Supporting the Design of Machine Learning Workflows with a Recommendation System

DIETMAR JANNACH, MICHAEL JUGOVAC, and LUKAS LERCHE, TU Dortmund, Germany

Machine learning and data analytics tasks in practice require several consecutive processing steps. RapidMiner is a widely used software tool for the development and execution of such analytics workflows. Unlike many other algorithm toolkits, it comprises a visual editor that allows the user to design processes on a conceptual level. This conceptual and visual approach helps the user to abstract from the technical details during the development phase and to retain a focus on the core modeling task. The large set of preimplemented data analysis and machine learning operations available in the tool, as well as their logical dependencies, can, however, be overwhelming in particular for novice users.

In this work, we present an add-on to the RapidMiner framework that supports the user during the modeling phase by recommending additional operations to insert into the currently developed machine learning workflow. First, we propose different recommendation techniques and evaluate them in an offline setting using a pool of several thousand existing workflows. Second, we present the results of a laboratory study, which show that our tool helps users to significantly increase the efficiency of the modeling process. Finally, we report on analyses using data that were collected during the real-world deployment of the plug-in component and compare the results of the live deployment of the tool with the results obtained through an offline analysis and a replay simulation.

CCS Concepts: • **Information systems** → **Data mining**; **Recommender systems**; *Data analytics*; *Collaborative filtering*; • **Computing methodologies** → *Machine learning*; • **Software and its engineering** → *Integrated and visual development environments*; • **Human-centered computing** → *Interactive systems and tools*; Laboratory experiments; User studies

Additional Key Words and Phrases: Data analysis workflows, RapidMiner, visual process modeling

ACM Reference Format:
Dietmar Jannach, Michael Jugovac, and Lukas Lerche. 2016. Supporting the design of machine learning workflows with a recommendation system. ACM Trans. Interact. Intell. Syst. 6, 1, Article 8 (February 2016), 35 pages.
DOI: http://dx.doi.org/10.1145/2852082

1. INTRODUCTION

The application of modern machine learning algorithms in the context of data mining or predictive analytics projects typically requires the implementation of a nontrivial processing workflow. Such workflows, for example, include the retrieval of the raw data, the application of certain preprocessing steps, or the training or evaluation of the actual machine learning model.

The reviewing of this article was managed by the associate editors of the special issue on Highlights of IUI 2015, Shimei Pan and Giuseppe Carenini.
This article is a significantly extended version of Jannach et al. [2015]. We thank Simon Fischer and RapidMiner GmbH for their support.
Authors' addresses: D. Jannach, M. Jugovac, and L. Lerche, Department of Computer Science, TU Dortmund, Dortmund, Germany; emails: {dietmar.jannach, michael.jugovac, lukas.lerche}@tu-dortmund.de.

DOI: http://dx.doi.org/10.1145/2852082

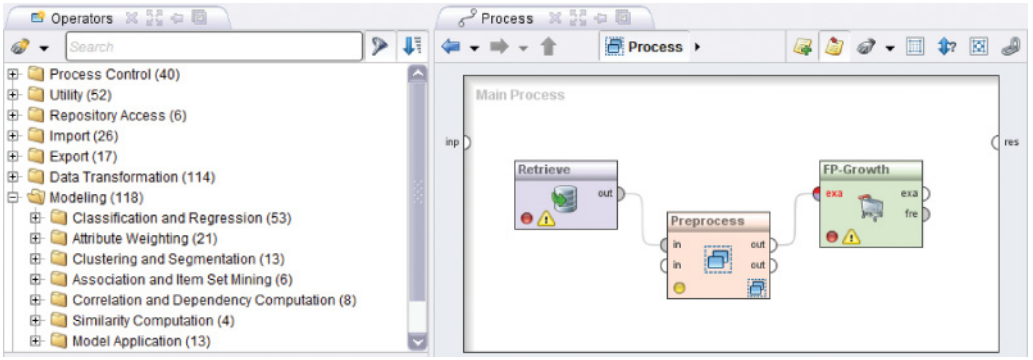

Fig. 1. A process fragment (right-hand side) developed with the RapidMiner GUI consisting of operators linked by edges that denote the data flow. The left-hand side shows the hierarchically organized set of available operators, which can be dragged and dropped to the modeling area.

RapidMiner is a widespread open-source software framework for the design and execution of data analysis workflows (http://www.rapidminer.com). The framework comprises a number of pre-implemented software components that provide often-required functionalities for data mining and data analysis including, for example, methods for data retrieval and data preprocessing, feature selection techniques, various machine learning algorithms, as well as methods for performance evaluation.

In contrast to many other data processing frameworks or machine learning libraries, RapidMiner includes a visual modeling environment (Figure 1), which supports the definition of complete data analysis workflows (processes). The elements of the processes are (1) a set of process steps called "operators" and (2) a set of edges connecting the operators. The operators are used to model data processing tasks and the control flow; the edges specify the required data flows.

The right-hand side of Figure 1 shows a small example of a process in RapidMiner consisting of three operators and their predefined input and output ports. The "preprocessing" operator in this case is by the way not an atomic step but rather a subprocess that encapsulates other operators. The left-hand side of the screenshot shows the set of available operators.

While visual tools like RapidMiner—and more recently also Microsoft Azure's Machine Learning Studio or KNIME[1]—make it easier for users to incrementally develop and immediately test data analysis workflows, the variety of available process steps can be challenging, in particular for novice users: A standard installation of RapidMiner without extensions already comprises several hundred operators. Furthermore, some operators are semantically or logically connected and often only make sense in a process when used in combination. In the example in Figure 1, the user might be in the act of creating a data analysis workflow for a typical shopping-basket analysis. The "FP-Growth" operator [Han et al. 2000] was already inserted into the model for the detection of "frequent patterns." The next typical step after applying the "FP-Growth" algorithm is the generation of association rules [Agrawal et al. 1993]. An intelligent user-interface (UI) component would therefore automatically propose the insertion of a corresponding "rule generation" operator, thus relieving the user from looking up a suitable operator in the complex hierarchical structure shown on the left-hand side of the screen.

[1]https://studio.azureml.net, https://www.knime.org.

Besides such forward-completion recommendations, operator recommendations can also be helpful to point the inexperienced user to missing operators in the process chains. The "FP-Growth" algorithm, for example, requires that its inputs are binary (true or false). If the input data that is retrieved and preprocessed in the example in Figure 1 is not binary, then the automated recommendation of the "numerical-to-binomial" conversion operator might therefore prevent the user from creating a faulty processes definition.

To better support the users in these situations, we developed a UI extension for RapidMiner which makes adaptive recommendations to the user regarding suitable operators to extend the currently developed data analysis process.[2] The new UI component bases its suggestions on a prediction model that is learned in a preprocessing phase using a pool of several thousand real-world data analysis workflows. The "operator recommender" is fully integrated into the RapidMiner modeling framework as a plug-in component, supports standard drag-and-drop operations, and constantly updates the recommendations depending on the user's recent actions. A first version of the recommendation plug-in has been successfully integrated and shipped with the latest releases of RapidMiner (starting from version 6 under the name "Wisdom of Crowds").

The article is organized as follows. In the next section, we propose algorithms for operator recommendation that take the user's current modeling *context* into account and benchmark these methods with the algorithms presented in Jannach and Fischer [2014]. After outlining the technical architecture of the overall recommendation framework, we report the results of a laboratory study that aimed to assess the utility of the system during the modeling process. We then summarize the key observations that were made during the first weeks of the ongoing live deployment of the plug-in within the RapidMiner framework and re-execute the accuracy analysis using the new process definitions collected during the real-world deployment. Furthermore, we run a "replay simulation" based on the process construction logs that were collected during the live deployment to analyze the accuracy of the different algorithms when the insertion order of the operators is taken into account. The article ends with a discussion of previous works and an outlook on future developments.

2. CONTEXTUALIZED OPERATOR RECOMMENDATION TECHNIQUES

2.1. Existing Approaches

In Jannach and Fischer [2014], the results of an experimental evaluation of different operator recommendation schemes were presented. The evaluation was based on an offline experimental design in which the goal was to predict additional operators given a partial machine learning workflow. We will use similar experimental procedures in this article; details will be given in a later section.

The prediction approaches in Jannach and Fischer [2014] were mostly based on operator co-occurrence patterns within a larger pool of historical data analysis workflows. Among others, different association rule mining techniques like those proposed in Fournier-Viger et al. [2012] or Hu and Chen [2006] were evaluated and compared with a "popularity-based" baseline.

From their algorithms, we will use the following two as baselines, (i) a *k*-nearest neighbors method (KNN) because of its high accuracy and (ii) a computationally more simple method based on pairwise operator co-occurrences, which is used in the live deployment of the plug-in.

[2]The term *process* and *workflow* and the terms *machine learning workflow* and *data analysis workflow* are used interchangeably in the text.

*2.1.1. A K-Nearest-Neighbors Method (*KNN*).* The most successful approach in most settings tested in Jannach and Fischer [2014] was a weighted *k-nearest-neighbor* (kNN)-based technique, which was designed as follows.

(1) Neighborhood formation: Each historical process p was represented as a vector containing Boolean values. Each vector element corresponds to one of the operators in the process and is set to "1" if the operator is contained at least once in p. The similarity of two processes was then determined by calculating the cosine of the angle between the vectors.
(2) Scoring function: To determine the prediction *score* for an operator *op* for a given partial process $\hat{p}$, the k most similar processes to $\hat{p}$ were considered and the similarity values for those neighbors that contained the operator *op* were summed up. More formally, given a target process $\hat{p}$, its k nearest neighbors $N_k(\hat{p})$, and an operator *op*,

$$score(op, \hat{p}) = \frac{\Sigma_{p_i \in N_k(\hat{p})} sim(\hat{p}, p_i)}{|N_k(\hat{p})|}, \quad (1)$$

where $sim(\hat{p}, p_i)$ is zero if p_i does not contain *op*, and the cosine similarity of $\hat{p}$ and p_i otherwise.
(3) The operators are finally ranked and recommended based on their score as computed in Equation (1) in decreasing order.

The main idea of the *kNN* method is therefore that operators appearing in processes that are similar to the currently developed one are likely to appear in the current process as well.

*2.1.2. A Pairwise Co-Occurrences Method (*CoOccur*).* In Jannach and Fischer [2014], additional experiments with a quite simple method called CoOccur were made. The method relies on co-occurrence probabilities for any pair of operators, which can be efficiently determined offline by scanning all historical processes once. To create an operator recommendation list for a given process $\hat{p}$, we can iterate over all operators of $\hat{p}$ and return a ranked list of other operators that co-occurred with the elements of $\hat{p}$ most often. The experiments in Jannach and Fischer [2014] showed that CoOccur performed surprisingly well and was often better than the computationally more expensive rule mining techniques.

2.2. Context-Aware Operator Recommendation Approaches

The different techniques presented in Jannach and Fischer [2014] significantly outperformed the "popularity"-based baseline with respect to their ability of predicting which operator was hidden from an existing process during the evaluation procedure. The predictions made by both proposed algorithms (CoOccur and kNN) are either directly or indirectly based on co-occurrence patterns.

In this work, we build on these co-occurrence-based approaches but in addition try to take the current stage of the modeling process—the "context"—into account. The development of a complex data analysis workflow is typically an incremental and possibly iterative process. A recommendation tool should therefore take into account which part of the process the designer is currently working on.

Consider, for example, a situation in which the user is working on the details of a multielement *subprocess* of a larger process. The recommendations made in this context should therefore be based on the operators used in the subprocess; other operators appearing in the overall workflow might be less relevant. Another example is illustrated in Figure 2, where a process with multiple execution threads is shown. If we assume that the last user action was the insertion of the "FP-Growth" operator, then

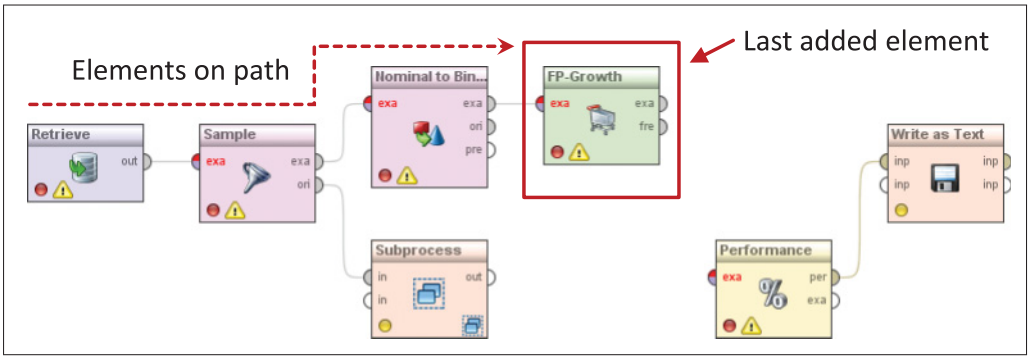

Fig. 2. The longest path of a RapidMiner process, which forms the *context* in our evaluation scheme.

it is reasonable to give more weight to operators in the recommendation process that co-occurred with this operator in the past. One assumption here is that the user is about to proceed with the construction of the process where the last operator insertion took place. Another reason to give more weight to these more "recent" operators is that maybe the user has already finished modeling the data preparation phase at the beginning of the process.

