Provenance- and Machine Learning-based Recommendation of Parameter Values in Scientific Workflows

Daniel Silva Junior[1], Aline Paes[2], Esther Pacitti[3], and Daniel de Oliveira[4]

[1]**danieljunior@id.uff.br**
[2]**alinepaes@ic.uff.br**
[3]**Esther.Pacitti@lirmm.fr**
[4]**danielcmo@ic.uff.br**

Corresponding author:
Daniel de Oliveira[1]

Email address: danielcmo@ic.uff.br

ABSTRACT

Scientific Workflows (SWfs) have revolutionized how scientists in various domains of science conduct their experiments. The management of SWfs is supported by complex tools named Scientific Workflow Management Systems that provide support for workflow composition, monitoring, execution, capturing, and storage of the data generated during execution. In some cases, they even provide components to ease the visualization and analysis of the generated data. During the workflow's composition phase, programs must be selected to perform the activities defined in the workflow. These computer programs, which act as experiment steps, often require additional parameters that serve to adjust the programs according to the experiment's goals. Consequently, workflows have many parameters to configure manually, encompassing even more than one hundred in many cases. Choosing wrongly the parameters' values can lead to unsuccessful executions of the workflow. As the execution of data- and compute-intensive workflows is commonly performed in a high-performance computing environment (e.g., a cluster, a supercomputer, or a public cloud), an unsuccessful execution configures a waste of time and resources. In this manuscript, we present *FReeP - Feature Recommender from Preferences*, a parameter value recommendation method that is designed to suggest values for workflow parameters, taking into account user preferences. *FReeP* is based on Machine Learning techniques, particularly in Preference Learning. *FReeP* is composed of three algorithms, where two of them aim at recommending the value for one parameter at a time, and the third makes recommendations for n parameters at once. The experimental results obtained with provenance data from two broadly used workflows showed *FReeP* usefulness in the recommendation of values for one parameter. Furthermore, the results indicate FReeP's potential to recommend values for n parameters in scientific workflows.

INTRODUCTION

Scientific experiments are the basis for evolution in several areas of human knowledge (de Oliveira et al., 2019; Mattoso et al., 2010b; Hey and Trefethen, 2020; Hey et al., 2012). Based on observations of open problems in their research areas, scientists formulate hypotheses to explain and solve those problems (Gonçalves and Porto, 2015). Such hypothesis may be confirmed or refuted, and also can lead to new hypotheses(Gonçalves and Porto, 2015). For a long time, scientific experiments were manually conducted by scientists, including instrumentation, control of the environment, annotation of results, and analysis. Despite the advances obtained with this approach, time and resources were wasted since the smallest amount of error could compromise the whole experiment. The analysis of errors in the results was also far from trivial.

The evolution in computer science field allowed for the development of technologies that provided useful support for scientists in their experiments. One of these technologies are the *scientific workflows* (de Oliveira et al., 2019; Deelman et al., 2005). Scientific workflows (or simply *workflows*) are an

abstraction that represents each step of the experiment expressed as an *activity*, which has input data and relationships with other activities, according to the stages of the experiment (Yong Zhao, 2008).

Such workflows commonly require the execution of data-intensive operations as loading, transformation, and aggregation (Mattoso et al., 2010b). Several computational paradigms can be used for the design and execution of workflows, *e.g.*, shell and Python scripts (Marozzo et al., 2013), but they are usually managed by complex engines named Workflow Management Systems (WfMS). A key feature that a WfMS must address is the efficient and automatic management of parallel processing activities in High-Performance Computing (HPC) environments (Eduardo Ogasawara and et.al, 2010). Besides managing the execution of the workflow in HPC environments, WfMSs are also responsible for recording metadata associated to all the data generated during the execution: input data, intermediate data, and the results. These metadata is well-known as *provenance* (Juliana Freire and Silva, 2008). Based on provenance data, it is possible to analyze the results obtained and guarantee the reproducibility of the experiment, which is essential to prove the veracity of a produced result.

In this article, the concept of an experiment is seen as encompassing the concept of a workflow, and not as a synonym. A workflow may be seen as a controlled action of the experiment. Hence, the workflow is defined as one of the *trials* conducted in the context of an experiment. In each trial, the scientist needs to define the parameter values for each activity of the workflow. It is not unusual that a workflow has more than 100 parameters to set. Setting up these parameters may be simple for an expert, but not so simple for non-expert users. Although WfMSs represent a step forward by providing the necessary infrastructure to manage workflow executions, they provide a little help (or even no help at all) on defining parameter values for a specific workflow execution. The correct choice of parameter values in a workflow execution is crucial not only for the quality of the results but also influences if a workflow will execute or not. A poor choice of parameter values can cause failures, which leads to a waste of execution time. Failures caused by poor choices of parameter values are even more severe when workflows are executing in HPC environments that follow a pay-as-you-go model, *e.g.*, clouds, since they can increase the overall financial cost.

This way, if the WfMS could "learn" from previous successfully executions of the workflow and recommend parameter values for scientists, some failures could be avoided. This recommendation is especially useful for non-expert users. Let us take as an example a scenario where an expert user has modeled a workflow and executed several trials of the same workflow varying the parameter values. If a non-expert scientist wants to execute the same workflow with a new set of parameter values and input data, but does not know how to set the values of some of the parameters, one can benefit from parameter values used on previous executions of the same (or similar) workflow. The advantage of the WfMS is provenance data already contains the parameter values used on previous executions and can be a rich resource to be used for recommendation. Thus, this article hypothesis is that *by adopting an approach to recommend the parameters values of workflows in a WfMS, we can increase the probability that the execution of workflow will be completed*. As a consequence, the financial cost associated with execution failures is reduced.

In this article, we devise a method named *FReeP - Feature Recommender From Preferences*, which aims at recommending values for parameters of workflow activities. The proposed approach is able to recommend parameter values in two modes: (i) a single parameter value at a time, and (ii) multiple parameter values at once. The proposed approach relies on user preferences, defined for a subset of workflow parameters, together with the provenance of the workflow. The idea of combining user preferences and provenance is novel and allows for producing a personalized recommendation for scientists. FReeP is based on Machine Learning algorithms (Mitchell, 2015), particularly, Preference Learning (Fürnkranz and Hüllermeier, 2011), and Recommender Systems (Ricci et al., 2011). We evaluated FReeP using workflow benchmarks such as Montage (Hoffa et al., 2008) and SciPhy (Ocaña et al., 2011) and results indicated the potential of the proposed approach. This manuscript is an extension of the conference paper "FReeP: towards parameter recommendation in scientific workflows using preference learning" (Silva Junior et al., 2018) published in the Proceedings of the 2018 Brazilian Symposium on Databases (SBBD). This extended version provides new empirical shreds of evidence regarding several workflow case studies as well as a broader discussion on related work.

This article is organized in five sections besides this introduction. *Background* section details the theoretical concepts used in the proposal development. *FReeP - Feature Recommender from Preferences* section presents the algorithm developed for the problem of parameters value recommendation using

user preferences. *Experimental Evaluation* section shows the results of the experimental evaluation of the approach in three different scenarios. Then, *Related Work* section presents a literature review with papers that have addressed solutions to problems related to the recommendation applied to workflows and the Machine Learning model hyperparameter recommendation. Lastly, *Conclusion* section brings conclusions about this article and points out future work.

BACKGROUND

This section presents key concepts for understanding the approach presented in this article to recommend values for parameters in workflows based on users' preferences and previous executions. Initially, it is explained about scientific experiments. Following, the concepts related to Recommender Systems are presented. Next, the concept of *Preference Learning* is presented. This section also brings a *Borda Count* overview, a non-common voting schema that is used to decide which values to suggest.

Scientific Experiment

A scientific experiment arises from the observation of some phenomena and questions raised from the observation. The next step in the experiment is the hypotheses formulation, that is, to develop possible answers to the questions raised. With a developed hypothesis, it is necessary to test it to verify if an output produced is a possible solution. Regardless of the final result with the formulated hypothesis, there are many iterations of refinement, which may consist, for example, of testing the hypothesis under other conditions, until it is possible to have enough elements to support the hypothesis.

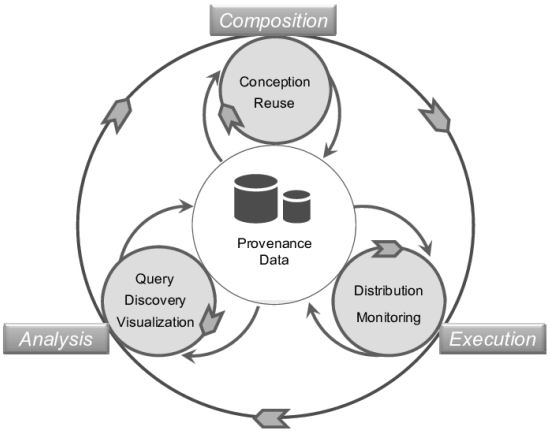


Figure 1. Mattoso et al. (2010a) Life Cycle of Scientific Experiments.

The scientific experiment can be divided into three major phases: *composition*, *execution* and *analysis*. Figure 1 illustrates what is called the life cycle of the experiment comprising these three phasesMattoso et al. (2010a). The *composition* phase is where the experiment is designed and structured and can still be divided into two stages: *conception* and *reuse*. The *conception* stage is responsible for creating an abstraction that represents the experiment, while the search for other experiments that have parts that can be adapted and used in the development is in charge of *reuse*. *Execution* is the phase where all the necessary instrumentation for the accomplishment of the experiment must be finished. Instrumentation means the definition of input data, parameters to be used at each stage of the experiment, and monitoring mechanisms. Finally, the *analysis* phase is where the data generated by the composition and execution phases are studied to understand the results obtained. The approach presented in this article focus on the *Composition* phase, where the workflow is being configured for execution.

How "big" are these systems?

Scientific Workflows

Scientific workflows have become a *de facto* standard for modeling *in-silico* experiments (Zhou et al., 2018). *Workflows* are abstractions that represent experiment steps and the dataflow through each of these steps. A workflow can be formally defined as a directed acyclic graph $W(A, Dep)$. The nodes $A = \{a_1, a_2, \ldots, a_n\}$ are the activities and the edges Dep represent the data dependencies among activities in A. Thus, given $a_i \mid (1 \leq i \leq n)$, the set $P = \{p_1, p_2, ..., p_m\}$ represents the possible input parameters for

↳ Maybe subscript by i?

activity a_i that defines the behavior of a_i. Therefore, a *workflow* can be seen as a graph where the vertices act as experiment steps and the edges are the relations, or the dataflow between the steps.

A workflow can also be categorized according to the level of abstraction into *conceptual* or *concrete*. A conceptual workflow represents the highest level of abstraction, where the experiment is defined in terms of steps and dataflow between them. This definition does not explain how each step of the experiment will execute. The concrete *workflow* is an abstraction where the activities are represented by the computer programs that will execute them. Workflow activities executions are called *activations* (de Oliveira et al., 2010a), where each program that represents an activity has its parameters defined. However, managing this execution, which involves setting the correct parameter values for each program, capturing the intermediate data and execution results, becomes a challenge. It was with this in mind, and with the help of the composition of the experiment in the workflow format, that *Workflow Management Systems (WfMS)*, such as Kepler (Altintas et al., 2006), Pegasus (Deelman et al., 2005) and SciCumulus (de Oliveira et al., 2010a) emerged.

SciCumulus is an important component of the proposed approach since it provides a framework for parallel scientific workflows to be combined with FReeP. It is necessary to highlight that other SWfMSs such as Pegasus and Kepler could also benefit from FReeP as long as they provide necessary provenance data. SciCumulus architecture is composed of four main components: *SCSetup*, *SCStarter*, *SCCore*, and *SCQP* (SciCumulus Query Processor). SCSetup is responsible for storing and retrieving prospective provenance to/from the provenance database. Using this component, scientists insert/update the structure of the workflows in the database. When the structure of the workflow is already inserted in the provenance database, SCStarter can be invoked. SCStarter component is responsible for configuring the environment for executing the workflow. In case of executions in cloud environments, SCStarter is responsible for deploying virtual machines and configuring storage services before the workflow execution. SCStarter has to be compatible to the cloud API. In the current version, SCStarter works with Amazon AWS API.

When all virtual machines and storage services are running, SCStarter invokes SCCore in each virtual machine. SCCore is an MPJ (MPI-like)[1] application, so it runs in all virtual machines at the same time using message passing (each virtual machine contains an instance of SCCore according to the rank of the virtual machine, *i.e.,* SCCore_0, SCCore_1, etc). SCCore follows a Master/Worker architecture. The SCCore-Master (SCCore_0) is responsible for scheduling the workflow activities in the several virtual machines. In addition, SCCore-Master is responsible for collecting retrospective provenance data and store it in the provenance database. All other instances of SCCore-Workers receive activations to execute and request more activations. In the original version of SciCumulus, all data is stored in one single bucket in the Amazon S3 service, thus it is not fragmented neither distributed in different buckets. The same problem occurs with other existing SWfMSs since they do not consider data file distribution and results confidentiality issues. The SCQP component is responsible for querying the provenance database during or after the workflow execution. It can be used by scientists to steer the workflow or to perform a *post-mortem* analysis of the results. For more information about SciCumulus please refer to Oliveira *et al.* (de Oliveira et al., 2012, 2010b).

Provenance

An workflow activation has input data, and generates intermediate and output data. WfMS has to collect all metadata associated to the execution in order to foster reproducibility. This metadata is called *provenance* (Juliana Freire and Silva, 2008). According to Goble (2002), the provenance must verify data quality, path audit, assignment verification, and information querying. Data quality check is also related to verifying the reliability of workflow generated data. Path audit is the ability to follow the steps taken at each stage of the experiment that generated a given result. The assignment verification is linked to the ability to know who is responsible for the data generated. Lastly, an information query is essential to analyze the data generated by the experiment's execution.

Especially for workflows, provenance can be classified as prospective (p-prov) and retrospective (r-prov) (Juliana Freire and Silva, 2008). *p-prov* represents the specification of the workflow that will be executed. It corresponds to the steps to be followed to achieve a result. *r-prov* is given by executed activities and information about the environment used to produce a data product, consisting of a structured and detailed history of the execution of the workflow.

[1] http://mpj-express.org/

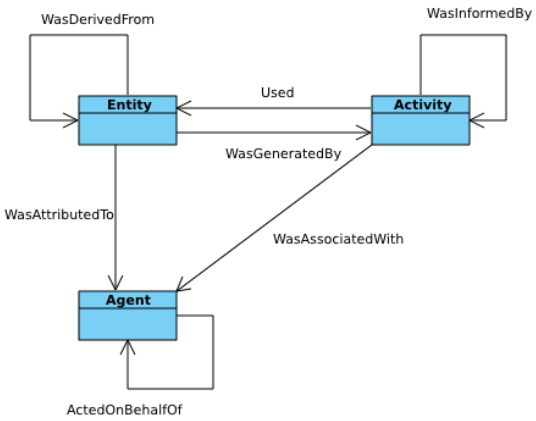


Figure 2. Belhajjame et al. (2013) *PROV* data model.

Provenance is vital for the scientific experiment analysis phase. It allows for verifying what caused an activation to fail or generated an unexpected result, or in the case of success, what were the steps and parameters used until the result. Another advantage of provenance is the reproducibility of an experiment, which is essential for the validation of the results obtained by third parties.

Considering the provenance benefits in scientific experiments, it was necessary to define a model of representation of provenance (Bose et al., 2006). The standard W3C model is *PROV* (Gil et al., 2013). *PROV* is a generic data model and is based on three basic components and their links, being the components: *Entity*, *Agent* and *Activity*. Figure 2 shows how the *PROV* representation uses the three components and how their links are used to represent the dataflow during the execution of the experiment.

