatically converted into executable workflows. Desolda et al. [12] report on the results of a study to identify possible visual paradigms to compose trigger-action rules in the IoT, and present a model and an architecture to execute them. Ghiani et al. [22] propose a method and a set of tools for end users to personalize the contextual behavior of their IoT applications through trigger-action rules. Similarly, Corno et al. propose EUPont [9], an ontological representation of End-User Development in the IoT for the creation of abstract rules that adapt to different contextual situations.

In our work, we adopt a different approach. As pointed out by Haines et al. [24], recommendation approaches could be useful to help end users without programming skills to use EUD systems, and advances in EUD have expanded the opportunities for offering recommendations. Therefore, the goal of *RecRules* is to directly suggest rules to be selected rather than acting on the underlying languages and models. Recommendations could be used for assisting users in reusing rules, e.g., by suggesting relevant rules from those publicly available, or for completing the definition of a new rule, e.g., with suggestions that include the already defined trigger.

2.2 Recommendations for Software Engineering

Recommendation opportunities in EUD have not yet been consistently explored, and most contemporary EUD solutions still continue to offer limited types of suggestions, e.g., by promoting the most popular rules, only. From the early works, EUD systems like EAGER [10] and Dynamic Macro [38] were using simple proactive suggestions to help end users define their programs. Moreover, in the last decade, recommendation technologies have been studied in the field of software engineering, mainly, from systems for feature recommendations [25, 32] to tools for source code suggestions [40]. The goal of these works, however, is to assist developers, instead of end users. In the same field, Ye and Fisher propose *CodeBroker* [60], a development environment that autonomously locates and delivers task-relevant and personalized components into the current software development

environment. Malheiros et al. [37], instead, developed a recommender system to help novice developers solve change requests in their source code. Other previous works targeting developers aim at suggesting APIs to facilitate expert-users in performing different tasks. Duala-Ekoko and Robillard [15] propose an approach that leverages the structural relationships between API elements to make the related methods more discoverable. Nguyen et al. [42] present a novel API recommendation approach, based on statistical learning, that taps into the predictive power of repetitive code changes to provide relevant API recommendations. D'Souza et al. [14] developed *PyReco*, an intelligent code completion system for Python that uses the mined API usages from open source repositories to order the results according to relevance rather than the conventional alphabetic order.

Besides developers, only a few recent works takes into account end users, e.g., by suggesting relevant smart "things" based on user preferences and interests, to optimize the time and cost of using IoT in a particular situation [59]. Instead of suggesting smart "things," *RecRules* suggests trigger-action rules that relate pairs of entities: to the best of our knowledge, this is the first work in the literature that proposes a recommendation algorithm with the aim of helping users define trigger-action rules for their Internet-enabled devices and web applications.

2.3 Top-N Recommendation Systems

Referring to the *top-N* recommendation problem, several works have been proposed in the last few years. One of the most popular algorithm in this field is SLIM [43], an algorithm that uses a sparse linear method for learning a sparse aggregation coefficient matrix to be used for computing *top-N* recommendations. SLIM has been extended to incorporate both users and side information about items [44]. More recently, Wu et al. [58] propose the Collaborative Denoising Auto-Encoder (CDAE) algorithm, a novel method for top-N recommendation that utilizes the idea of Denoising Auto-Encoders.

Other works represent the *top-N* recommendation task as a ranking problem using *learning to rank*. Rendle et al. [48] propose a Bayesian Personalized Ranking (BPR) criterion for optimizing a ranking loss. Also BPR has been extended in other works, e.g., to compute useful recommendations in cold start scenarios (BPR-MF [19]). Shi et al. [51] developed CLiMF, a novel collaborative filtering approach in which the model parameters are learned by directly maximizing the Mean Reciprocal Rank (MRR), which is a well-known information retrieval metric for measuring the performance of *top-N* recommendations. The same authors developed TFMAP [51], a model that directly maximizes Mean Average Precision with the aim of creating an optimally ranked list of items for individual users under a given context. TFMAP uses tensor factorization to model implicit feedback data (e.g., purchases, clicks) with contextual information.

In our work, we compare *RecRules* with several state-of-the-art *top-N* recommendation algorithms, ranging from BPR-MF to Least Square SLIM [18].

2.4 Semantic-Aware Recommendation Systems

To compute *top-N* recommendations, *RecRules* follows a semantic approach able to capture the different relationships among IF-THEN rules. Such relationships range from technology-based similarities to similarities in terms of shared functionality, independently of the involved devices or web applications. Several works on ontological recommender systems have been proposed in the literature [2, 6, 41, 49] with the aim of improving the performance of recommender systems, and to overcome some drawbacks of collaborative methods such as cold start and data sparsity. Furthermore, in the last 10 years, the advent of the Linked Open Data (LOD) [3] initiative opened the way for a new class of ontological recommender systems based on data freely available on the Web. One of the first approaches that exploits LOD to build a recommender system is the work of

Heitmann and Hayes [26]. Here, the authors demonstrate that the usage of Linked Data mitigates the new-user, new-item, and sparsity problems of collaborative recommender systems. Fernández-Tobías et al. [16] show a knowledge-based framework for cross-domain recommendations leveraging DBpedia[4]. Khrouf and Troncy [31], instead, present a novel hybrid approach built on top of Semantic Web for event recommendations. Their system is enhanced by the integration of a user diversity model designed to detect user propensity towards specific topics.

Contextually to the progression of LOD-based recommender systems, recommendation methods based on generic heterogeneous networks have recently emerged. Knowledge graph embedding approaches, in particular, have proven to be effective to improve recommendations: they connect various types of information related to items (e.g., genre, director, actor of a movie) in a unified global space, which helps to develop insights on recommendation problems that are difficult to uncover with user-item interaction data only [53]. State of-the-art methods in this context mainly extend the latent factor model by considering similarities between items derived from paths in a knowledge graph. Yu et al. [61], for instance, present a network-based entity recommendation method that uses path-based latent features to represent the connectivity between users and items along different types of paths. A global and local matrix factorization model, in particular, are learnt by using the BPR [48] approach. Lao and Cohen [33] improve the classic Random Walk with Restart approach by proposing a proximity measure defined as a weighted combination of path types, called *path experts*, obtained by fitting a logistic regression model. Ostuni et al. propose SPRank [47], a hybrid recommender system to compute *top-N* recommendations from implicit feedback using linked data sources. SPRank directly formulates the problem of computing *top-N* recommendations in the standard learning to rank setting adopted in Web search [36] by replacing queries with users and document with items, and using user's ratings as relevance scores. It exploits an RDF graph, and it computes recommendations by training a learning to rank algorithm through path-based features. SPRank has also been extended to work with explicit feedback [45]. Similarly to SPRank, but with a more specific application area, Oramas et al. [46] describe how to create and exploit a knowledge graph to supply a hybrid recommendation engine with information that builds on top of a collection of documents describing musical and sound items. They link the items to be recommended to external graphs such as WordNet and DBpedia through tags and textual descriptions, thus semantically enriching the initial data, and they add to the knowledge graph collaborative filtering information by connecting users to musical and sound items on the basis of their ratings. Then, they use the resulting knowledge graph in two versions of the algorithm, by formulating different explicit graph feature mappings based on entities and paths, respectively. Our approach is similar to previous works based on heterogeneous networks, in the way that we combine semantic and collaborative information to build a hybrid knowledge graph, and we use path-based features for mining the user-item interactions. However, we exploit a semantic reasoning process on OWL ontologies to enrich IF-THEN rules with side information. Such a process allows the algorithm to uncover hidden connections between rules, and to capture the meaning of different types of path-based features.

3 RECOMMENDING BY FUNCTIONALITY: KNOWLEDGE ENRICHMENT VIA SEMANTIC REASONING

RecRules aims at suggesting rules on the basis of the final behaviors users would like to define, e.g., increasing the temperature in a room. The idea is to abstract details such as involved technologies, brands or manufactures. In this way, *RecRules* can be exploited to compute recommendations for yet unknown or rarely used devices or web applications, thus helping users to discover new

[4]http://dbpedia.org, last visited on February 12, 2019

functionality, starting from their actual needs. The goal, in particular, is to overcome interoperability and expressiveness issues of contemporary EUD tools for trigger-action programming. Such tools, in fact, mainly model devices and web applications on the basis of the underlying brands or manufacturers, thus describing the IoT ecosystem with a low-level of abstraction [9]. As a result, trigger-action rules that differ from each other for some minor details, only, e.g., the manufacturer of a device, are considered different even if they are conceptually similar. A user that has already defined a rule for turning on her *Philips Hue* lamp in the kitchen, however, could be also interested in turning on her *LIFX* lamp in the bedroom, or in opening the living room blinds. Specifically, she could be interested in the following rules:

R1 *"if the kitchen Nest Cam recognizes me, then turn on the kitchen Philips Hue"*
R2 *"if the living room Homeboy Cam detects a movement, then open the Hunter Douglas blinds"*
R3 *"if I open the SmartThings bedroom door, then turn on the bedroom LIFX lamp."*

Even if they share a common goal, i.e., to light up a room, **R1**, **R2**, and **R3** are considered different in contemporary EUD tools because they refer to different manufacturers. With such a technological-centric approach, the users experience with EUD tools is very limited, and the actual end-user needs' are not taken into account. As suggested by Ur et al. [55], the continuous growth of trigger-action programming in the real world, and its application to a range of online services and physical devices, suggests the need to provide users with more support for discovering *functionality*, i.e., the behaviors that rules aim to define.