2.2.1. Adding Context Information. In the following, we propose new context-aware versions of the discussed methods, which receive the current modeling context as an additional parameter. In our experimental setup, we consider the operators that "precede" the hidden operator as the context. The context is defined as an *ordered* list of operators where the operators and their order can be determined in different ways.

—**Structure:** In our first set of experiments (Section 3), the context is derived from the structure of a given process. The list is determined by following the incoming edges of the hidden operator backwards until an operator is reached which has no input or a predefined maximum context length is reached. The ordering is therefore dependent on how the operators are connected in the process model. Our assumption is that the users build their processes left to right and that the way in which the operators are connected therefore corresponds to the insertion order of the operators. Thus, we assume that the last operator in the context is also the last operator that was inserted before the hidden one was introduced into the process model. In the example in Figure 2, we would pass this assumedly last added operator and the three other elements on the path to the recommender algorithm. In case there are multiple input paths for an operator, we return the elements on the longest path. If the last added operator is not yet connected with the rest, the additional context information would be empty.
—**Insertion Time:** In the second set of experiments, which we base on more detailed information about how the processes were actually created, we determine the ordered list of context operators using the insertion time of the individual operators. However, this approach requires the existence of the construction history of the processes. In Section 5, we will we compare the results obtained with the position-based context definition with the time-based one.

Note that while the context information passed to the recommendation algorithms can contain all the elements that were already inserted or are on the path, not all of them might be equally useful to adapt the recommendations. Operators appearing at

the very beginning of the context could even be misleading the algorithms as this part of the process model might already be completed. An algorithm could therefore try to focus on the operators that immediately precede the last operator, either based on the position in the graph or using a time-based metric.

In the following, we provide the details of a number of new context-aware operator recommendation schemes.

2.2.2. A Context-Aware kNN Method (kNN-Ctx). The general idea of the proposed kNN-Ctx method is to increase the weight (importance) of neighbors that contain an operator that is also part of the context of the currently constructed process, that is, we assume that these neighbors are better predictors for the current situation than others that are similar as well but have no overlap with the context. The score function shown in Equation (2) correspondingly has an additional parameter—the context—and uses an updated similarity function sim_{ctx},

$$score(op, \hat{p}, ctx) = \frac{\Sigma_{p_i \in N_k(\hat{p})} sim_{ctx}(\hat{p}, p_i, ctx)}{|N_k(\hat{p})|}. \tag{2}$$

Various ways of adding more weight to certain neighbors are possible. In our experiments we used a scheme which increases the original similarity by adding the similarity of the context. We additionally weight the context similarity with a factor α to determine the importance of the context in relation to the original similarity. The context-based weighting scheme is shown in Equation (3):

$$sim_{ctx}(\hat{p}, p_i, ctx) = sim(\hat{p}, p_i) + \alpha \cdot sim(p_i, ctx), \tag{3}$$

where $\alpha \geq 0$ and sim is again calculated as the cosine similarity, in this case between the context and the neighbor process p_i, which allows us to take the degree of overlap into account in the weighting scheme.

2.2.3. A Context-Aware Co-Occurrence Method (CoOccur-Ctx). Since the computationally simple but comparably effective method of looking at pairwise operator co-occurrences (CoOccur) led to good results in Jannach and Fischer [2014], we created a context-aware version of it (CoOccur-Ctx). The difference to the original method is that we only consider co-occurrence scores for operators that are part of the current context *ctx*. Because this context-based approach is independent of operators outside of the context, it requires even less computation time than its noncontextualized counterpart.

2.2.4. A Linked-Based Method (Link-Ctx). So far, we only looked at operator co-occurrences but did not take the structure of the processes into account. The method Link-Ctx is a first step to harness such structural process characteristics for recommendations.

Specifically, the idea, again, is to look at the current modeling context and count in the training set which other operators were *frequently linked* with the context elements. Following this approach, we try to rank those operators higher which are "closer" to the current modeling area. At the same time, we hope to rule out frequently used elements (e.g., for input data retrieval) which have high co-occurrence statistics but are no longer relevant at the current stage of modeling.

The algorithm Link-Ctx uses the following score function, in which P is the set of all training processes, *op* is the operator to be scored, and *ctx* is the given context (of the currently constructed process $\hat{p}$):

$$score(op, \hat{p}, ctx) = \Sigma_{p_i \in P} \Sigma_{op_j \in p_i} linkScore(op, op_j, ctx). \tag{4}$$

The function *linkScore* can implement different strategies to assign scores to the linked operators depending, for example, on the position of the compared element in the context. One possible strategy is to simply look at the last element of the context

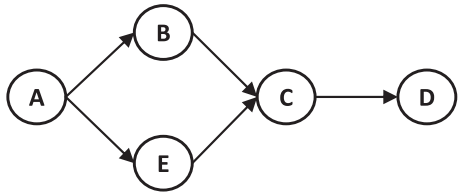

Fig. 3. An example process containing operators connected by edges forming various chains and sub-chains.

(*ctx.last*()) and return 1 if the operator *op* was a direct successor of this element in a training process or 0 otherwise (Equation (5)).

$$linkScore(op, op_j, ctx) = \begin{cases} 1, & \text{if}(op_j = ctx.last() \wedge linked(op_j, op)) \\ 0, & \text{otherwise} \end{cases}. \tag{5}$$

The score for a given operator *op* is therefore determined by the number of times it is a successor in all training processes to the last operator of the context. In case no or only a few connected operators can be found (e.g., because the last operator is a rarely used or it is a custom operator), we append elements from the standard CoOccur method to the recommendation list.

2.2.5. A Chain-Based Method (Chain-Ctx). The described Link-Ctx approach recommends items that are directly linked to the elements of the current editing context. This can be considered as a special case of finding operator "chains" of length 2. In this section, we propose the Chain-Ctx technique, which generalizes this idea. It searches for operator chains of larger sizes in the context and determines the most frequent successors of these chains in the training processes.

Let $chains(p, n)$ be a function that determines all existing chains with a minimum length of 2 and a certain maximum length n within a process p. Given, for example, the process definition p in Figure 3, the method $chains(p, 3)$ would then return the set {AB, AE, BC, EC, CD, ABC, AEC, BCD, ECD}.

Let us furthermore define a function $chainFreq(c, P)$, which returns how often a chain c appeared in the training data P. To compute this function, we have to count the occurrence frequencies of all existing chains and their subchains appearing in the training processes $p \in P$. Technically, to obtain these frequencies, we iterate over all processes p, set n to the number of operators appearing in the process, and then calculate $chains(p, n)$, which returns all chains and subchains of p.

At runtime, given a context ctx with a designated "last" element ($ctx.last()$), we first compute all operator chains C_{ctx} in ctx, this time using the context size as the maximum length n, that is, $C_{ctx} = chains(ctx, n)$, and then extract $C'_{ctx} \subseteq C_{ctx}$ as the set of chains that end with the element $ctx.last()$.

To compute the score for an operator *op*, context ctx, and target process $\hat{p}$, we extend each chain appearing in C'_{ctx} with *op* and sum up the occurrences of these concatenated chains to a score. For example, for a target process with the context (B, C) the set of context chains C'_{ctx} would be {BC, C}. Assuming our training set only consist of the process depicted in Figure 3 (making it the only element of P), we would extend the chains in C'_{ctx} with all known operators (A, B, C, D, E) to form the concatenations {BCA, BCB, BCC, BCD, DCE, CA, CB, CC, CD, CE}. A call to $chainFreq(c, P)$ would then return a zero score for all chains except *BCD* and *CD*, which were both contained once in our training process. Because both chains end with *D*, the operator *D* would receive a score of 2 while all other operators are scored with zero.

The formal definition of our scoring function is

$$score(op, \hat{p}, ctx) = \Sigma_{c_{cxt} \in C'_{ctx}} chainFreq(c_{cxt} + op, P) \cdot \lambda^{|c_{cxt}+op|}, \tag{6}$$

where λ is a parameter (with $\lambda \geq 1$), $|c_{cxt} + op|$ denotes the length of the chain $c_{cxt} + op$, and "+" is a concatenation operator. The additional factor $\lambda^{|c_{cxt}+op|}$ is introduced to return higher scores if longer chains in *ctx* are matched in the training processes. The parameter λ can be used to fine-tune the relative importance of matching operator chains of larger size. For the above-mentioned example this means that by increasing λ we can determine how much more important the chain *BCD* is in comparison to the shorter one *CD*.

Later, we will show that CHAIN-CTX leads to higher recommendation accuracy when compared to LINK-CTX, which is limited to chain lengths of size 2. Our experiments will also show that the required computation times of CHAIN-CTX are only marginally higher even for larger chain lengths.

2.2.6. A Hybrid Strategy (HYBRID-CTX). The last recommendation strategy evaluated in this article, HYBRID-CTX, combines the two contextualized strategies, KNN-CTX and LINK-CTX, because the two methods focus on different patterns in the training data.[3] We use a weighting strategy, where the weight factor w is dependent on the number of operators in the process. The more operators exist, the more weight is given to the KNN-CTX method. For shorter processes, our experiments show that the link-based method is preferable; for longer processes the kNN method performs better. Thus, we devised the following scheme to distribute the algorithm's weights:

$$w(\hat{p}) = \begin{cases} log_{10}(|\hat{p}|) \cdot \gamma, & \text{if } |\hat{p}| \leq \eta \\ \gamma, & \text{otherwise} \end{cases} \tag{7}$$

in which $|\hat{p}|$ corresponds to the number of operators in $\hat{p}$, γ is an empirically determined maximum weight, and $\eta \in \mathrm{N}$ is a threshold parameter. If the number of operators in the process is higher than η, then the maximum weight of γ is chosen. The score for a target operator *op* is given in Equation (8):

$$\begin{aligned} score(op, \hat{p}, ctx) \; = \; & w(\hat{p}) \cdot score_{kNN\text{-}Ctx}(op, \hat{p}, ctx) \\ & + (1 - w(\hat{p})) \cdot score_{Link\text{-}Ctx}(op, \hat{p}, ctx). \end{aligned} \tag{8}$$

3. ACCURACY EVALUATION USING EXISTING PROCESSES

Our first set of experiments is based on a pool of existing process definitions created by users of RapidMiner. For these process definitions, which were partially collected from a public website, only the (assumedly final) version is available. Since no information about the order of the insertion of the operators is available, the current modeling context is approximated based on the process structure as described in Section 2.2.1.

3.1. Evaluation Protocol

We adopt the evaluation approach from Jannach and Fischer [2014] and split the dataset into training and test subsets. The task of the algorithm is to predict one operator that was hidden from each of the processes in the test dataset. Tenfold cross-validation was applied in all experiments. The following protocol variants were examined.

—*Leave-one-out*: In this setting, we remove one operator from each process in the test set and use the following strategies to determine the element to be removed: RANDOM, FIRST, MIDDLE, SECOND-TO-LAST, LAST. Removing and predicting the first and the last element is, however, not very informative because these operators are typically

[3]We made additional experiments in which we combined KNN-CTX and CHAIN-CTX, which did not lead to better results.

related to raw data retrieval (e.g., from a file) and result output. In this article we will report the results of the SECOND-TO-LAST variant; the results are similar for the RANDOM and MIDDLE strategy.

—*Hide-last-two*: This is similar to SECOND-TO-LAST, but we additionally hide the very last element in the process. Again, the hidden second-to-last operator has to be predicted. We tested this specific variant to be able to better compare the offline analysis on historical processes with the data that we collected during the live deployment of the plug-in (for which the insertion history is known); see Section 5.

—*Given-n*: The idea of this evaluation variant is to remove all but the first n elements from the process. The operator that directly follows the nth operator has to be predicted. By increasing the value of n step by step, we can simulate the incremental development of the machine learning workflow by the user. This evaluation procedure can thus also help to determine how good the individual algorithms are in the "cold-start" setting, in which limited information is available for the recommendation process.