Recommender Systems

The massive data available in the information age, although beneficial to users, can also be seen as an avalanche that overloads them and, in some cases, leaves them lost in the search for relevant information. This difficulty in searching relevant information reflects the degree of freedom that the user has in searching for available data. To address this problem *recommender systems (RS)* (Resnick and Varian, 1997) (Ricci et al., 2011) (Bobadilla et al., 2013) are developed. recommender systems are based on techniques that aim to suggest the most relevant items to the user to perform their task, such as choosing one book. The *Amazon.com* (Linden et al., 2003) recommender system is an example of how useful it is to suggest some items from all the search space available on the platform.

Generally, recommender systems are specialized in only one type of item, where the objects that the system was developed to recommend are understood by item. The recommender systems can be of the type *personalized*, which means that each user of the system will have their recommendations based on their preferences. There are also *non-personalized* recommender systems that return *top-n* lists. Freep is based on the first type.

Personalized recommendations seek to suggest the most relevant items for carrying out the user's task. For this, it is necessary to collect the user's preferences, which can be *explicit* or *implicit*. An example of explicit preference is the evaluation of a book, giving it zero to five stars. An implicit preference may be to visit a page about a book, how long the page remains, *etc*. The development of a recommender system is based on the observation that other close individuals' suggestion influences individuals' decisions. An example of this behavior is music choice, where a system can suggest songs to a user based on songs that other same age group users have heard, are listening or have heard and that the current user has not yet heard.

There are three essential elements for the development of a recommender system: *Users*, *Items*, and *Transactions*. The *Users* are the target audience of the recommender system, and each user has their characteristics and objectives. The recommender system must be able to make suggestions for items relevant to each user's objectives. As mentioned, *Items* are the recommendation objects and also have their characteristics. Its relevance varies from user to user. The *Transactions* are records that hold a tuple *(user, interaction)*, where the interaction is the actions that the user performed when using the recommender system. These interactions are generally user feedbacks, which can be interpreted as their preferences.

The most common is to use transactions that only consider explicit feedback of preference, such as evaluating an item as bad, regular, or good. Implicit feedback is also useful but needs a deeper assessment of what behaviors should be considered as feedback and how to use it to improve the recommendations' usefulness.

The recommender system task can be defined as: given the elements *Items, Users* and *Transactions*, find the most useful items for each user. Adomavicius and Tuzhilin (2005) formalizes that a recommender system must satisfy Equation 1, considering U as the space for all users, I the space for all items and F the utility function that calculates the utility of an item i *inI* for a u *inU* user. Note that in Equation 1 the tuple (u,i) will not be defined in the entire search space, so the recommender system must extrapolate the F function in these cases.

$$\forall u \in U, i'_u = \arg\max_{i \in I} F(u,i) \tag{1}$$

Calculating the utility function varies according to the approach that will be used in the recommender system. It is based on the different strategies used to calculate the utility function that recommender systems can be categorized. The most common approaches to recommender systems are: *Content Based, Collaborative Filtering* and *Hybrids*. There are still *Demographic* and *Knowledge-based*, however their applications are restricted to specific cases (Ricci et al., 2011). Figure 3 provides a taxonomy of the interesting types of recommender systems for this work.

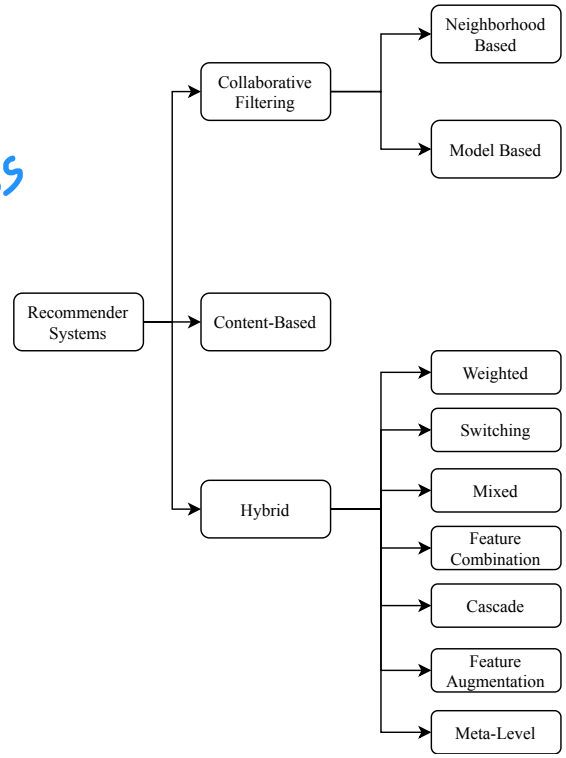


Figure 3. Interesting types of recommender systems taxonomy.

Collaborative Filtering

In *Collaborative Filtering Recommender Systems*, a recommendation is based on other users' experience with items in the system domain. The idea is very intuitive and is related to the human behavior of, at times, giving credit to another person's opinion about what should be done in a given situation. The experience of other users with the recommender system's domain items, in general, is expressed through *evaluations*. Assessments can be expressed explicitly or implicitly (Schafer et al., 2007). An explicit evaluation is one captured employing a direct request from the system to the user, such as ask the user for rating a seen film on a scale from zero to five. The implicit evaluation is inferred by the behavior

user interaction with the system, such as the time spent viewing information about a film genre. In *Collaborative Filtering*, evaluations can be of the following types: scalar, binary, or unary (Schafer et al., 2007). Scalar evaluations are like the example of a film evaluation; its values are included in a numerical range. The binary assessment type is like good/bad, agree/disagree. Unary evaluation is present in item purchase or non-purchase information.

Collaborative Filtering can be *Neighborhood Based* or *Model Based*. *Neighborhood Based Collaborative Filtering* strictly follows the principle that users with similar profiles have similar preferences. Thus, given a neighborhood, the most popular items among them are used as a recommendation, where popularity is measured concerning an item's ratings.

A user's neighborhood can be defined concerning the users of the system themselves or about the items evaluated by the users. When the neighborhood is defined concerning users, the system predicts a r_{ui} rating of a new i item for a u user from the average of the nearest k neighbors' ratings. It is important to note that the nearest neighboring k must be those that have evaluated the item i. A problem with using the average of the ratings of the nearest neighboring k is not to differentiate the ratings of users with the closest profiles. In this case, an alternative is to include a weighting factor in the calculation of an item's valuation prediction. Alternatively, a user's neighborhood can be defined in terms of items in the recommendation domain. This approach works as follows: let u be an user with a I set of evaluated items and a new i' item a candidate for a recommendation item. The k items, k inI, more similar to i' can be used to predict how appropriate the i' recommendation to u is.

The neighborhood-based recommender system needs a similarity measure to define a neighborhood. A similarity measure traditionally used to define a neighborhood is *Cosine Similarity* (Bell and Koren, 2007). In the context of the recommendation, for example, based on users' similarities, this measure requires that users have a vector representation. Therefore, let u be an user represented by a vector A and v an user represented by a vector B, the similarity of these two users using *Cosine Similarity* is given by Equation 2. Another similarity measure widely used in the definition of neighborhoods in *Collaborative Filtering* is *Pearson's Correlation* (Massa and Avesani, 2007). Still, using the notation where a user u is represented by a vector A and another user v is represented by a vector B, *Pearson's correlation* can be calculated using Equation 3.

$$\cos\theta = \frac{A.B}{\|A\|\|B\|} \quad (2) \qquad \rho = \frac{cov(A,B)}{\sqrt{var(A).var(B)}} \quad (3)$$

Neighborhood Based Collaborative Filtering has as strengths the straightforward interpretation and explanation of the recommendations made and the ability to incorporate new users and items into the recommendation process without needing a *offline* training phase. However, as the training base grows, the recommendation becomes more expensive due to the algorithm's quadratic nature, due to the need to calculate similarity measures.

In contrast, instead of loading the entire database into memory and calculating similarity measures for each new recommendation, *Model Based Collaborative Filtering* seeks to generate a hypothesis from the data and use it to make recommendations instantly. Two of the most used approaches in this *Collaborative Filtering* category are *clustering* and *matrix factorization* (Thi Do et al., 2010). *Clustering* (Ungar and Foster, 1998) use in the recommendation is based on the idea that users in the same group have similar interests. The clustering-based recommendation also requires vector representation of users and is divided into three steps: 1) dividing users into k groups; 2) assignment of a user who will receive the recommendation to one of the groups; 3) use of item evaluations by users of the assigned group to recommend a new item.

Collaborative Filtering based on *matrix factorization* (Koren et al., 2009) uses an evaluation matrix generated from users and items. This R evaluation matrix has m lines representing each system user and n columns representing the items. Thus, a cell r_{ij} represents the user rating u_i for the item i_j. This matrix$R$ in general is sparse due to the large number of items not evaluated by all users of a recommender system. Matrix factorization uses matrix algebra to decrease the size of the R matrix, thus addressing the R sparsity problem. Two of the main *matrix factorization* models in the context of *Collaborative Filtering* are *Principal Component Analysis (PCA)* (Smith, 2002) and *Singular Values Decomposition (SVD)* (Ma, 2008).

The *PCA* factorization idea is to find the most relevant evaluation matrix components without loss of information. In the context of the R evaluation matrix, the most relevant components can be seen as

the users who perform the most evaluations and the items that receive the most evaluations. Differently, *SVD* looks for two matrices: U a matrix of user characteristics; and V a matrix of characteristics of recommendation items. The prediction of a user rating u_i for an item i_j is then given by a prediction function $p(U_i, V_j)$.

Content-Based

Content-based Recommender Systems make recommendations similar to items that the user has already expressed a positive rating in the past. To determine the similarity degree between items, this approach is highly dependent on extracting their characteristics. This extraction is necessary because the original representation of an item is often not structured, which is a prerequisite for automating recognizing similarities. Three main components can represent this recommender system type: Content Analyzer, Profile Learner, and Filtering, as seen in Figure 4.

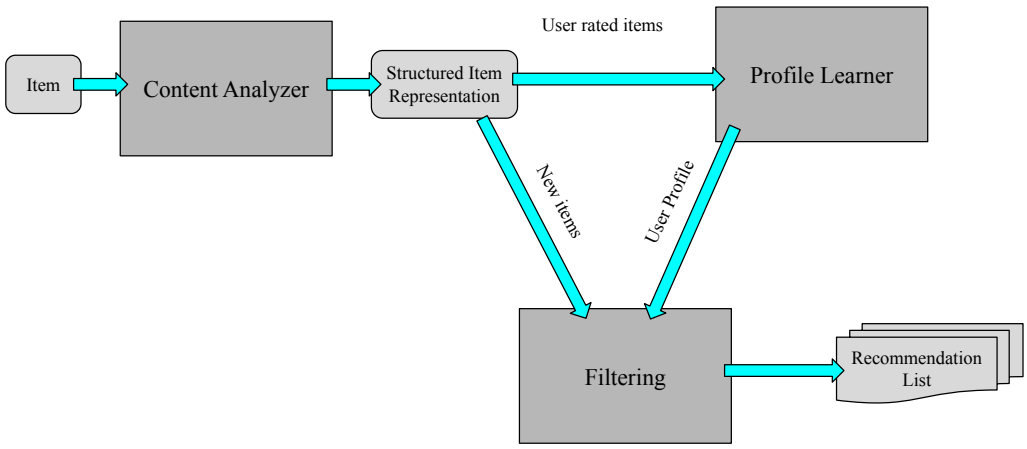


Figure 4. Content-Based recommender system architecture overview: the Content Analyzer component is responsible for extracting the characteristics of the items, producing a structured representation; the Profile Learner component is responsible for building a model that represents a user's profile based on past preferences; lastly, the Filtering component uses other items not yet evaluated as input to the user profile model, and its output is the recommended items.

Content Analyzer: The target items of recommendation are, in most cases, not adequately represented for the use of the computer in automating the recommendation. This component is responsible for producing a structured representation of the items. One of the most used representations is *TF-IDF* (Salton, 1989), which has its origin in the area of *Information Retrieval*. Equation 4 formalizes *TF-IDF*, considering a feature x and an item y , $tf_{x,y}$ is the frequency of the feature x in y, N is the total of items, and df_x is the number of items that contain x. The variable $w_{x,y}$ is a weight vector where each component of the vector represents the relevance of the feature x to the item y. This type of representation serves as input for the other recommender system components.

$$w_{x,y} = tf_{x,y} \times \log\left(\frac{N}{df_x}\right) \quad (1) \tag{4}$$

Profile Learner: This component produces a model, induced by Machine Learning techniques, that represents the profile of an user based on the items evaluated by this user. The *Naive Bayes* (Lewis, 1998) is a probabilistic model based on Bayes' Theorem, and its objective is to estimate the probability *a posteriori* of an event A to occur, given that an event B occurs. In the recommender system domain, Naive Bayes estimates the probability of an item be relevant, given its characteristics.

Filtering: With the structured representation of the items and a model of the user profile at hand, it is possible to filter the items not yet evaluated, creating a list of these "new items" ordering them according to their similarity to the others positively assessed items.

Hybrid

Hybrid recommender systems arise out of an attempt to minimize the weaknesses that traditional recommendation techniques have when used individually. Also, it is expected that a hybrid strategy can aggregate the strengths of the techniques used together. There are some methods of combining recommendation techniques in creating a hybrid recommender system, among them: *Weighting*, *Switching*, *Mixing*, *Feature Combination*, *Cascade*, *Feature Augmentation* and *Meta-Level* (Burke, 2002).

Weighting: In this method, each recommendation item has a score that is defined by combining the prediction results from all recommendation techniques that are in use. It is a simple method that can be implemented, for example, as a linear combination of predictions, which can be assessments, that each recommendation technique assigns to an item.

Switching: This method defines a criterion responsible for determining which recommendation technique will be used in each recommendation situation. An example of this method is to use the Content-Based approach, and if the recommendation of this method does not reach a predetermined degree of confidence, use the Collaborative Filtering method to make the recommendation. A major drawback of this method is the inclusion of a configuration parameter in the system, which is the alternation criterion responsible for defining when to use one technique or another according to the recommendation situation.

Mixing: This method is an alternative when the system can make more than one recommendation at a time. The mixed-method consists of using all recommendations from all recommendation techniques employed in the system. An example of using this method is a list of suggestions for TV or film programming, where generally more than one recommendation item is presented at a time.

Feature Combination: this method is based on the hybridization between the Content-Based recommendation and the Collaborative Filtering. The method treats the neighborhood information in Collaborative Filtering as additional information from the dataset and then applies the Content-Based recommendation technique.

Cascade: here, the method is divided into steps as follows: 1) a recommendation technique is used first, performing something similar to a filter of the candidate items for the recommendation; 2) a second technique refines candidates from the first stage, looking for the best alternative.

Feature Augmentation: in this method, there is also a chain of recommendation techniques. The method uses a recommendation technique on the original data, and the output of this first technique is used as an additional resource when entering a second technique. Although it is also a chain of techniques like the *Cascade*, it is important to note that the *Feature Augmentation* is different because the set of input attributes for one is different from the input for the other technique.

Meta-Level: this method uses the model generated by one technique as input for the second. Note that the difference between this method and the *Feature Augmentation* is that this method uses the generated model itself as an input to the second technique, while the *Feature Augmentation* uses the model output generated by the first technique as an additional database resource for the second technique.

The Content Based and Collaborative Filtering methods combination allows better performance in recommendations in different domains due to the possibility of minimizing the weaknesses of each approach when used separately. However, it is not an easy task to combine the two approaches since the methods used by each approach can generate representations of incompatible data.

Several studies have emerged to propose combining the recommendation approaches in Content and Collaborative Filtering: in (Balabanovic and Shoham, 1997) the recommendations are based on the content that users with similar profiles are evaluated positively; Basilico and Hofmann (2004) uses *Joint Feature Maps* to integrate feature vector of the users and the items; Lekakos and Caravelas (2008) proposed a hybrid recommender system based on the *Switching* method, starting from using collaborative filtering as the main method and if, it is not possible, it goes to the content-based recommendation.