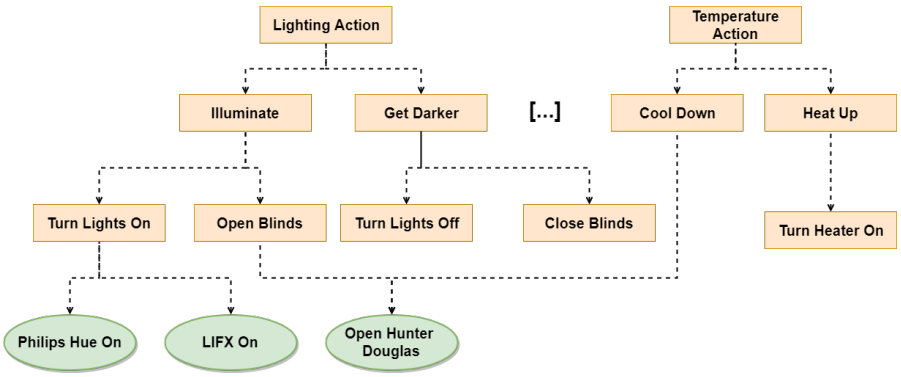


Fig. 3. A partial view of the hierarchy of some lighting and temperature-related actions modeled in EUPont. Rectangles are OWL classes, while ovals are OWL individuals, that in this case represent actions of the IFTTT ecosystem.

To capture the similarities between **R1**, **R2**, and **R3**, thus characterizing them on the basis of their final purpose, we exploit a semantic reasoning process that enrich IF-THEN rules with semantic information. Differently from previous works on semantic recommender systems, that simply associate items with relevant entities defined in online semantic datasets [21], the usage of reasoning capabilities allows *RecRules* to capture detailed information and to discover hidden relationships between trigger-action rules, e.g., in terms of shared functionality. The reasoning process uses an OWL high-level representation of trigger-action programming, named EUPont [9]. EUPont allows the classification of trigger-action rules by considering the behaviors they aim to define, by abstracting triggers and actions independently of the involved technologies, brands, manufacturers, and user context. The EUPont representation can be instantiated and used to model rules of any EUD tools for trigger-action programming, including the most complex ones that

allow multiple triggers and actions. With its hierarchical functionality representation (Figure 3), the similarities between **R1**, **R2**, and **R3** can be retrieved by semantically reasoning on the classes and super-classes of the involved triggers and actions. Both the actions *"turn on the Philips Hue lamp"* (**R1**) and *"turn on the LIFX lamp"* (**R3**), for example, are instances of the `Turn Lights On` action class, which is a subclass of the `Illuminate` action class. Furthermore, the action *"open the Hunter Douglas blinds"* (**R2**) is an instance of the `Open Blinds` class, which is a sibling class with respect to the `Turn Lights On` one. The semantic reasoning process is detailed in Section 4.3.1.

4 RECRULES

RecRules is designed as a hybrid and semantic recommendation system for suggesting IF-THEN rules to end users. In this section, we first describe the underlying semantic model we used in the recommendation process. Then, we formulate the addressed problem, and we describe our approach by detailing the different parts of our algorithm.

4.1 Knowledge Graph Model

Linked Open Data (LOD) datasets can be viewed as vast decentralized knowledge graphs, where nodes correspond to RDF[5] entities, and labeled edges are properties connecting them. Furthermore, the semantic graph can be enriched with collaborative filtering information by embedding the collaborative filtering problem in the semantic graph as well, where users and items to be recommended are the nodes, and users' feedback, either implicit or explicit, are the links [45]. By reasoning on OWL ontologies, the semantic graph can be further enriched with the additional properties and features given by OWL classes and sub-classes (Figure 4).

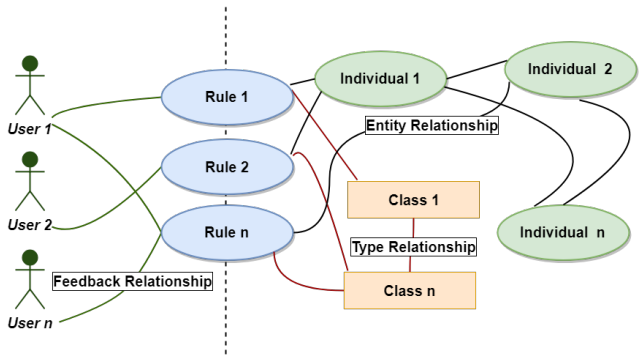


Fig. 4. The knowledge graph built in *RecRules*. Users are connected to rules by means of feedback relationships. Rules are in turn connected to RDF individuals through individual relationships, and to OWL classes through class relationships.

The resulting multi-relational graph can be modeled as a directed graph $G = \{V, \Sigma\}$ where V denotes the set of vertices and Σ indicates the set of properties that connect them. In our model, the set of vertices is defined as $V = U \cup R \cup I \cup C$, where $u \in U$ are *users*, $r \in R$ are *rules*, $i \in I$ are RDF *individuals*, and $c \in C$ are OWL *classes*.

Semantic properties may have different nature, so each property is labeled with one label from the set $L = \{F, E, T\}$. The set of properties $\Sigma \subseteq V \times V \times L$ is the union of such different types of relationships, i.e., $\Sigma = \Sigma_F \cup \Sigma_E \cup \Sigma_T$:

[5] https://www.w3.org/RDF/, last visited on February 12, 2019

- $\Sigma_F \subseteq U \times R \times \{F\}$ represents the set of *feedback relationships* (label F), which connects users to rules;
- $\Sigma_E \subseteq I \times R \times \{E\} \cup I \times I \times \{E\}$ represents the set of *entity relationships* (label E), which connects rules with RDF individuals, or RDF individuals among themselves;
- $\Sigma_T \subseteq C \times R \times \{T\} \cup C \times C \times \{T\}$ represents the set of *type relationships* (label T), which connects rules with OWL classes, or OWL classes among themselves.

4.2 Problem Formulation

Given the described model, the problem is formulated as follows. Considering the set of all users U, and the set of all rules R, the *user profile* of each user $u \in U$ is defined as $H_u = \{r \in R \mid \hat{r}_{ur} \in \hat{R}\}$, where:

- in case of explicit feedback, $\hat{R}$ is the user-item matrix, with each $\hat{r}_{ur} \in \mathbb{R}$ representing the rating of the user u on the rule r;
- in case of implicit feedback, $\hat{R}$ is the implicit feedback matrix, with each $\hat{r}_{ur} \in \mathbb{R}$ representing the implicit relevance of the rule r for the user u.

Regardless of the feedback type, the matching between user interests and rule content for each user-rule pair $(u, r) \in U \times R$ is mapped in the feature vector $\vec{x_{ur}} \in \mathbb{R}^D$, where D is the dimension of the feature space: each component $x_{ur} \in \mathbb{R}$ represents the connection of the rule r to the user u according to a specific feature k. The definition of the features is detailed in Section 4.3.2. In particular, we rely on different types of path-based features, able to characterize the interaction between users and rules with respect to collaborative information, technology-based similarities, and functionality-based similarities. Eventually, the training set TS and the recommendation set RS are defined as:

$$TS = \bigcup_u \{< u, r, \vec{x_{ur}}, \hat{r}_{ur} > \mid r \in H_u\} \tag{1}$$

$$RS = \bigcup_u \{< u, r, \vec{x_{ur}}, \hat{r}^*_{ur} > \mid r \in (R \backslash H_u)\} \tag{2}$$

The final goal of the recommender system is to learn a scoring function from the training data TS able to predict $\hat{r}^*_{ur}$, in order to replicate for each user their perfect ranking.

4.3 Recommendation Approach

The architecture of *RecRules* is shown in Figure 5. The algorithm has been implemented in Java by using the Linked Open Data Recommender Systems Library [45] (lodreclib[6]), and it is composed of two main phases, i.e., *Knowledge Graph Construction* and *Learning*, joined by a contextual filter (*User Context Filter*). The implementation we adopted for our experiments is available at https://git.elite.polito.it/public-projects/recrules.

4.3.1 Knowledge Graph Construction. Starting from a set of recommendable IF-THEN rules and the users' history, i.e., which rules have been already defined or reused, *RecRules* first build a knowledge graph that combines RDF individuals, OWL classes, and collaborative information. Individuals, in particular, represent the *technology information* that characterize the involved rules, i.e., involved devices and web applications. Classes, instead, represent the *functionality information* that are used to characterize the rules in terms of their final goal, independently of the involved technological details.

[6] https://github.com/sisinflab/lodreclib, last visited on February 12, 2019

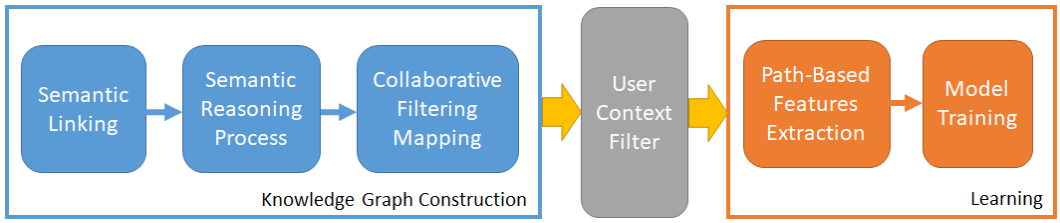


Fig. 5. A schematic representation of the *RecRules* algorithm.