3.1.1. Determining the Context and the "Last" Inserted Elements. Besides the test processes from which certain elements were removed, the recommendation algorithms were also provided with the information about the current context as described above, that is, they received the list of operators that preceded the hidden one (see Section 2.2.1). In our first set of experiments, we will rely on a set of publicly shared and other process definitions, which means that no information is available about how the process was actually constructed. The underlying assumption in our approximation of computing the context of the last inserted element is that processes are generally created from "left-to-right" in the direction of the data flow. The context is therefore constructed by going backwards along the data flow path as described earlier. In a later section, we will revisit this assumption based on the data that is available from the live deployment, which includes the construction history.

3.1.2. Evaluation Measures. As evaluation measures, we use *Recall* and the *Mean Reciprocal Rank* (MRR) for the top-10 recommendation lists. In all measurements, we are interested to determine if an algorithm is capable of recommending one specific hidden element. In the *leave-one-out* and *hide-last-two* configurations, the goal is predict the single hidden (leave-one-out) or the first of the two hidden elements (hide-last-two). In the *given-n* configuration, the goal is to predict the immediate successor of the first n given elements.

Recall has therefore the value of "1," in case the recommendation algorithm manages to place the hidden operator in the top-n list. Otherwise, no hit was made and the Recall is zero. In this particular setup, Precision is proportional to Recall, that is, Precision is Recall divided by the recommendation list length [Cremonesi et al. 2010].

The MRR measure in addition takes the position of the "hit" in the list into account. The MRR value for a given recommendation is 0 if the hidden element is not in the list and $1/pos$ otherwise, where pos denotes the position of the hit in the recommendation list. Both for Recall and the MRR we report the mean value over cross-validation runs.

3.2. Dataset

The data that was used in the subsequently described evaluation consisted of a collection of more than 6,600 real-world machine learning workflows created with RapidMiner. The process definitions contained processes that were publicly shared by users on the "myexperiment" website,[4] the example process sets from the RapidMiner framework, and data from other sources. Since there were a number of duplicate processes

[4]http://www.myexperiment.org.

Table I. Dataset Characteristics for the Data Used in Section 3

Number of processes	5,463		
Number of unique operators	695		
	Mean	*Median*	*Std. dev.*
Operators per process	9.2	7	6.7
Unique operators per process	7.0	6	3.9
Duplicate operators per process	1.2	1	1.7
Longest connected path	6.2	5	4.2
Unconnected operators per process	13.3%		

stemming from, for example, duplicate posts on the forums, we applied a comparably strict deduplication strategy. Specifically, we considered process definitions to be duplicates when they contained the same operator sequences. Furthermore, we ignored all processes that contained fewer than 3 or more than 50 operators. The single-element and two-element processes are most probably incomplete processes and not suitable for evaluation (e.g., in the *given-n* protocol). Some outliers that contained a large number of operators were also removed as these models were most probably used only for test purposes by the users or contained multiple unconnected workflows in one model.

Table I shows some statistics of the final dataset after deduplication and filtering. RapidMiner allows the definition of subprocesses at several levels. In our experiments, we only considered the main process flow, that is, the top level workflow definition.

The following observations regarding the dataset characteristics can be made:

—Nearly 700 different operators were used at the top level of the processes (and over 800 on all levels). This confirms that finding the right operator can be challenging.
—On average, the processes comprise about 9 operators. However, half of the processes have 7 or fewer operators (median) but there are also some very large processes.
—The average number of unique operators per process is 7, which means that it is not unusual that an operator appears several times in a process definition. A typical example would be multiple data preprocessing steps.
—In at least half of the processes one operator appeared twice (row "duplicate operators per process," median).
—The longest connected path on average is 6.2 operators long, which is less than the average number of 9.2 operators in a process. This is caused by two reasons. First, some processes contain multiple branches or parallel execution threads and, second, there are process definitions that contain operators which are not connected with the other operators.
—To quantify this latter aspect, we measured how many operators in all process definitions were not connected with the rest (13.3%).

Another important aspect to note is that the distribution of how often certain operators are used is highly skewed. Many of the several hundred operators are only used in a small number of processes. Others, for example, those for data import, sampling, or the computation of derived attributes, are a substantial subset of the processes. As a result, using a popularity-based recommendation strategy and recommending the most frequently used operators can be a comparably hard baseline.

The distribution of operator usage frequencies is shown in Figure 4. To visualize the unevenness of the distribution, we sorted the operators based on their absolute usage frequency in the process pool in descending order and grouped them in bins of size 10. The first bin (leftmost bar in the chart) therefore contains the 10 most frequently used operators. Overall, about 40% of all operator usages in the processes correspond to these first 10 very general operators. On the other hand, the long-tail distribution

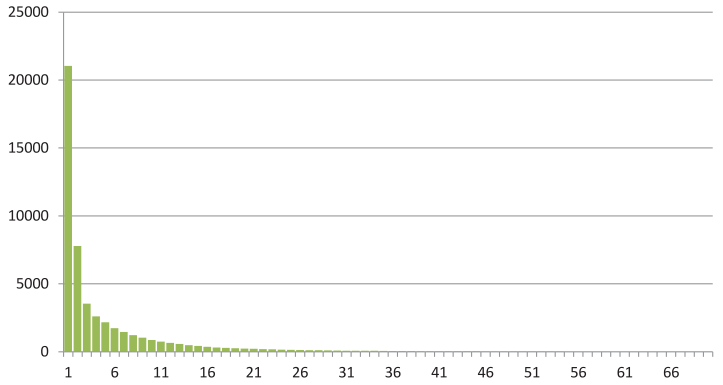

Fig. 4. Distribution of the usage frequencies of the operators. Each bar represents a bin of 10 operators.

Table II. Recall, MRR (Top-10), Training and Recommendation Time (TT & RT) for the Leave-One-Out and Hide-Last-Two Configurations

	Leave-one-out			Hide-last-two			
Algorithm	Recall	MRR	RT (ms)	Recall	MRR	RT (ms)	TT (s)
CoOccur	0.423	0.130	0.7	0.421	0.132	0.7	0.066
MostFreq	0.425	0.089	0.0	0.425	0.089	0.0	0.010
CoOccur-Ctx	0.468	0.158	0.5	0.468	0.158	0.4	0.062
Link-Ctx	0.690	0.420	0.9	0.690	0.419	0.9	0.082
kNN	0.773	0.250	2.8	0.725	0.222	2.7	0.020
Chain-Ctx	0.853	0.705	1.0	0.852	0.705	1.0	0.144
kNN-Ctx	0.861	0.582	2.7	0.832	0.566	2.3	0.016
Hybrid-Ctx	0.871	0.579	4.2	0.848	0.564	3.9	0.097

indicates that many of the operators are used fewer than 10 times in the several thousand process definitions.

3.3. Evaluation Results and Discussion

We compared the best performing algorithms from Jannach and Fischer [2014] with the context-aware ones proposed in the previous section using the above-described protocols and evaluation measures. In addition, we included a popularity-based baseline algorithm called MostFreq, which simply ranks the recommended operators based on their usage frequency in the training processes.

3.3.1. Results for the Leave-One-Out Setting. Table II shows the obtained Recall and MRR results using the best parameter settings for each algorithm, which we obtained through a manual fine-tuning process.[5] We chose a list length of 10, assuming that this corresponds to a number users would probably inspect and which will not require scrolling in the UI. Experiments with different list lengths did not lead to different results in terms of the ranking of the algorithms.

The numbers provided on the left-hand side of Table II correspond to a setting in which we hid the second-to-last element in the process. Different hiding strategies—for example, hiding a random element—led to different absolute values for the measures but not to a different algorithm ranking. The right-hand side of Table II shows the

[5]We determined optimal neighborhood sizes of 5 for the kNN method and 9 neighbors for kNN-Ctx, for which we used $\alpha = 100$. The optimal context-lengths were 2 (CoOccur-Ctx) and 1 (Link-Ctx). The values of 0.85 for γ and 5 for η led to the best results for the hybrid and $\lambda = 100$ worked best for Chain-Ctx.

results for the hide-last-two evaluation scheme, in which the last two operators were hidden and the second-to-last operator had to be predicted.

Besides the accuracy measures, we report the average time required to compute one recommendation list on a desktop computer equipped with 16 GB RAM and an Intel i7 CPU. We implemented the context-aware algorithms in such a way that a context-agnostic algorithm of the same family is used as fallback in case the contextualized variant did not produce enough results.

The results for the leave-one-out setting can be summarized as follows.

—The contextualized versions of the algorithms are generally significantly[6] better than their noncontextualized counterparts. The kNN method is the only "competitive" noncontextualized approach.
—In terms of the Recall, the hybrid method works best and is slightly but statistically significantly ($p = 0.02$) better than the contextualized kNN algorithm. In terms of the MRR, the difference between kNN-Ctx and Hybrid-Ctx is not significant. Considering longer operator chains, using Chain-Ctx works quite well especially in terms of the MRR. With regards to this measure it significantly ($p < 0.001$) outperforms all other algorithms.
—The popularity-based baseline MostFreq as expected (see Figure 4) achieves a Recall of about 40%. The simple CoOccur method is to some surprise not better in terms of the Recall than MostFreq but leads to significantly higher MRR values ($p < 0.001$).
—The recommendations can be computed very efficiently with all tested techniques. Computing one single recommendation list needed only 42 ms in the worst case. The low computation times for the kNN-based algorithms are partially due to our internal bitset encoding of the processes, which allows us to compute neighborhoods—typically a comparably costly operation—very quickly.

The observations for the hide-last-two setting mostly correspond with those for the leave-one-out scheme both in terms of the ranking as well as the absolute results. The only noteworthy difference is that in this configuration the Chain-Ctx method very slightly outperforms both kNN-Ctx and the hybrid method (but not significantly) in terms of the Recall. However, Chain-Ctx retains its advantage in comparison to all other algorithms in terms of the MRR ($p < 0.001$). This seems to be due to the fact that kNN-Ctx and Hybrid-Ctx show a weakened performance when only the operators left of the hidden one are revealed instead of both the left and the right neighbors. For the Chain-Ctx method the additionally revealed right neighbor serves no purpose because it works in a strictly left-to-right fashion. Thus, it is as accurate as for the leave-one-out scenario. Apparently, the kNN-based algorithms lead to slightly lower performance in the hide-last-two scheme because in this setup the target processes contain one element less that can be used to find neighbors than in the leave-one-out configuration. For the Chain-Ctx method, in contrast, the additionally revealed right neighbor in the leave-one-out configuration is irrelevant because only the process chains leading to the hidden element are considered. As a result, the differences between the kNN-based algorithms and the Chain-Ctx method are smaller in the hide-last-two setup.

The results moreover show that the Recall and MRR measures are not necessarily correlated and the results have to be interpreted with care. A higher value for Recall

[6]We use two-tailed Student's t-tests (paired or nonpaired, depending on the nature of the experiment) throughout the article with a $p < 0.05$ confidence level (if not specified explicitly otherwise) to test the statistical significance of any observed differences in the experiments. Bonferroni correction to account for multiple testing is utilized where applicable.

Table III. Recall/MRR for Different Given-*N* Configurations

	Given-1		Given-3		Given-5	
Algorithm	Recall	MRR	Recall	MRR	Recall	MRR
MostFreq	0.429	0.116	0.430	0.110	0.434	0.120
CoOccur	0.499	0.151	0.447	0.160	0.452	0.178
CoOccur-Ctx	0.499	0.151	0.468	0.183	0.495	0.187
kNN	0.440	0.106	0.592	0.164	0.679	0.173
Link-Ctx	0.644	0.297	0.690	0.389	0.706	0.391
kNN-Ctx	0.271	0.136	0.604	0.370	0.732	0.444
Hybrid-Ctx	0.644	0.297	0.707	0.391	0.798	0.453
Chain-Ctx	0.644	0.297	0.852	0.652	0.893	0.763

means that an algorithm was more often able to place the hidden element correctly in the top-10 list. The MRR measure, on the other hand, penalizes algorithms quite strongly if a "hit" occurred close to the end of the recommendation list. Considering the results shown in Table II this means that Hybrid-Ctx more often identified the hidden operator correctly, but Chain-Ctx had the correct element further up the list. The importance of the list position can be domain and application dependent. If users typically scan all recommendations, then the MRR value is probably less important. If the users stop after inspecting the first few items, then achieving a higher MRR value can be advantageous.

As we will see later (Section 5), the findings of this section will be corroborated by a rerun of our evaluation scheme on a larger real-world dataset. However, in Section 5 we will also present the results of a replay simulation on this real-world dataset and will observe that this more realistic evaluation scheme will partially lead to different observations.

3.3.2. Results for the Given-N Setting. Table III shows the results when using the *given-n* procedure to simulate the incremental development process. The following observations can be made.