Preference Learning

User preferences play a crucial role in most decision systems. Thus, preferences have become the object of study in economics and psychology, among others, since these areas sometimes use decision support tools (Pigozzi et al., 2016). In Computer Science, this concept has gained more relevance with the growth of recommender systems (Feynman and Vernon Jr., 1963) (Viappiani and Boutilier, 2009) and the development of personal assistants (Mitchell et al., 1994) (Myers et al., 2007).

The first important concept to be clarified when talking about *Preference Learning* is the meaning of *preference*. From an Artificial Intelligence perspective, a *preference* is a problem restriction that allows

some degree of relaxation. Fürnkranz and Hüllermeier (2011) refers to Learning Preferences as "inducing preference models from empirical data". Regarding the empirical data, it can be said that many times a preference is not explicitly expressed. An example is choosing a product from the same category that another product is as an *ecommerce* user. As all products are in the same category, a TV, for example, the user's preference is expressed by clicking on the most interesting products.

A Preference Learning task consists of learning a predictive function that, given a set of items where preferences are already known, predicts preferences for a new set of items. It is necessary to define an adequate representation for the preferences to create an algorithm that solves the problem for which the Preference Learning is proposed. The most common way of representing preferences is through binary relationships. An example of a tuple that would be part of this representation in a classic literature format is $x_i > x_j$, which means a preference for the value i over j for the attribute x. There are also other representations for preferences that aim to facilitate variations of the problem related to Machine Learning, such as the Conditional Preferences Networks (Koriche, 2012), or the Generalized Additive Independence Networks (Gonzales and Perny, 2004).

The task with more research in the Preference Learning area is the *Learning to Rank*. This task arises from the characteristic of requiring a relationship of order between preferences. The task can be divided into three categories: *Label Ranking* (Vembu and Gärtner, 2011), *Instance Ranking* (Bergeron et al., 2008) and *Object Ranking* (Nie et al., 2005). *Label ranking* is a generalization of the traditional Machine Learning Classification task. This *ranker* makes an ordering of the set of classes of a problem for each instance of the problem. An example is the categorization of a news story, where it can be included in several topics simultaneously, such as sports and entertainment. Since the classes of a problem have an order among themselves, the *instance ranking* task consists of ordering the instances of a problem. The instances belonging to the "highest" classes precede the instances that belong to the "lower" classes. An example is the categorization of articles submitted to a congress: *reject, weak reject, weak accept* and *accept*. The task will produce a list where the articles will appear according to their classes, from accepted to rejected ones. In *object ranking* an instance is not related to a class. This task's objective is, given a subset of items referring to the total set of items, to produce a ranking of the objects in that subset—for example, the ranking of web pages by a search engine.

Pairwise Label Ranking (Hüllermeier et al., 2008) is a Preference Learning technique used specifically for the Label Ranking problem and is based on preference relationships learning. This technique consists of learning preferences by decomposing the problem into small binary preference problems, that is, preference between pairs of items. The *Pairwise Label Ranking* technique is based on *Pairwise Preference Learning*, which in turn is based on *Pairwise Classification*. Pairwise Classification is well known in the context of Classification (Fürnkranz, 2002) involving more than one class. Instead of using a single classifier that must make predictions between m classes, given a set L of m classes, learning binary preferences generates $m(m-1)/2$ binary classifiers, where a classifier $M_{i,j}$ only predicts between classes i, j *inL*. With this, each classifier $M_{i,j}$ prediction is treated as a vote for the class i or j, and, in the end, the most voted category is returned as a prediction. The Pairwise Classification technique can be generalized to *Pairwise Preference Learning* (Fürnkranz and Hüllermeier, 2003). The generalization consists of using each instance with a preference type $a > b$, representing that a is preferable to b, as a record of the training base of a classifier $M_{a,b}$ which returns 1 as a prediction that a is preferable to b and 0 otherwise. With this, each classifier $M_{i,j}$ also represents the preference between two categories, and the combination of the preferences expressed by each classifier builds a complete preference relationship. For this, the strategy defined by *Pairwise Label Ranking* uses the prediction of each classifier as a vote and uses a voting system that defines an ordered list of preferences.

To bringing together recommender systems and preference learning, *MovieLens* (Miller et al., 2003) is a good example. MovieLens, recommends films based on ratings, and the recommendation problem is defined in terms of a triples set (*user, item, rating*) representing a system user's ratings for a given item. Other approaches using the MovieLens dataset, in general, focus on learning a Machine Learning model capable of predicting a user's evaluation for an item not yet evaluated. With a trained Machine Learning model, the best-rated n items are recommended. However, the ordering of the items in a descending way in relation to the predicted evaluations does not necessarily mirror the user's order of preference. This is because the models used for the predictions are trained and evaluated based on error metrics. The simple exchange of evaluation between two pairs of items defines the same error measure as the configuration with the correct evaluations. In this way, the recommendation would have its efficiency degraded.

Ranking algorithms can be used to alleviate the recommendation ordering problem, since they are prepared to obtain the order relationship between the predicted evaluations. (Pessiot et al., 2007) uses the approach of exchanging the predictive model for a ranking model using the MovieLens (Harper and Konstan, 2016) dataset. Lu et al. (2012) refined the *Area Under the Curve (AUC)* metric that is traditionally used in recommender systems that work with custom ranking. A new metric called *SAUC* has been proposed, which adds a term to the original equation of *AUC*, which aims to increase the likelihood that an unwanted item, but that fulfills the user's preferences, is well placed in a recommendation list.

Borda Count

An election is a process that, in a democratic environment, aims to seek a consensus between conflicts and interests (Koffi, 2015). Generally, in an election, there is a set of candidates and a set of voters. *Voting Theory* (Taylor and Pacelli, 2008) is an area of Mathematics aimed at the study of voting systems.

In an election between two candidates, it is simple to achieve a fair election, obeying the majority criterion (Strøm et al., 1990), that is, when the winning candidate obtains more than half the votes of the election. However, elections involving more than two candidates require a more robust voting system. With this in mind, other two voting systems developed to handle the case of elections with more than two candidates are *Preferential Voting* (Karvonen, 2004) and *Borda Count* (Emerson, 2013). In *Preferential Voting* voting system, voters vote in an ordered list from the most preferred to the least preferred candidate. The elected candidate is the one most often chosen as the most preferred by voters.

Borda Count is also a voting system in which voters draw up a list of candidates arranged according to preference. Each position in the user's preference list is assigned a score. In a list of n candidates, the candidate in the i position on the list receives the score $n - i$. To determine the winner, the sum of each candidate's scores for each voter is the final score, and the candidate with the highest score is the elected one.

Position / Votes	14	10	8	4	1
1st choice	A	C	D	B	C
2nd choice	B	B	C	D	D
3rd choice	C	D	B	C	B
4th choice	D	A	A	A	A

Table 1. Candidate votes example.

An example of how *Borda Count* works can be given with the help of Table 1, in which there are four candidates: A, B, C and D. In Table 1 the columns represent the number of votes received by the candidates, and the lines represent the preference positions occupied by each candidate. As there are four candidates, the candidate preferred by a voter receives three points. The calculation of the score for the candidate D is performed as follows:

- 8 voters elected the candidate D as the preferred candidate; $8 * 3 = 24$ points
- $4 + 1$ voters elected the candidate D as the second most preferred candidate; $5 * 2 = 10$ points
- 10 voters elected the candidate D as the third most preferred candidate; $10 * 1 = 10$ points
- 14 voters elected the candidate D as the least preferred; $14 * 0 = 0$ points
- **Candidate *D* total score** $= 24 + 10 + 10 + 0 = 44$

Voting algorithms were also used together with recommender systems previously. In the Collaborative Filtering recommender systems, after recovering the users who have the profiles most similar to another user, it is necessary to choose which items these users liked best to make a good recommendation. At this point in choosing the items to recommend, given the various opinions, voting algorithms' use becomes an interesting tool. Rani et al. (2017) proposed a recommendation algorithm based on clustering and Voting Theory that can be applied in different domains. The algorithm consists of extracting clusters with *KMeans* (Kanungo et al., 2002) and using the cluster to which a recommendation target user belongs to perform Collaborative Filtering. The $KMeansPlus^{Log\ power}$ variant is used to lessen the impact of the

random choice of centroids on the cluster-based recommendation. After clustering and selecting the target user's cluster, the *Borda Count* is used to select the most popular items in the cluster and use them in the recommendation. Making a performance comparison between the *Borda Count* and the *Copeland Score* Al-Sharrah (2010) in the recommendation for Collaborative Filtering, the Lestari et al. (2018) method makes a clustering based on the age demographic data and alternates the two techniques in making the recommendation. Still using the *Borda Count*, Tang and Tong (2016) proposes the *BordaRank*. The method consists of using the *Borda Count* method directly in the sparse matrix of evaluations, without predictions, to make a recommendation.

FREEP - FEATURE RECOMMENDER FROM PREFERENCES

To address the workflow parameter recommendation problem , we propose *FReeP, Feature Recommender from Preferences* approach. FReeP suggests parameter values that maximize a probability to make the workflow run flawlessly until its end. As input, the approach receives a user preferences set, which makes personalized recommendations. Machine Learning and Preference Learning techniques are the basis of the recommendation approach.

The recommendation approach is presented in three versions. In the first two versions, the algorithm aims at recommending a value for only one parameter at a time. While the first version assumes that all parameters have a discrete domain, the second version is an extension of the first, dealing with cases where a parameter presents a continuous domain, in addition to other improvements regarding the first version. Meanwhile, the third version is a proposal for the recommendation of values for n parameters at a time.

FReeP algorithm can be seen as a hybrid Recommendation technique since Collaborative Filtering, and Content-Based concepts are used. We start by presenting the first version of the algorithm, which makes the recommendation for a single parameter at a time and helps to evaluate the hypothesis. Then, we follow to the improved version with some variants to decrease performance problems. Finally, a generic version of the algorithm is presented, aiming at making the recommendation of values for multiple (n) parameters at a time.

Discrete Domain Parameter Value Recommendation

Given a provenance database D, a parameter $y \in Y$, where Y is the workflow parameters set, and a preferences or restrictions set P defined by the user, where $p_i \in P(y_i, val_k)$, the first *FReeP* approach described here aims to solve the problem of recommending a r value for y, so that the P preferences together with the r recommendation to y maximize the chance of workflow activation to run to the end.

Figure 5 presents an architecture overview of *FReeP*'s first version. The algorithm receives as input the provenance database, a target workflow and user preferences. User preferences are also input data as this work is based on the premise that the user already has a subset of parameters for which has already defined values to use. In this version of the proposed approach, user preferences are only allowed in the form $a = b$, where a is a parameter, and b is the parameter desired value.

It is important to note that, based on the user's preferences, it would be possible to query the provenance database from which the experiment came from to retrieve records that could assist in the search for other parameters values that had no preferences defined. However, FReeP is based on a model generation that generalizes the provenance database, removes the user's need to perform this query and can still provide results that the simple data query would not be able to return.

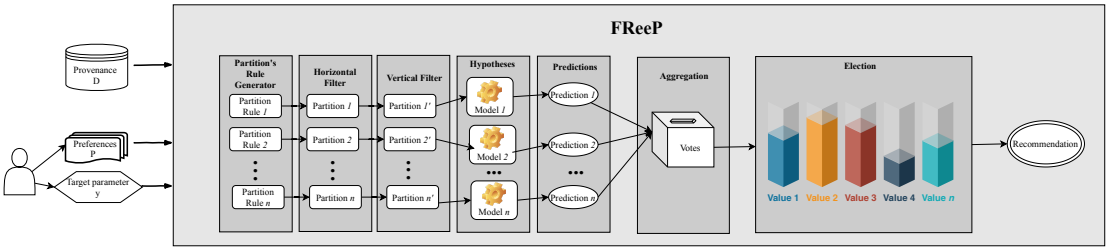


Figure 5. *FReeP* Architecture Overview.

To obtain a recommendation from *FReeP*'s first version, seven steps are required: **partitions generation**, **horizontal filter**, **vertical filter**, **hypothesis generation**, **predictions**, **aggregation** and finally a **election-based recommendation** is performed. Algorithm 1 shows the proposed algorithm to perform the parameter recommendation, considering the preferences for a subset or all other workflow execution parameters.

Algorithm 1 First Version of FReeP

Require:

y : recommendation target parameter

$P : \{(param, val) \mid param$ is a workflow parameter, val is the preference value for $param\}$

$D : \{\{(param_1^1, val_1^1), ..., (param_1^l, val_1^l), ...(param_l^m, val_l^m)\} \mid l$ is the workflow parameters number, m is the provenance dataset length$\}$

1: **procedure** FREEP(y, P, D)
2: $partitions \leftarrow partitions\_generation(P, D)$
3: $votes \leftarrow \emptyset$
4: **for each** $partition \in partitions$ **do**
5: $data \leftarrow horizontal\_filter(D, partition)$
6: $data \leftarrow vertical\_filter(data, partition)$
7: $data' \leftarrow preprocessing(data)$
8: $model \leftarrow hypothesis\_generation(data', y)$
9: $vote \leftarrow recommend(model, y)$
10: $votes \leftarrow votes \cup \{vote\}$
11: $recommendation \leftarrow elect\_recommendation(votes)$
12: **return** $recommendation$

The algorithm input data are:

- Target parameter for which the algorithm should make the recommendation, y
- User preferences set, such as a list of key-values, where the key is a workflow parameter and value is the user's preference for that parameter, P
- Provenance database, D

It is important to note that the storage of provenance data for an experiment may vary from one WfMS to another. For example, SciCumulus, which uses a provenance representation derived from *PROV*, stores provenance in a relational database. Using SciCumulus example, it is trivial for the user responsible for the experiment to elaborate a SQL query that returns the provenance data related to the parameters used in each activity in a key-value representation. The key-value representation can be easily stored in a *csv* format file, which is the required format expected as provenance dataset in *FReeP* implementation. Thus, convert provenance data to the *csv* format is up to the user. Still, regarding the provenance data, the records present in the algorithm input data containing information about the parameters must be related only to executions that were successfully concluded, that is, there was no failure that resulted in the execution abortion. It should be noted that the inclusion of components, such as the conversion of provenance and successful executions parameters selection, in the algorithm, would require implementations for each type of WfMS, which is out of the scope of this article.

The initial step, **partitions_generation**, builds partitioning rules set based on the user's preferences. Initially, the preference set parameters P are used to generate a *powerset*. This first step returns all generated *powerset* as a *partitions* ruleset. Figure 6 shows an example of how this first step works, with some parameters from *SciPhy* workflow.

Then, FReeP initializes an iteration over the partitioning rules generated by the previous step. Iteration begins selecting only the records that follow the user's preferences contained in the current ruleset, named in the algorithm as **horizontal_filter**. Figure 7 uses the *partitions* presented in Figure 6 to show how the **horizontal_filter** step works.

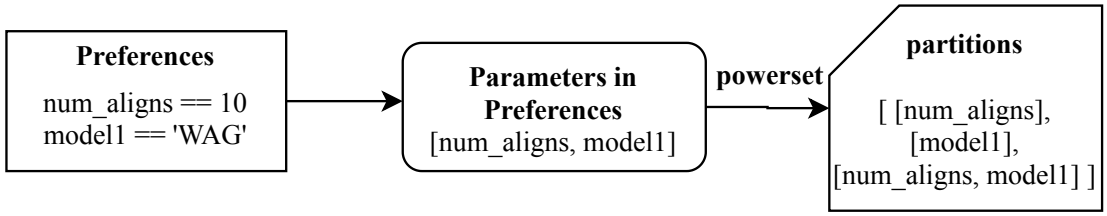


Figure 6. Example of *FReeP*'s Partitioning Rules Generation for *Sciphy* provenance dataset using user's preferences.

