Guidelines for the Analysis and Design of Argumentation-based Recommendation Systems

Mario Leiva[a], Maximiliano C.D. Budán[a,b], Gerardo I. Simari[a,*]

[a]*Departamento de Cs. e Ing. de la Computación, Universidad Nacional del Sur (UNS) & Instituto de Cs. e Ing. de la Computación (UNS—CONICET), San Andres 800, (8000) Bahía Blanca, Argentina*
[b]*Departamento de Matemática, Universidad Nacional de Santiago del Estero, Belgrano(s) 1912, (4200) Capital, Sgo. del Estero, Argentina*

Abstract

Recommender systems study the characteristics of its users and, applying different kinds of processing to the available data, find a subset of items that may be of interest to a given user in a specific situation. Argumentation-based tools offer the possibility of analyzing complex and dynamic domains by generating and analyzing arguments for and against recommending a specific item based on the users' preferences. This approach allows to analyze the qualitative and quantitative characteristics of the recommended items, and to provide explanations to increase transparency. In this work, we develop a set of software engineering guidelines for the analysis and design of recommender systems leveraging this approach.

Keywords: Recommendation Systems; Defeasible Argumentation; Methodological Guidelines; Decision Support Systems.

1. Introduction

Existing recommender systems ("RS", for short) cannot formally address the defeasible nature of user preferences in complex environments [1]. Decisions about preferences are driven mainly by heuristics, which are typically based on classifying the choices of previous users or on gathering information from other users with similar interests. In addition, as discussed in [2], most quantitative approaches do not have a clear underlying model, making it difficult to provide users with a simple explanation of how the system arrived at its recommendations. Another problem is that modeling users' preference criteria is not an easy task, since it generally requires dealing with incomplete and potentially inconsistent knowledge.

Tools developed in the area of argumentation-based reasoning offer the possibility of analyzing complex and dynamic domains by studying the arguments for and against recommending a specific item based on user preferences. Specifically, *defeasible argumentation* leverages models that contain inconsistency, evaluating arguments that support contradictory conclusions and deciding which ones to keep. An argument supports a conclusion from a set of premises [3]; a conclusion `C` constitutes a piece of tentative information that an agent is willing to accept. If the agent then acquires new information, the conclusion `C`—along with the arguments that support it—could be invalidated. The validity of a conclusion `C` is guaranteed when there is an argument that provides justification for `C` that is undefeated. This process involves the construction of an argu-

*Corresponding author
Email addresses: `mario.leiva@cs.uns.edu.ar` (Mario Leiva), `mcdb@cs.uns.edu.ar` (Maximiliano C.D. Budán), `gis@cs.uns.edu.ar` (Gerardo I. Simari)

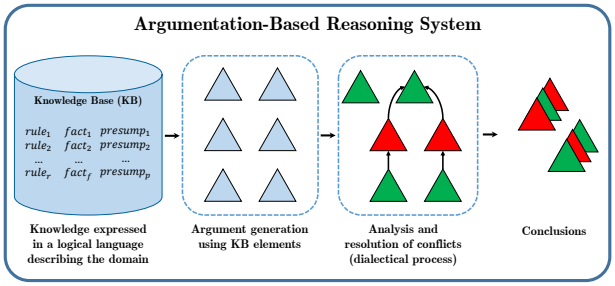

Figure 1: Outline of an argumentation-based reasoning system.

ment $\mathcal{A}$ for $\mathtt{C}$, and the analysis of counterarguments that are possible defeaters of $\mathcal{A}$; as these defeaters are arguments, it must be verified that they are not themselves defeated, and so on. This analysis has a valuable byproduct: the set of all arguments can be used to provide *explanations* for recommendations provided by the system, increasing its transparency.

There is a large body of work on frameworks to carry out this kind of reasoning; the most closely related to this work are those *based on rules*, which consider the structure of the arguments that model a discussion [4, 5, 6]. Figure 1 presents a brief outline of their basic elements. Such systems have a knowledge base (KB) that allows storing information expressed in a logical language. Inference rules allow to leverage certain information (antecedents) to derive new information (consequents). Other elements of the KB include facts or presumptions, representing evidence obtained from the environment; such evidence typically plays a central role in firing rules and thus building arguments, which are then evaluated via an exhaustive analysis to decide which are accepted and which conclusions can be guaranteed from the current knowledge. A key property of argumentation-based reasoning is non-monotonicity—the incorporation of new information can generate new arguments that contradict existing ones and, therefore, invalidate statements that were previously guaranteed.