—The structure-based Chain-Ctx is generally among the best-performing methods in the cold-start simulation, both in terms of the Recall and the MRR. In the extreme *given-1* setting, the results for the best three algorithms are identical by design, because Chain-Ctx with the maximum chain length of 1 is the same as the Link-Ctx method and the hybrid method embeds the Link-Ctx method.
—When more information about the processes is available (*given-3*, *given-5*), Chain-Ctx performs best with significantly better results (all $p < 0.001$) than the other contenders. However, the hybrid method shows performance values relatively close to the Chain-Ctx method.
—The CoOccur method is significantly ($p < 0.001$) better than the popularity-based baseline in the cold-start situation and even significantly ($p < 0.001$) better than the kNN method when the user is at the very beginning of the modeling process.
—Adding context information "hurts" only in the extreme *given-1* situation for the kNN method.

Note that the absolute values for Recall and MRR cannot be compared across the different evaluation configurations as different underlying dataset subsets were used. To be able to evaluate, for example, the *given-5* configuration, we only included those test processes in our evaluation which had at least six operators *on the longest path*. This reduced the set of usable test processes to about 2,000, which is a much smaller set

Table IV. Number of Recommended Operators (Catalog Coverage) for the Different Recommendation Schemes

	Number of recommended operators for ...				
Algorithm	Leave-one-out	Hide-last-two	Given-1	Given-3	Given-5
MostFreq	10.0	10.0	10.0	10.0	10.0
CoOccur	43.8	49.3	149.8	80.7	44.3
CoOccur-Ctx	104.7	104.7	149.8	115.7	88.4
Link-Ctx	189.3	190.7	197.9	201.9	164.0
Hybrid-Ctx	232.5	229.0	197.9	206.2	179.1
Chain-Ctx	251.4	252.8	198.0	265.4	218.7
kNN	280.1	279.9	185.4	280.9	231.9
kNN-Ctx	289.0	290.4	207.9	303.7	245.6

than the more than 5,000 processes that could be tested in the *given-1* configuration. The detailed dataset characteristics are given in the Appendix in Table XV.

3.3.3. Item Coverage and Popularity Bias. As a last part of our first evaluation, we compare the different operator recommendation techniques with respect to how strongly they adapt their recommendations to the currently developed process model. Comparable to the recommendation bias analyses provided in Jannach et al. [2013], we hypothesize that some algorithms have a stronger focus on more popular items (operators) or concentrate on a small set of operators in most situations.

Table IV shows the results of measuring (a) catalog coverage,[7] that is, how many different items ever appear in a top-10 recommendation list, and (b) the average popularity of the recommended items measured in terms of their occurrence frequency in the training data.

The results show that the differences between the algorithms can be quite strong and in particular the CoOccur method focuses its recommendations on a very small subset of comparably popular items. The kNN and kNN-Ctx methods, on the other hand, generate more diverse recommendations and cover the space of possible operators to a much larger extent.

In e-commerce scenarios, a strong focus on the most popular items can be undesirable, as it can lead to a "blockbuster effect" and a starvation of the niche items. In the operator recommendation scenario addressed in this article, focusing on the most popular items cannot necessarily be considered as negative, but it might influence how the recommendation plug-in is used. If the focus is on popular operators, then the plug-in could represent a convenient set of shortcuts to frequently used operators. If the recommendations are very specifically adapted to the current process model, then the plug-in can serve as a pointer to useful operators that the user might not have even thought of.

Table V shows how popular the recommendations of the different algorithms were on average. The results mostly correspond with those from Table IV, that is, algorithms that recommend a larger set of operators also tend to recommend on average less popular items.

3.4. Summary: Offline Analysis Using Existing Processes

Overall, the results of the offline evaluation indicate that combining different aspects like process structure and neighborhood-based co-occurrence patterns—as done in Chain-Ctx or Hybrid-Ctx—is the most promising approach to obtain high recommendation accuracy. In our ongoing work we are currently investigating the use of more complex structural patterns than the chains of operators that were used in our

[7]Sometimes this is called "aggregate diversity".

Table V. Average Popularity of the Operators Recommended by the Different Algorithms Based on the Usage Frequencies of the Operators in the Training Set

Algorithm	Popularity of the recommended operators for ...				
	Leave-one-out	Hide-last-two	Given-1	Given-3	Given-5
MostFreq	1893.9	1893.9	1893.9	1893.9	1893.9
CoOccur	1824.5	1819.5	1563.0	1775.3	1811.5
CoOccur-Ctx	1692.7	1692.7	1563.0	1721.5	1707.7
Hybrid-Ctx	1200.1	1210.0	1291.1	1293.0	1247.4
Link-Ctx	1161.7	1159.8	1291.1	1175.1	1161.4
kNN	1134.3	1148.2	1512.9	1360.4	1245.3
Chain-Ctx	1017.1	1015.2	1291.1	1012.1	1015.6
kNN-Ctx	981.3	979.2	904.7	979.2	1037.3

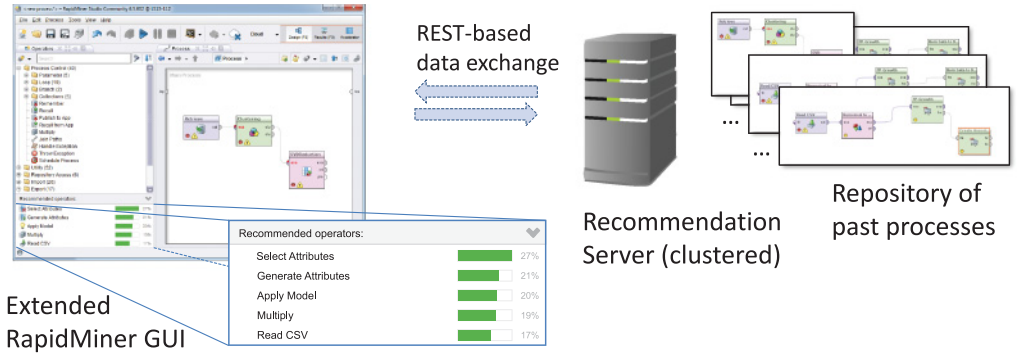

Fig. 5. Architecture of the recommendation service for RapidMiner. On the left, the RapidMiner UI with its plug-in recommender component is shown. The plug-in communicates with the recommendation server (right-hand side), which generates recommendations based on a repository of past processes and the user's currently modeled process. The communication is enabled through a web service interface.

experiments. Specifically, given that finding identical subgraphs in the processes is generally a computationally hard problem, we will look at heuristic techniques to identify similar structures in the processes.

4. LABORATORY STUDY

After performing the offline analysis on existing RapidMiner processes and before the deployment of the system to the live environment, we carried out a laboratory study in order to assess to which extent the automated system-side recommendation of operators can actually help to make the design process for data analysis workflows more efficient. To be able to conduct such a test, we developed a fully functional plug-in to RapidMiner. In a between-groups study design, half of the participants then used the plug-in to accomplish a certain task, while the other half did not receive any recommendations by the tool. Before we describe the details of the laboratory study, we will first outline the overall architecture of the recommendation service of RapidMiner (Figure 5).

4.1. Implementation Architecture

4.1.1. The End User's View. The recommendation service is visible to the end user (process designer) as one additional "view" (UI plug-in), which is integrated into the Graphical User Interface of RapidMiner, see Figure 5. Once activated, the plug-in monitors the user actions related to the process definition, in particular the insertion or removal of operators. After such a relevant user action is observed, the UI component collects

the information about the currently modeled process and forwards it to a remote recommendation service to receive a new set of recommendations.

The server responds with a ranked list of operators which—together with a confidence value[8]—is then presented in the designated area of RapidMiner. Figure 5 shows the default position of the recommendation lists, which is close to the existing tree- or search-based operator selection window. In case the user finds a relevant operator in the recommendation list, they can drag and drop the operator into the modeling window, which in turn leads to a request to the server for a new recommendation list.

4.1.2. An Adaptive Recommendation Service. The recommendation algorithms run on a dedicated server, which receives requests for recommendations from the clients over a REST and JSON-based Application Programming Interface. The requests contain the current partial process model developed by the user.[9] The server then creates the recommendations and also updates the new partial process received by the client in the pool of known processes. This helps us to constantly grow our "knowledge base" of training process models.[10] It furthermore allows us to make recommendations for new operators that are added to the RapidMiner system over time.

As stated earlier, each recommendation list can be computed in a fraction of a second. This nearly unnoticeable delay is more than sufficient to stay within the narrow time frame required in this interactive application setting. Scalability for situations of heavy parallel access to the recommendation server can be easily achieved by running the algorithms on a cluster of servers.

4.2. Experiment Design: Expectations

The goal of the user study was to obtain a better understanding of the value of the proposed recommendation service within RapidMiner and to get additional feedback from the users regarding, for example, the general usability. The experiment consisted of a workflow modeling exercise, in which half of the participants had to accomplish the task with the plug-in and the other half was not supported by the plug-in (i.e., the plug-in was not visible in the UI at all).

Inspired by the work presented in Koschmider et al. [2011], who made a comparable user study in the domain of business process modeling, we specifically aimed to verify the following expectations with our experiment.

Expected differences between the groups (with and without recommendation support):

—**E1** Participants that use the plug-in are more efficient than the other group in terms of required system interactions and the overall task completion time.
—**E2** Plug-in users are more confident in their solution as their solution was based on system-side recommendations.
—**E3** Participants that use our tool perceive the modeling exercise to be easier and the overall task to be more understandable than the other group.

Expectations within subgroups of plug-in users (experience/nonexperienced):

—**E4** Nonexperienced users find the recommendations to be more helpful than experienced ones and have a higher tendency to recommend the tool to others.

[8]The confidence values presented to the user depend on the chosen algorithm.
[9]All specific pieces of information like parameters, data source names, and so on, are removed for privacy reasons before the transmission to the server.
[10]This feature was switched off for the user study.

—**E5** The plug-in leads to efficiency improvements both for experienced and inexperienced users.

4.3. Experimental Procedure

The experiment had three phases. In the first phase, the users were instructed in the use of RapidMiner. In the second phase, the participants had to complete a partial workflow model according to a written specification. At the end of the experiment, the participants had to fill out a questionnaire.[11]

All subjects completed the task individually in the same office room at our department and used identical computer equipment. We screen-recorded the interactions of the subjects with RapidMiner. We furthermore encouraged the users to comment on what they are thinking during the exercise in the sense of a "think-aloud" protocol and recorded the user utterances.

4.3.1. Tutorial Phase. The goal of this phase was to make the participants acquainted with the RapidMiner system. We used detailed scripts that were used by the experimenter and the participant to make the process independent of the experimenter. In the tutorial, the participants completed a partial process with the tool. In case the participant was part of the group using the plug-in, the tutorial included a step that showed how to use the recommendations.

4.3.2. Modeling Exercise. All participants received the same detailed and written task description which did not mention the recommendation tool. The experimenter was not allowed to help the subject in any form. The experimenter was instructed to stop the experiment after a time limit of 15min was reached.

The problem to solve in the exercise consisted of a partial four-element data analysis workflow to which three more operators had to be added to achieve the described process goals that were specified in the task description. The exact names of the operators to be inserted into the process were not provided in the task description. Besides the retrieval of the correct operators, the participants were asked to connect the operators via their input and output ports. Overall, the process to model therefore contained seven operators, which is a realistic and common size as mentioned above. Figure 6 shows a screenshot of a correctly completed process.

At the server side, we used the kNN method from Jannach and Fischer [2014] to compute the operator recommendations since this was the best-performing method we had available at the time of the study.

4.3.3. Questionnaire. In the postexperiment questionnaire, we asked several questions to validate our hypotheses. We asked questions around the following topics:

—experience in data mining in general and with RapidMiner,
—the perceived complexity of the modeling task,
—the user's self-assessment of the quality of the final process and the level of encountered difficulties,
—and their satisfaction with the tutorial material.

The users that could use the recommendation plug-in were additionally asked

—whether they found the tool helpful,
—if they felt well-supported by the tool,
—and if they would recommend the use of the plug-in to others.

For all questions we used 5-point or 10-point Likert scales.

[11]The detailed questionnaire can be found in the auxiliary materials of Jannach et al. [2015] at https://dl.acm.org/citation.cfm?id=2701380.