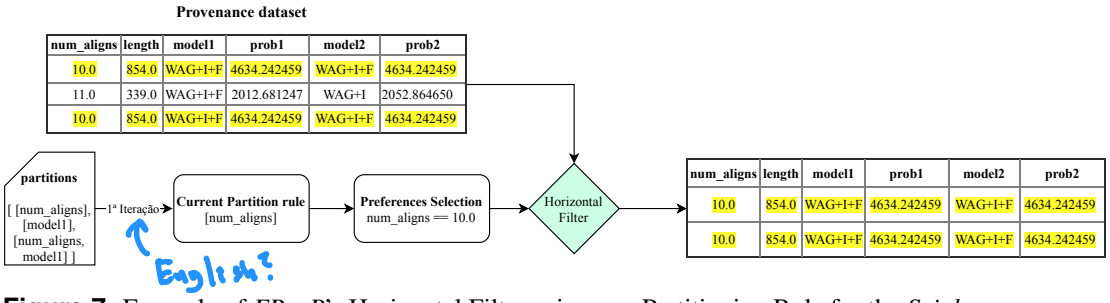


Figure 7. Example of *FReeP*'s Horizontal Filter using one Partitioning Rule for the *Sciphy* provenance dataset.

Subsequently, in the **vertical_filter** step, there is a parameter removal that aims to keep only the recommendation target parameter, the parameters present in the current set of partitioning rules, and those that are not the recommendation target parameter nor are present in any of the original user preferences. The last parameters mentioned remain because, in a next step, they can help to build a more consistent hypothesis. Formalizing: let PW be all workflow parameters set, PP the workflow parameters for which preference values have been defined; PA the parameters present in the partitioning rules of an iteration over the partitioning rules and $PV = (PW - PP) \cup (PP \cap PA) \cup \{y\}$; the output from **vertical_filter** is the data from **horizontal_filter** for parameters in PV. Figure 8 uses data from the examples in Figures 6 and 7 to show how the **vertical_filter** step works.

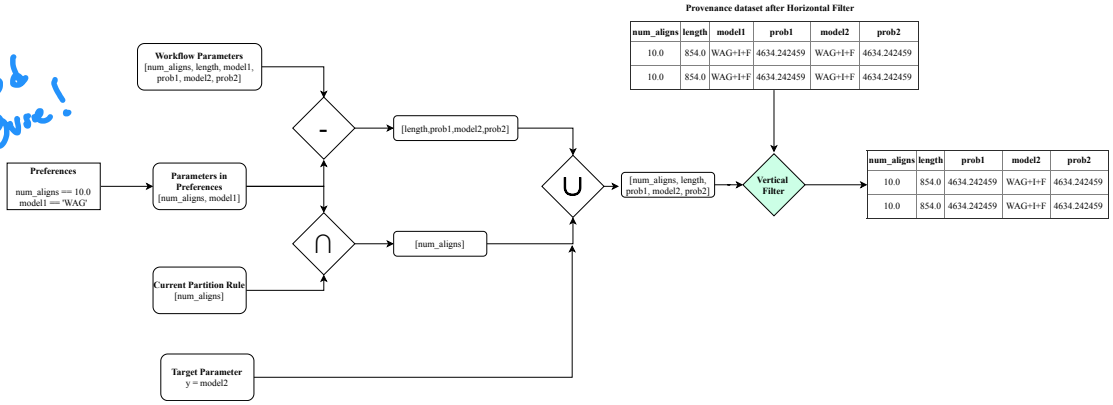


Figure 8. *FReeP*'s Vertical Filter step.

The chain comprising the partitions generation and the horizontal and vertical filters is crucial to minimize the *Cold Start* problem (Lika et al., 2014). *Cold Start* problem is caused by the lack of ideal operating conditions for an algorithm, specifically in the recommender systems. This problem occurs, for example, when there are few users for the neighborhood definition with a similar user profile or lack of ratings for enough items. *FReeP* can also be affected by *Cold Start* problem. If only all preferences were used at one time for partitioning the provenance data, in some cases, it could be observed that the resulting partition would be empty. This is because there could be an absence of any of the user's

preferences in the provenance data. Therefore, generating multiple partitions with subsets of preferences decreases the chance of obtaining only empty partitions. However, in the worst case where none of the user's preferences are present in the workflow provenance, *FReeP* will not perform properly, thus failing to make any recommendations.

After the partitions generation and horizontal and vertical filters are discovered, there is a filtered data set that follows part of the user's preferences. These provenance data that will generate the hypotheses, represented by Machine Learning models, have numerical and categorical domain parameters. However, traditional Machine Learning models generally work with numerical data because the generation of these models, in most cases, involves many numerical calculations. Therefore, it is necessary to codify these categorical parameters to numerical representation. The technique used here to encode categorical domain parameters to numerical representation is *One-Hot encoding* (Coates and Ng, 2011). This technique consists of creating a new binary attribute, that is, the domain of this new attribute is 0 or 1, for each different attribute value present in dataset.

The encoded provenance data allows building Machine Learning models to make predictions for the target parameter under the step **hypothesis_generation**. The hypothesis generated has parameter y as class variable, and the other parameters present in **vertical_filter** step output data are the attributes used in generalizing the hypothesis. In this algorithm approach, the model representing the generated hypothesis can be a classifier, where the model's prediction is a single recommendation value, or a *ranker*, where its prediction is an ordered list of values, of the value most suitable for the recommendation to the least suitable.

With a hypothesis created, we can use it to recommend the value for the target parameter. This step is represented in FReeP as **recommended**, and the recommendation of parameter y is made from the user's preferences. It is important to emphasize that the model's training data may contain parameters that the user did not specify any preference. In this case, an attribute of the instance submitted for the hypothesis does not have a defined value. To clarify the problem: let PW be all workflow parameters set, PP the parameters of workflow for which preference values have been defined; PA the parameters present in the partition rules of an iteration over the partitioning rules; and $PV = (PW - PP) \cup (PP \cap PA) \cup \{y\}$, there may be parameters $p \in PV \mid p \notin PP$, and for those parameters p there are no values defined *a priori*. To handle this problem, the average values present in the provenance data are used to fill in the numerical attributes' values and the most frequent values in the provenance date for the categorical attributes.

All predictions generated by **recommend** step, which is within the iteration over the partitioning rules, are stored. The last algorithm step, **elect_recommendation**, uses all of these predictions as votes, which will define which value should be recommended for the target parameter. When an algorithm instance is setup to return a *classifier* type model in **hypothesis_generation** step, the election takes place through the majority strategy most frequent value, that is, the most voted is elected as the recommendation. On the other hand, when an algorithm instance is setup to return a *ranker* type model in **hypothesis_generation** step, the strategy is *Borda Count*. The use of the *Borda Count* strategy seeks to take advantage of the list of lists form that the saved votes acquire when using the *ranker* model. This list of lists format occurs because the *ranker* prediction is a list, and since there are as many predictions as partitioning rules, the storage of these predictions takes the list of lists format.

Discrete and Continuous Domain Parameter Value Recommendation

The first version of *FReeP* allowed evaluating the algorithm's proposal. The proposal showed relevant results after initial tests (presented in next section), so efforts were focused on improving its performance and utility. In particular, the following problems have been identified:

1. User has some restriction to set his/her parameters preferences;
2. The categorical domain parameters when used as a class variable;
3. Machine Learning models used in FReeP to represent the hypotheses created, *classifier* and *ranker* are used only for categorical class parameters;
4. All partitions generated by workflow parameters *powerset* present in user preferences are used as partitioning rules for the algorithm.

Regarding problem 1, in Algorithm 1, the user was limited to define his preferences with the equality operator. Depending on the user's preferences, the equality operator is not enough. With this in mind, the

second version of FReeP allows for the user to have access to the relational operators: $==, >, >=, <, <=$ and $!=$ to define his/her preferences. In addition, two logical operators are also supported in setting preferences: $|$ and &. Preferences with combination of supported operators is also allowed, for example: $(a > 10)|(at < 5)$.

However, by allowing users to define their preferences in this way we create a problem when setting up the instances for **recommendation** step. As seen, PW represents all workflow parameters set , PP are workflow parameters that preference values have been set; PA the parameters present in the partitioning rules of an iteration over the partition rules; and $PV = (PW - PP) \cup (PP \cap PA) \cup \{y\}$. Thus, there may be parameters $p \in PV \mid p \notin PP$, and for those parameters p, there are no values defined *a priori*. As in this second algorithm version, since the user's preferences can be expressed in a more relaxed way, and it is necessary to create the instances used in the step **recommendation** that include a range (or set of values). To handle these cases, all possible instances from preference values combinations were generated. In case the preference is related to a numerical domain parameter and is defined in terms of values range, like $a \leq 10.5$, FReeP uses all values present in the source provenance database that follows the preference restriction. It is important to note that for both numerical and categorical parameters, the combination of possible values are those present in the provenance database and respect the user's preferences. Then, predictions are made for a set of instances using the hypothesis learned during the training phase.

Difference between versions could be communicated better.

Regarding problem 2, the provenance database, which is one algorithm input data, in general, present attributes with numerical and categorical domains. As before, it is FReeP converts categorical values to numerical ones.

This pre-processing step was included in Algorithm 2 as **preprocessing_classes** step. The preprocessing consists in exchange each distinct categorical value for a distinct integer. Note that the encoding of the parameter used as a class variable in the model generation is different from the encoding applied to the parameters used as attributes represented by the step **preprocessing**.

Concerning problem 3, using Machine Learning models developed for the classification task for continuous numerical domain class variables degrades the performance results. This problem happens when *classifier* and *ranker* are used in the first version of FReeP. Performance degradation is because the numerical class variables are considered as categorical. For continuous numerical domain class variables, the Machine Learning models suggested are *Regressors* (Myers and Myers, 1990). In this way, the second version of *FReeP* checks the parameter y domain, which is the recommendation target parameter, represented as **model_select** step in Algorithm 2.

To analyze problem 4, it is important to note that after converting categorical attributes *One-Hot encoding* in **preprocessing** step, the provenance database will have a considerable increase in the number of attributes. Also, after categorical attributes encoding in **preprocessing** step, the parameters extracted from the user's preferences, are also encoded for **partitions_generation** step. In Algorithm 1, the partitioning rules *powerset* is calculated on all attributes derived from the original parameters after *One-Hot encoding*. If FReeP uses the *powerset* generated from the parameters present in the user's preferences set as partitioning rules (in the **partitions_generation** step), it can be very costly. Thus, using the *powerset* makes the complexity of the algorithm becomes exponential according to the parameters present in the user's preferences set. Alternatives to select the best partitioning rules and handle the exponential cost are represented in Algorithm 2 as **optimized_partitions_generation** step. The two strategies proposed were based on *Principal Components Analysis (PCA)* (Garthwaite et al., 2002) and the *Analysis of variance (ANOVA)* (Girden, 1992) statistical metric.

The strategy based on the *PCA* consists of extracting x principal components from all provenance database, pca_D, and for each $pt \in partitions$, pca^i_{pt}, which are pt partition principal components. Then, the norms are calculated $\left\| pca_D - pca^i_{pt} \right\|$, and from that are selected n partitioning rules that generated pca^i_{pt} such that $\left\| pca_D - pca^i_{pt} \right\|$ resulted in the lowest calculated values. Note that both x and n are defined parameters when executing the algorithm. In summary, the *PCA* strategy will select the partitions where the main components extracted are the closest to the principal components of the original provenance dataset.

ANOVA strategy seeks the n partitioning rules that best represent D, selecting those that generate partitions where the data variance is closest to D data variance. In short, original data variance and data variance for each partition are calculated using the *ANOVA* metric, then partitions with most similar variance to the original provenance data are selected. Here, the n rules are defined in terms of the data

percentage required to represent the entire data set, and that parameter must also be defined in algorithm execution. Using *PCA* or *ANOVA* partitioning strategies means that the partitioning rules used by FReeP can be reduced, depending on the associated parameters that need to be defined.

Algorithm 2 Second Version of FReeP

Require:

y : recommendation target parameter

P : $\{(param, val) \mid param$ is a workflow parameter, val is the preference value for $param\}$

D : $\{\{(param_1^1, val_1^1), ..., (param_1^l, val_1^l), ...(param_l^m, val_l^m)\} \mid l$ is the workflow parameters number, m is the provenance dataset length$\}$

1: **procedure** FREEP(y, P, D)
2: $D' \leftarrow classes\_preprocessing(D)$
3: $partitions \leftarrow optimized\_partitions\_generation(P, D')$
4: $votes \leftarrow \emptyset$
5: **for each** $partition \in partitions$ **do**
6: $data \leftarrow horizontal\_filter(D', partition)$
7: $data \leftarrow vertical\_filter(data, partition)$
8: $model\_type \leftarrow model\_select(data, y)$
9: $data' \leftarrow preprocessing(data)$
10: $model \leftarrow hypothesis\_generation(data', y, model\_type)$
11: $vote \leftarrow recommend(model, y)$
12: $votes \leftarrow votes \cup \{vote\}$
13: $recommendation \leftarrow elect\_recommendation(votes)$
14: **return** *recommendation*

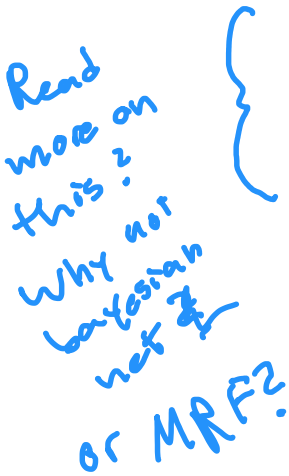
Recommendation for n Parameters at a time

Algorithms 1 and 2 aims at producing single parameter recommendation at a time. However, in a real usage scenario of scientific workflows, the WfMS will probably need to recommend more than one parameter at a time. A naive alternative to handle this problem is to execute Algorithm 2 for each of the target parameters, always adding the last recommendation to the user's preference set. This alternative assumes that the parameters to be recommended are independent random variables. One way to implement this strategy is to use a *classifiers chain* (Read et al., 2011).

Nevertheless, this naive approach neglects that the order in which the target parameters are used during algorithm interactions can influence the produced recommendations. The influence is due to parameter dependencies that can be found between two (or more) workflow activities (e.g., two activities consume a parameter produced by a third activity of the workflow). In Figure 9, the circles represent the activities of workflow, so it can be seen that activities 2 and 3 are preceded by activity 1 (e.g., they consume the output of activity 1). Using this example, we can see that it is possible that there is a dependency relationship between the parameters *param2* and *param3* with the parameter *param1*. In this case, the values of *param2* and *param3* parameters can be influenced by parameter *param1* value.

In order to deal with this problem, FReeP leverages the *Classifiers Chains Set* (Read et al., 2011) concept. This technique allows for estimating the joint probability distribution of random variables based on a *Classifiers Chains Set*. In this case, random variables are the parameters for which values are to be recommended, and the joint probability distribution concerns the possible dependencies between these parameters. The *Classifiers Chains* and *Classifiers Chains Set* are techniques from *Multi-label Classification* (Tsoumakas and Katakis, 2007) Machine Learning task.

An architecture overview for the proposed algorithm named as *Generic FReeP* that recommends n parameters simultaneously is shown in Figure 10. The architecture presented in Figure 10 shows that the solution developed to make n parameter recommendations at a time is a packaging of *FReeP* algorithm to one parameter. This final approach is divided into five steps: identification of parameters for the recommendation, generation of ordered sequences of these parameters, iteration over each of the sequences generated with the addition of each recommendation from *FReeP* to the user preferences set, separation of recommendations by parameter and finally the choice of value recommendation for each target parameters. The formalization can be seen in Algorithm 3.