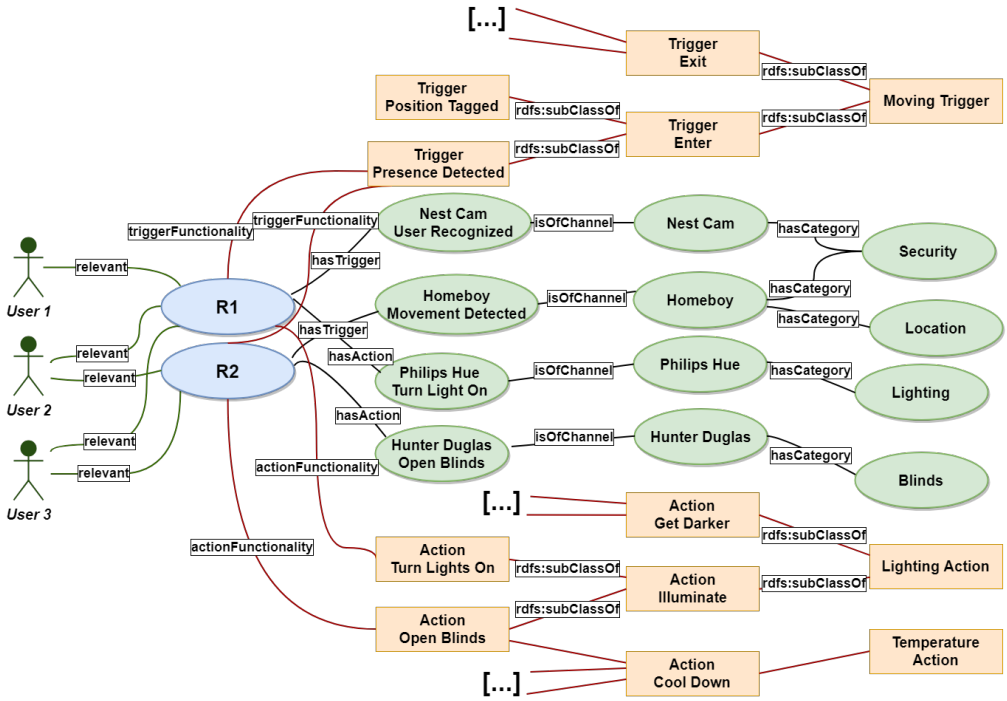


Fig. 6. The knowledge graph built by *RecRules* by considering the rules R1 and R2, reported in Section 3. Rules are linked with individuals (ovals), i.e., the technology information, classes (rectangles), i.e., functionality information, and users, i.e., collaborative information.

Figure 6 shows an example of a simple knowledge graph built by considering R1 and R2, already reported in Section 3. Both rules have been extracted from the dataset exploited in the evaluation and rephrased for the sake of readability:

R1 *"if the kitchen Nest Cam recognizes me, then turn on the kitchen Philips Hue;"*
R2 *"if the living room Homeboy Cam detects a movement, then open the Hunter Douglas blinds."*

In the remainder of this section, we describe the creation of the graph, and we show how it is used in the recommendation process. In the example, we deliberately chose to represent a simple scenario with 2 users and 2 rules, only, to allow a better understanding of the underlying process. In a normal recommendation process, the algorithm models much more complex scenarios, where multiple users and rules are connected through different complex paths.

Semantic Linking & Graph Instantiation. To build a semantic graph such as the one reported in Figure 6, trigger-action rules need to be linked to ontological sources. Different techniques and available ontological sources can be used for this purpose: the identification of entities in text-based resources is a well-known task in the Natural Language Processing community and it has recently gained momentum thanks to the availability of knowledge bases publicly available on the Web [50]. In our work, we link trigger-action rules with the EUPont ontology. Since the dataset exploited in the evaluation (Section 5) is composed of trigger-action rules extracted from IFTTT, we exploit the instantiation of EUPont for IFTTT[7], that offers a hierarchical functionality representation of more than 500 triggers and actions supported by the popular platform. For linking rules to such an ontology, in particular, we use the translation procedure described in [8], with which triggers and actions are linked to the corresponding ontological entities on the basis of their description and the involved devices and web applications. After the linking process, *RecRules* instantiates a semantic graph by adding the trigger-action rules to be recommended, and by connecting them to the *technology information* that can already be extracted from contemporary EUD tools. Such information are represented as RDF individuals, while their connections as entity relationships. In the graph of Figure 6 individuals are represented as ovals, and their connections as the `hasAction`, `hasTrigger`, `isOfChannel`, and `hasCategory` entity relationships. The rule *"if the kitchen's Nest Cam recognizes me, then turn on the kitchen Philips Hue"* (**R1**) is therefore linked with the `Nest Cam User Recognized` trigger and with the `Philips Hue Turn Light On` action, for example.

Semantic Reasoning Process. To enrich IF-THEN rules with semantic information, and to characterize them in terms of shared functionality, we use a semantic reasoning process. The process analyzes triggers and actions of each rule, and recursively extracts their OWL classes and superclasses that represent the hierarchical characterization of triggers and actions offered by EUPont (*functionality information*, represented as rectangles in Figure 6). Such information are then linked in the form of OWL classes to the involved IF-THEN rules by means of the `triggerFunctionality`, `actionFunctionality`, and `subclassOf` type relationships. To implement the semantic reasoning process, *RecRules* uses OWL API [27], a high level Application Programming Interface (API) for working with OWL ontologies in Java, and the HermiT reasoner [23]. A semantic reasoner is a piece of software able to infer logical consequences from a set of asserted facts or axioms. HermiT, in particular, supports several specialized reasoning services such as class and property classification, as well as a range of features outside the OWL standard such as DL-safe rules and description graphs. We employ it to discover all the EUPont classes that can be used to characterize a trigger or an action, including the cases in which this information is not explicitly available, e.g., when a trigger or an action can be classified under different branches of the EUPont hierarchy. The rule *"if the living room Homeboy Cam detects a movement, then open the Hunter Douglas blinds"* (**R2**), for example, is connected both to the `Lighting Action` and the `Temperature Action` hierarchy branches (Figure 6).

Collaborative Filtering Mapping. The last part of Knowledge Graph Construction phase is the Collaborative Filtering Mapping, with which *RecRules* adds *collaborative information* to the semantic graph, i.e., how the rules are used and supported by the community of end users. In this way, the algorithm is able to discriminate between relevant rules, i.e., rules that are appreciated by the users, and not relevant rules, i.e., rules that are not appreciated and therefore should not be recommended. To add such feedback relationships, the algorithm defines a labeling function

[7]available at http://elite.polito.it/ontologies/eupont-ifttt.owl

$$\gamma : \mathbb{R} \longrightarrow \{\text{relevant, not relevant}\} \tag{3}$$

Different strategies can be used to define the labeling function, ranging from implicit feedback metrics (e.g., how many times a rule has been reused by other users), to explicit feedback metrics (e.g., rule ratings). In our evaluation of the approach (Section 5) we exploit a Graded Implicit Feedback [34] strategy by normalizing between 1 and 5 the number of times a rule has been reused by others.

4.3.2 Learning. The generated semantic graph is then used in the Learning phase to extract different types of path-based features and to train a *learning to rank* model. Depending on the user for whom the recommendations need to be computed, the graph can be dynamically filtered through the *User Context Filter*. After the filtering process, the graph is restricted to trigger-action rules that can be actually recommended to that user, i.e., the rules involving devices or web applications that the user is authorized to control.

Path-Based Features Extraction. To recommend IF-THEN rules, *RecRules* exploit the underlying connections modeled in the semantic graph. From the graph, in particular, the algorithm extracts path-based features able to characterize the interaction between users and rules. Given the sets of users U and rules R, a path $p_{u,r}$ is an acyclic sequence of adjacent relationships of length greater or equal than 2 that links a user $u \in U$ to a rule $r \in R$ in the semantic graph. The first step of the path $p_{u,r}$ links the user u to a rule belonging to her user profile H_u, while the rest of the path reaches the terminal rule r that does not necessarily belong to the user profile. Thanks to the information coded in the semantic graph, *RecRules* is able to distinguish paths on the basis of their meaning:

- **Collaborative Paths.** A collaborative path involves feedback relationships, only, i.e., it connects a user to a rule by means of other users. Considering the property labels L, collaborative paths have the form $(F, F, F, ..., F)$. Figure 7 shows a collaborative path extracted from the semantic graph of Figure 6. Here, u_1 is linked to r_2 by means of the path $p1_{u_1,r_2} = \{relevant, relevant^{-1}, relevant\}$. Besides this simple example, the algorithm is able to deal with more complex collaborative paths, i.e., that involve multiple rules connected through feedback relationships.
- **Technology Paths.** A technology path is a path that, excluding the first relationship, involves entity relationships, only, i.e., it connects a user to a rule by means of technological information. Technology paths have the form $(F, E, E, ..., E)$. Figure 8 shows a technology path extracted from the semantic graph of Figure 6. Here, u_1 is linked to r_2 by means of the path $p2_{u_1,r_2} = \{relevant, hasTrigger, isOfChannel, hasCategory, hasCategory^{-1}, isOfChannel^{-1}, hasTrigger^{-1}\}$. Besides this simple example, the algorithm is able to deal with more complex technology paths, i.e., that involve multiple rules connected through entity relationships.
- **Functionality Paths.** A functionality path is a path that, excluding the first relationship, involves type relationships, only, i.e., it connects a user to a rule by means of functionality information. Functionality paths have the form $(F, T, T...T)$. Figure 9 shows a functionality path extracted from the semantic graph of Figure 6. Here, u_1 is linked to r_2 by means of the path $p3_{u_1,r_2} = \{relevant, actionFunctionality, subClassOf, subClassOf^{-1}, actionFunctionality^{-1}\}$. Also in this case, besides this simple example, the algorithm is able to deal with more complex technology paths, i.e., that involve multiple rules connected through type relationships.

We define the signature k of a path $p_{u,r}$ as the sequence of the actual properties traversed by the path. The signature $k = 1$ of the path $p1_{u_1,r_2}$ is, for example, $\{relevant, relevant^{-1}, relevant\}$. Each

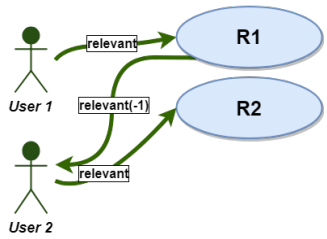


Fig. 7. Connection between User 1 and Rule 2 through a collaborative path.

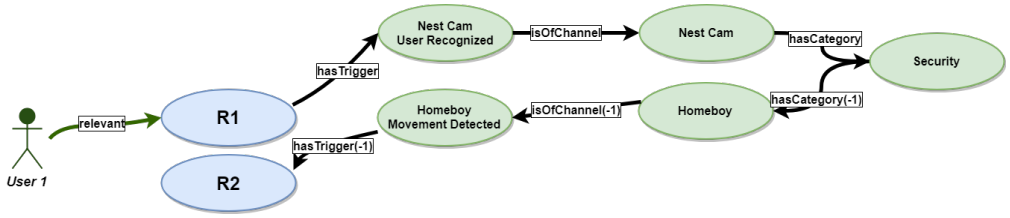


Fig. 8. Connection between R1 and R2 through a technology path.