In the domain of RS, the frameworks developed in defeasible argumentation offer the possibility of analyzing complex and dynamic situations by studying arguments for/against recommending an item based on user preferences, focusing on both qualitative and quantitative features. Though the process of obtaining recommendations in this manner is very different from traditional approaches, they share the same main idea: establish a similarity between items and users, and use that similarity to derive recommendations. The main difference is that traditional methods establish similarities through purely numerical analyses, while ABRS use a dialectical process similar to how human beings debate to establish similarity—in particular, we can prioritize different elements depending on users' preferences. Thus, based on the set of rules that define the behavior of the recommender system together with knowledge of the domain, the system will establish the set of arguments for recommending an item, carry out the dialectical process, and execute the corresponding actions.

State of the Art

There are several works that propose using argumentation to enhance recommendations. Early work includes [1], and [7, 2] present an application in the domain of film recommendation, stressing the importance of considering both qualitative and quantitative aspects; furthermore, explanations that support the recommendations are generated in natural language. Other recent efforts leveraging argumentation-based tools are [8, 9]. Finally, other relevant work involves applying data-driven approaches, as we will discuss below [10, 11]. None of these efforts focus on software development methodologies—this work thus aims to present a set of guidelines to support knowledge and software engineers in the analysis and design of argumentation-based recommender systems (ABRS), aiming to fill this gap in the current intelligent systems development literature.

2. Analysis and Design of Argumentation-based Recommender Systems

When executing the analysis and design of a RS with these characteristics, it is desirable to focus on four central aspects: (i) knowledge base design; (ii) recommendation mechanism; (iii) design and presentation of explanations; and (iv) design of user interactions. We now introduce a series of methodological guidelines defined around these aspects. Figure 2

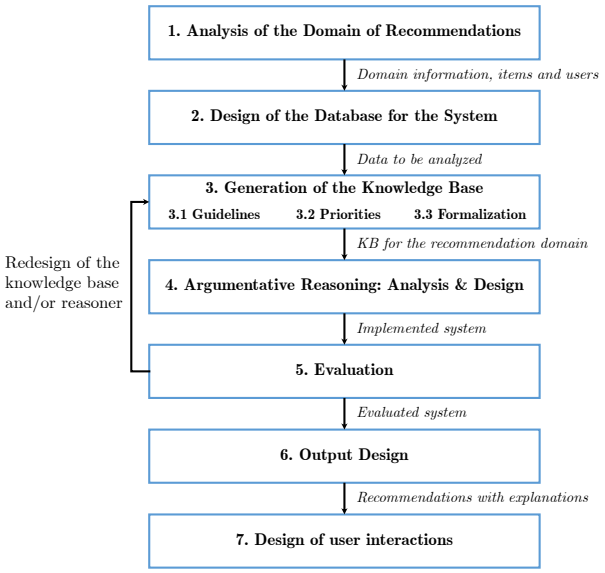

Figure 2: Schematic view of the stages for the analysis and design of an ABRS, including intermediate results.

illustrates the relationships that exist among these different tasks.

2.1. Stages 1–2: Domain Analysis and DB Design

The two main entities in RS are *items* and *users*—tt is essential to analyze the relationships between them, since they are central to user preferences. Depending on the domain, it is necessary to refine their description to obtain more information from the participating entities. Examples of relationships include: "number of times the user listened to a song and the rating given" and "number of times that a recommended article was shared or valuation was provided to it". The result of Stage 1 offers a detailed and clear description of the domain, and defines the attributes associated with the entities that are most relevant for the task.

The next step is to design the database—there are three basic options: (i) create and populate the database that feeds the system; (ii) reuse an existing dataset related to the domain; or (iii) extend an existing dataset or merge several datasets to enhance the available information. The latter two options allow to determine the recommendation mechanism without having to design and populate the underlying database, which can be a very complex and expensive process until enough relevant data is obtained.