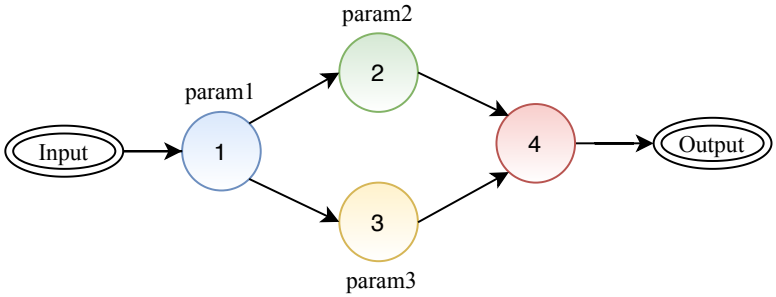


Figure 9. A generic workflow representation: circles represent activities, arrows between the circles represent the link between activities, and the labels for each circle represent the configuration parameters for each activity.

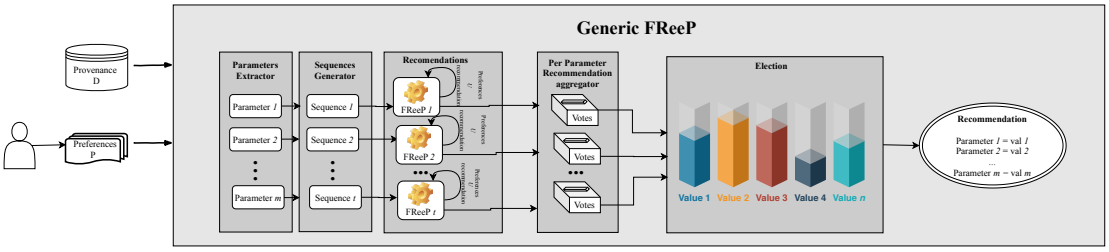


Figure 10. *Generic FReeP* architecture overview

Algorithm 3 Generic FReeP

Require:

P : $\{(param, val) \mid$ $param$ is a workflow parameter, val is the preference value for $param\}$

D : $\{\{(param_1^1, val_1^1), ..., (param_1^l, val_1^l), ...(param_l^m, val_l^m)\} \mid$ l is the workflow parameters number, m is the provenance dataset length$\}$

N :*number of random sequences orders to be generated*

procedure GENERIC FREEP(P, D, N)
  $target\_parameters \leftarrow parameters\_extractor(P, D)$
  $votes \leftarrow \emptyset$
  for each $param \in target\_parameters$ **do**
    $votes \leftarrow votes \cup \{(param, [])\}$
  $ordered\_sequences \leftarrow sequence\_generator(target\_parameters, N)$
  for each $sequence \in ordered\_sequences$ **do**
    $preferences\_tmp \leftarrow P$
    for each $param \in sequence$ **do**
      $recommendation \leftarrow FReeP(param, preferences\_tmp, D)$
      $votes[param] \leftarrow votes[param] \cup recommendation$
      $new\_preference \leftarrow generate\_preference(param, recommendation)$
      $preferences\_tmp \leftarrow preferences\_tmp \cup new\_preference$
  $response \leftarrow \emptyset$
  for each $(param, values) \in votes$ **do**
    $response[param] \leftarrow most\_voted(values)$
  return $response$

The first step **parameters_extractor** extracts the workflow parameters that are not present in the users' preferences and will be the targets of the recommendations. Thus, all other parameters that are not in the user's preferences will have recommendation values.

Some sense of runtime/ complexity?

Lines 4 and 5 of the algorithm comprise the initialization of the variable responsible for storing the different recommendations for each parameter during the algorithm execution. Then, the list of all parameters that will be recommended is used for generating different ordering of these parameters, indicated by **sequence_generators** step. For example, w be a workflow with 4 p parameters and let u be an user with pr_1 and pr_3 preferences for the p_1 and p_3 parameters respectively. The parameters to be recommended are p_2 and p_4, in this case two possible orderings are: $\{p_2, p_4\}$ and $\{p_4, p_2\}$. Note that the number of sorts used in the algorithm are not all possible sorts, in fact N of the possible sorts are selected at random.

Then, the algorithm initializes an iteration over each of the sorts generated by the step **sequence_generators**. Another nested iteration over each parameter present in the current order also begins. An intuitive explanation of the algorithm between lines 9 and 13 is that each current sequence parameter is used together with the user's preferences for its recommendation. At the end of the recommendation of one of the ordering parameters, the recommendation is incorporated into the preferences set used in the recommendation of the next ordering parameter. In this iteration, the recommendations are grouped by parameter to facilitate the election of the recommended value for each target parameter.

The step of iterating over the generated sequences, always adding the last recommendation to the set of preferences, is the *Classifiers Chains* concept. To deal with the dependency between the workflow parameters that can influence a parameter value recommendation, the step that generates multiple sequences of parameters, combined with the *Classifiers Chains*, is the *Classifiers Chains Set* concept.

Finally, to choose the recommendation for each target parameter, a vote is taken on lines 15 and 16. The **most_voted** procedure makes the majority election that defines the target parameter recommendation value. This section presented three algorithms that are part of the FReep approach developed for the parameter recommendation problem in workflows. The proposals covered two main scenarios for parameters value recommendation (single and multiple parameter at a time).

EXPERIMENTAL EVALUATION

All *FreeP* algorithms presented in this article were implemented using the *Python* programming language. Some of the most used *Python* libraries to develop Machine Learning tools were used, for example *Scikit-Learn* (Pedregosa et al., 2011), which provides several classifiers and regressors, in addition to tools that assist in the proposal evaluation, such as *K Fold Cross Validation* (Arlot et al., 2010); *numpy* (Walt et al., 2011), a numerical data manipulation library; and *pandas* (McKinney, 2011), which provides tabular data functionalities.

To measure recommendations performance when the parameter is categorical, *precision* and *recall* are used as metrics. *Precision* and *recall* are metrics widely used for the quantitative assessment of recommender systems (Herlocker et al., 2004) (Schein et al., 2002). Equation 5 defines *precision* and Equation 6 defines *recall*, following the recommender vocabulary, where TR is the correct recommendation set and R is all recommendations set. An intuitive explanation to *precision* is that it represents the most appropriate recommendations fraction. Still, *recall* represents the appropriate recommendation fraction that was made.

$$precision = \frac{\|TR \cap R\|}{\|R\|} \quad (5) \qquad recall = \frac{\|TR \cap R\|}{\|TR\|} \quad (6) \qquad MSE = \frac{1}{n}\sum_{i=1}^{n}(RV - TV)^2 \quad (7)$$

When the parameter to be recommended is numerical, the performance of FReeP is evaluated with *Mean Square Error (MSE)*. The MSE formula is given by Equation 7 where n is the recommendations number, TV is the correct recommendation values set, and RV is the recommended values set.

Dataset

The datasets used are provenance data extracted from past executions of the workflows *SciPhy* (Ocaña et al., 2011) and *Montage* (Hoffa et al., 2008). *SciPhy* is a *Bioinformatics* workflow aiming at generating phylogenetic trees, that is, trees that represent an organism evolutionary history. *Montage* is a well-known astronomical workflow that generates mosaics from several sky images.

Table 2 summarizes the main characteristics of the datasets. The *Total Records* column shows the amount of data from past executions of each workflow. Each provenance dataset record can be used as an example for generating Machine Learning models during the algorithm's execution. As seen, the *Sciphy* dataset is relatively small compared to *Montage*. The column *Total Attributes* shows how many activity

Dataset	Total Records	Total Attributes	Categorical Attributes	Numerical Attributes
Sciphy	376	6	2	4
Montage	1565	8	2	6

Table 2. Dataset characteristics.

parameters are present in each workflow execution. Both *workflows* have the same number of categorical domain parameters, as can be attested by the column *Categorical Attributes*. *Montage* has more numeric domain parameters than *Sciphy*, as shown in the *Numerical Attributes* column.

Parameter	Minimum Value	Maximum Value	Standard Deviation
num_aligns	9.000000	11.000000	0.209135
length	85.000000	1039.000000	169.904263
prob1	634.673994	5753.519623	1103.431812
prob2	635.874188	5795.280758	1101.762483

Table 3. *Sciphy* dataset statistics.

Statistics on the *Sciphy* dataset numerical attributes are shown in Table 3. This table presents the minimum and maximum values of each attribute, in addition to the standard deviation. As can be seen, the attribute *prob1* has the highest standard deviation, and its range of values is the largest among all attributes. The *prob2* attribute has both a range of values and the standard deviation similar to *prob1*. The standard deviation of the values of *num_aligns* is very small, while the attribute *length* has a high standard deviation, considering its values range.

Parameter	Minimum Value	Maximum Value	Standard Deviation
cntr	0.000000	134.000000	35.335662
ra	83.118935	323.898836	91.131423
dec	-27.166501	28.845708	17.903018
crval1	83.118785	323.898697	91.131426
crval2	-27.166362	28.845847	17.903018
crota2	0.000084	359.999884	178.643719

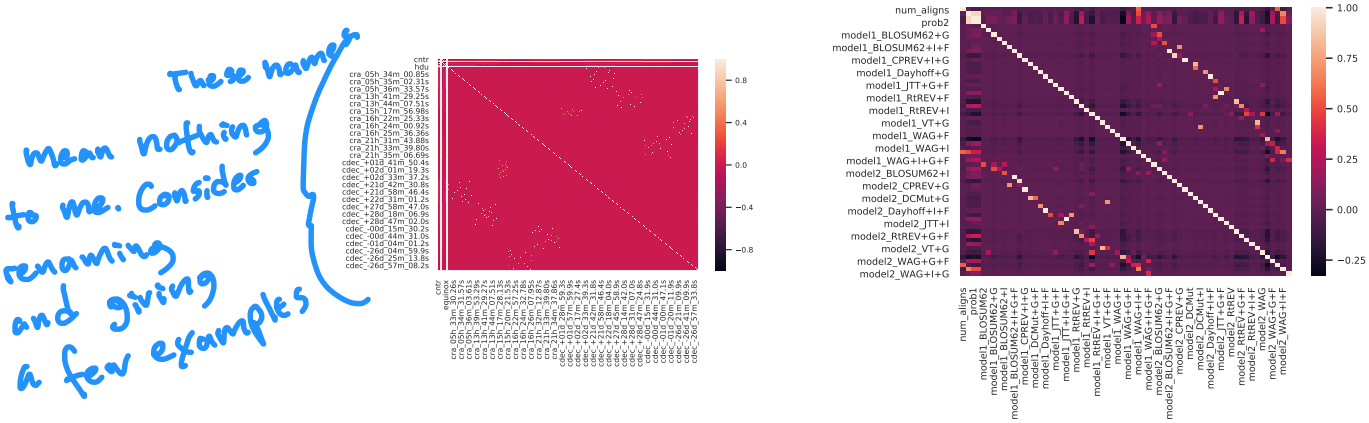
Table 4. *Montage* dataset statistics.

The *Montage* dataset numerical attributes, in most of the cases, have smaller standard deviation than the *Sciphy* dataset. On average, *Montage* attributes also have a smaller values range than *Sciphy* dataset attributes. Also, in *Montage* dataset, the *crota2* attribute has the largest values range and the largest standard deviation. The *dec* and *crval2* attributes have close statistics and are the attributes with the smallest data range and the smallest *Montage* data standard deviation.

In Figure 11, it is possible to check the correlation between the different attributes in the datasets. It is notable in both Figure 11a and Figure 11b that the attributes (i.e., workflow parameters) present a weak correlation. All those statistics are relevant to understand the results obtained by the experiments performed from each version of FReeP algorithm.

Discrete Domain Recommendation Evaluation

This experiment was modeled to evaluate *FReeP*'s algorithm key concepts using the first version presented in Algorithm 1, that was developed to recommend one discrete domain parameter at a time. This experiment aims at evaluating and comparing the performance of FReeP when its *hypothesis_generation* step instantiates either a single *classifier* or a *ranker*. The *ranker* tested as a model was implemented using the *Pairwise Label Ranking* technique. *K Nearest Neighbors* (Keller et al., 1985) classifier is used



(a) *Montage* dataset attributes Correlation Matrix.

(b) *Sciphy* dataset attributes Correlation Matrix.

Figure 11. Datasets Attributes Correlation matrices.

as the classifier of this *ranker* implementation. The k parameter of *K Nearest Neighbors* classifier was set as $3, 5, 7$ for both the *ranker* and *classifier*. The choice of $k \in \{3, 5, 7\}$ is because small datasets are used, and thus k values greater than 7 do not return any neighbors in the experiments.

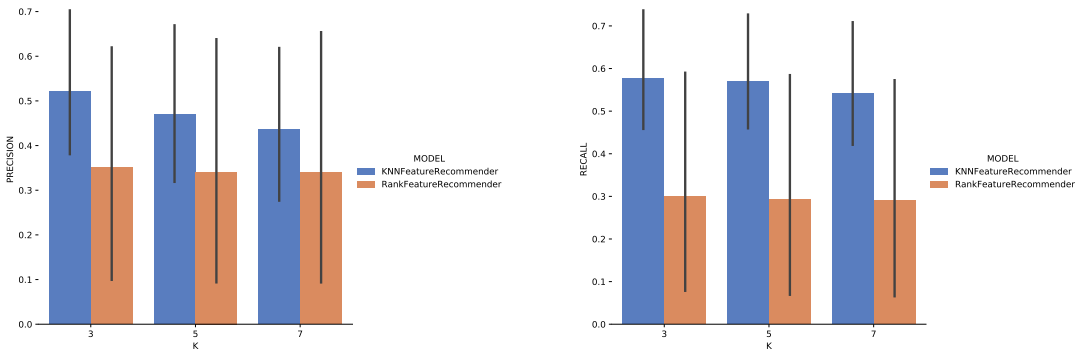
Experiment 1. Algorithm 1 Evaluation Script.

1. The algorithm is instantiated with the *classifier* or *ranker* and a recommendation target workflow parameter.
2. The provenance database is divided using *K-Fold Cross Validation* (Kohavi, 2001), which consists of dividing the data set into k parts, and at each step $k-1$ parts for model training and 1 parts for model prediction. In this experiment was used $k = 5$.
3. Each workflow parameter is used as recommendation target parameter.
4. Each provenance record in test data is used to retrieve target parameter real value.
5. Parameters that are not the recommendation target are used as preferences, with values from current test record.
6. Then, algorithm performs recommendation and both the result and the value present in the test record for the recommendation target parameter are stored.
7. *Precision* and *recall* values are calculated based on all *K-Fold Cross Validation* iterations.

Results

Experiment 1 results are presented and analyzed based on the values of *precision* and *recall*, in addition to the execution time. Figure 12a shows that Algorithm 1 execution with *Sciphy* provenance database, using both the *classifier* and the *ranker*. Only KNN classifier with $k = 3$ gives a precision greater than 50%. Also, a high standard deviation is noticed. Even with unsatisfactory performance, Figure 14b shows that *KNN* classifier presented better *recall* results than those for *precision*, both in absolute values terms and standard deviation, which had a slight decrease. In contrast, the *ranker recall* was even worse with the *precision* results and still present a very high standard deviation.

Figure 13 shows the execution time spent, in seconds, to obtain the experiment's recommendations for SciPhy. The execution time of *ranker* is much more significant when compared to the time spent by the *classifier*. This behavior can be explained by the fact that the technique used to generate the *ranker* creates multiple *binary classifiers*. Another point to note is that the execution time standard deviation



(a) Precision results with *Sciphy* data.

(b) Recall results with *Sciphy* data.

Figure 12. precision and Recall results with *Sciphy* data.