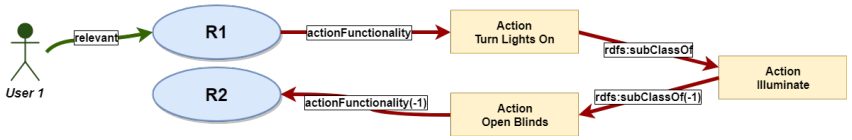


Fig. 9. Connection between R1 and R2 through a functionality path.

distinct signature k maps to a distinct dimension $x_{ur}(k)$ of the feature vector $\vec{x_{ur}}$. Considering a path with signature k, in particular, the feature component $x_{ur}(k)$ is computed by counting $\#p_{u,r}(k)$, i.e., how many *instances* of paths having signature k exist between a user u and a rule r. For the graph of Figure 6, for example, $\#p_{u_1,r_2}(1) = 2$, since there are 2 instances of the path $\{relevant, relevant^{-1}, relevant\}$ that connect u_1 to r_2. To compare features between different users, we follow the strategy of Di Noia et al. [45] by introducing a user-based normalization. At the end, the components of the feature vector are computed with the following formula:

$$x_{ur}(k) = \frac{\#p_{u,r}(k) - \min_{w \in R}(\#p_{u,w}(k))}{\max_{w \in R}(\#p_{u,w}(k)) - \min_{w \in R}(\#p_{u,w}(k))} \tag{4}$$

Equation (4) represents the importance of the path of type k between user u and rule r.

Model Training. To finally compute *top-N* recommendations, a ranking model from training data is built using a learning to rank technique. The goal of the Model Training phase is to learn a scoring function in such a way the model can sort new items according to their relevance. Different techniques can be used for this purpose. They can be classified into three main categories: pointwise, pairwise, and listwise. In our evaluation of *RecRules* (Section 5) we explore three standard learning to rank algorithms, one for each learning to rank category.

- **Pointwise Learning to Rank.** Pointwise approaches look at a single instance at a time, and transform the ranking problem into a regression or a classification one. In particular, they take a single instance at a time, and train a classifier (or a regressor) to predict its relevance.

Each instance is therefore independent from each other, and the final ranking is achieved by simply sorting the result list looking at the predicted scores. For the pointwise category, *RecRules* uses Random Forest [4], an algorithm that constructs a multitude of decision trees at training time and outputs the class label for classification (or the mean prediction for regression) of the individual trees.

- **Pairwise Learning to Rank.** Pairwise approaches look at a pair of instances at a time, and try to find out their optimal ordering. The goal for a pairwise ranker is to minimize the number of inversions in ranking, i.e., cases where the pair of results are in the wrong order relative to the ground truth. For the pairwise category, *RecRules* uses RankBoost [17], an algorithm that builds a linear combination of weak rankers, and optimizes a loss function based on the exponential difference between the relevance of pairs of items.
- **Listwise Learning to Rank.** Listwise approaches directly look at the entire list of instances. In particular, they try to come up with the optimal ordering by minimizing a loss function defined over the ranked list of instances. For the listwise category, *RecRules* uses LambdaMart [57] (LMART), an algorithm that exploits the normalized Discounted Cumulative Gain (nDCG) metric for fitting the parameters of the regression trees

5 EVALUATION

We evaluated *RecRules* through different experiments. First, we assessed the effectiveness of *recommending by functionality* by exploring the accuracy of 3 learning to rank techniques trained with different types of path-based features. Finally, we compared *RecRules* with state-of-the-art collaborative filtering, ranking-oriented, and semantic-aware recommendation algorithms.

In the remainder of this section, we first provide a description of the evaluation process. Then, we report on the results coming from the different evaluations.

5.1 Evaluation Description

5.1.1 Exploited Dataset. Our evaluations are based on a dataset of trigger-action rules extracted from IFTTT [55], one of the most popular EUD tools. To the best of our knoweldge, this is the only publicly available dataset of IF-THEN rules defined and shared by different users. It was obtained by Ur et al. [55] with a web scrape of the IFTTT platform as of September 2016. The dataset contains 295,156 rules created and shared by 129,206 different authors, and has a high degree of sparsity (97.51%, Table 1).

Table 1. IFTTT Dataset statistics.

Metric	Value
Users (#)	129,206
Items (#)	295,156
Sparsity (%)	97.51

Table 2 shows a rule extracted from the dataset. Beside the information about the trigger (Trigger Channel, Trigger, and Trigger Description) and the action (Action Channel, Action, and Action Description), each rule includes data about the author who created it (Author), a description of the entire rule (Description), and the information about how many times the rule has been reused by other users (Shares). According to the dataset, some rules were reused more than 400,000 times, while others were reused only by one user (M=837.79, SD=9452.77, $range$=1 : 476355). We used this information to calculate the labeling function γ (Eq. 3), in order to define relevant and not relevant

Table 2. Example of a rule stored in the IFTTT dataset exploited for the evaluation.

Field	Value
Id	100301
Description	Save Soundcloud likes to Google Drive.
Author	gigaphon
Trigger Channel	SoundCloud
Trigger	New public like
Trigger Description	This Trigger fires every time you like a public track.
Action Channel	Google Drive
Action	Upload file from URL
Action Description	This Action will download a file at a given URL and add it to Google Drive at the path you specify. NOTE: 30 MB file size limit.
Shares	2.2k

rules. As suggested in [34], instead of randomly choosing negative data points, we computed a Graded Implicit Feedback (GIF) by normalizing the number of reuse between 1 and 5. Then, we defined γ as

$$\gamma = \begin{cases} \text{relevant}, & \text{if GIF} > 3 \\ \text{not relevant}, & \text{otherwise} \end{cases} \tag{5}$$

The threshold was empirically set to 3 to obtain two homogeneous groups. Our idea was to promote rules that were already reused by a consistent number of users. Table 3 reports the distribution stemming from the labeling function γ: at the end, 45.88% of the rules were considered as "not relevant", while the remaining 54.12% were considered as "relevant." Finally, since the dataset does not contain any information about which devices or web applications can be actually controlled by each user, we set up the *User Context Filter* by supposing that each user is authorized to control *any* connected entity. For all the experiments, we follow a k-fold cross-validation approach by randomly splitting up the dataset into 10 groups. Similarly, the tuning of the parameters in the learning to rank algorithms was performed through cross validation on validation data obtained by selecting 15% of items for each user from the original dataset.

Table 3. Distribution of the computed Graded Implicit Feedback (GIF), obtained by normalizing between 1 and 5 the number of shares.

GIF	Label	Coverage
1	not relevant	32.46%
2	not relevant	13.42%
3	relevant	5.38%
4	relevant	14.3%
5	relevant	34.44%

5.1.2 Evaluation Protocol. In all the evaluations, we used the "all unrated items" methodology [52], that consists in computing a *top-N* recommendation list for each user and by predicting a score for every item not rated by that particular user, including those that are not in the test set. The main assumption in this methodology is that all the unrated items are considered to be irrelevant for the

user, with the effect of underestimating the recommendation quality. However, the user experience in *top-N* recommendation applications depends on the ranking of all items. This implies that, in this case, the "all unrated items" methodology is more suitable than the "rated test-items," where only the test data are considered for generating the *top-N* recommendations [52].

We used different metrics for investigating the results. All of them were computed for each single user and then averaged. In particular, we measured the recommendation accuracy with three standard performance metrics, i.e., precision, recall, and normalized discounted cumulative gain.

For the definition of the metrics, we considered:

- R, i.e., the set of all the trigger-action rules;
- S, i.e., the set of *top-N* recommendations;
- $m^{+,N}$, i.e., the number of recommendations in S that are relevant;
- m^{+}, i.e., the number of all relevant rules.

Precision represents the fraction of the *top-N* recommended rules that are relevant among all the recommended items S. It is computed with the formula

$$prec@N = \frac{m^{+,N}}{N} \tag{6}$$

Recall represents the fraction of the *top-N* recommended rules that are relevant over the total amount of relevant rules. The recall is computed with the formula

$$rec@N = \frac{m^{+,N}}{m^{+}} \tag{7}$$

Differently from precision and recall, which are binary metrics that consider the relevance of the items, only, **normalized Discounted Cumulative Gain (nDCG)** can handle graded values by taking into account both relevance and rank position. Given $\hat{r}_{ur_j}$, i.e., the GIF of u to the rule r_j in the j-th position of the recommended rules S, and $IDCG@N$, i.e., a normalization factor that represents the score obtained by an ideal or perfect *top-N* ranking, $nDCG@N$ is computed with the formula

$$nDCG@N = \frac{1}{IDCG@N} \sum_{j=1}^{N} \frac{2^{\hat{r}_{ur_j}} - 1}{\log_2(1+j)} \tag{8}$$

In addition to precision, recall, and nDCG, we used diversity, coverage, and serendipity metrics to investigate our results beyond accuracy. Standard accuracy metrics, in fact, cannot measure all the different aspects of a recommendation process, and the recommendations that are most accurate according to the standard metrics are sometimes not the recommendations that are most useful to users [39].

Diversity measures how dissimilar recommended items are for a user. A popular metric for measuring diversity is the Intra-List Similarity (ILS) [63]. We used the Jaccard similarity coefficient J to calculate the similarity between 2 rules in terms of technologies, i.e., involved devices or web applications. We firstly defined $\tau(r)$ as the set of technologies involved in triggers and actions of a given rule r. Then, we define ILS as

$$ILS@N = \frac{1}{2} \sum_{r_i \in S} \sum_{r_j \in S} J(\tau(r_i), \tau(r_j)) \tag{9}$$

The Jaccard similarity coefficient measures similarity between finite sample sets, e.g., A and B. The coefficient is defined as the size of the intersection divided by the size of the union of the sample sets:

$$J(A, B) = \frac{|A \cap B|}{|A \cup B|} \tag{10}$$

In our case, $A = \tau(r_i)$ and $B = \tau(r_j)$ represent the set of technologies involved in triggers and actions of 2 different rules.