Stages 1–2 can be mapped directly to those of traditional methods.

2.2. Stage 3: KB Generation

The KB is the structure where knowledge of the domain is formally represented. Its generation can be carried out in three steps:

- Analyze the domain and establish the criteria to be used in the generation of recommendations.
- Specify a preference criterion to apply in case the rules established in the previous step generate contradictory results.
- Specify the KB in a formal logical language.

During the first step, we create statements in natural language that express how items should (not) be recommended. Though here it may be necessary to appeal to domain experts to generate rules and priorities, it is also possible to leverage existing tools to process large volumes of data, such as data mining, machine learning, genetic algorithms, and information retrieval [10, 12]. For instance, with association rule mining it is possible to find which characteristics best describe similarity between items, and prioritize rules. In the second step, a preference criterion is established among the criteria to reflect the domain's characteristics and the users' preferences. Finally, one must specify the patterns in a formal logical language to be interpreted, analyzed, and manipulated by the reasoner. Traditional methods cannot be applied directly in this stage, since they seek to formalize metrics based on quantitative aspects to characterize similar elements, while in ABRS similar elements are characterized through rules. It is possible to adapt traditional approaches to generate a more general representation of such metrics so that they can be mapped into rules to feed dialectical processes.

2.3. Stage 4: Analysis & Design of the Reasoner

The reasoner is the system's main component—it interprets available knowledge, create a model based on that knowledge, analyzes the relationships between arguments, resolves conflicts between them, and ultimately issues recommendations.

The first component of an argumentative reasoner is the *Inference Engine*, which provides the ability to analyze domain knowledge and infer new knowledge to be used in the recommendation process. The literature highlights three alternatives to represent and formalize logic-based arguments: as a proof tree based on the premises [13], as a sequence of proofs (or derivations) [14], or as a pair of conclusion-premises implying that there is a proof for the conclusion from the premises in the underlying logic [3, 4].

The second component is the *Relationship Interpreter*. One of the essential definitions for any argumentative system is the definition of *conflict* (also known as *counter-argumentation* or *attack*) among arguments, which characterizes disagreement. This component is responsible for interpreting the relationships between arguments, creating a model, and establishing a preference order among arguments. Argumentative systems typically parameterize the comparison criterion between arguments, which is generally specified by the knowledge engineer in relation to the application domain.

The last component is the *Semantic Analyzer*, which determines acceptability of arguments by considering their interactions—given an argument, it considers its defeaters, the defeats of its defeaters, and so on. The definition of a mechanism for deciding acceptability of arguments determines precisely how inferences are obtained. In the literature, there are various proposals for this; in particular, either a declarative approach or a procedural approach can be followed. The former establishes conditions that a set of acceptable arguments must meet [15, 16], while the in the latter a specific algorithm is provided [4, 14].

2.4. Stage 5: Evaluation

The performance of the proposed system should now be evaluated regarding its capability to characterize users' preferences—i.e., determine if it is capable of predicting the items that its users would like, including additional considerations such as variety and surprise.

In [17], the authors present a study that emphasizes the evaluation of RS, defining three possible experimental setups: *offline experiments*, *user studies*, and *large-scale online experiments*. They also describe the most important properties that systems must satisfy, together with their satisfaction criteria. Typical properties are: *accuracy* of predictions, *coverage* of recommendations offered, *cold start* capability, *confidence*, *credibility*, *novelty and originality*, *diversity*, *usefulness*, *risk*, *robustness*, *privacy*, *adaptability*, *scalability*, and *performance*. The selection of experiments, and the subset of properties to emphasize in the evaluation, is part of the activities carried out by knowledge engineers based on their evaluation of the domain.

2.5. Stage 6: Design of Outputs

Once all the machinery for producing recommendations is in place, the next step is to design their presentation to users—this consists of designing the system interface, along with any justifications for these selections. In particular, the knowledge engineer must establish how the entities that support the recommendations will be analyzed and represented, to facilitate their understanding and generate a satisfactory explanation. According to [18], an explanation must be "understandable", allow "to improve knowledge", and be "satisfactory" in the sense of fulfilling the interlocutor's expectations. The generation of an understandable explanation refers to a justification or coherent explanation requested by the interlocutor. On the other hand, [19] defines an explanation as "a transfer of knowledge from one interlocutor to another within the context of a dialogue". Finally, according to [20], an explanation "has to be planned and then transmitted in an appropriate way"; i.e., it is an "object to be designed" and a "communicative act" to be achieved. The agent who acts as interlocutor only has partial knowledge of the subject in question, for which it requests an explanation in hopes that the agent generating it can fill their knowledge gaps.