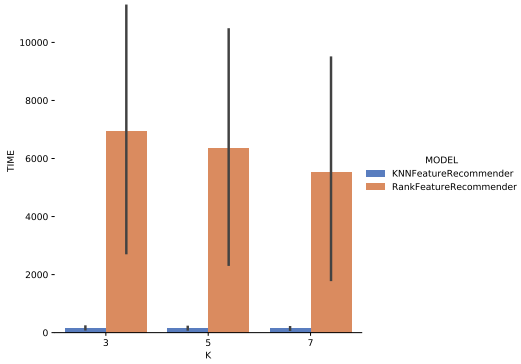


Figure 13. Experiment recommendation execution time with *Sciphy* data.

from *ranker* is also very high. It is important to note that when *FReeP* uses KNN, it is memory-based, since each recommendation needs to be loaded into main memory.

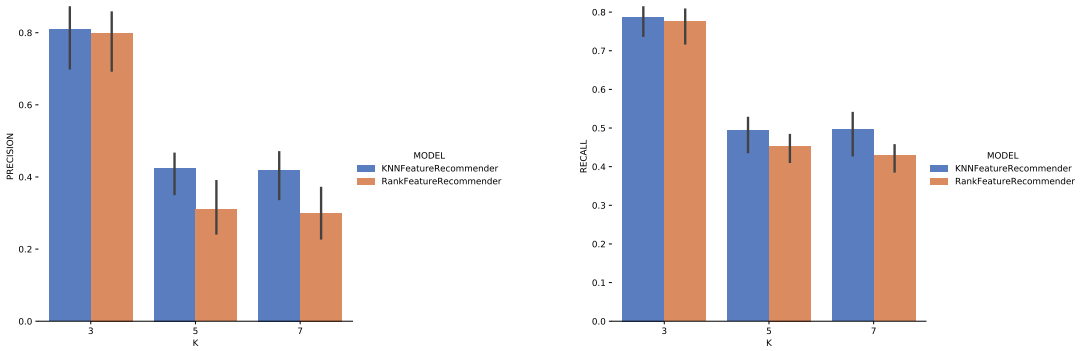
Analyzing Figure 14a (Montage) one can conclude that with the use of $k = 3$ for the classifier and for the *ranker* produces relevant results. The *precision* for this case reached 80%, and the standard deviation was considerably smaller compared to the *precision* results with *Sciphy* dataset in Figure 12a. For $k \in \{5,7\}$, the same results behavior was observed, considerably below those expected.

Considering the precision, Figure 14b shows that the results for $k = 3$ were the best for both the classifier and for *ranker*, although for this case they did not reach 80% (although it is close). It can be noted that the standard deviation was smaller when compared to the standard deviations found for *precision*. One interesting point about the execution time of the experiment with *Montage* presented in Figure 15 is that for $k \in \{3,7\}$ the *ranker* spent less time than the *classifier*. This behavior can be explained because the *ranker*, despite being generated by a process where several classifiers are built, relies on binary classifiers. When used alone, the classifier needs to handle all class variables values, in this case, parameter recommendation values, at once. However, it is also important to note that the standard deviation for *ranker* is much higher than for the *classifier*.

In general, it was possible to notice that the use of *ranker* did not bring encouraging results. In all cases, *ranker precision* and *recall* were lower than those presented by the *classifier*. Besides, the standard deviation of *ranker* in the execution time spent results was also very high. Another point to be noted is that the best *precision* and *recall* results were obtained with the data from *Montage workflow*. These results may be linked to the fact that the *Montage* dataset has more records than the *Sciphy* dataset.

Discrete and Continuous Domain Recommendation Evaluation

Experiment 1 was modified to evaluate the Algorithm 2 performance, yielding Experiment 2. Algorithm 2 was executed with variations in the choice of classifiers and regressors, partitions strategies, and records



(a) Precision results with *Montage* data.

(b) Recall results with *Montage* data.

Figure 14. precision and Recall results with *Montage* data.

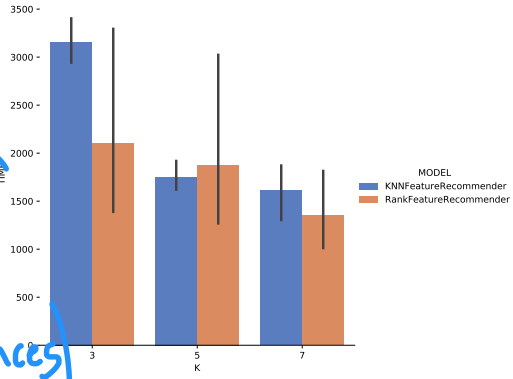


Figure 15. Experiment recommendation execution time with *Montage* data.

percentage from provenance database. All values per algorithm parameter are presented in Table 5.

Classifiers	Regressors	Partition Strategy	Percentage
KNN	Linear Regression	PCA	30
SVM	KNR	ANOVA	50
Multi-Layer Perceptron	SVR		70
	Multi-Layer Perceptron		

Table 5. Algorithm 2 values per parameter used in Experiment 2

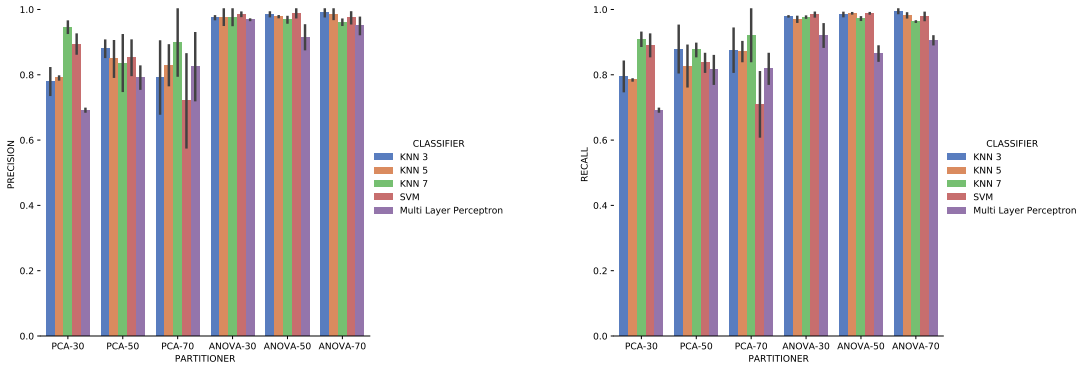
Results

Experiment 2 results are presented using *precision*, *recall*, and execution time for categorical domain parameters recommendations, while numerical domain parameters recommendations are evaluated using *MSE* and the execution time. Based on the results obtained in Experiment 1, only classifiers were used as Machine Learning models in Experiment 2, *i.e.*, we do not consider rankers.

The first observation when analyzing the *precision* data in Figure 16a is that *ANOVA* partitioning strategy obtained better results than *PCA*. *ANOVA* partitioning strategy *precision* in absolute values is generally more significant, and variation in *precision* for each attribute considered for recommendation is lower than *PCA* strategy. The *classifiers* have very similar performance for all percentages of partitions in the *ANOVA* strategy. On the other hand, the variation in the percentages of elements per partition also reflects a more significant variation in results between the different classifiers. The *Multi Layer Perceptron (MLP)* classifier, which was trained using the *Stochastic Descending Gradient* (Bottou, 2010) with a single hidden layer, presents the worst results except in the setup that it follows the *PCA* partitioning

Experiment 2. Algorithm 2 Evaluation Script.

1. Algorithm 2 is instantiated with a *classifier* or *regressor*, a partitioning strategy, percentage data to be returned by partitioning strategy, and a target workflow parameter.
2. Provenance database is divided using *K-Fold Cross Validation*, $k = 5$
3. Each provenance record on test data is used to retrieve the target parameter's real value.
4. A random number x between 2 and parameters number present in provenance database is chosen to simulated preference number used in recommending target parameter.
5. x parameters are chosen from the remaining test record to be used as preferences.
6. Algorithm performs recommendation, and both result and test record value for the target parameter are stored.
7. Precision and recall, or *MSE* values are calculated based on all K-Fold Cross Validation iterations.



(a) Precision results with categorical domain *Sciphy* data.

(b) Recall results with categorical domain *Sciphy* data.

Figure 16. Precision and Recall results with *Sciphy* data.

strategy with a percentage of 70% elements in the partitioning. The *MLP* model performance degradation may be related to the fact that the numerical attributes are not normalized before algorithm execution.

Recall results, in Figure 16b were very similar to *precision* results in absolute values. A difference is the smallest variation, in general, of *recall* results for each attribute used in the recommendation experiment. The *Multi Layer Perceptron* classifier presented a behavior similar to the *precision* results, with a degradation in the setup that includes *ANOVA* partitioning with 70% of the elements in the partitioning.

Figure 17 shows the average execution time in seconds during the experiment with categorical domain parameters in each setup used. Execution time of *ANOVA* partitioning strategy was, on average, half the time used with the *PCA* partitioning strategy. The execution time using different classifiers for each attribute is also much smaller and stable for *ANOVA* strategy than for *PCA*, regardless of element partition percentage.

Analyzing *precision*, *recall*, and execution time spent data jointly, *ANOVA* partitioning strategy showed the best recommendation performance for the categorical domain parameters of the *Sciphy* provenance database. Going further, the element partition percentage generated by the strategy has no significant impact on the results. Another interesting point is that a simpler classifier like *KNN* presented results very similar to those obtained by a more complex classifier like *SVM*.

Figure 18a brings the data from results obtained for the numerical domain parameter *Sciphy* provenance database. The data shows zero *MSE* in all cases, except for the use of *Multi Layer Perceptron* in the

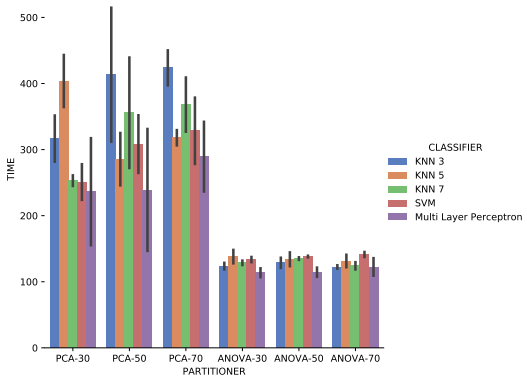
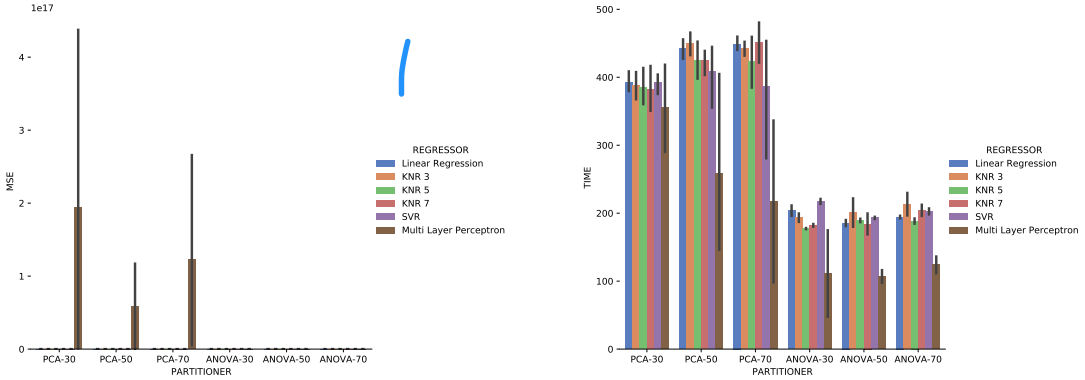


Figure 17. Experiment recommendation execution time with categorical domain *Sciphy* data.



(a) *MSE* results with categorical domain *Sciphy* data.

(b) Experiment recommendation execution time with numerical domain *Sciphy* data.

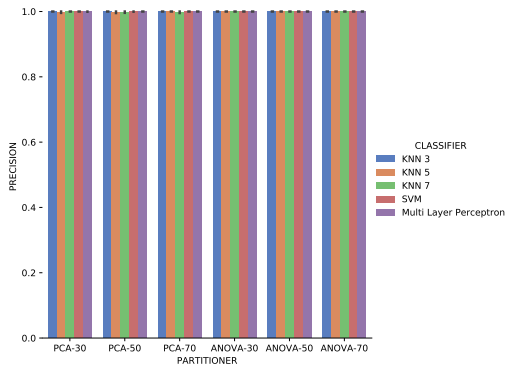
Figure 18. *MSE* results and recommendation execution time with *Sciphy* data.

regression. This result can be explained by the small database and the few different values for each numerical domain parameter. Small values difference per parameter suggests that the *regressors* have no work to generate a result equal to what is already present in the database.

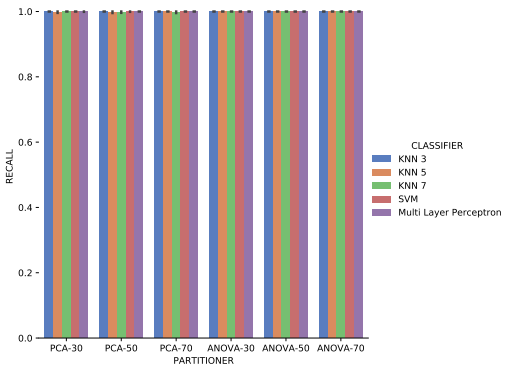
Looking at Figure 18b, one can notice that, similar to the categorical domain parameters results, the execution time of *ANOVA* partitioning strategy is much less than the time used by the *PCA* strategy. Another similar point with categorical domain parameter results is the smaller and more stable *ANOVA* strategy results variation.

From all results obtained in the Experiment 2 using *Sciphy* provenance database, it can be noticed that the *ANOVA* partitioning strategy had the best performance. Further *precision*, *recall*, and *MSE* results, for the Algorithm 2 setup with *ANOVA* partitioning strategy also proved to be the one that performed the recommendations in the shortest time, generally in half the time that the *PCA* partitioning strategy. Note that the recommendation time can be treated as training time since the proposed algorithm has a memory-based approach. Finally, the choice of the generated partition size and the *classifier* or *regressor* used have no significant impact on the final result unless the *classifier* or *regressor* is based on *Multi Layer Perceptron* with the same parametrization used in this work.

Analyzing Figure 19a, *precision* results obtained with categorical domain parameters from *Montage* workflow provenance database is observed that in almost all the experiment setup variations evaluated, maximum performance is reached. As seen in Table 2, the *Montage* workflow provenance database used in the experiments has only two categorical domain parameters. The small variation in possible values in the database is an explanation for the *precision* results. The *recall* results in Figure 19b are similar to the *precision* ones.



(a) Precision results with categorical domain *Montage* data.



(b) Recall results with categorical domain *Montage* data.

Figure 19. precision and Recall results with *Montage* data.

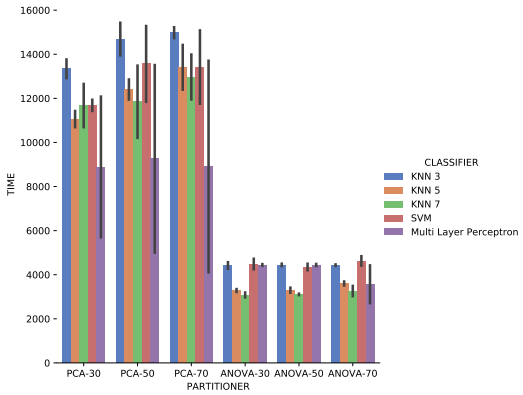


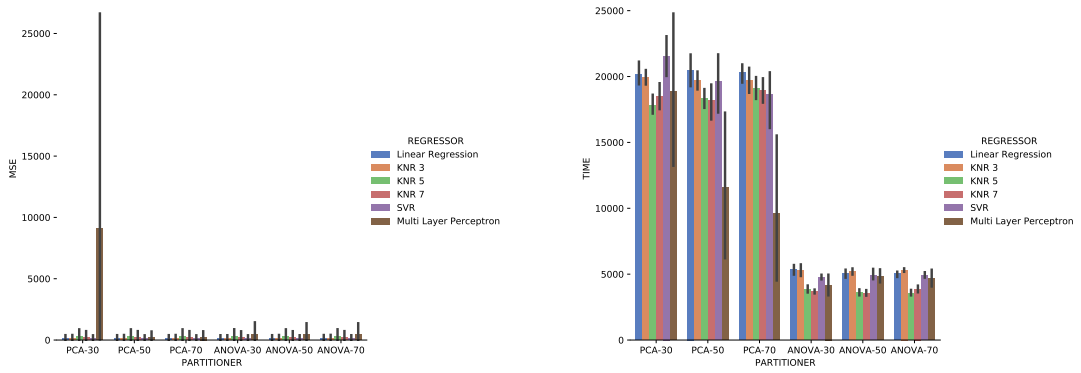
Figure 20. Experiment recommendation execution time with categorical domain *Montage* data.