Coverage represents the percentage of things (items, users, or ratings) that the recommender system is able to recommend. Not being able to predict a particular set of users or items is usually caused by an insufficient number of ratings, and is generally known as the cold start problem. In our work, we used the item coverage metric, i.e., the number total number of recommended rules (n) over the total number of trigger-action rules

$$COV@N = \frac{n}{|R|} 100 \tag{11}$$

Serendipity is the measure of how surprising the successful or relevant recommendations are. We assessed serendipity through the unserendipity metric [62], by using the Jaccard similarity coefficient J to measure the technological similarity in terms of involved devices or web applications between rules in the user's profile (H_u) and new recommendations in S. Lower values of this metric indicate that recommendations deviate from a user's traditional behavior, and hence are more surprising. For a given user u, in particular, unserendipity is defined as

$$UNSER@N = \frac{1}{|H_u|} \sum_{r_i \in H_u} \sum_{r_j \in S} \frac{J(\tau(r_i), \tau(r_j))}{20} \tag{12}$$

5.2 Evaluation I: Effectiveness of the Functionality Features

In the first evaluation, we analyzed the main characteristic of *RecRules*, i.e., recommending by functionality, by exploring to what extent the different types of path-based features influence the recommendation process. For this purpose, we explored the accuracy of 3 learning to rank techniques for the *Model Training* phase of RecRules, i.e., Random Forest, RankBoost, and LambdaMart, in two different configurations. In particular, *i)* we first trained the algorithms by exploiting **c**ollaborative and **t**echnology paths ($RecRules_{ct}$), only, and *ii)* we then trained the same algorithms by using all the **c**ollaborative, **t**echnology, and **f**unctionality paths ($RecRules_{ctf}$), with the aim of understanding whether the functionality information improved the recommendation process.

Table 4 reports the results in terms of precision, recall, and nDCG for each learning to rank algorithm in the $RecRules_{ct}$ configuration. As shown in the table, Random Forest and LambdaMart (the pointwise and listwised approaches, respectively) provided similar results, and performed better than RankBoost (the pairwise approach) in all the three metrics.

Table 5 reports the results in terms of precision, recall, and nDCG for each learning to rank algorithm in the $RecRules_{ctf}$ configuration. Also in this case, the results for Random Forest and LambdaMart are similar, and, in average, such two algorithms performed better than RankBoost for all the three metrics.

Table 4. Comparative results of three learning to rank algorithms trained with collaborative and technolgy-based features, only. Results are the average of 5 equals experiments.

	Category	prec@5	rec@5	nDCG@5	prec@10	rec@10	nDCG@10
Random Forest	Pointwise	**0.1077**	**0.2090**	**0.4920**	**0.0772**	**0.2901**	**0.5830**
RankBoost	Pairwise	0.0743	0.1309	0.4558	0.0570	0.1998	0.5515
LambdaMart	Listwise	0.0900	0.1918	0.4884	0.0633	0.2589	0.5756

Table 5. Comparative results of three learning to rank algorithms in *RecRules* trained with collaborative, technology, and functionality-based features. Results are the average of 5 equals experiments.

	Category	prec@5	rec@5	nDCG@5	prec@10	rec@10	nDCG@10
Random Forest	Pointwise	**0.1211**	**0.2177**	**0.5054**	0.0813	**0.3019**	**0.6452**
RankBoost	Pairwise	0.0967	0.1894	0.4861	0.0660	0.2536	0.5753
LambdaMart	Listwise	0.1115	0.2123	0.4893	**0.0858**	0.2941	0.5754

Furthermore, as shown in Table 4 and Table 5, the exploitation of functionality paths in the recommendation process resulted in an increase of the recommendation accuracy. By extracting similarities between rules in terms of shared functionality *RecRules* is therefore able to uncover useful hidden connections between rules that cannot be identified in contemporary EUD tools. Ideally, this could introduce numerous advantages: the algorithm could overcome technological constraints, and suggest IF-THEN rules even for rarely used devices and web applications, on the basis of the actual user's needs. To investigate such an hypothesis, we analyzed the recommendations computed by the 2 *RecRules* configurations with diversity (ILS), coverage (COV), and serendipity (UNSER) metrics. The goal was to go beyond accuracy, thus identifying other advantages of recommending by functionality. Table 6 reports the results for the ILS, COV, and UNSER metrics for $RecRules_{ct}$ and $RecRules_{ctf}$ using Random Forest.

Table 6. Diversity (ILS metric), coverage (COV metric), and serendipity (UNSER metric) results for $RecRules_{ct}$ and $RecRules_{ctf}$ using Random Forest. Results are the average of 5 equals experiments.

	ILS@10	COV@10	UNSER@10
RecRules$_{ct}$	0.1088	**13.94%**	0.6774
RecRules$_{ctf}$	**0.0439**	13.57%	**0.6243**

While the 2 versions of the algorithm computed recommendations that resulted in similar coverage and serendipity, recommendations computed by $RecRules_{ctf}$ were less similar in terms of involved technologies with respect to the suggestions proposed by $RecRules_{ct}$ ($ILS = 0.0439$ *vs.* $ILS = 0.1088$, respectively). This confirms that, by using functionality-based features, *RecRules* encourages the recommendation of IF-THEN rules for controlling new devices and web applications, without affecting accuracy.

5.3 Evaluation II: Comparison With Other Algorithms

Beside investigating the effectiveness of the functionality-based features, we compared *RecRules* in its best setting, i.e., $RecRules_{ctf}$ with Random Forest, with state-of-the-art collaborative filtering, ranking-oriented and semantic recommendation algorithms, namely:

- **Item-KNN.** The Item-KNN algorithm used for the evaluation is a baseline item-based K-Nearest Neighbours (KNN) algorithm [20]. KNN is a non-parametric, lazy learning method that uses a database in which the data points, i.e., the items, are separated into several clusters to make inference for new samples. Item-KNN does not make any assumptions on the underlying data distribution but it relies on item feature similarity.
- **User-KNN.** The User-KNN algorithm used for the evaluation is a baseline user-based K-Nearest Neighbours algorithm [20]. Differently from Item-KNN, User-KNN recommends items by analyzing similar users: it firstly finds the K-nearest neighbors to a specific user *a*, and it then predicts the rating that *a* will give to all items the k neighbors have consumed but *a* has not.
- **Soft Margin Ranking MF.** The Soft Margin Ranking Matrix Factorization is a matrix factorization model for item ranking which uses ordinal regression score as loss function [56]. Matrix factorization methods represent the state-of-the-art for rating prediction tasks.
- **BPR-MF.** The BPR-MF algorithm [19] is a hybrid extension of the Bayesian personalized ranking (BPR) [48] that learns a linear mapping on the user/item features from the factorization matrix. This extension of BPR is able to compute useful recommendations in cold-start scenarios.
- **BPR-SLIM.** The BPR-SLIM algorithm is an extension of the SLIM algorithm [43] that uses the BPR criterion. SLIM, in particular, uses a Sparse Linear method for learning a sparse aggregation coefficient matrix.
- **WRMF.** The WRMF algorithm [28] is a weighted matrix factorization method that interprets the number of times an item is observed by a user as a measure for the user preference, and uses regularization to prevent overfitting.
- **Least Square SLIM.** The Least Square SLIM algorithm is a variant of the SLIM algorithm in which the the model is learned using a coordinate descent algorithm with soft thresholding [18].
- **Item Attribute KNN.** The Item Attribute KNN algorithm used for the evaluation is an attribute-based K-Nearest Neighbours approach [20]. Besides items, users, and ratings, the algorithm can access other types of side information in the form of item attributes. In our evaluation, we provided the algorithm with the semantic information used in *RecRules*, i.e., OWL classes and super-classes of each trigger and action according to the EUPont ontology.
- **BPR-Linear.** BPR-Linear [19] is a hybrid matrix factorization method able to work with sparse datasets. As for the Item Attribute KNN algorithm, BPR-Linear is able to manage side information. Also in this case, we enriched the algorithm with the semantic information provided by EUPont.
- **Entity Graph-Embedding.** The Entity Graph-Embedding is a hybrid *semantic* recommender system proposed by Oramas et al. [46]. The algorithm exploits a knowledge graph and two different embedding approaches to encode knowledge graph information into a linear feature representation. In the Entity Graph-Embedding, in particular, features are calculated by analyzing the neighborhood of each entity composing the knowledge graph.

To compute the recommendations with Item-KNN, User-KNN, Soft Margin Ranking MF, BPR-MF, BPR-SLIM, WRMF, Least Square SLIM, Item Attribute KNN, and BPR-Linear we used *MyMediaLite* [20], a publicly available software library for recommender systems. For implementing the Entity-Based Graph-Embedding algorithm, instead, we used the *lodreclib* library [45], by building the same knowledge graph built in *RecRules*, i.e., with the semantic information of EUPont.

Table 7. Comparison of *RecRules* with other state-of-the-arts algorithms in terms of precision, recall, and normalized discounted cumulative gain on top-N recommendations (with N=5 and 10). Results are the average of 5 equals experiments.

	prec@5	rec@5	nDCG@5	prec@10	rec@10	nDCG@10
RecRules$_{ctf}$	**0.1211**	**0.2177**	**0.5054**	**0.0813**	**0.3019**	**0.6452**
Item-KNN	0.0847	0.1807	0.1939	0.0514	0.2383	0.2095
User-KNN	0.0961	0.2103	0.2410	0.0520	0.2277	0.2419
Soft Margin Ranking MF	0.0760	0.1716	0.1942	0.0452	0.2019	0.1905
BPR-MF	0.1085	0.1898	0.2082	0.0664	0.2148	0.2131
BPR-SLIM	0.1110	0.1976	0.2224	0.0616	0.2200	0.2216
WRMF	0.1155	0.2045	0.2228	0.0618	0.2217	0.2223
Least Square SLIM	0.1105	0.1970	0.2196	0.0604	0.2158	0.2229
Item Attribute KNN	0.0273	0.0845	0.2398	0.0207	0.1302	0.2357
BPR-Linear	0.0504	0.1708	0.2957	0.0356	0.2383	0.2890
Entity Graph-Embedding	0.0975	0.1918	0.4728	0.0656	0.2467	0.5625

Table 7 reports the results of the comparison evaluation in terms of accuracy metrics. Looking at the results, *RecRules* outperformed baseline collaborative filtering methods (i.e., Item-KNN and User-KNN) and Matrix Factorization approaches (i.e., Soft Margin Ranking MF, BPR-MF, and WRMF) in terms of precision, recall, and nDCG. Furthermore, *RecRules* also outperformed other hybrid learning to rank methods such as BPR-SLIM and Least Square SLIM. This suggests that using semantic information improved the recommendation accuracy. Moreover, by comparing *RecRules* with Item Attribute KNN and BPR-Linear, i.e., the 2 algorithms that used semantic information as item attributes, we can conclude that, in addition to the semantic information, the usage of an underlying graph provided clear advantages in terms of recommendation accuracy. Indeed, the usage of semantic information in a graph based setting, along with the extraction of different path-based features, resulted in an increase of precision, recall, and nDCG.