In the domain of argumentation, structured approaches provide important advantages in the task of

translating the structure of arguments into natural language propositions. Another interesting aspect is the possibility they afford to visualize, through tree-based structures, the dialectical process generated to support a recommendation, which can be part of an extended explanation.

This stage is modular—it depends on the domain, since we consider that explaining and showing the process to generate a recommendation is very useful for certain domains (such as investments, medical diagnosis, or risk analysis), but may prove to be overwhelming and unhelpful in more mundane ones (such as multimedia).

2.6. Stage 7: Design of User Interactions

This stage is common to the design of user interactions any system—when the interface with the user is well designed, the user "slides through" the interaction smoothly and effortlessly. In RS, the GUI is focused on what the user sees when requesting a recommendation and how the system outputs (including possible explanations) are presented.

3. Case Study: Music Recommendations

Methodologies aiding in the successful engineering of software systems must be properly validated. In this section, we implement our proposal to analyze and design an ABRS in the music domain, comparing the results obtained with a baseline RS that resembles those designed under classical schemes. A music RS aims to suggest songs, videos, albums, or artists that appeal to its users—well-known examples include *Spotify, YouTube, Last.fm, Pandora, Genius*, among others.

For reasons of space, we only present a reduced version of the analysis carried out in each stage, focusing on showing the intermediate results towards obtaining the final system.

3.1. Case Study: Stages 1–2

The main attributes that describe a song are: *ID, Title, Author, Album, Genre*—for our purposes, these attributes suffice to build reasoning patterns to support issuing recommendations. In most RS, users are represented by their personal information: *Name, LastName, Sex, Age, Country.*

Relationships represent the interaction between users and items, which are also called *transactions* and can be encoded via tuples *User ID, Song ID, Score, #reproductions.* The latter is a secondary parameter to measure the degree to which the song was liked by the user.

Once the domain of the recommender system has been defined, the second stage of the process involves designing how available information is stored in a database.

3.2. Case Study: Stage 3

As described in Section 2.2, the generation of the KB will be carried out in three steps.

3.2.1. Step 1: Analyze the domain

The system will reason based on the following general criteria:

- If a user likes a song by a certain artist, they can also be expected to like another song by the same artist. For greater specificity, we can extend this to sharing the same artist and tags.
- If a user likes a song of a certain genre, they may be expected to like another song of the same genre.
- If a particular song is liked by many users in the system, a given user can also be expected to like it.
- Given two similar users, a song liked by one of them can be expected to be liked by the other.

In general, the dual of each of these criteria can also be applied; for instance, if a user does not like a song from a particular genre, they can be expected to not like other songs in that genre. Based on these general statements, criteria were defined to represent that knowledge and direct the recommendation process.

Next, we establish the events that must be considered to trigger reasoning chains. These events are:

$E1$: Two songs are by the same artist.

$E2$: A song and an artist have similar tags whenever they share a number of tags greater than or equal to half the tags assigned to the entity with least number of tags.

$E3$: Two songs have similar tags (genre); determined as above.

$E4$: A song is considered to be "good" whenever the ratio between the number of times it is played and number of listeners is greater than or equal to 6. Otherwise, the song is considered to be "bad".

Clearly, some of the parameters used here can be readjusted or modified as needed. For example, a weaker similarity notion can be used to specify closeness between two songs or artists, or the threshold for determining song quality can be adjusted.

Once these foundations are defined, the conditions that trigger the recommendations are derived as follows. Recommendations based on artist:

$R1$: A user may like a given song if there is another song by the same artist that was positively valued by them.

$R2$: A user may *not* like a given song if there is another song by the same artist and it was negatively valued by them.