Concerning the results about the experiment time with categorical domain parameters from the *Montage* provenance database, presented in Figure 20, one can see that the *KNN* classifier, $k = 3$, with *PCA* partitioning strategy was the most time-consuming. On the other hand, with the same *PCA* partitioning strategy, the *Multi Layer Perceptron* classifier used less time, but with a wide variation in recommendation times for different parameters. The *ANOVA* partitioning strategy continued to be a partitioning strategy that delivers the fattest recommendations. Still analyzing *ANOVA* partitioning strategy results, it is possible to see that the *KNN* classifier, with $k \in \{5,7\}$, was the fastest in recommending *Montage* workflow categorical domain parameters.

Making a general analysis of results in Figure 19 and 20, the setup that uses *ANOVA* partitioning strategy with the *KNN* classifier, $k = 7$ it's the best. This setup was the one that obtained the best results for *precision*, *recall*, and execution time spent simultaneously. *MSE* results for *Montage* numerical domain parameters presented in Figure 21a show that, in general, the *MSE* was very close to zero for all cases, except in algorithm setup using *PCA* partitioning strategy with 30% elements in the generated partition and the *regressor* implemented by *Multi Layer Perceptron*. The *MSE* and its variation were very close to zero.

Regarding the execution time of Experiment 2 for numerical domain parameters recommendations for *Montage* data, Figure 21b indicates the same behavior shown by results with *SciPhy* provenance database. Using *ANOVA* partitioning strategy and *KNR* regressors with $k \in \{5,7\}$ as setup for Algorithm 2 produced the fastest recommendations.

The experiment execution time of *Montage* provenance database was much greater than the time used with the data from the workflow *Sciphy*. The explanation is the difference in the database size. Another



(a) *MSE* results with categorical domain *Montage* data.

(b) Experiment recommendation execution time with numerical domain *Montage* data.

Figure 21. *MSE* results and recommendation execution time with *Montage* data.

observation is that the *ANOVA* partitioning strategy produces the fastest recommendations. Another point is that the percentage of the elements in partitioning generated by each partitioning strategy has no impact on the algorithm performance. Finally, it was possible to notice that the more robust *classifiers* and *regressors* had their performance exceeded by simpler models in some cases for the data used.

Generic *FReeP* Recommendation Evaluation

A third experiment was modeled to evaluate Algorithm 3 performance. As in Experiment 2, different variations, following Table 5 values, were used in algorithm execution. *Precision*, *recall*, and *MSE* are also the metrics used to evaluate the recommendations made by each algorithm instance.

Experiment 3. Algorithm 3 Evaluation Script.

1. n Records from the provenance database were chosen as random examples.
2. $m \geq 2$ random parameters were chosen for each example record as preferences, and their values are the same as those present in the example record.
3. Algorithm 3 was instantiated with a classifier or a regressor, a partitioning strategy, the partitions percentage to be returned by the partitioning strategy, and the selected m preferences.
4. Each returned recommendation is separated into numeric and categorical and is stored.
5. Precision and recall values were calculated for categorical recommendations and Mean Square Error (*MSE*) for numerical recommendations.

Results

Results showed here were obtained by fixing parameter $n = 10$ in Experiment 3, and using only *SciPhy* provenance database. Based on Experiment 2 results, it was decided to use the *ANOVA* partitioning strategy with 50% recovering elements from the provenance database. This choice is because the *ANOVA* partitioning strategy was the one that obtained the best results in previous experiment. As the percentage of data recovered by the strategy was not an impacting factor in the results, an intermediate percentage used in the previous experiment is selected. In addition, only *KNN*, with $k \in \{5,7\}$, and *SVM* were kept as classifiers, whereas only *KNR*, with $k \in \{5,7\}$, and *SVR* was chosen as regressors. These choices are supported by, in general, are the ones that present the best *precision*, *recall*, and *MSE* results in Experiment 2.

Table 6 presents the results obtained with the Algorithm 3 instance variations. Each row in the table represents an Algorithm 3 instance setup. The column that draws the most attention is the *Failures*. What

Classifier	Regressor	Partitioning Strategy	*MSE*	precision	recall	Failures
KNN 5	KNR 5	ANOVA 50	0.0	1.0	1.0	6
KNN 5	KNR 7	ANOVA 50	0.0	1.0	1.0	6
KNN 5	SVR	ANOVA 50	1.1075	1.0	1.0	6
KNN 7	KNR 5	ANOVA 50	4279.2240	1.0	1.0	5
KNN 7	KNR 7	ANOVA 50	0.0	1.0	1.0	5
KNN 7	SVR	ANOVA 50	0.444	1.0	1.0	5
SVM	KNR 5	ANOVA 50	1148.1876	0.75	0.75	6
SVM	KNR 7	ANOVA 50	0.0	1.0	1.0	7
SVM	SVR	ANOVA 50	0.0	1.0	1.0	7

Table 6. Experiment 3 results with *Sciphy* dataset

happens is that, for some cases, the algorithm was not able to carry out the recommendation together and therefore did not return any recommendations. It is important to remember that each algorithm setup was tested on a set with 10 records extracted randomly from the database. The random record selection process can select records in which parameter values can be present only in the selected record. For this experiment, the selected examples are removed from the dataset, and therefore there is no other record that allows the correct execution of the algorithm.

Analyzing Table 6 results, focusing on the column *Failures* and taking into account that 10 records were chosen for each setup, it is possible to verify that in most cases, the algorithm was not able to make recommendations. However, considering only the recommendations made, it can be seen that the algorithm had satisfactory results for the precision and recall metrics. The values presented for the *MSE* metric were mostly satisfactory, differing only in the configurations of lines 4 and 7, both using the regressor *KNR* with $k = 5$. Another point to note is that the algorithm had more problems to make recommendations when the *SVM* classifier was used. Furthermore, it is possible to note that algorithm setups with more sophisticated Machine Learning models such as *SVM* and *SVR* do not add performance to the algorithm, specifically for *Sciphy* provenance dataset used.

RELATED WORK

Previous literature works had already relied on recommender systems to support scientific workflows. In general, the works that seek to assist scientists with some type of recommendation involving *scientific workflow* are focused on the composition phase. Zhou et al. (2018) uses a graph-based clustering technique to recommend *workflows* that can be reused in the composition of a developing workflow. De Oliveira et al. (2008) uses workflow provenance to extract connection patterns between components in order to make recommendations of new components for a workflow in composition. For each new component used in the composition of workflow, new components are recommended. Halioui et al. (2016), uses Natural Language Processing combined with specific ontologies in the field of Bioinformatics to extract concrete *workflows* from works in the literature. After the reconstruction of concrete *workflows*, tool combinations patterns , its parameters, and input data used in these *workflows* are extracted. All this data extracted can be used as assistance for composing new ones *workflows* that solve problems related to the mined *workflows*.

Yet concerned with assistance during the workflow composition phase, Mohan et al. (2015) proposes the use of *Folksonomy* (Gruber, 2007) to enrich the data used for the recommendation of others *workflows* similar to a workflow under development. A design workflow tool was developed that allows free specification tags to be used in each component, making it possible to use not only the recommendation strategy through the workflow syntax, but also component semantics. Soomro et al. (2015) uses domain ontologies as a knowledge base to incorporate semantics into the recommendation process. A hybrid recommender system was developed using ontologies to improve the already known recommendation strategy based on the extraction of standards from other *workflows*. Zeng et al. (2011) uses data and control dependencies between activities, stored in the workflow provenance to build a causality table and another weights table. Subsequently, a *Petri network* (Zhou and Venkatesh, 1999) is used to recommend other components for the composition of workflow.

The works that uses recommender system methods to support the scientific process are closely linked

to the experiment's composition phase. The execution phase, where there is a need to adjust parameters, still lacks alternatives. This work proposes a hybrid recommendation algorithm capable of making value recommendations for one or *n* parameters of a *scientific workflow*, taking into account the user's preferences.

A very similar problem to the recommendation of parameters in scientific workflows is Machine Learning algorithms hyperparameters tuning. A Machine Learning model can perform poorly with a set of hyperparameters, but a simple adjustment of the values can significantly improve the results. Then, the use of information from similar use cases can also help in the arduous task of looking for additional parameters to execute a Machine Learning model training process. In order to improve the search for the best sets of hyperparameters, some initiatives (Hutter et al., 2011) (Bergstra et al., 2011) sought to do a combination of manual search and *Grid Search* (Hsu et al., 2003).

Another approach that has shown relevant results and influenced other variants' development is *Sequential Model-Based Bayesian Optimization*. Brochu et al. (2010) proposed the method of optimizing a function f through an iterative process. The process needs to define a function f, a set $T : \{t_0, t_1, ..., t_n\}$ of hyperparameters, θ, a set of values for T, θ the search space for θ and probabilistic model μ. The process starts by calculating f for each of the θ_i and the pairs $< \theta_i, f(\theta_i) >$ are stored. That done, there is an iteration over three steps: 1) Training the *mu* model with the pairs $< \theta_i, f(\theta_i) >$; 2) Use of the *mu* model to select the next θ' set that is promising; 3) New calculation for the pairs $< \theta'_i, f(\theta'_i) >$. In step 2, the new set θ' is chosen with the help of a function called *acquisition function* that seeks a balance between the choice in the search space and the quality of the improvement obtained. Through this *acquisition function*, the number of evaluations by the search space decreases dramatically.

To further assist the use of Machine Learning algorithms, Thornton et al. (2013) proposed a work that aims to indicate which hyperparameters and which Machine Learning model to use for a dataset. The proposal was developed based on the classification task and consisted of using the algorithm as an input hyperparameter in a *Sequential Model-Based Bayesian Optimization* algorithm. The good results obtained were due to the use of the implementation of *Sequential Model-Based Optimization* (Hutter et al., 2011) and *Tree-Structured Parzen Estimator* (Bergstra et al., 2011) that allow the search for values of parameters with restriction related to the Machine Learning model taken into question during an iteration.

Also based on *Sequential Model-Based Bayesian Optimization*, Bardenet et al. (2013) makes a replacement for the f function used in *Sequential Model-Based Bayesian Optimization* by an f' function. This f' function is such that it will not be influenced by the different orders of quantities present in different databases, allowing a choice of an initial set of hyperparameters in a more generalized way, called *Surrogate-Based Collaborative Tuning*.

However, the works presented do not consider that the user may prefer values for a subset of parameters. The present work proposes an algorithm that considers the user's preferences to make recommendations for one or *n* parameters for which there is no defined preference.

CONCLUSION

The scientific process is mostly responsible for several advances in human knowledge. The process involves observing phenomena from different areas, formulating hypotheses, testing, and refining them. Arguably, this is an arduous job for the responsible scientist. With the advances in computational resources, there is a growing concern about helping scientists in scientific experimentation. A significant step towards a more robust aid was the adoption of *scientific workflows* as a model for representing scientific experiments. From the representation of *workflows*, several tools emerged to support the management of experiment executions, storage of the data generated during the execution, and analysis of the data, called *Scientific Workflow Management Systems*.

Computational execution of the experiments represented as *scientific workflows* relies on the use of computer programs that play the role of each stage of the experiment. In addition to input data, these programs often need additional configuration parameters to be adjusted to simulate the experiment's conditions. The scientist responsible for the experiment ends up developing an intuition about the sets of parameters that lead to satisfactory results. However, another scientist who runs the same experiment will not have the same experience, which may lead him/her to define a set of parameters that will not result in a successful experiment.

Several proposals in the literature have aimed at supporting the composition phase of the experiments, but recommending parameter values for the experiment execution phase is still an open field. This article

presented an algorithm for recommending values for parameters in *scientific workflows* considering the user's preferences. The goal was to allow a new user to express their preferences of values for a subset of workflow parameters and recommend values for the parameters that had no preference defined. The developed algorithm was called *FReeP: Feature Recommender From Preferences*, and has three versions, all of them relying on Machine Learning concepts and techniques. Two approaches focused on the value recommendation for one parameter at a time. The third instance addresses recommending values for all the other parameters of a workflow for which a user preference was not defined.

The first approach was developed as a proof of concept. The second approach addressed the problems perceived by the first approach and increases the performance of the algorithm. Finally, the third approach is a proposal for recommending *n* parameters at once. A different experiment evaluated each approach. The first experiment showed that a classifier came out as a better option than a *ranker* for a Machine Learning model in the algorithm. The second experiment was carried out under several variants of the second version of the algorithm, using different classifiers, regressors, and partitioners. The second experiment results showed that the algorithm's first approach's changes had the desired effects, increasing the algorithm's performance. The third experiment clarifies that the algorithm approach for recommending simultaneously *n* parameters still needs to be refined to obtain satisfactory results. Empirically, the *FReeP* algorithm was able to make valuable recommendations for one *scientific workflow* parameter at a time, especially the second approach presented.

The proposed algorithm proved to be useful for recommending one parameter, indicating a path for the recommendation of *n* parameters. Nevertheless, there are some limitations, such as:

- *FReeP*, as a memory-based algorithm, faces scalability issues as its implementation can consume a lot of computational resources.
- The recommendations of *FReeP* are limited to the existence of examples on the provenance dataset. This means that the algorithm cannot make any "default" recommendations if there are no examples for the algorithm's execution or recommend values that are not present in the provenance dataset.
- The recommendation algorithm may have a longer processing time than the experiment itself.
- All the instances have the same weight during the recommendation process. The algorithm does not consider the user's expertise that performed the previous execution to adjust an example's weight.
- The algorithm considers only the set of parameters of the workflow; however, a set of parameters may be more or less relevant according to the input data.
- The recommendation algorithm may end up recommending a set of values present in the provenance base that causes a workflow execution failure.

Based on those limitations, some proposals for future work are:

- Parallelizing the processing of the generated partitions, which should decrease the time spent on the recommendation
- Evaluating *FReeP* on data from other domains.
- Evaluating the tradeoff between the recommendation time and the algorithm execution time.
- Associating weights with examples from the provenance dataset according to the user's profile.
- Using instances from the provenance dataset that failed to execute the workflow as a constraint to improve the recommendations' results.

ACKNOWLEDGMENTS

Authors would like to thank FAPERJ, CAPES and CNPq for partially sponsoring this research.

REFERENCES

Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749.

Al-Sharrah, G. (2010). Ranking using the copeland score: a comparison with the hasse diagram. *Journal of chemical information and modeling*, 50(5):785–791.

Altintas, I., Ludaescher, B., Klasky, S., and Vouk, M. A. (2006). Introduction to scientific workflow management and the kepler system. In *SC'06: Proceedings of the 2006 ACM/IEEE conference on Supercomputing*, page 205.

Arlot, S., Celisse, A., et al. (2010). A survey of cross-validation procedures for model selection. *Statistics surveys*, 4:40–79.

Balabanovic, M. and Shoham, Y. (1997). Fab: content-based, collaborative recommendation. *Communications of the ACM*, 40(3):66–73.

Bardenet, R., Brendel, M., Kégl, B., and Sebag, M. (2013). Collaborative hyperparameter tuning. In *International conference on machine learning*, pages 199–207.