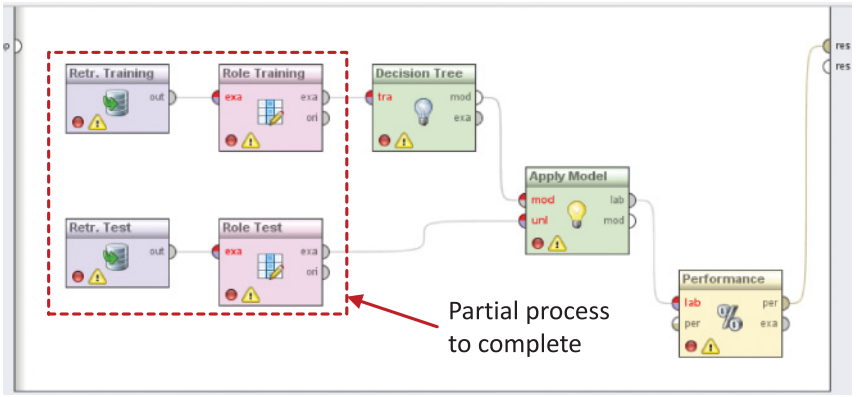

Fig. 6. The figure shows the process used in the modeling exercise of the user study. A partial process consisting of four operators was given as a starting point. The three rightmost operators had to be added to the process during the modeling assignment.

Table VI. Efficiency Results in Terms of Processing Time and Number of Clicks

Group	Time (sec)	Std. dev.	Clicks	Std. dev.
no plug-in	290.85	105.55	35.77	6.34
with plug-in	119.94	104.70	9.86	16.76

4.4. Participants

In total, 28 subjects aged between 20 and 30 participated in the study. Almost all were male undergraduate students of computer science; the rest had at least basic knowledge in the field of computer science. Of the 28 participants, 24 had at least some prior knowledge about data mining and 6 participants considered themselves data mining experts; however, only 6 participants had previous experience with RapidMiner.

The participants were split randomly into two groups of 14 subjects. One group could complete the modeling task with the help of the recommendation plug-in but they could also use the standard operator browsing and search feature of the tool. The other group did not have the plug-in at their disposal at all. We recruited the students via an announcement to a mailing list for CS students. We conducted a raffle and one participant received a price worth 100 € after the study was completed.

4.5. Results of the Laboratory Study

4.5.1. Efficiency of Modeling (E1). As a measurement method for testing our expectation E1, that is, for determining how quickly users can complete the task, we measured (a) the elapsed time until all required operators were correctly inserted by the participant and (b) the number of required clicks to insert the operators (either through the recommender or through the standard tree-based representation of RapidMiner). Table VI summarizes these observations.

Note that all subjects could successfully identify and insert the three missing operators within the 15-min time limit. However, not all subjects managed to correctly wire the operators in time. Since the purpose of the plug-in is to point users to relevant operators and is not related to the wiring of the operators, our measurement covers

Table VII. Confidence in Results

Group	Confidence in result	Std. dev.
no plug-in	3.36	1.23
with plug-in	3.93	0.64

Table VIII. Perceived Ease/Comprehension of Modeling Task

Group	Ease of task	Std. dev.	Comprehension of task	Std. dev.
no plug-in	3.42	0.88	3.29	0.82
with plug-in	3.72	0.88	4.29	0.96

the time from the beginning of the task until the moment when the operators were successfully inserted.[12]

The observations show that the users that were supported by the plug-in needed significantly ($p < 0.001$) less time and number of clicks to find and insert the correct operators. The average task completion time without the plug-in was close to 5min. The provision of recommendations helped to reduce the average time to about 2min. The comparably large variance in comparison to the completion time for the task observed for the plug-in users is explained by the fact that two of the subjects found and inserted the three operators into the model in less than 30s.

4.5.2. Confidence in Results (E2). For this measurement, we compared the self-reported confidence of the users after the experiment in the quality of their solution (Table VII). The mean value reported by plug-in users was at 3.93 on the 5-point Likert scale, which means that most of them were highly confident that their solution was correct. The mean value for users without recommendation support was slightly lower (3.36). A t-test reveals that the differences were not significant ($p = 0.14$). Nonetheless, we see this as some indication that the participants assumed that the recommendations made by the system were correct.

4.5.3. Perceived Task Complexity (E3). Our expectation before the experiment was that through the recommendation support, users (a) perceived the construction of the specific workflow to be less complex when guided by the tool and (b) that they found it overall clearer from the beginning what they were supposed to do in general, that is, find and connect the right operators and complete the workflow. The mean values of the provided answers in the questionnaire are shown in Table VIII.

As for the modeling complexity, plug-in users found the exercise on average (3.72) to be easier than the other group (3.42); note that higher values for this questionnaire item correspond to lower complexity. The differences were, however, not statistically significant and our expectation was therefore not fulfilled. However, plug-in users found the overall task significantly clearer (4.29 vs. 3.29 with $p = 0.01$) than users who were not supported by the tool. We see this as an indicator that the tool can be particularly useful for novice users as the system guides the user through the model development process.

4.5.4. Utility of Plug-In for Different User Groups (E4). We differentiate between experienced and nonexperienced users based on their self-reported assessment at the beginning of the experiment. We expect that nonexperienced users find the tool more helpful than

[12]Given that all participants at the end had correctly inserted the three required operators, no differences in the final solution quality were observed. Testing whether our tool leads to higher quality solutions, was not in the focus of the study. Furthermore, we report absolute duration values in Table VI; potential timing effects that can occur with think-aloud protocols were not considered.

Table IX. Relative Efficiency Improvement
(Experienced vs. Inexperienced)

Group	Experienced	Inexperienced
no plug-in	214.6	301.2
with plug-in	52.5	111.3
rel. improvement	75.5%	63.0%

experienced ones and would also have a higher tendency to recommend the tool to others.

The general acceptance of the tool was very high. The mean answer for the questions regarding the helpfulness of the recommendations in general and whether the users felt well supported by the recommendations was at about 4.8, which is close to the optimum on the given 5-point scale. The average value for the answer regarding the recommendation of the plug-in to friends was at 8.75 on a 10-point scale.

A correlation analysis using the Bravais-Pearson coefficient showed that the questionnaire answer regarding the helpfulness of the recommendations correlated with the self-reported expertise values. Lower expertise in RapidMiner, for example, led to higher values for perceived helpfulness ($r = -0.85$) and a higher tendency to recommend the tool to friends ($r = -0.56$). Lower general experience in data mining correlated with a slightly higher friend-recommendation tendency ($r = -0.23$).

In general, the mean self-reported value regarding experience in data mining was modest (2.46 on a 5-point scale). Further experiments involving a larger group of more advanced and experienced users are therefore required to better assess the helpfulness of the tool for this user group.

4.5.5. Efficiency Gains for Different User Groups (E5). To analyze whether both experienced and novice users profit from the tool, we made the following comparison. First, we split all participants into experienced users and nonexperienced ones. We used the mean self-reported value as a split point. Then we looked at the average time needed for these two groups when the plug-in was *not used*. Experienced users required about 217s and inexperienced about 301s. *With* the plug-in, the average time needed by experienced users went down to about 52s. Inexperienced users needed about 111s. As a result, the relative efficiency gain was about three fourths for experienced users and more than two thirds for inexperienced users. Table IX summarizes these effect.

Overall, the results suggest that efficiency gains are even stronger for experienced users, at least when they are given a task which might be relatively easy for them.

4.5.6. General Observations and Limitations of the Study.

General Observations Regarding the User Interfaces. According to the user utterances during the experiments, the UI of RapidMiner was generally considered to be quite complex in particular by the inexperienced participants. The default UI layout of RapidMiner contains six subwindows, of which four have more than one tab. At the time when the experiments were conducted, the recommendation functionality was implemented as an additional view, which can be activated and positioned floating on the whole screen or in docked mode within the UI.

The recommendations therefore further add to the complexity of the UI and "compete" with the other subwindows for the attention of the user. We conducted initial experiments regarding the best positioning of the recommendations because we observed in a pilot study that they were often overlooked by users. For example, users paid less attention to the recommendations when they were displayed on the right-hand side of the screen, even though this is a typical position for recommendations

on popular websites such as YouTube or Facebook. Even when positioned between the operator selection tree and the main drawing window—which was our choice for the experiments—some users only noticed the recommendations after the list was automatically updated when the first operator was added by the user.[13] Further experiments are therefore required to determine the optimal default position and a layout of the recommendations that does not overwhelm the users.

Research Limitations. The size of the user study is clearly a limitation of our research. As for the selection of the participants, we have recruited mostly computer science students. Some of them had some background in data mining, a small number already knew the RapidMiner tool, and one of them worked with it professionally. The group is therefore probably not representative of the average professional RapidMiner user. Still, graduates in computer science might be well representative of inexperienced (beginners) or infrequent users of RapidMiner.

In this study, we have mainly focused on the efficiency improvements that can be achieved through the recommendations for a specific modeling problem. A detailed study on how the recommendations possibly affect the quality of the resulting models has not been done so far. Furthermore, experiments with professional users—as planned in our future work—are needed to better assess the utility of the plug-in for experienced process modelers.

4.6. Summary: User Study

Overall, the results from the user study are promising as they indicate that the tool can measurably help to substantially increase the efficiency of the modeling process both for novice and expert users. Furthermore, the overall satisfaction with the tool was generally very high. The study finally indicates that the UI design has to be done with particular care and has to be self-explaining because of the already quite complex user interface of the RapidMiner toolkit.

5. AN ANALYSIS BASED ON REAL-WORLD RECOMMENDATION LOG DATA

After the user study was conducted, the software was further developed and, as of 2015, is shipped as a standard component of RapidMiner 6 that users can switch on during process design. In this section, we report first observations regarding the practical use of the system.

As a basis for our analysis, we use an entirely new dataset consisting of process definitions that were collected during the live deployment of the system based on the usage logs of the recommendation component. The data was collected during a live deployment of the tool which took place during a few weeks in June 2015.

We performed the following analyses:

(1) We measure how often the provided recommendations were actually accepted by the users in real-world use based on a sample of more than 3,300 processes that were created with the recommendation plug-in switched on.
(2) We repeat the offline analysis done in Section 3 using an even larger sample of more than 16,000 processes. These processes were created by users during the live deployment and help us validate that the results obtained in Section 3 generalize to other datasets.
(3) We conduct a replay-based accuracy analysis using the temporal information about the insertion of the operators. This measurement helps us assess to which extent

[13]On average, the participants selected about 2.4 (of 3) of the inserted operators from the recommendation plug-in.

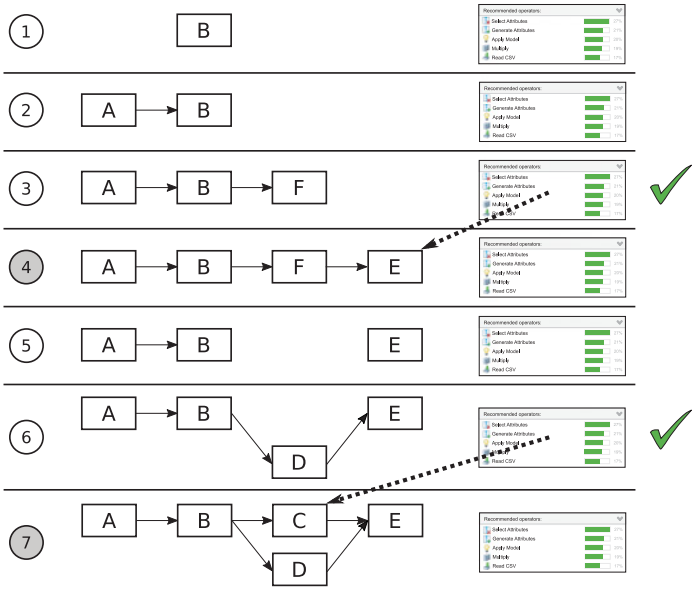

Fig. 7. Visualization of a process construction and recommendation log. Each row represents a modeling action by the user, which can either be the insertion or removal of an operator. The check marks on the very right and the dashed arrows indicate that the user incorporated one of the recommendations presented by the system.

the offline analysis done in Section 3 is realistic in particular with respect to the assumption on how people create the process models.

5.1. General Data Collection Method

The CoOccur method described in Section 3 was used as a recommendation technique on the company's server when the data sample was taken. Even though our results in Section 3.3 indicate that recommendations can be computed efficiently with the kNN method, our industrial partner preferred to rely on the even faster CoOccur technique in the live deployment. When the recommendation plug-in is switched on, it constantly monitors if the current process model is changed and sends a snapshot of the process to the server whenever an operator is added or deleted.[14] The server stores the snapshot and returns a list of recommendations along with a confidence value to be displayed by the plug-in. Whenever the user selects one of the operators in the recommendation list and adds it to the current process, this event is sent to the server, which again logs this information and returns a new set of recommendations.