Recommendations based on artist and tags:

$R3$: A user may like a particular song if there is another song by the same artist valued positively by them and whose artist has the same tags as the song to be recommended.

$R4$: A user may *not* like a particular song if there is another song by the same artist that is negatively valued by them and this artist has the same tags as the song to be recommended.

As explained above, we can derive analogous guidelines based on genre, ratings, and user similarity.

These criteria model knowledge of the application domain—they are the building blocks used to build arguments for/against specific recommendations. Note that they combine qualitative and quantitative approaches, taking advantage of the flexibility offered by argumentation-based reasoning.

3.2.2. Step 2: Establish priorities

We consider that the preference criterion that best responds to this particular domain is *rule priority*. Therefore, in this step an order of priorities must be defined between the defined criteria. We use the symbol "$\succ$" to denote "greater priority than". An example of priority using the rules defined above is: $R1 \succ R4$. The next step involves the formalization of these priorities in a formal logical language.

3.2.3. Step 3: Specify the KB

We now present a subset of the rules and priorities formalized in DeLP, whose language is based on logic programming, where the basic concepts such as variables and functions are defined in the usual way. Literals are atoms that can be preceded by the symbol $\sim$ denoting a strict negation; facts are positive literals. Strict rules are ordered pairs $L_0 \leftarrow L_1, \ldots, L_n$, where the first component, L_0, is a literal and the second component, $L_1, \ldots, L_n$,is a set finite and not empty if literals. Similarly, a defeasible rule is an ordered pair $L_0 -\!\prec L_1, \ldots, L_n$, where the first component, L_0, is a literal and the second component, $L_1, \ldots, L_n$, is a finite and not empty set of literals. Strict rules are used to represent incontrovertible information, while defeasible rules are used to represent defeasible knowledge (that is, tentative information that can be used if nothing opposes it).

In this formalism, the state of the domain is modeled through a defeasible logic program, essentially a set of facts, strict rules and defeasible rules. Given a defeasible logic program $\mathcal{P}$, the subset of strict rules and facts is denoted with Π, and the subset of defeasible rules with Δ. In this way, a $\mathcal{P}$ program can be denoted with (Π, Δ). Since the set Π represents non defeasible information, this must be non-contradictory. We use the convention that names of variables begin with capital letters, while the constants and names of predicates begin with lowercase letters. Finally, given a program $\mathcal{P}$ and query Q, the reasoner must provide a response based on the domain knowledge.

Recommending a song based on artist:

$R1$ (defeasible rule):

likes_by_artist(*Track1*, *User*) $-\!\prec$
listen_artist(*User*, *Track2*, *Artist*, l),

Track1 = \ = *Track2*,
same_artist(*Track1*, *Track2*).

$R2$ (defeasible rule):

$\sim$*likes_by_artist*(*Track1*, *User*) $-\prec$
listen_artist(*User*, *Track2*, *Artist*, b),
Track1 = \ = *Track2*,
same_artist(*Track1*, *Track2*).

$E1$ (strict rule):

same_artist(*Track1*, *Track2*) $\leftarrow$
artist_track(*Artist*, *Track1*)
artist_track(*Artist*, *Track2*).

Priorities:

We use "$better\_rule(rule1, rule2)$" to encode that R1 has priority over R2.

General rules:
Based on all the rules defined, some further general rules are formalized:

recommend(*Track*, *User*)$-\prec$
likes_by_artist(*Track*, *User*)
recommend(*Track*, *User*)$-\prec$
likes_by_artist_track(*Track*, *User*).

The formalization of all the criteria, and the priorities over them, define the logic program (model) used by the reasoner to guide the recommendation process.

3.3. Case Study: Stage 4

The reasoning machinery provided by Defeasible Logic Programming (*DeLP*) [4] is used as the argumentation-based tool in the system. An important design aspect is the integration between DeLP programs and relational databases, which affords the necessary information from entities to create arguments. In our proposal, such integration is provided by the *Database Integration for DeLP* (DBI-DeLP) framework [21]. A DBI-DeLP program is an extended DeLP program with information obtained from one or more databases. An important point is the need to consider the possible presence of contradictory information linked to the use of several databases, which for instance could lead to reasons both in favor of the recommendation of an element and against it. Since facts in DeLP cannot be contradictory, here we adopt the notion of *presumption* to represent "defeasible" information.