The benefits of the path-based features used in *RecRules*, in particular, can be glimpsed by looking at the results of the Entity Graph-Embedding approach. The Entity Graph-Embedding outperformed all the evaluated state of the art algorithms in terms of recommendation accuracy: it provided results in line with those obtained with *RecRules*, especially for what concern the nDCG metric. This further confirms the benefits of using a graph based model in our domain. Moreover, precision, recall, and nDCG were slightly higher with *RecRules*: we can reasonably conclude that capturing connections between IF-THEN rules in terms of shared functionality by means of functionality paths , i.e., the most important feature of *RecRules*, is a promising approach.

To further investigate the potential of *RecRules*, we repeated our analysis of diversity, coverage, and serendipity by looking at the recommendations of the previous state-of-the-art approaches.Table 8, in particular, reports the results of the ILS, COV, and UNSER metrics for all the evaluated algorithms.

Only the recommendations computed with the Entity Graph-Embedding approach covered a higher number of rules with respect to *RecRules* ($COV = 18.72\%$ *vs.* $COV = 13.57\%$, respectively), but its recommendations were in general less surprising than the rules suggested by *RecRules* ($UNSER = 0.9812$ *vs.* $UNSER = 0.6243$, respectively). Looking at the table, also the recommendations computed by the other 2 algorithms that used semantic information, i.e., Item Attribute KNN and BPR-Linear, were in general less surprising than the rules suggested by *RecRules*: this may suggest that using different semantic path-based features, based on technology, collaborative, and functionality

Table 8. Comparison of *RecRules* with other state-of-the-arts algorithms in terms of diversity (ILS metric), coverage (COV metric), and serendipity (UNSER metric) on top-N recommendations (with N=10). Results are the average of 5 equals experiments.

	ILS@10	COV@10	UNSER@10
RecRules$_{ctf}$	0.0439	13.57%	0.6243
Item-KNN	**0.0362**	4.53%	0.0167
User-KNN	0.0476	3.14%	**0.0129**
Soft Margin Ranking MF	0.0419	3.64%	0.0208
BPR-MF	0.0571	3.66%	0.5083
BPR-SLIM	0.0685	5.18%	0.5360
WRMF	0.0857	4.28%	0.5253
Least Square SLIM	0.0476	3.46%	0.4851
Item Attribute KNN	0.1125	12.01%	1.1616
BPR-Linear	0.1328	11.90%	1.2695
Entity Graph-Embedding	0.0471	**18.72%**	0.9812

information, resulted in a higher serendipity. Furthermore, even if the *RecRules* suggestions were in general less surprising than the rules suggested by the other collaborative filtering and ranking-oriented methods, i.e., Item-KNN, User-KNN, Soft Margin Ranking MF, BPR-MF, BPR-SLIM, WRMF, and Least Square SLIM, recommendations computed by *RecRules* were less similar in terms of involved technologies, brands, and manufacturers with respect to the majority of the other evaluated algorithms: only for the Item-KNN and Soft Margin Ranking MF the ILS metric was slightly lower.

6 DISCUSSION

We evaluated *RecRules* through different experiments by leveraging a dataset of trigger-action rules extracted from IFTTT [55], one of the most popular EUD platforms. Results show that *RecRules* outperformed several state-of-the-art recommendation algorithms, demonstrating that the usage of semantic information in a graph based setting, along with the extraction of different path-based features, resulted in an increase of different metrics.

In terms of accuracy metrics, i.e., precision, recall, nDCG, *RecRules* outperformed all the other evaluated approaches, ranging from baseline collaborative filtering methods, e.g., Item-KNN and User-KNN, to other semantic-based approaches that used the same underlying semantic information, e.g., Entity Graph-Embedding. Furthermore, the usage of different semantic path-based features increased the coverage and the diversity of the computed recommendations, by allowing *RecRules* to remain competitive (and in some cases better) in terms of serendipity. Moreover, functionality paths increased the recommendation accuracy of the explored learning to rank approaches: this suggest that the main characteristic of RecRules, i.e., *recommending by functionality*, is effective in suggesting trigger-action rules, and could help end users discover new rules based on their final purpose, rather than the involved devices and web applications.

Our approach is therefore able to uncover useful hidden connections between rules that cannot be identified in contemporary EUD tools and through conventional and established recommendation systems based on popularity or involved technologies. While a conventional recommender system would probably suggest, regardless of the user goal, rules that involve the same technologies, *RecRules* is able to establish, in high-level terms, what the user is trying to achieve with the rules she has already defined. Such a feature can be glimpsed by qualitatively analyzing the recommendations computed by *RecRules*. For a specific user (User 254), for example, we found

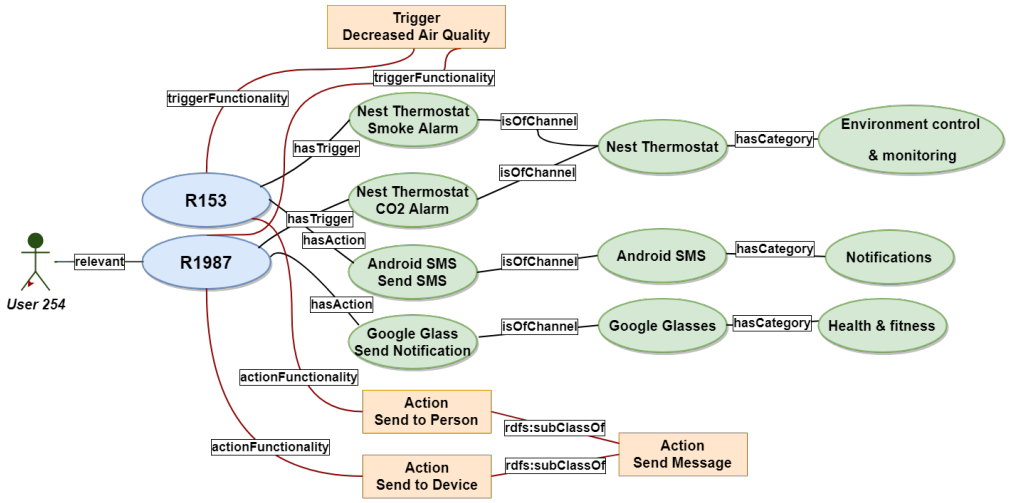


Fig. 10. A partial view of the knowledge graph from which *RecRules* recommended R153 to User 254. R153 is connected to R1987, i.e., a rule that is in the user's training set, by means of 2 functionality paths that relate both the triggers and the actions of the 2 rules.

the following recommended rule: *"if my Nest detects a smoke alarm, then send me an Android SMS (R153)"*. By analyzing the training set for that user, we found that it contained *"if my Nest detects a carbon monoxide alarm, then send me a notification on my Google Glasses (R1987)"*, that is conceptually equivalent to R153 in terms of final functionality, i.e., "let me know if something is wrong in my home". Figure 10 shows a partial view of the knowledge graph built for User 254. Here, R153 is connected to R1987, i.e., a rule that is in the user's training set, by means of 2 functionality paths that relate both triggers and actions of the 2 rules. The 2 triggers, in fact, share a common OWL class in the EUPont model, i.e., `Decreased Air Quality`. Furthermore, also the two actions *"send me a notification on my Google Glasses"* and *"send me an Android SMS"* share an EUPont class, i.e., `Send Message`.

In another case, the recommended rule *"if the door connected to my Abode system opens, then set the temperature to x on the Netatmo thermostat"* contained two technologies, i.e., `Abode` and `Netatmo`, that the analyzed user had never used. However, the same user created the rule *"if the Nest Cam recognizes me, then set the temperature to x on the Nest thermostat"*, equivalent to the recommended rule in terms of functionality. Also in this case, triggers and actions are similar from the point of view of EUPont.

6.1 Limitations

We are aware that the results presented in this paper could depend on numerous factors, including the high degree of sparsity of the evaluated dataset. Unfortunately, differently from other domains such as movies and songs, recommendations in the EUD are in their early stages. To the best of our knowledge, the dataset of Ur et al. [55] is the only publicly available collection of IF-THEN rules. A possible reason is that, excluding IFTTT, none of the most popular EUD tool publicly share collections of rules created by different users, making it *impossible* to crawl data from such

platforms. In our work, in particular, we unsuccessfully tried to crawl data from Zapier, Microsoft Flow[8], Tasker[9], and Resonance AI[10].

Even if further investigation is needed, however, we may conclude that *RecRules* opens the way for a new class of recommender systems in EUD based on the actual end-users' needs.

7 CONCLUSIONS AND FURTHER DEVELOPMENTS

In this paper, we tackled the problem of recommending IF-THEN rules to end users by presenting *RecRules*, a hybrid and semantic recommendation algorithm. Through a mixed content and collaborative approach, *RecRules* exploits different path-based features and learning to rank techniques, and it is able to interact with OWL ontologies to compute *top-N* recommendations. For the semantic part, *RecRules* exploits EUPont, an ontological high level representation of trigger-action programming. With such a representation, *RecRules* not only takes into account technologies already used by users, but is able to recommend trigger-action rules on the basis of their final functionality, i.e., the behavior that they aim to define. By exploiting a dataset of trigger-action rules created and shared by real users on IFTTT, we demonstrated the effectiveness of our approach by evaluating three different learning to rank algorithms, and by investigating to what extent the different path-based features affected the computed recommendations. Furthermore, we showed that *RecRules* outperforms state-of-the-art ranking-oriented and semantic recommendation algorithms, by demonstrating the potential of *recommending by functionality*.