Figure 7 visualizes the contents of the usage logs for one selected process. The example process was created in seven construction steps in which individual operators were added or removed. In each step, a list of recommendations for the current situation was created and displayed to the user. The detailed contents of the recommendation lists and the corresponding process "snapshot" are stored in the database. The green marks on the right-hand side of the figure at steps 3 and 6 indicate that the user has used one of the recommendations. The process snapshots from step 4 and 7 therefore contain one of the recommended elements.

A snapshot was taken whenever the set of operators of the current process was changed. For example, in step 5 the removal of an operator was also logged. Multiple

[14]Again, all parameters of the operators and other sensitive information are removed before transmission.

Table X. Dataset Statistics for the Data Collected During Live Deployment

Number of processes	3,337		
Number of recommendations	77,811		
Number of unique operators (1st level)	466		
Number of unique operators (incl. subprocesses)	600		
Fraction of unconnected operators per process (%)	9.7		
	Mean	*Median*	*Std. dev.*
Number of snapshots per process	43.1	9	201.2
Number of recommendations per process	22.3	5	56.1
Operators per process (1st level)	6.4	5	4.8
Operators per process (incl. subprocesses)	11.0	8	8.9
Unique operators per process (1st level)	4.9	4	2.8
Unique operators per process (incl. subprocesses)	8.0	7	4.9
Duplicate operators per process	0.9	0	1.4
Longest connected path	4.7	4	2.9

deletions at a time are also possible. Generally, the difference between two consecutive snapshots in the database is not necessarily limited to one operator because, for example, process modifications and recommendations are not logged when the user is not online during modeling. For privacy reasons, no information about the creators of the processes was stored.

5.2. Recommendation Usage Analysis

In the offline experimental analysis in Section 3, we have artificially hidden certain elements from the *final process models* and tested the capability of different algorithms to predict these hidden elements. With the available log data, we can now evaluate *for each step during the construction* of the processes if (a) the provided recommendations were accurate, that is, the right things were recommended, and (b) if the users actually used the recommendation plug-in to insert the operators.

5.2.1. Data Filtering and Final Dataset Characteristics. As already described in Section 3.2, a number of the recorded process snapshots are not particularly useful in our evaluation. These are, for example, processes which were most probably created for test purposes and which only contain one or two unconnected operators. We again removed such processes and their histories from our database and only retained those processes and their snapshots which had at least three and not more than 50 operators *in the last snapshot*.

The characteristics of the dataset resulting after removing these processes are given in Table X. The characteristics of the "final" processes are in many respects similar to those that were used in the analysis in Section 3. The average process length and the number of unique operators is generally slightly smaller for the processes that were collected in the live environment. One possible explanations is that some processes might not be finished by the users at the time when we collected the last snapshot. The distribution of the popularity of the individual operators are comparable to the long-tail distribution of the dataset used in Section 3. The average number of snapshots per process is about 43, although the median of 9 might be a better indicator of the usual amount a captured snapshots, because the relatively high standard deviation suggests that some outliers with very large snapshot histories exist.

5.2.2. Usage of Recommended Operators. We start with the most obvious measure, that is, how often users picked items from the recommendation list during the construction of the processes. We measure this in a straightforward way. Consider the example in Figure 7 above. There were *six* recommendations—we do not count the last one

as the process is considered to be finished—and in two cases the user adopted the recommendations. The fraction of used recommendations would therefore be $2/6 \approx 33.3\%$.

In our log dataset, we have recorded 77,811 ten-item recommendations. In about 7.8% of the cases, a user picked one of the recommended items and added it into the current process. In other words, in cases where the user turned on the plug-in, about every 13th operator inserted into the models was drag-and-dropped from the recommendation list and not looked up in the long existing "operator tree" UI component. The average position of the "accepted" recommendations in the list was at 3.36, that is, between the third and fourth positions (median $= 2$; $\sigma = 2.70$).

Note that in case that multiple operators of one single recommendation list were relevant in a certain situation (i.e., inserted in a later step into the process), we only increase the counter if the user actually picked one of them. After that, a new recommendation list will be created.

In our view this is a very promising result, given the hundreds of possible operators that can be inserted. It shows that already the recommendations created with a comparably simple method are helpful to the users. Furthermore, the recommendation UI component seems to be generally well accepted by users despite being a very recent addition to the already quite complex UI of the RapidMiner tool.

5.2.3. Theoretical Hit Rate. The previous measurement informed us about how many recommendations were actually picked by the users from the list displayed in the plug-in component. There might, however, be situations where the plug-in made useful recommendations but the user nonetheless decided to use the traditional operator tree widget to locate the needed operator. Different reasons for such a behavior are possible. Users might simply not notice the existence of the new plug-in, assume that the recommendations will not be helpful anyway, do not inspect the full 10-item list, or prefer to use the operator tree to see if there are alternatives for a desired functionality.

The previous measurement therefore represents a lower bound of the actually useful recommendations. To determine a "theoretical" hit rate, we made an additional analysis. Specifically, we counted how many of the operators that were part of the final process ever appeared in a recommendation list during the construction of the process. This measurement is therefore independent of any particular user preferences regarding the usage of the plug-in in the modeling process and works even when a user did not understand that they could directly drag-and-drop items from the recommendation list to the main modeling panel.

When taking the measurement, we first determine the set of unique operators in the final process, that is, we remove multiple occurrences of the same operator. Next, we determine for each snapshot and its corresponding recommendation list if the recommendation list contains an operator that appeared in the final process but is not yet part of the snapshot. Figure 8 shows an example. Besides the two "accepted" recommendations E and C, the user could also have picked A from the recommendation list in the very beginning. When calculating the theoretical hit rate, we therefore include the recommendation list in which A was recommended and end up with a $3/6 = 50\%$ success rate in the example. Note that in Step 2 operator F was also recommended and included in the next snapshot of the process. However, since F was later on removed again from the process, we do not count it as a successful recommendation.

The *theoretical* hit rate in our log sample is 19.1% . This result shows that the theoretical utility of the plug-in is much higher than what we observed in the previous measurement (7.8% of adopted recommendations) in which we could only make conjectures why some users did not adopt the recommendation.

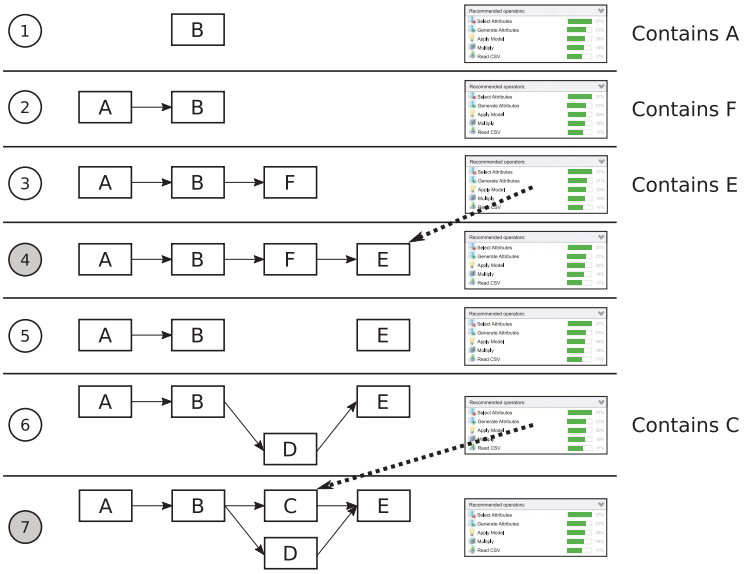

Fig. 8. Visualization of a process construction and recommendation log ("Theoretical" hit rate). In addition to Figure 7, we show on the very right which operators would in theory have been helpful—as they were inserted manually by the user—but were not picked from the recommendations.

Note that the theoretical hit rate cannot be directly compared with the results obtained in the offline analysis from Section 3 as a slightly different measurement method had to be applied. The repeated recommendation of operators that already exist in the process according to our offline evaluation leads to a strong increase in the hit rates; such repeated recommendations are, however, by design not counted in the theoretical hit rate. Additionally, we divide the number of theoretical hits by the number of operators to make the measure comparable with the previously mentioned fraction of accepted recommendations. There are, however, also many processes which were designed before the live experiment started, that is, they were already in a mature state before June 2015, and maybe only a small number of operators were changed during the experiment, leading to a small number of recommendations compared to the overall process size.

5.2.4. Removed Recommendations. In our last measurement, we determined how often users picked a recommendation from the list and later decided that the choice was not correct or optimal. Looking at Figure 8, such a situation would have happened if the user accepted recommendation F in step 2 but later decided to remove the operator again in step 5. In our log sample this fraction of accepted but later-removed recommendations is 34.2%.

The "normal" removal rate of operators that were added and later removed from the models, that is, when they are not picked from the recommendation list, is at 24.5%. This in general means that the operator recommender leads to a slightly increased insertion rate for operators that are later removed again. A deeper investigation of the reasons of this phenomenon is part of our ongoing work. One of our hypotheses is that inexperienced users are tempted to "try out" different recommended operators and thereby explore the options that the tool offers them. Another hypothesis could be that users pick one recommended operator because it is generally usable for the given problem (e.g., a certain and popular classification algorithm) and later decide to use a very specific variant of the algorithm when the process is further refined.

Table XI. Summary of the Recommendation Usage Analysis Based on the Recommendation Plug-In's Log Data Gathered in Live-Deployment

Number of recorded recommendations	77,811
Fraction of accepted recommendations (%)	7.8
Theoretical hit rate (%)	19.1
Fraction of accepted but later-on removed recommendations (%)	34.2
Operator removal rate when picked from operator tree (%)	24.5

5.2.5. Discussion. Table XI summarizes the observations from the live deployment. Overall, we can see that the general adoption of the tool after a few weeks of deployment is good and several thousand processes were built with the recommendation plug-in switched on. This observation is particularly promising because the plug-in is a very recent addition to the tool and competes with the traditional operator tree widget. In addition, the plug-in has to be explicitly activated before use and users have to agree that the anonymized process models are evaluated at the server side. The analysis furthermore shows that the recommendations can be truly useful for the users during process development.

As a next step, we plan to look closer into the logs to understand, for example, if there are stages in the development cycle in which the recommendations are particularly useful or if the recommendations are specifically valuable when the process models are either complex or comparably simple.

5.3. Accuracy Analysis without Sequence Information on New Dataset

The goal of the next analysis was to assess if the results obtained with "historical" process models reported in Section 3 can be reproduced on the data collected during the live deployment.

5.3.1. Harnessing Even More Data. The recommendation functionality was released in two phases. In a preliminary phase, we only recorded the process construction histories to obtain a dataset of reasonable size that can be used for learning the co-occurrence patterns. In the second phase, we started recording the recommendations and which of the recommendations were accepted. The data collected in this second phase was used in the analysis in the previous section and contained about 3,300 processes that fulfill the described length constraints.

Since we do not need the detailed usage logs of the plug-in component for our subsequent analysis, we can use all the process construction histories that were recorded during the live deployment, that is, also those from the preliminary phase.[15] This larger dataset comprises about 16,000 processes. The detailed characteristics are given in Table XII.

5.3.2. Accuracy Results. We reexecuted all evaluations reported in Section 3 (Recall, MRR, different protocol variants) on the new dataset. Table XIII shows the results for the hide-last-two protocol.

The results show that the ranking and performance of the different algorithms is mostly in line with the observations made in Section 3. Regarding the absolute magnitude of the values, we can observe that the Recall values for the algorithms are often higher, which can at least partially explained by the fact that more training data is available.

[15]This does not include the process definitions from the very first offline evaluation in Section 3.2, which were gathered from different sources including websites, but only process models based on logs of the live deployment.