In DBI-DeLP, the tuples of the database (in our case, the information from the dataset) are represented as a particular type of presumptions called operative presumptions, which are literals of the form $predicate(q_1, \ldots, q_m) -\prec true$. A DBI-DeLP program is a DeLP program with a set of operative presumptions, associated with the dataset records used in the RS, which are retrieved at the request of the system to answer a particular query associated with a recommendation and then discarded. In summary, to obtain data relevant to the argumentation process, elements from the literal that the dialectical process is trying to warrant are used to determine relevant records in the database, and the relevant SQL queries are issued. Finally, all the recovered results are transformed into operative presumptions—this dynamic search for relevant information is crucial in adequately leveraging available datasets.

The use of DeLP as the argumentation engine was chosen based on our familiarity with its implementation, which simplified the development of the prototype used for this case study; we also consider that DeLP is a powerful tool based on intuitive concepts. Note, however, that this choice is modular and there are alternative structured argumentation systems in the literature, such as ASPIC+ [5], ABA [6], and hybrid methods [8]. In this last work the authors present a method for making predictions in RS, and show experimentally that it is competitive in the movie domain; they also illustrate how it can be used to generate effective explanations, which is a valuable byproduct of many dialectical processes, not just DeLP.

3.4. Case Study: Stage 5

The goal of the evaluation is to determine if the system is able to make good predictions regarding items that users like. Here we report on the results of an offline experiment to evaluate our case study.

3.4.1. Experiment setup

The dataset consists of 1,200 valuations, selected at random with the sole condition that they come from

different users; our goal was to avoid the possible introduction of biases stemming from the behavior of specific users. There may be repeated songs in the ratings, since the same song may appear in multiple ratings from different users.

The experiment consists of issuing a series of queries to the system to evaluate its capability of predicting whether or not a given user likes a song. Such queries are of the form "*recommend*(*Track*, *User*)?". Then, each answer was classified in one of the following categories, according to the response obtained and the evaluation that the user gave to that song within the dataset[1]: *True Positive* (TP): recommended song rated "l" (love) by the user; *True Negative* (TN): song not recommended, and user rated it "b" (bad); *False Positive* (FP): recommended song rated "b" (bad); *False Negative* (FN): song not recommended, and user rated it "l" (love); and *Undecided* (U): the system neither suggests nor denies the recommendation. The last case occurs when the system cannot guarantee either recommendation or non-recommendation of a song to a user, thus arriving at an undecided position. To carry out the test, 1,200 queries were made to the recommender system (one for each valuation in the dataset).

The experiment was run twice, one for each DeLP program (mixed and quantitative models) in order to present a comparison of the performance of the two approaches—the DeLP programs were derived based on the guidelines presented above. The first program follows the mixed approach, and is composed of all the previously defined preference criteria so the system is able to work with both quantitative and qualitative features of the entities that participate in the domain. The second program is based on criteria that consider purely quantitative features, as in traditional RS.

3.4.2. Results

The results are presented in Figure 3; on the left we show how each approach performed regarding answer categories, and on the right we have the classical metrics derived from these values. As we can see, the mixed approach outperforms the purely quantitative one in all metrics; its only disadvantage is observed in the true negative category, but this is offset by its performance in the rest. These result clearly show the advantages of applying a mixed approach, which allows to "refine" the answer given by the quantitative approach based on a more complete analysis of the available information.

3.5. Case Study: Stage 6

We propose explanations based on the approach by [2], where explanations are derived based on the structures of the arguments involved, making a simple replacement of the logical structure to their corresponding colloquial interpretations. As an example, we have

Argument structure:

recommend(*Track*, *User*)—≺
likes_by_artist(*Track1*, *User*),
listen_artist(*User*, *Track2*, *Artist*, *l*),
Track1 = \ = *Track2*,
same_artist(*Track1*, *Track2*)

Explanation: "*User*, the song *Track* was recommended since you liked another song by the same artist.

Such explanations can then be offered to users upon request.

3.6. Case Study: Stage 7

For reasons of space, and since the user interface does not differ from typical ones in similar systems, we do not include details of this stage here.