This work opens the way for a new class of recommender systems in EUD based on the actual end-users' needs, rather than the involved technologies, brands, or manufactures. To overcome the lack of available datasets in this domain, we are planning to integrate *RecRules* in a real platform for composing trigger-action rules, with the aim of further assessing the benefits of our approach in helping end users personalize their devices and web applications.

REFERENCES

[1] Pierre A. Akiki, Arosha K. Bandara, and Yijun Yu. 2017. Visual Simple Transformations: Empowering End-Users to Wire Internet of Things Objects. *ACM Transactions on Computer-Human Interaction* 24, 2, Article 10 (April 2017), 43 pages. https://doi.org/10.1145/3057857

[2] Sarabjot Singh Anand, Patricia Kearney, and Mary Shapcott. 2007. Generating Semantically Enriched User Profiles for Web Personalization. *ACM Transactions on Internet Technology* 7, 4, Article 22 (Oct. 2007). https://doi.org/10.1145/1278366.1278371

[3] C. Bizer and T. Heathand T. Berners-Lee. 2009. Linked data - the story so far. *International Journal on Semantic Web and Information System* 5, 3 (2009), 1–22.

[4] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (Oct. 2001), 5–32. https://doi.org/10.1023/A:1010933404324

[5] Julia Brich, Marcel Walch, Michael Rietzler, Michael Weber, and Florian Schaub. 2017. Exploring End User Programming Needs in Home Automation. *ACM Transaction on Computer-Human Interaction* 24, 2, Article 11 (April 2017), 35 pages. https://doi.org/10.1145/3057858

[6] Iván Cantador, Alejandro Bellogín, and Pablo Castells. 2008. A Multilayer Ontology-based Hybrid Recommendation Model. *AI Communications* 21, 2-3 (April 2008), 203–210.

[7] Vint Cerf and Max Senges. 2016. Taking the Internet to the Next Physical Level. *IEEE Computer* 49, 2 (Feb 2016), 80–86. https://doi.org/10.1109/MC.2016.51

[8] F. Corno, L. De Russis, and A. Monge Roffarello. 2017. A Semantic Web Approach to Simplifying Trigger-Action Programming in the IoT. *Computer* 50, 11 (November 2017), 18–24. https://doi.org/10.1109/MC.2017.4041355

[9] Fulvio Corno, Luigi De Russis, and Alberto Monge Roffarello. 2019. A high-level semantic approach to End-User Development in the Internet of Things. *International Journal of Human-Computer Studies* 125 (2019), 41 – 54. https://doi.org/10.1016/j.ijhcs.2018.12.008

[8] https://flow.microsoft.com/, last visited on February 12, 2019

[9] https://tasker.joaoapps.com/, last visited on February 12, 2019

[10] http://www.resonance-ai.com/, last visited on February 12, 2019

[10] Allen Cypher. 1991. EAGER: Programming Repetitive Tasks by Example. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '91)*. ACM, New York, NY, USA, 33–39. https://doi.org/10.1145/108844.108850

[11] Jose Danado and Fabio Paternò. 2014. Puzzle: A mobile application development environment using a jigsaw metaphor. *Journal of Visual Languages & Computing* 25, 4 (2014), 297–315. https://doi.org/10.1016/j.jvlc.2014.03.005

[12] G. Desolda, C. Ardito, and M. Matera. 2017. Empowering End Users to Customize Their Smart Environments: Model, Composition Paradigms, and Domain-Specific Tools. *ACM Transaction on Computer-Human Interaction (TOCHI)* 24, 2, Article 12 (April 2017), 52 pages. https://doi.org/10.1145/3057859

[13] Anind K. Dey, Timothy Sohn, Sara Streng, and Justin Kodama. 2006. iCAP: Interactive Prototyping of Context-aware Applications. In *Proceedings of the 4th International Conference on Pervasive Computing (PERVASIVE'06)*. Springer-Verlag, Berlin, Heidelberg, 254–271. https://doi.org/10.1007/11748625_16

[14] A. R. D'Souza, D. Yang, and C. V. Lopes. 2016. Collective Intelligence for Smarter API Recommendations in Python. In *2016 IEEE 16th International Working Conference on Source Code Analysis and Manipulation (SCAM)*. 51–60. https://doi.org/10.1109/SCAM.2016.22

[15] Ekwa Duala-Ekoko and Martin P. Robillard. 2011. *Using Structure-Based Recommendations to Facilitate Discoverability in APIs*. Springer Berlin Heidelberg, Berlin, Heidelberg, 79–104. https://doi.org/10.1007/978-3-642-22655-7_5

[16] Ignacio Fernández-Tobías, Iván Cantador, Marius Kaminskas, and Francesco Ricci. 2011. A Generic Semantic-based Framework for Cross-domain Recommendation. In *Proceedings of the 2Nd International Workshop on Information Heterogeneity and Fusion in Recommender Systems (HetRec '11)*. ACM, New York, NY, USA, 25–32. https://doi.org/10.1145/2039320.2039324

[17] Yoav Freund, Raj Iyer, Robert E. Schapire, and Yoram Singer. 2003. An Efficient Boosting Algorithm for Combining Preferences. *Journal of Machine Learning Research* 4 (Dec. 2003), 933–969.

[18] J. Friedman, T. Hastie, and R. Tibshirani. 2010. Regularization Paths for Generalized Linear Models via Coordinate Descent. *Journal of Statistical Software* 33, 1 (2010), 1–22. http://www.jstatsoft.org/v33/i01/

[19] Z. Gantner, L. Drumond, C. Freudenthaler, S. Rendle, and L. Schmidt-Thieme. 2010. Learning Attribute-to-Feature Mappings for Cold-Start Recommendations. In *Proceedings of the 2010 IEEE International Conference on Data Mining (ICDM '10)*. IEEE Computer Society, Washington, DC, USA, 176–185. https://doi.org/10.1109/ICDM.2010.129

[20] Zeno Gantner, Steffen Rendle, Christoph Freudenthaler, and Lars Schmidt-Thieme. 2011. MyMediaLite: A Free Recommender System Library. In *Proceedings of the Fifth ACM Conference on Recommender Systems (RecSys '11)*. ACM, New York, NY, USA, 305–308. https://doi.org/10.1145/2043932.2043989

[21] Andrés Garcí-a silva, Oscar Corcho, Harith Alani, and Asunción Gómez-pérez. 2012. Review: Review of the State of the Art: Discovering and Associating Semantics to Tags in Folksonomies. *The Knowledge Engineering Review* 27, 1 (Feb. 2012), 57–85. https://doi.org/10.1017/S026988891100018X

[22] G. Ghiani, M. Manca, F. Paternò, and C. Santoro. 2017. Personalization of Context-Dependent Applications Through Trigger-Action Rules. *ACM Transactions on Computer-Human Interaction (TOCHI)* 24, 2, Article 14 (April 2017), 33 pages. https://doi.org/10.1145/3057861

[23] Birte Glimm, Ian Horrocks, Boris Motik, Giorgos Stoilos, and Zhe Wang. 2014. HermiT: An OWL 2 Reasoner. *Journal of Automated Reasoning* 53, 3 (Oct. 2014), 245–269. https://doi.org/10.1007/s10817-014-9305-1

[24] Will Haines, Melinda Gervasio, Aaron Spaulding, and Bart Peintner. 2010. Recommendations for End-User Development. In *Proceedings of the ACM RecSys 2010 Workshop on User-Centric Evaluation of Recommender Systems and Their Interfaces (UCERSTI)*.

[25] Mostafa Hamza and Robert J. Walker. 2015. Recommending Features and Feature Relationships from Requirements Documents for Software Product Lines. In *Proceedings of the Fourth International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE '15)*. IEEE Press, Piscataway, NJ, USA, 25–31.

[26] Benjamin Heitmann and Conor Hayes. 2010. Using linked data to build open, collaborative recommender systems. *Artificial Intelligence* (2010), 76–81.

[27] Matthew Horridge and Sean Bechhofer. 2011. The OWL API: A Java API for OWL Ontologies. *Semantic Web* 2, 1 (Jan. 2011), 11–21.