Basilico, J. and Hofmann, T. (2004). Unifying collaborative and content-based filtering. In *Proceedings of the twenty-first international conference on Machine learning*, page 9. ACM.

Belhajjame, K., B'Far, R., Cheney, J., Coppens, S., Cresswell, S., Gil, Y., Groth, P., Klyne, G., Lebo, T., McCusker, J., et al. (2013). Prov-dm: The prov data model. *W3C Recommendation. http://www. w3. org/TR/prov-dm*.

Bell, R. M. and Koren, Y. (2007). Improved neighborhood-based collaborative filtering. In *KDD cup and workshop at the 13th ACM SIGKDD international conference on knowledge discovery and data mining*, pages 7–14. Citeseer.

Bergeron, C., Zaretzki, J., Breneman, C., and Bennett, K. P. (2008). Multiple instance ranking. In *Proceedings of the 25th international conference on Machine learning*, pages 48–55.

Bergstra, J. S., Bardenet, R., Bengio, Y., and Kégl, B. (2011). Algorithms for hyper-parameter optimization. In *Advances in neural information processing systems*, pages 2546–2554.

Bobadilla, J., Ortega, F., Hernando, A., and Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-based systems*, 46:109–132.

Bose, R., Foster, I., and Moreau, L. (2006). Report on the international provenance and annotation workshop:(ipaw'06) 3-5 may 2006, chicago. *ACM SIGMOD Record*, 35(3):51.

Bottou, L. (2010). Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer.

Brochu, E., Cora, V. M., and De Freitas, N. (2010). A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *arXiv preprint arXiv:1012.2599*.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370.

Coates, A. and Ng, A. Y. (2011). The importance of encoding versus training with sparse coding and vector quantization. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 921–928.

de Oliveira, D., Ocaña, K. A. C. S., Baião, F. A., and Mattoso, M. (2012). A provenance-based adaptive scheduling heuristic for parallel scientific workflows in clouds. *J. Grid Comput.*, 10(3):521–552.

de Oliveira, D., Ogasawara, E., Baião, F., and Mattoso, M. (2010a). Scicumulus: A lightweight cloud middleware to explore many task computing paradigm in scientific workflows. In *2010 IEEE 3rd International Conference on Cloud Computing*, pages 378–385. IEEE.

de Oliveira, D., Ogasawara, E., Baião, F., and Mattoso, M. (2010b). Scicumulus: A Lightweight Cloud Middleware to Explore Many Task Computing Paradigm in Scientific Workflows. In *3rd International Conference on Cloud Computing*, pages 378–385.

de Oliveira, D. C. M., Liu, J., and Pacitti, E. (2019). *Data-Intensive Workflow Management: For Clouds and Data-Intensive and Scalable Computing Environments*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers.

De Oliveira, F. T., Murta, L., Werner, C., and Mattoso, M. (2008). Using provenance to improve workflow design. In *International Provenance and Annotation Workshop*, pages 136–143. Springer.

Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Kesselman, C., Mehta, G., Vahi, K., Berriman,

G. B., Good, J., et al. (2005). Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237.

Eduardo Ogasawara, D. d. O. and et.al (2010). An Algebraic Approach for Data-Centric Scientific Workflows. *Proceedings of the VLDB Endownment*, 4:1328–13369.

Emerson, P. (2013). The original borda count and partial voting. *Social Choice and Welfare*, 40(2):353–358.

Feynman, R. and Vernon Jr., F. (1963). The theory of a general quantum system interacting with a linear dissipative system. *Annals of Physics*, 24:118–173.

Fürnkranz, J. (2002). Round robin classification. *Journal of Machine Learning Research*, 2(Mar):721–747.

Fürnkranz, J. and Hüllermeier, E. (2003). Pairwise preference learning and ranking. In *European conference on machine learning*, pages 145–156. Springer.

Fürnkranz, J. and Hüllermeier, E. (2011). Preference learning. In *Encyclopedia of Machine Learning*, pages 789–795. Springer.

Garthwaite, P. H., Jolliffe, I. T., Jolliffe, I., and Jones, B. (2002). *Statistical inference*. Oxford University Press on Demand.

Gil, Y., Miles, S., Belhajjame, K., Deus, H., Garijo, D., Klyne, G., Missier, P., Soiland-Reyes, S., and Zednik, S. (2013). Prov model primer. *W3C Working Group Note*, 30.

Girden, E. R. (1992). *ANOVA: Repeated measures*. Number 84. Sage.

Goble, C. (2002). Position statement: Musings on provenance, workflow and (semantic web) annotations for bioinformatics. In *Workshop on Data Derivation and Provenance, Chicago*, volume 3.

Gonçalves, B. and Porto, F. (2015). Managing scientific hypotheses as data with support for predictive analytics. *Comput. Sci. Eng.*, 17(5):35–43.

Gonzales, C. and Perny, P. (2004). Gai networks for utility elicitation.

Gruber, T. (2007). Ontology of folksonomy: A mash-up of apples and oranges. *International Journal on Semantic Web and Information Systems (IJSWIS)*, 3(1):1–11.

Halioui, A., Valtchev, P., and Diallo, A. B. (2016). Towards an ontology-based recommender system for relevant bioinformatics workflows. *bioRxiv*, page 082776.

Harper, F. M. and Konstan, J. A. (2016). The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19.

Herlocker, J. L., Konstan, J. A., Terveen, L. G., and Riedl, J. T. (2004). Evaluating collaborative filtering recommender systems. *ACM Transactions on Information Systems (TOIS)*, 22(1):5–53.

Hey, T., Gannon, D., and Pinkelman, J. (2012). The future of data-intensive science. *Computer*, 45(5):81–82.

Hey, T. and Trefethen, A. E. (2020). The fourth paradigm 10 years on. *Inform. Spektrum*, 42(6):441–447.

Hoffa, C., Mehta, G., Freeman, T., Deelman, E., Keahey, K., Berriman, B., and Good, J. (2008). On the use of cloud computing for scientific workflows. In *eScience, 2008. eScience'08. IEEE Fourth International Conference on*, pages 640–645. IEEE.

Hsu, C.-W., Chang, C.-C., Lin, C.-J., et al. (2003). A practical guide to support vector classification.

Hüllermeier, E., Fürnkranz, J., Cheng, W., and Brinker, K. (2008). Label ranking by learning pairwise preferences. *Artificial Intelligence*, 172(16-17):1897–1916.

Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration. In *International Conference on Learning and Intelligent Optimization*, pages 507–523. Springer.

Juliana Freire, David Koop, E. S. and Silva, C. T. (2008). Provenance for Computational Tasks: A Survey. *Computing in Science & Engineering*, pages 20–30.

Kanungo, T., Mount, D. M., Netanyahu, N. S., Piatko, C. D., Silverman, R., and Wu, A. Y. (2002). An efficient k-means clustering algorithm: Analysis and implementation. *IEEE transactions on pattern analysis and machine intelligence*, 24(7):881–892.

Karvonen, L. (2004). Preferential voting: Incidence and effects. *International Political Science Review*, 25(2):203–226.

Keller, J. M., Gray, M. R., and Givens, J. A. (1985). A fuzzy k-nearest neighbor algorithm. *IEEE transactions on systems, man, and cybernetics*, (4):580–585.

Koffi, C. (2015). Exploring a generalized partial borda count voting system.

Kohavi, R. (2001). A study of cross-validation and bootstrap for accuracy estimation and model selection. 14.

Koren, Y., Bell, R., and Volinsky, C. (2009). Matrix factorization techniques for recommender systems. *Computer*, (8):30–37.

Koriche, F. (2012). Relational networks of conditional preferences. *Machine learning*, 89(3):233–255.

Lekakos, G. and Caravelas, P. (2008). A hybrid approach for movie recommendation. *Multimedia tools and applications*, 36(1-2):55–70.

Lestari, S., Adji, T. B., and Permanasari, A. E. (2018). Performance comparison of rank aggregation using borda and copeland in recommender system. In *2018 International Workshop on Big Data and Information Security (IWBIS)*, pages 69–74. IEEE.

Lewis, D. D. (1998). Naive (bayes) at forty: The independence assumption in information retrieval. In *European conference on machine learning*, pages 4–15. Springer.

Lika, B., Kolomvatsos, K., and Hadjiefthymiades, S. (2014). Facing the cold start problem in recommender systems. *Expert Systems with Applications*, 41(4):2065–2073.

Linden, G., Smith, B., and York, J. (2003). Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, 7(1):76–80.

Lu, Q., Chen, T., Zhang, W., Yang, D., and Yu, Y. (2012). Serendipitous personalized ranking for top-n recommendation. In *2012 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, volume 1, pages 258–265. IEEE.

Ma, C.-C. (2008). A guide to singular value decomposition for collaborative filtering. *Computer (Long Beach, CA)*, pages 1–14.

Marozzo, F., Talia, D., and Trunfio, P. (2013). Scalable script-based data analysis workflows on clouds. In *WORKS*, pages 124–133.

Massa, P. and Avesani, P. (2007). Trust-aware recommender systems. In *Proceedings of the 2007 ACM conference on Recommender systems*, pages 17–24. ACM.

Mattoso, M., Werner, C., Travassos, G. H., Braganholo, V., Murta, L., Ogasawara, E., de Oliveira, D., da Cruz, S. M. S., and Martinho, W. (2010a). Towards Supporting the Life Cycle of Large-scale Scientific Experiments. *International Journal of Business Process Integration and Management*, pages 79–92.

Mattoso, M., Werner, C., Travassos, G. H., Braganholo, V., Ogasawara, E. S., de Oliveira, D., da Cruz, S. M. S., Martinho, W., and Murta, L. (2010b). Towards supporting the life cycle of large scale scientific experiments. *IJBPIM*, 5(1):79–92.

McKinney, W. (2011). pandas: a foundational python library for data analysis and statistics. *Python for High Performance and Scientific Computing*, pages 1–9.

Miller, B. N., Albert, I., Lam, S. K., Konstan, J. A., and Riedl, J. (2003). Movielens unplugged: experiences with an occasionally connected recommender system. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 263–266. ACM.

Mitchell, T. M. (2015). *Machine Learning*. McGraw-Hill Science/Engineering/Math.

Mitchell, T. M., Caruana, R., Freitag, D., McDermott, J., Zabowski, D., et al. (1994). Experience with a learning personal assistant. *Communications of the ACM*, 37(7):80–91.

Mohan, A., Ebrahimi, M., and Lu, S. (2015). A folksonomy-based social recommendation system for scientific workflow reuse. In *Services Computing (SCC), 2015 IEEE International Conference on*, pages 704–711. IEEE.

Myers, K., Berry, P., Blythe, J., Conley, K., Gervasio, M., McGuinness, D. L., Morley, D., Pfeffer, A., Pollack, M., and Tambe, M. (2007). An intelligent personal assistant for task and time management. *AI Magazine*, 28(2):47.

Myers, R. H. and Myers, R. H. (1990). *Classical and modern regression with applications*, volume 2. Duxbury Press Belmont, CA.

Nie, Z., Zhang, Y., Wen, J.-R., and Ma, W.-Y. (2005). Object-level ranking: bringing order to web objects. In *Proceedings of the 14th international conference on World Wide Web*, pages 567–574.

Ocaña, K. A., de Oliveira, D., Ogasawara, E., Dávila, A. M., Lima, A. A., and Mattoso, M. (2011). Sciphy: a cloud-based workflow for phylogenetic analysis of drug targets in protozoan genomes. In *Brazilian Symposium on Bioinformatics*, pages 66–70. Springer.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., et al. (2011). Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830.

Pessiot, J.-F., Truong, V., Usunier, N., Amini, M., and Gallinari, P. (2007). Learning to rank for

collaborative filtering.
Pigozzi, G., Tsoukias, A., and Viappiani, P. (2016). Preferences in artificial intelligence. *Annals of Mathematics and Artificial Intelligence*, 77(3-4):361–401.
Rani, P., Shokeen, J., and Mullick, D. (2017). Recommendations using modified k-means clustering and voting theory.
Read, J., Pfahringer, B., Holmes, G., and Frank, E. (2011). Classifier chains for multi-label classification. *Machine learning*, 85(3):333.
Resnick, P. and Varian, H. R. (1997). Recommender systems. *Communications of the ACM*, 40(3):56–59.
Ricci, F., Rokach, L., and Shapira, B. (2011). Introduction to recommender systems handbook. In *Recommender systems handbook*, pages 1–35. Springer.
Salton, G. (1989). Automatic text processing: The transformation, analysis, and retrieval of. *Reading: Addison-Wesley*.
Schafer, J. B., Frankowski, D., Herlocker, J., and Sen, S. (2007). Collaborative filtering recommender systems. In *The adaptive web*, pages 291–324. Springer.
Schein, A. I., Popescul, A., Ungar, L. H., and Pennock, D. M. (2002). Methods and metrics for cold-start recommendations. In *Proceedings of the 25th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 253–260. ACM.
Silva Junior, D., Paes, A., Pacitti, E., and de Oliveira, D. (2018). Freep: towards parameter recommendation in scientific workflows using preference learning. In *XXXIII Brazilian Symposium on Databases (SBBD)*, pages 211–216, Rio de Janeiro, Brazil.
Smith, L. I. (2002). A tutorial on principal components analysis. Technical report.
Soomro, K., Munir, K., and McClatchey, R. (2015). Incorporating semantics in pattern-based scientific workflow recommender systems: Improving the accuracy of recommendations. In *Science and Information Conference (SAI), 2015*, pages 565–571. IEEE.
Strøm, K., Strøm, K., et al. (1990). *Minority government and majority rule*. Cambridge University Press.
Tang, Y. and Tong, Q. (2016). Bordarank: A ranking aggregation based approach to collaborative filtering. In *2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS)*, pages 1–6. IEEE.
Taylor, A. D. and Pacelli, A. M. (2008). *Mathematics and politics: strategy, voting, power, and proof*. Springer Science & Business Media.
Thi Do, M.-P., Van Nguyen, D., and of Loc Nguyen, A. N. (2010). Model-based approach for collaborative filtering.
Thornton, C., Hutter, F., Hoos, H. H., and Leyton-Brown, K. (2013). Auto-weka: Combined selection and hyperparameter optimization of classification algorithms. In *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 847–855. ACM.
Tsoumakas, G. and Katakis, I. (2007). Multi-label classification: An overview. *International Journal of Data Warehousing and Mining (IJDWM)*, 3(3):1–13.
Ungar, L. H. and Foster, D. P. (1998). Clustering methods for collaborative filtering. In *AAAI workshop on recommendation systems*, volume 1, pages 114–129.
Vembu, S. and Gärtner, T. (2011). *Label Ranking Algorithms: A Survey*, pages 45–64. Springer Berlin Heidelberg, Berlin, Heidelberg.
Viappiani, P. and Boutilier, C. (2009). Regret-based optimal recommendation sets in conversational recommender systems. In *Proceedings of the third ACM conference on Recommender systems*, pages 101–108. ACM.
Walt, S. v. d., Colbert, S. C., and Varoquaux, G. (2011). The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30.
Yong Zhao, Ioan Raicu, I. F. (2008). Scientific Workflow Systems for 21st Century, New Bottle or New Wine? *IEEE Congress on Services*, page 1.
Zeng, R., He, X., and van der Aalst, W. M. (2011). A method to mine workflows from provenance for assisting scientific workflow composition. In *2011 IEEE World Congress on Services*, pages 169–175. IEEE.
Zhou, M. and Venkatesh, K. (1999). *Modeling, simulation, and control of flexible manufacturing systems: a Petri net approach*, volume 6. World Scientific.
Zhou, Z., Cheng, Z., Zhang, L.-J., Gaaloul, W., and Ning, K. (2018). Scientific workflow clustering and recommendation leveraging layer hierarchical analysis. *IEEE Transactions on Services Computing*,

11(1):169–183.