4. Conclusions and Future Work

The main contribution of this work lies in the development of methodological guidelines for the analysis and design of an ABRS capable of: (i) making recommendations to its users from an incomplete or inconsistent knowledge base, (ii) providing the possibility of adapting the knowledge-based analysis that is carried out according to its users' preferences, (iii)

[1] To ensure that the triple (*Track*, *User*, *Rating*) had no influence on the prediction made by the system, the corresponding record was removed from the test dataset.

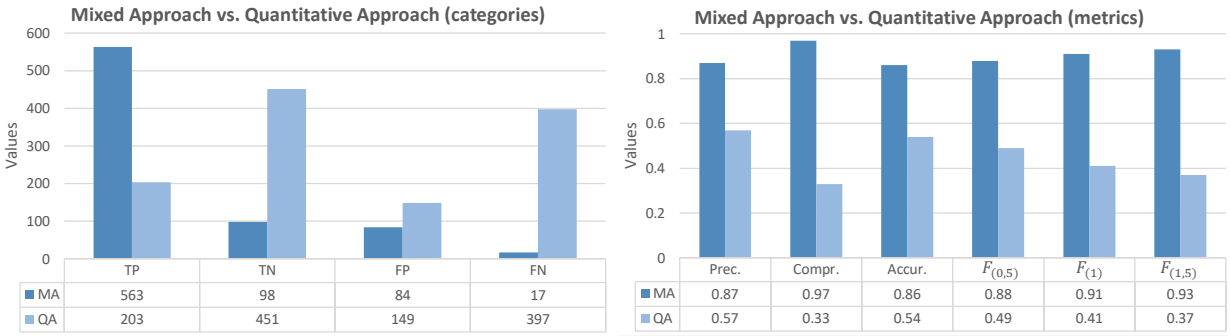

Figure 3: Comparison of approaches: Categories (*left*) and Metrics (*right*).

analyzing qualitative and quantitative information, and (iv) providing explanations. The state of the art of ABRS focuses on studying the characteristics of such systems and the development of prototypes, without applying a methodology to guide the process; i.e., exploiting argumentation-based tools in RS, without referring to the specific design choices made. Our work is therefore a first approach to the investigation of software development methodologies tailored to this type of system.

Future work involves further evolving these guidelines towards a formal and solid software development methodology for creating high-quality ABRS.

Acknowledgments.. This work was funded in part by Universidad Nacional del Sur (UNS) under grants PGI 24/N046 and PGI 24/ZN34, by Consejo Nacional de Investigaciones Científicas y Técnicas (CONICET) under grant PIP 11220170100871CO, and by Agencia Nacional de Promoción Científica y Promoción Tecnológica under grant PICT-2018-0475 (PRH-PIDRI-2014-0007).

References

[1] C. I. Chesñevar, A. G. Maguitman, and M. P. González, "Empowering recommendation technologies through argumentation," in *Argumentation in artificial intelligence*, pp. 403–422, Springer, 2009.

[2] C. E. Briguez, M. C. Budan, C. A. Deagustini, A. G. Maguitman, M. Capobianco, and G. R. Simari, "Argument-based mixed recommenders and their application to movie suggestion," *Expert Systems with Applications*, vol. 41, no. 14, pp. 6467–6482, 2014.

[3] G. R. Simari and R. P. Loui, "A mathematical treatment of defeasible reasoning and its implementation," *Artificial intelligence*, vol. 53, no. 2-3, pp. 125–157, 1992.

[4] A. J. García and G. R. Simari, "Defeasible logic programming: DeLP-servers, contextual queries, and explanations for answers," *Argument & Computation*, vol. 5, no. 1, pp. 63–88, 2014.

[5] S. Modgil and H. Prakken, "The ASPIC+ framework for structured argumentation: a tutorial," *Argument & Computation*, vol. 5, no. 1, pp. 31–62, 2014.

[6] F. Toni, "A tutorial on assumption-based argumentation," *Argument & Computation*, vol. 5, no. 1, pp. 89–117, 2014.

[7] C. E. Briguez, M. Capobianco, and A. G. Maguitman, "A theoretical framework for trust-based news recommender systems and its implementation using defeasible argumentation," *International Journal on Artificial Intelligence Tools*, vol. 22, no. 04, p. 1350021, 2013.