Table XII. Statistics of the Dataset Collected During the Live Deployment Including the Data Gathered During the "Preliminary Phase" in which Only Data about Process Construction Was Collected But No Recommendations Were Displayed

Number of processes	16,516		
Number of unique operators	731		
Fraction of unconnected operators per process (%)	8.2		
	Mean	*Median*	*Std. dev.*
Number of snapshots per process (incl. processes with only one snapshot)	15.6	1	65.7
Number of snapshots per process (excl. processes with only one snapshot)	28.5	11	86.7
Operators per process	8.1	6	6.3
Unique operators per process	5.7	5	3.0
Duplicate operators per process	1.3	1	1.8
Longest connected path	5.6	5	3.5

Table XIII. Recall, MRR (top-10), Training and Recommendation Time (TT & RT) for the Hide-Last-Two Protocol on the Real-World Dataset. For Easier Comparison, the Results for the Previous Dataset, that is, the Dataset Containing Processes Gathered from the Web That Were Originally Reported in Table II, Are Repeated on the Right Side of the Table

	Real-world dataset				Previous dataset	
Algorithm	Recall	MRR	RT (ms)	TT (s)	Recall	MRR
MostFreq	0.501	0.118	0.0	0.032	0.425	0.089
CoOccur	0.537	0.156	0.9	0.206	0.421	0.132
CoOccur-Ctx	0.566	0.191	0.5	0.201	0.468	0.158
Link-Ctx	0.738	0.433	1.0	0.246	0.670	0.419
kNN	0.778	0.236	8.9	0.063	0.725	0.222
kNN-Ctx	0.782	0.510	9.1	0.055	0.832	0.566
Hybrid-Ctx	0.862	0.539	10.4	0.311	0.848	0.564
Chain-Ctx	0.890	0.745	1.1	0.389	0.852	0.705

In this measurement, the CoOccur method is also capable of outperforming the MostFreq baseline significantly ($p < 0.001$) on both measures. This might also be attributed to the fact that CoOccur improves stronger with larger training set sizes while the simple recommendation scheme of MostFreq does not benefit as much from a bigger training corpus.

The results for the other configurations (Leave-one-out, given-n) on the new dataset (see Table XVI in the Appendix) are also comparable to those obtained with the dataset of historical processes used in Section 3. The algorithm performances and the corresponding measurement results therefore seem to be stable across different datasets.

5.4. Accuracy Analysis with Sequence Information on the New Dataset: Replay Simulation

The accuracy analyses based on the completed process models, which we reported on in the previous subsection and in Section 3, can only to a certain extent approximate the true value and accuracy of the different recommendation techniques. In particular the context-based and structure-based methods rely on certain assumptions about how the processes are developed, that is, from left to right according to the data flow. The evaluation protocol might therefore lead to somewhat too-optimistic absolute Recall values as we hide certain elements in a chain and the algorithms can implicitly rely on the existence of an additional operator that continues the current operator chain.

Table XIV. Recall, MRR (top-10), and Recommendation Time (RT) for the *replay simulation* with the Hide-Last-Two Protocol. For Comparison Purposes, the Table Contains All Previous Hide-Last-Two Accuracy Results, that is, the Results from Table XIII for the Real-World Dataset But with the Evaluation Scheme That Is Agnostic of the Insertion Time and from Table II for the Previous Dataset with the Same Evaluation Scheme

	Real-world dataset Replay evaluation			Real-world dataset Classic evaluation		Previous dataset Classic evaluation	
Algorithm	Recall	MRR	RT (ms)	Recall	MRR	Recall	MRR
Chain-Ctx	0.376	0.189	1.1	0.890	0.745	0.852	0.705
Link-Ctx	0.377	0.174	1.1	0.738	0.433	0.690	0.419
MostFreq	0.500	0.242	0.0	0.501	0.118	0.425	0.089
CoOccur	0.535	0.262	0.8	0.537	0.156	0.421	0.132
Hybrid-Ctx	0.547	0.224	11.5	0.862	0.539	0.848	0.564
CoOccur-Ctx	0.568	0.285	0.4	0.566	0.191	0.468	0.158
kNN	0.675	0.228	10.2	0.778	0.236	0.725	0.222
kNN-Ctx	0.695	0.238	13.8	0.782	0.510	0.832	0.566

In reality, users might, however, follow different strategies of how they develop the process. In the subsequent experiments we will therefore use the known sequence of operator insertions as a basis to hide and predict the operators instead of the position in the data flow.

Specifically this means for the evaluation protocols:

—When elements are hidden (*hide-last-two*) or retained (*given-n*), the selection of the hidden and given elements is based on the chronological insertion order of the operators.
—The information about the recent modeling *context* provided to the context-aware algorithms is no longer based on structural information but on the insertion sequence of the operators that were added to the process immediately before the hidden element.

5.4.1. Results for the Hide-Last-Two Configuration. Table XIV shows the accuracy results for the *hide-last-two* configuration when the time-based insertion order is used to hide the last two elements. A comparison with the results shown in Table XIII—where no time information was used—reveals that the algorithms that were relying on structural information (Link-Ctx, Chain-Ctx, Hybrid-Ctx) do not perform well anymore using the time-based evaluation protocol. The kNN algorithm and its contextualized variant show the best results in terms of the Recall, with kNN-Ctx being slightly but significantly ($p < 0.001$) better than kNN. In terms of the MRR, the popularity-based baseline is generally hard to beat. Only the co-occurrence-based methods (CoOccur, CoOccur-Ctx), which also have a strong popularity bias, lead to competitive results and are significantly better than the other techniques (all $p < 0.02$). The results are similar for the *given-n* configurations; see Table XVII in the Appendix.

5.5. Discussion of the Observed Differences

These observations indicate that the offline evaluation protocol that was used in Section 3 and Section 5.3 might make a too-strong assumption about how users create the process models, that is, in a left-to-right order following the data flow in the process. The recommendation methods that are based on the existence of such chains (including Link-Ctx or Chain-Ctx) only work well when this assumption holds.

In the offline experimental evaluations in which no sequence information was available, this assumption was implicitly encoded in the protocol. Whenever an item, for example, close to the end of a process (*leave-one-out)*, was hidden, we generally assumed that this element was connected with a sequence of preceding operators. We

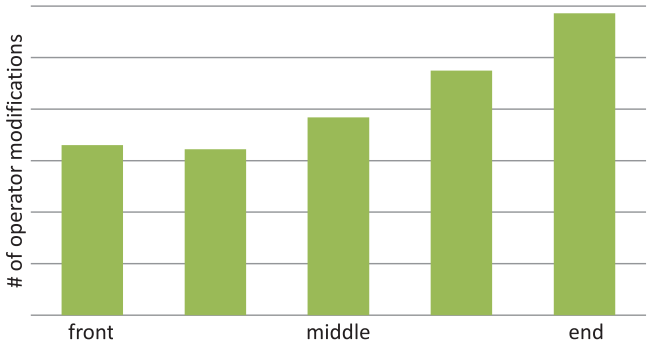

Fig. 9. Distribution of the positions in the process at which operators were added or deleted.

could therefore always assume that a successor of the current modeling context exist. Similarly, in the *given-n* protocol, we assumed that there is a chain of at least $n + 1$ connected operators in the tested processes. Overall, in many cases, it seems that there is some logical connection between these operators, which is why looking for similar sequences or chains in the training processes worked so well.

To understand how users actually create their process, we analyzed at which positions of the currently modeled process users add the operators. To that purpose, we split the each process snapshot in five "areas" (from front to end) and counted the insertions and deletions identified in the subsequent snapshot for each position. Figure 9 shows the resulting distribution, which indicates that people actually have some tendency to extend the processes at the "end" (right-hand side according to the data flow), but insertions at the beginning or middle are also quite common. Furthermore, we measured how many of the operators in each analyzed snapshot were not connected with the rest and found that between 15% and 30% of the operators were unconnected in the *given-n* configurations.[16] Some users therefore seem to prefer to first add a number of operators and add the data flow connections only later on.

As a result, the link structure of the developed processes is not the best predictor for the next position of an operator insertion in the replay setting and both structure-agnostic *kNN* methods (kNN and kNN-Ctx), which we used as a basis in the laboratory study, show the best results in terms of the Recall. Note, however, that this measurement result does not necessarily mean that the recommendations of the structure-based approaches are not useful in reality. The measurement only tells us that some of the users did not consistently continue the development of data processing chains but edited the processes at varying places. The structure-based methods actually *might have* included a suitable operator, but the users did simply decide not to continue to model the data flow of the last inserted operator but worked on other parts of the process first.

Similarly to the experiments done without sequence information in Section 3, we can, however, observe that the context-aware methods CoOccur-Ctx and kNN-Ctx perform better than their noncontextualized counterparts. This corroborates our findings that context information can be important and should be considered in the recommendation models. Note, however, that the context in the sequence-aware evaluation scheme used here is not defined based on structural properties of the process but on the insertion sequence of the operators.

[16]In the *given-n* evaluations that were only based on the final processes, the given operators were always assumed to be connected, leading to an advantageous situation for link-based techniques.

Nonetheless, a structure-based approach of defining the context can in theory be very effective in case users follow a left-to-right modeling approach as shown in Section 3. We therefore currently explore the use of "dynamic" schemes which adapt the selection of the context elements to be considered based, for example, on the user's modeling behavior or the connectedness of the currently developed process.

Overall, our offline analyses and the replay simulation revealed some important insights regarding the importance of the context and the users' modeling behavior, which should be considered when making recommendations. Some other aspects can, however, only be analyzed through A/B tests in which different methods are compared with each other with real users. This, for example, includes the common question in recommender systems research if small differences in the offline accuracy results actually lead to measurable differences with respect to the user acceptance or the effectiveness of the plug-in in practice.

5.6. Real-World Evaluation: Limitations and Future Works

The results of our analyses reported in this article are based on a comparably short period of time after the new functionality was released in RapidMiner. The user base might therefore be a subgroup of *early adopters* and have some characteristics that differ from the general user population of RapidMiner.

Furthermore, due to the preferences of our industrial partner, only the comparably simple CoOccur method was used at the server side in this initial phase of the live deployment. No true A/B test with different versions of the tool and a control group without the tool was possible so far in a live setting in particular as no process snapshots are taken when the recommendation plug-in is *not* activated. The incorporation and evaluation of algorithms which are more accurate than CoOccur in the offline analysis are part of our ongoing work.

A comparison of novice and expert users as done in Section 4 was so far not possible as well, as the identities of the users are not revealed for privacy reasons. Nonetheless, in our future work we will try to identify patterns of how users create their processes (e.g., left-to-right vs. inside-out etc.) and try to categorize and cluster users based on their modeling behavior. Based on such patterns, the use of a *personalized* selection of the most promising recommendation strategy for a particular user could be possible.

Finally, a survey-based evaluation about the perceived usefulness and value of the recommendation plug-in of real users has not been done so far but is planned as one of the immediate next steps.

6. RELATED WORK

The use of recommendation techniques to support the user in the specific area of modeling data analysis workflows has—to the best of our knowledge—not been explored so far. There are, however, a number of research works that have focused on techniques for assisting the user in the development of more general process models, in particular for business processes. According to the overview of such approaches in this area from Kluza et al. [2013], our work would fall into the category of *single-element, structural recommendations*. Our Given-*n* simulation corresponds to a *forward-completion* approach.

Koschmider et al. [2011] present a visual and recommendation-based modeling system for business processes. The environment supports both a search interface for process model fragments based on semantic annotations (tags) as well as a recommendation-based ranking component which is based on a combination of factors such as a query-based score, process fragment usage frequencies, and indicators of the structural match or the quality of the recommended fragments. The work is similar to

ours in that the recommendations are based on a repository of past models (fragments) and that the current modeling state is essential to determine suitable suggestions for process extensions. Similarly to our work, the authors conducted a laboratory study involving 24 pairs of students who had to accomplish a modeling exercise with and without tool support to assess the value of the recommendation support.

The evaluation in Koschmider et al. [2011] was based only on one recommendation technique and a comparably small repository of process fragments that were developed for the particular modeling exercise. In our work, in contrast, we could rely on a pool of thousands of real-world workflows and additionally compare a number of different strategies in an offline experimental design based on process snapshots from the live deployment of the plug-in.

Generally, however, there are several approaches in the area of business process modeling support that we plan to explore in the future in our specific application domain. This includes, for example, the use of semantic information to identify the user's modeling *intention*, see Hornung et al. [2008] or Koschmider et al. [2011]. In our application domain, semantic information is, for example, available in terms of operator descriptions, user-specified operator names, or the given operator hierarchy.

Furthermore, we plan to go beyond our first link-based analyses and explore more complex *structural* patterns in the currently developed process and the process fragments in the repository. Such analyses were done, for example, in Li et al. [2014] through an efficient encoding of the graph matching problem or can rely on other process similarity measures as proposed in Dijkman et al. [2011].

Several approaches in the literature like the ones mentioned above or the Bayesian Networks one proposed in Bobek et al. [2013] aim at the *recommendation of process fragments* or more complex structures instead of single elements (operators) as done in our approach. In our application domain, we plan to specifically look at patterns in subprocesses that are used for common tasks like cross-validation. The possibly required adaptation according to the specific process could then be supported, for example, by a case-based reasoning approach as proposed in Minor et al. [2010].

Finally, some other modeling support approaches in the literature in the domain of business processes aim to the suggestion of appropriate textual names [Leopold et al. 2011] or present recommendations based on the syntactical rules of the used modeling language [Mazanek and Minas 2009]. These approaches seem to have limited applicability in our domain, in which (a) the modeling language has only a few syntactic rules and operators have quite a limited set of connection types and (b) the activities already have defined semantics and names.