[28] Yifan Hu, Yehuda Koren, and Chris Volinsky. 2008. Collaborative Filtering for Implicit Feedback Datasets. In *Proceedings of the 2008 Eighth IEEE International Conference on Data Mining (ICDM '08)*. IEEE Computer Society, Washington, DC, USA, 263–272. https://doi.org/10.1109/ICDM.2008.22

[29] Justin Huang and Maya Cakmak. 2015. Supporting Mental Model Accuracy in Trigger-action Programming. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing (UbiComp '15)*. ACM, New York, NY, USA, 215–225. https://doi.org/10.1145/2750858.2805830

[30] Ting-Hao K. Huang, A. Azaria, and J. P. Bigham. 2016. InstructableCrowd: Creating IF-THEN Rules via Conversations with the Crowd. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems (CHI EA '16)*. ACM, New York, NY, USA, 1555–1562. https://doi.org/10.1145/2851581.2892502

[31] Houda Khrouf and Raphaël Troncy. 2013. Hybrid Event Recommendation Using Linked Data and User Diversity. In *Proceedings of the 7th ACM Conference on Recommender Systems (RecSys '13)*. ACM, New York, NY, USA, 185–192. https://doi.org/10.1145/2507157.2507171

[32] Jacob Krüger. 2018. When to Extract Features: Towards a Recommender System. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings (ICSE '18)*. ACM, New York, NY, USA, 518–520. https://doi.org/10.1145/3183440.3190328

[33] Ni Lao and William W. Cohen. 2010. Relational Retrieval Using a Combination of Path-constrained Random Walks. *Mach. Learn.* 81, 1 (Oct. 2010), 53–67. https://doi.org/10.1007/s10994-010-5205-8

[34] Lukas Lerche and Dietmar Jannach. 2014. Using Graded Implicit Feedback for Bayesian Personalized Ranking. In *Proceedings of the 8th ACM Conference on Recommender Systems (RecSys '14)*. ACM, New York, NY, USA, 353–356. https://doi.org/10.1145/2645710.2645759

[35] Henry Lieberman, Fabio Paternò, Markus Klann, and Volker Wulf. 2006. *End User Development*. Springer Netherlands, Dordrecht, Netherlands, Chapter End-User Development: An Emerging Paradigm, 1–8. https://doi.org/10.1007/1-4020-5386-X_1

[36] Tie-Yan Liu. 2009. Learning to Rank for Information Retrieval. *Foundations and Trends In Information Retrieval* 3, 3 (March 2009), 225–331. https://doi.org/10.1561/1500000016

[37] Y. Malheiros, A. Moraes, C. Trindade, and S. Meira. 2012. A Source Code Recommender System to Support Newcomers. In *2012 IEEE 36th Annual Computer Software and Applications Conference*. 19–24. https://doi.org/10.1109/COMPSAC.2012.11

[38] Toshiyuki Masui and Ken Nakayama. 1994. Repeat and Predict: Two Keys to Efficient Text Editing. In *Conference Companion on Human Factors in Computing Systems (CHI '94)*. ACM, New York, NY, USA, 31–32. https://doi.org/10.1145/259963.259999

[39] Sean M. McNee, John Riedl, and Joseph A. Konstan. 2006. Being Accurate is Not Enough: How Accuracy Metrics Have Hurt Recommender Systems. In *CHI '06 Extended Abstracts on Human Factors in Computing Systems (CHI EA '06)*. ACM, New York, NY, USA, 1097–1101. https://doi.org/10.1145/1125451.1125659

[40] Kim Mens and Angela Lozano. 2014. *Source Code-Based Recommendation Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 93–130. https://doi.org/10.1007/978-3-642-45135-5_5

[41] Bamshad Mobasher, Xin Jin, and Yanzan Zhou. 2004. *Web Mining: From Web to Semantic Web*. Springer Berlin Heidelberg, Berlin, Heidelberg, Chapter Semantically Enhanced Collaborative Filtering on the Web, 57–76. https://doi.org/10.1007/978-3-540-30123-3_4

[42] Anh Tuan Nguyen, Michael Hilton, Mihai Codoban, Hoan Anh Nguyen, Lily Mast, Eli Rademacher, Tien N. Nguyen, and Danny Dig. 2016. API Code Recommendation Using Statistical Learning from Fine-grained Changes. In *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE 2016)*. ACM, New York, NY, USA, 511–522. https://doi.org/10.1145/2950290.2950333

[43] X. Ning and G. Karypis. 2011. SLIM: Sparse Linear Methods for Top-N Recommender Systems. In *2011 IEEE 11th International Conference on Data Mining*. 497–506. https://doi.org/10.1109/ICDM.2011.134

[44] Xia Ning and George Karypis. 2012. Sparse Linear Methods with Side Information for Top-n Recommendations. In *Proceedings of the Sixth ACM Conference on Recommender Systems (RecSys '12)*. ACM, New York, NY, USA, 155–162. https://doi.org/10.1145/2365952.2365983

[45] Tommaso Di Noia, Vito Claudio Ostuni, Paolo Tomeo, and Eugenio Di Sciascio. 2016. SPrank: Semantic Path-Based Ranking for Top-N Recommendations Using Linked Open Data. *ACM Transactions on Intelligent Systems and Technology* 8, 1, Article 9 (Sept. 2016), 34 pages. https://doi.org/10.1145/2899005

[46] Sergio Oramas, Vito Claudio Ostuni, Tommaso Di Noia, Xavier Serra, and Eugenio Di Sciascio. 2016. Sound and Music Recommendation with Knowledge Graphs. *ACM Trans. Intell. Syst. Technol.* 8, 2, Article 21 (Oct. 2016), 21 pages. https://doi.org/10.1145/2926718

[47] Vito Claudio Ostuni, Tommaso Di Noia, Eugenio Di Sciascio, and Roberto Mirizzi. 2013. Top-N Recommendations from Implicit Feedback Leveraging Linked Open Data. In *Proceedings of the 7th ACM Conference on Recommender Systems (RecSys '13)*. ACM, New York, NY, USA, 85–92. https://doi.org/10.1145/2507157.2507172

[48] Steffen Rendle, Christoph Freudenthaler, Zeno Gantner, and Lars Schmidt-Thieme. 2009. BPR: Bayesian Personalized Ranking from Implicit Feedback. In *Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence (UAI '09)*. AUAI Press, Arlington, Virginia, United States, 452–461.

[49] Giovanni Semeraro, Pasquale Lops, Pierpaolo Basile, and Marco de Gemmis. 2009. Knowledge Infusion into Content-based Recommender Systems. In *Proceedings of the Third ACM Conference on Recommender Systems (RecSys '09)*. ACM, New York, NY, USA, 301–304. https://doi.org/10.1145/1639714.1639773

[50] W. Shen, J. Wang, and J. Han. 2015. Entity Linking with a Knowledge Base: Issues, Techniques, and Solutions. *IEEE Transactions on Knowledge and Data Engineering* 27, 2 (Feb 2015), 443–460. https://doi.org/10.1109/TKDE.2014.2327028

[51] Yue Shi, Alexandros Karatzoglou, Linas Baltrunas, Martha Larson, Alan Hanjalic, and Nuria Oliver. 2012. TFMAP: Optimizing MAP for Top-n Context-aware Recommendation. In *Proceedings of the 35th International ACM SIGIR Conference on Research and Development in Information Retrieval (SIGIR '12)*. ACM, New York, NY, USA, 155–164. https://doi.org/10.1145/2348283.2348308

[52] Harald Steck. 2013. Evaluation of Recommendations: Rating-prediction and Ranking. In *Proceedings of the 7th ACM Conference on Recommender Systems (RecSys '13)*. ACM, New York, NY, USA, 213–220. https://doi.org/10.1145/2507157.2507160

[53] Zhu Sun, Jie Yang, Jie Zhang, Alessandro Bozzon, Long-Kai Huang, and Chi Xu. 2018. Recurrent Knowledge Graph Embedding for Effective Recommendation. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys '18)*. ACM, New York, NY, USA, 297–305. https://doi.org/10.1145/3240323.3240361

[54] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L. Littman. 2014. Practical Trigger-action Programming in the Smart Home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '14)*. ACM, New York, NY, USA, 803–812. https://doi.org/10.1145/2556288.2557420

[55] B. Ur, M. Pak Yong Ho, S. Brawner, J. Lee, S. Mennicken, N. Picard, D. Schulze, and M. L. Littman. 2016. Trigger-Action Programming in the Wild: An Analysis of 200,000 IFTTT Recipes. In *Proceedings of the 34rd Annual ACM Conference on Human Factors in Computing Systems (CHI '16)*. ACM, New York, NY, USA, 3227–3231. https://doi.org/10.1145/2858036.2858556

[56] Markus Weimer, Alexandros Karatzoglou, and Alex Smola. 2008. Improving Maximum Margin Matrix Factorization. *Mach. Learn.* 72, 3 (Sept. 2008), 263–276. https://doi.org/10.1007/s10994-008-5073-7

[57] Qiang Wu, Christopher J. Burges, Krysta M. Svore, and Jianfeng Gao. 2010. Adapting Boosting for Information Retrieval Measures. *Information Retrieval Journal* 13, 3 (June 2010), 254–270. https://doi.org/10.1007/s10791-009-9112-1

[58] Yao Wu, Christopher DuBois, Alice X. Zheng, and Martin Ester. 2016. Collaborative Denoising Auto-Encoders for Top-N Recommender Systems. In *Proceedings of the Ninth ACM International Conference on Web Search and Data Mining (WSDM '16)*. ACM, New York, NY, USA, 153–162. https://doi.org/10.1145/2835776.2835837

[59] Lina Yao, Quan Z. Sheng, Anne H.H. Ngu, Helen Ashman, and Xue Li. 2014. Exploring Recommendations in Internet of Things. In *Proceedings of the 37th International ACM SIGIR Conference on Research & Development in Information Retrieval (SIGIR '14)*. ACM, New York, NY, USA, 855–858. https://doi.org/10.1145/2600428.2609458

[60] Y. Ye and G. Fischer. 2002. Supporting reuse by delivering task-relevant and personalized information. In *Proceedings of the 24th International Conference on Software Engineering. ICSE 2002.* 513–523. https://doi.org/10.1109/ICSE.2002.1007995

[61] Xiao Yu, Xiang Ren, Yizhou Sun, Quanquan Gu, Bradley Sturt, Urvashi Khandelwal, Brandon Norick, and Jiawei Han. 2014. Personalized Entity Recommendation: A Heterogeneous Information Network Approach. In *Proceedings of the 7th ACM International Conference on Web Search and Data Mining (WSDM '14)*. ACM, New York, NY, USA, 283–292. https://doi.org/10.1145/2556195.2556259

[62] Yuan Cao Zhang, Diarmuid Ó Séaghdha, Daniele Quercia, and Tamas Jambor. 2012. Auralist: Introducing Serendipity into Music Recommendation. In *Proceedings of the Fifth ACM International Conference on Web Search and Data Mining (WSDM '12)*. ACM, New York, NY, USA, 13–22. https://doi.org/10.1145/2124295.2124300

[63] Cai-Nicolas Ziegler, Sean M. McNee, Joseph A. Konstan, and Georg Lausen. 2005. Improving Recommendation Lists Through Topic Diversification. In *Proceedings of the 14th International Conference on World Wide Web (WWW '05)*. ACM, New York, NY, USA, 22–32. https://doi.org/10.1145/1060745.1060754

Received February 2007; revised March 2009; accepted June 2009