Adaptive Recommendation-based Modeling Support for Data Analysis Workflows

Dietmar Jannach
TU Dortmund
44227 Dortmund, Germany
dietmar.jannach@udo.edu

Michael Jugovac
TU Dortmund
44227 Dortmund, Germany
michael.jugovac@udo.edu

Lukas Lerche
TU Dortmund
44227 Dortmund, Germany
lukas.lerche@udo.edu

ABSTRACT

RapidMiner is a software framework for the development and execution of data analysis workflows. Like many modern software development environments, the tool comprises a visual editor which allows the user to design processes on a conceptual level, thereby abstracts technical details, and thus helps the user focus on the core modeling task. The large set of pre-implemented data analysis operations available in the framework, as well as their logical dependencies, can, however, be overwhelming in particular for novice users.

In this work we present an intelligent add-on to the RapidMiner framework that supports the user during the modeling phase by recommending additional operations to insert into the currently developed data analysis workflow. In the paper, we first 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.

Author Keywords

Visual Process Modeling; Recommendation; User Interfaces

ACM Classification Keywords

H5.3 Information Interfaces and Presentation: Interaction Styles

INTRODUCTION

RapidMiner[1] is a wide-spread open-source software framework for the design and execution of data analysis workflows. 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 pre-processing, feature selection techniques, various machine learning algorithms, as well as methods for performance evaluation.

[1] `http://www.rapidminer.com`

IUI 2015, March 29–April 1, 2015, Atlanta, GA, USA.

ACM 978-1-4503-3306-1/15/03 ...$15.00.
http://dx.doi.org/10.1145/2678025.2701380

In contrast to many other data processing frameworks or machine learning libraries, RapidMiner includes a visual modeling environment (Figure 1) that 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 pre-defined 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 tools like RapidMiner make it easier for users to incrementally develop and immediately test the 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, a "rule generation step" would typically follow after the association rule mining operator "FP-Growth". Rule mining, on the other hand, might require some preceding data transformation step.

To better support the users, we propose a user interface (UI) extension for RapidMiner which makes adaptive recommendations to the user regarding suitable operators to extend the currently developed data analysis process. The new UI component bases its suggestions on a prediction model that is learned in a pre-processing 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 & drop operations, and constantly updates the recommendations depending on the user's recent actions.

The paper is organized as follows. In the next section, we propose new algorithms for operator recommendation that take the user's current modeling *context* into account and benchmark these methods with the algorithms presented in [12]. 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. The paper ends with a discussion of previous works and an outlook on future developments.

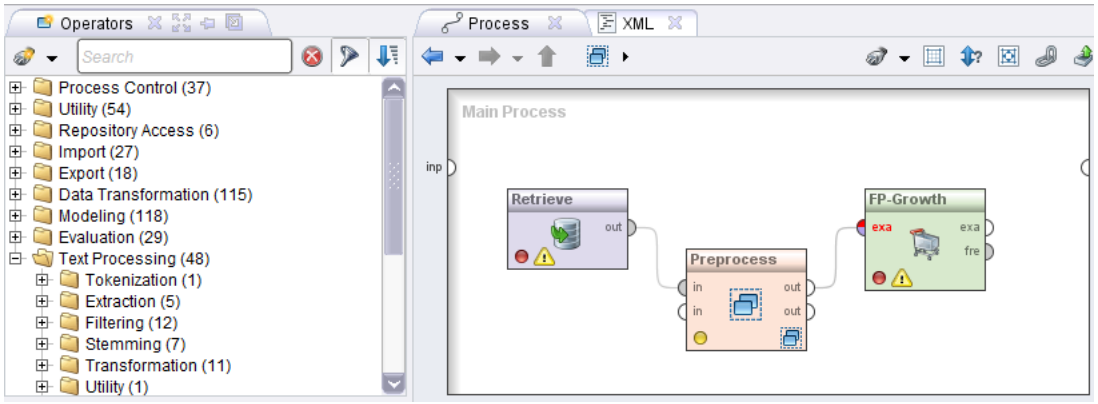

Figure 1. A process fragment developed in RapidMiner

CONTEXTUALIZED OPERATOR RECOMMENDATION

Results from a previous study

In [12], 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 data mining workflow. We will use the same experimental procedure in this paper; details will be given in a later section.

The prediction approaches in [12] 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 FP-Growth [1], sequential patterns [9], or CFP-Growth [11] were evaluated.

A k-nearest neighbors method

The most successful approach in most settings tested in [12] 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 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 that appeared in processes that are similar to the currently developed one, are likely to appear in the current process as well.

Using pairwise co-occurrences

In [12], 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 [12] showed that CoOccur performed surprisingly well and was often better than the computationally more expensive rule mining techniques.