[8] A. Rago, O. Cocarascu, and F. Toni, "Argumentation-based recommendations:

Fantastic explanations and how to find them," 2018.

[9] C. Diez, J. Palanca, V. Sanchez-Anguix, S. Heras, A. Giret, and V. Julián, "Towards a persuasive recommender for bike sharing systems: A defeasible argumentation approach," *Energies*, vol. 12, no. 4, p. 662, 2019.

[10] I. Portugal, P. Alencar, and D. Cowan, "The use of machine learning algorithms in recommender systems: A systematic review," *Expert Systems with Applications*, vol. 97, pp. 205–227, 2018.

[11] B. S. Neysiani, N. Soltani, R. Mofidi, and M. H. Nadimi-Shahraki, "Improve performance of association rule-based collaborative filtering recommendation systems using genetic algorithm," *Int. J. Inf Technol. Comput. Sci.*, vol. 2, pp. 48–55, 2019.

[12] M. K. Najafabadi, A. H. Mohamed, and M. N. Mahrin, "A survey on data mining techniques in recommender systems," *Soft Computing*, vol. 23, no. 2, pp. 627–654, 2019.

[13] F. Lin and Y. Shoham, "Argument systems: A uniform basis for nonmonotonic reasoning.," *KR*, vol. 89, pp. 245–255, 1989.

[14] H. Prakken and G. Sartor, "Argument-based extended logic programming with defeasible priorities," *Journal of applied non-classical logics*, vol. 7, no. 1-2, pp. 25–75, 1997.

[15] P. M. Dung, "On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games," *Artificial intelligence*, vol. 77, no. 2, pp. 321–357, 1995.

[16] C. Cayrol and M.-C. Lagasquie-Schiex, "On the acceptability of arguments in bipolar argumentation frameworks," in *ECSQARU*, pp. 378–389, Springer, 2005.

[17] G. Shani and A. Gunawardana, "Evaluating recommendation systems," in *Recommender systems handbook*, pp. 257–297, Springer, 2011.

[18] C. Lacave and F. J. Diez, "A review of explanation methods for heuristic expert systems," *The Knowledge Engineering Review*, vol. 19, no. 2, pp. 133–146, 2004.

[19] D. Walton, "A new dialectical theory of explanation," *Philosophical Explorations*, vol. 7, no. 1, pp. 71–89, 2004.

[20] B. Moulin, H. Irandoust, M. Bélanger, and G. Desbordes, "Explanation and argumentation capabilities: Towards the creation of more persuasive agents," *Artificial Intelligence Review*, vol. 17, no. 3, pp. 169–222, 2002.

[21] C. A. Deagustini, S. E. F. Dalibón, S. Gottifredi, M. A. Falappa, C. I. Chesñevar, and G. R. Simari, "Relational databases as a massive information source for defeasible argumentation," *Knowledge-Based Systems*, vol. 51, pp. 93–109, 2013.

Author Bios

Mario Leiva is a Ph.D. student at Universidad Nacional del Sur, Argentina. His main interests are in Artificial Intelligence, focusing on theoretical and practical aspects of argumentation-based reasoning. E-mail: mario.leiva@cs.uns.edu.ar.

Maximiliano C.D. Budán is a professor at Universidad Nacional de Santiago del Estero, and a researcher at the Institute of Computer Science and Engineering – CONICET, Argentina. His main interests are in Artificial Intelligence and Abstract Algebras, with a focus on reasoning under uncertainty considering qualitative and quantitative features varying over time. He obtained his Ph.D. from Universidad Nacional del Sur (Bahía Blanca, Argentina), in the area of Artificial Intelligence. E-mail: mcdb@cs.uns.edu.ar.

Gerardo I. Simari is a professor at Universidad Nacional del Sur, a researcher at CONICET, Argentina, and adjunct faculty at Arizona State University (USA). His main interests lie at the intersection of Artificial Intelligence and Databases, with a particular focus on reasoning under uncertainty. He obtained his Ph.D. from University of Maryland College Park (USA), in the area of Artificial Intelligence. E-mail: gis@cs.uns.edu.ar.