An alternative path potentially worth exploring in the context of our work is to apply ideas from "information foraging," where the goal is to aid the user in their "'hunt for information" (see Piorkowski et al. [2012]). For our scenario, this means that instead of supporting the search process by immediately providing potential results, our tool would rather support the user in their natural search process, for example, by highlighting search paths that contain the most promising operators. Such an approach may therefore help to combine the advantages of the user's own search abilities with the information reduction capabilities of the recommender algorithms.

Since the mid-2000s, interorganizational business processes, service oriented architectures, and the integration of applications through web services gained in importance. Recommendation-based modeling support approaches have been applied in that context, for example, for process completion, the recommendation of alternatives for broken services, or web service discovery [Chan et al. 2011, 2012; Sellami et al. 2009]. The discovery approach presented in Chan et al. [2011], for example, tries to identify fragments (composite services) in past processes that are *structurally* similar to the currently modeled one. In contrast to our work, the evaluation in Chan et al. [2011]

was made on a small set of purposely created fragments, while in our work we can rely on a larger set of real-world process definitions.

A different strategy for web service discovery was proposed in Chan et al. [2012]. Instead of relying on text-based (content-based) matching as done, for example, in Dong et al. [2004], they try to apply historical usage data and classic collaborative and content-based filtering techniques to determine suitable recommendations, for example, by comparing user profiles. Our kNN method has some similarity with the process-similarity-based approach in Chan et al. [2012]. We have, however, not looked at the behavior of individual users over time in our approach, as no author information is available in our workflow repository and collecting such information raises privacy issues.

A different approach to generate recommendations using historical data was made in Chan et al. [2014], where process *execution logs* were used instead of the models as a basis for mining co-occurrence patterns. Thereby, actual usage frequencies are taken into account, which is also the case in our recommendation techniques. Similarly to our work, Chan et al. [2014] propose to take the current modeling context into account and to generate recommendations depending on the selected activity in the workflow. Their evaluation was based on artificially created execution logs as real-world execution logs were not available. Generally, estimating the "relevance" of individual workflow models based on their usage (execution frequencies) could also be an interesting approach in our domain if such information were available.

Finally, adaptive action recommendation mechanisms that are based on historical interaction data can also be found in other types of software applications not related to process modeling. One example is the *CommunityCommands* system proposed in Matejka et al. [2009], which recommends new software functionality in terms of helpful commands to users of the AutoCAD design software. As a recommendation technique, user- and item-based collaborative filtering techniques were evaluated based on automatically collected command usage data. Similarly to our work, *CommunityCommands* in some form recommends user actions.[17] The main goal of the AutoCAD extension, however, is to point users to newly introduced functionality in the system in a personalized way. Even though our approach can also help users discover new operators, our work aims to point the user to operators that are supposed to be helpful in the current modeling context rather than recommendations based on the user's past actions.

7. CONCLUSIONS

We have presented an extension to the widespread RapidMiner software environment for modeling data analysis workflows, which recommends additional operators to the user during the modeling process. An offline evaluation revealed that taking the current modeling context into account is advisable to increase the prediction accuracy. A first user study showed that the tool actually helps both experienced and inexperienced users to develop the processes in a more efficient way. First results regarding the adoption and accuracy of the tool in a live deployment are promising. A replay-based simulation reveals that the used offline evaluation protocol can have some limitations and more work is required to develop approaches that help us predict the success of a strategy based on offline analyses. In our future work, we will further explore semantic and more complex structure-based approaches to identify patterns in the data and investigate the recommendation of process fragments in addition to the current single-element recommendations. Furthermore, we plan to conduct A/B tests in the live environment and instrument a user survey to obtain further feedback from the users regarding the usability and their satisfaction with the tool.

[17]See also Davison and Hirsh [1998] for an earlier work on predicting user actions.

APPENDIX

Table XV. Characteristics of the Test Data Subsets When Evaluating with Different Protocols (Section 3)

	Subset statistics of the test processes for the protocol ...														
	Leave-One-Out			Hide-Last-Two			Given-1			Given-3			Given-5		
Number of processes	3,090			3,090			5,371			3,916			1,992		
Number of unique operators	512			473			151			327			264		
Frac. of unconn. op. per proc. (%)	21.3			12.0			100.0			0.0			0.0		
	Mean/*Med.*/σ			*Mean*/*Med.*/σ			*Mean*/*Med.*/σ			*Mean*/*Med.*/σ			*Mean*/*Med.*/σ		
Operators per process	11.4	10	7.0	10.4	9	7.0	1.0	1	0.0	3.0	3	0.0	5.0	5	0.0
Unique operators per process	8.3	7	3.7	7.5	7	3.8	1.0	1	0.0	2.9	3	0.3	4.6	5	0.7
Duplicate operators per process	1.7	1	1.9	1.6	1	1.9	0.0	0	0.0	0.1	0	0.3	0.3	0	0.5
Longest connected path	6.9	6	4.2	6.2	5	4.2	1.0	1	0.0	3.0	3	0.0	3.0	3	0.0

Table XVI. Recall/MRR for Different Given-*N* Configurations on Data Collected During Live Deployment *without* Sequence Information (Section 5.3)

	Given-1		Given-3		Given-5	
Algorithm	Recall	MRR	Recall	MRR	Recall	MRR
MostFreq	0.496	0.154	0.443	0.107	0.429	0.094
CoOccur	0.543	0.177	0.467	0.145	0.457	0.157
CoOccur-Ctx	0.543	0.177	0.491	0.174	0.511	0.196
kNN	0.514	0.111	0.597	0.167	0.662	0.191
kNN-Ctx	0.191	0.095	0.508	0.341	0.695	0.445
Link-Ctx	0.648	0.326	0.692	0.388	0.696	0.404
Hybrid-Ctx	0.648	0.326	0.692	0.371	0.784	0.462
Chain-Ctx	0.648	0.326	0.858	0.639	0.894	0.769

Table XVII. Recall/MRR for Different Given-*N* Configurations for the Replay Simulation (Section 5.4)

	Given-1		Given-3		Given-5	
Algorithm	Recall	MRR	Recall	MRR	Recall	MRR
Link-Ctx	0.254	0.124	0.331	0.154	0.314	0.136
Chain-Ctx	0.254	0.124	0.343	0.164	0.325	0.154
MostFreq	0.411	0.266	0.427	0.226	0.396	0.198
Hybrid-Ctx	0.254	0.124	0.407	0.182	0.410	0.164
CoOccur	0.438	0.210	0.460	0.231	0.420	0.203
CoOccur-Ctx	0.438	0.210	0.474	0.226	0.445	0.197
kNN	0.466	0.212	0.515	0.196	0.493	0.172
kNN-Ctx	0.477	0.208	0.524	0.199	0.507	0.176

REFERENCES

Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. 1993. Mining association rules between sets of items in large databases. In *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data (SIGMOD'93)*. 207–216.

Szymon Bobek, Mateusz Baran, Krzysztof Kluza, and Grzegorz J. Nalepa. 2013. Application of Bayesian networks to recommendations in business process modeling. In *Proceedings of the 2013 Workshop AI Meets Business Processes (AIBP'13)*. 41–50.

Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. 2011. Composition context matching for web service recommendation. In *Proceedings of the 2011 IEEE International Conference on Services Computing (SCC'11)*. 624–631.

Nguyen Ngoc Chan, Walid Gaaloul, and Samir Tata. 2012. A recommender system based on historical usage data for web service discovery. *Serv. Orient. Comput. Appl.* 6, 1 (2012), 51–63.

Nguyen Ngoc Chan, Karn Yongsiriwit, Walid Gaaloul, and Jan Mendling. 2014. Mining event logs to assist the development of executable process variants. In *Proceedings 26th International Conference on Advanced Information Systems Engineering*. 548–563.

Paolo Cremonesi, Yehuda Koren, and Roberto Turrin. 2010. Performance of recommender algorithms on top-n recommendation tasks. In *Proceedings of the 2010 ACM Conference on Recommender Systems (RecSys'10)*. 39–46.

Brian D. Davison and Haym Hirsh. 1998. Predicting sequences of user actions. In *Proceedings of the AAAI/ICML '98 Workshop on Predicting the Future: AI Approaches to Time Series Analysis (AAAI'98)*. 5–12.

Remco Dijkman, Marlon Dumas, Boudewijn van Dongen, Reina Käärik, and Jan Mendling. 2011. Similarity of business process models: Metrics and evaluation. *Inform. Syst.* 36, 2 (2011), 498–516.

Xin Dong, Alon Halevy, Jayant Madhavan, Ema Nemes, and Jun Zhang. 2004. Similarity search for web services. In *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB'04)*. 372–383.

Philippe Fournier-Viger, Usef Faghihi, Roger Nkambou, and Engelbert Mephu Nguifo. 2012. CMRules: Mining sequential rules common to several sequences. *Knowledge-Based Syst.* 25, 1 (2012), 63–76.

Jiawei Han, Jian Pei, and Yiwen Yin. 2000. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data (SIGMOD'00)*. 1–12.

Thomas Hornung, Agnes Koschmider, and Georg Lausen. 2008. Recommendation based process modeling support: Method and user experience. In *Proceedings of the 27th International Conference on Conceptual Modeling (ER'08)*. 265–278.

Ya-Han Hu and Yen-Liang Chen. 2006. Mining association rules with multiple minimum supports: A new mining algorithm and a support tuning mechanism. *Decision Support Syst.* 42, 1 (2006), 1–24.

Dietmar Jannach and Simon Fischer. 2014. Recommendation-based modeling support for data mining processes. In *Proceedings of the 8th ACM Conference on Recommender Systems (RecSys'14)*. 334–340.

Dietmar Jannach, Michael Jugovac, and Lukas Lerche. 2015. Adaptive recommendation-based modeling support for data analysis workflows. In *Proceedings of the 20th International Conference on Intelligent User Interfaces (IUI'15)*. 252–262.

Dietmar Jannach, Lukas Lerche, Fatih Gedikli, and Geoffray Bonnin. 2013. What recommenders recommend - An analysis of accuracy, popularity, and sales diversity effects. In *Proceedings of the 21st International Conference on User Modeling, Adaptation and Personalization (UMAP 2013)*. Rome, Italy.

Krzysztof Kluza, Mateusz Baran, Szymon Bobek, and Grzegorz J. Nalepa. 2013. Overview of recommendation techniques in business process modeling. In *Proceedings of 9th Workshop on Knowledge Engineering and Software Engineering (KESE9)*. 46–57.

Agnes Koschmider, Thomas Hornung, and Andreas Oberweis. 2011. Recommendation-based editor for business process modeling. *Data Knowledge Eng.* 70, 6 (2011), 483–503.

Henrik Leopold, Jan Mendling, and Hajo A. Reijers. 2011. On the automatic labeling of process models. In *Advanced Information Systems Engineering*. Lecture Notes in Computer Science, Vol. 6741. 512–520.

Ying Li, Bin Cao, Lida Xu, Jianwei Yin, Shuiguang Deng, Yuyu Yin, and Zhaohui Wu. 2014. An efficient recommendation method for improving business process modeling. *IEEE Trans. Indust. Inform.* 10, 1 (2014), 502–513.

Justin Matejka, Wei Li, Tovi Grossman, and George W. Fitzmaurice. 2009. CommunityCommands: Command recommendations for software applications. In *Proceedings of the 22th Annual ACM Symposium on User Interface Software and Technology (UIST'09)*. 193–202.

Steffen Mazanek and Mark Minas. 2009. Business process models as a showcase for syntax-based assistance in diagram editors. In *Model Driven Engineering Languages and Systems*. Lecture Notes in Computer Science, Vol. 5795. 322–336.

Mirjam Minor, Ralph Bergmann, Sebastian Görg, and Kirstin Walter. 2010. Towards case-based adaptation of workflows. In *Case-Based Reasoning. Research and Development*. Lecture Notes in Computer Science, Vol. 6176. 421–435.

David Piorkowski, Scott Fleming, Christopher Scaffidi, Christopher Bogart, Margaret Burnett, Bonnie John, Rachel Bellamy, and Calvin Swart. 2012. Reactive information foraging: An empirical investigation of theory-based recommender systems for programmers. In *Proceedings of the 2012 Conference on Human Factors in Computing Systems (CHI'12)*. 1471–1480.

Mohamed Sellami, Samir Tata, Zakaria Maamar, and Bruno Defude. 2009. A recommender system for web services discovery in a distributed registry environment. In *Proceedings of the 4th International Conference on Internet and Web Applications and Services (ICIW'09)*. 418–423.

Received July 2015; revised November 2015; accepted November 2015