Context-Aware Algorithms

The different techniques presented in [12] 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 the proposed algorithms were mostly 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 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 multi-element *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 shown in Figure 2, where we have a process with multiple execution threads. If we assume that the last user action was the insertion of the "FP-Growth" operator, then it is reasonable to give more weight to operators in the recommendation process that co-occurred

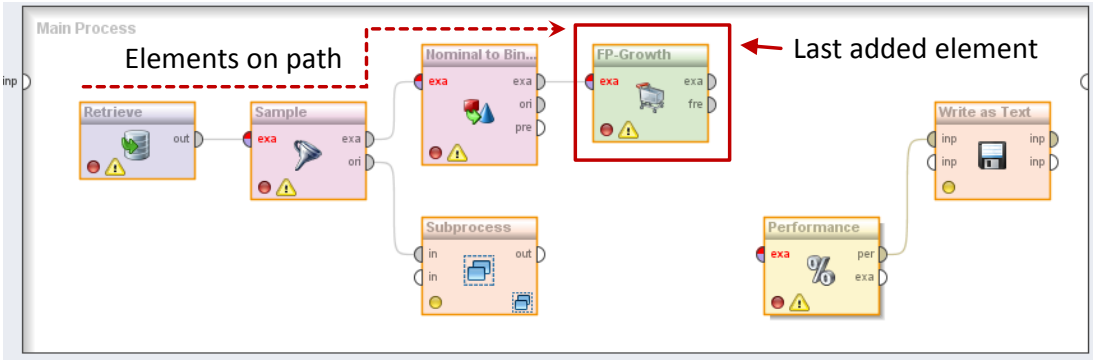

Figure 2. Looking at the most recently added elements

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.

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 set of operators that precede the last inserted one as the context. The list of preceding operators is determined by following the incoming edges backwards until an operator is reached which has no inputs. In the example in Figure 2, we would pass the last added operator and the three other elements on the path to the recommender. 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[2].

Note that while the context information passed to the recommendation algorithms contains all the elements of the path, not all of them might actually be useful to adapt the recommendations. Operators appearing at the very beginning of the process 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.

A context-aware kNN method

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, i.e., 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 recent context. The score function shown in Equation 2 correspondingly has an additional parameter – the context – and uses an updated similarity function sim_{ctx}.

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

[2]Alternative ways of determining the context are of course possible.

Various ways of adding more weight to certain neighbors are possible. In our experiments we used a scheme which increases the weight by a factor α for favorable neighbors (which contain elements of the current context) and decreases the weight of other neighbors using a factor β. The context-based weighting scheme is shown in Equation 3:

$$sim_{ctx}(\hat{p}, p_i, ctx) = \begin{cases} sim(\hat{p}, p_i) \cdot \alpha \cdot (1 + sim(p_i, ctx)), \\ \qquad \text{if } p_i \text{ overlaps with } ctx \\ sim(\hat{p}, p_i) \cdot \beta, \\ \qquad \text{otherwise} \end{cases} \quad (3)$$

where $\alpha > 0$; $0 < \beta < 1$; 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.

A context-aware co-occurrence method

Since the computationally simple but comparably effective method of looking at pairwise operator co-occurrences (COOCCUR) led to good results in [12], 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*, i.e., for the elements on the path in Figure 2. Because this context-based approach is independent of operators outside of the context, it requires even less computation time than its non-contextualized counterpart.

LINK-CTX*: Considering linked operators*

So far, we only looked at operator co-occurrences but did not take the structure of the processes into account. The proposed method LINK-CTX is a first step to look at 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 $\hat{p}$):

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

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

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

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 operators in 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.

A hybrid strategy

The last recommendation strategy evaluated in this paper, HYBRID-CTX, combines the two contextualized strategies KNN-CTX and LINK-CTX because the two methods focus on different patterns in the training data. 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:

$$w(\hat{p}) = \begin{cases} log_{10}(|\hat{p}|) \cdot \gamma, & \text{if } |\hat{p}| \leq 5 \\ \gamma, & \text{otherwise} \end{cases} \quad (6)$$

in which $|\hat{p}|$ corresponds to the number of operators in $\hat{p}$ and γ is an empirically determined maximum weight. The score for a target operator op is given in Equation 7.

$$score(op, \hat{p}, ctx) = w(\hat{p}) \cdot score_{kNN-Ctx}(op, \hat{p}, ctx) + (1 - w(\hat{p})) \cdot score_{Link-Ctx}(op, \hat{p}, ctx) \quad (7)$$

EXPERIMENTAL EVALUATION

Evaluation protocol

We adopt the evaluation approach from [12]. The evaluation is based on a pool of several thousand existing data mining workflows, which are split into training and test datasets. The task of the algorithm is to predict operators that were hidden in the processes from the test dataset. Ten-fold 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 operations are typically related to raw data retrieval (e.g., from a file) and result output. In this paper we will report the results of the SECOND-TO-LAST variant; the results are similar for the RANDOM and MIDDLE strategy.

- *Given-n*: The idea of this evaluation variant is to remove all but the first *n* elements from the process. By increasing the value of *n* step by step, we can simulate the incremental development of the data mining 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.

Beside the incomplete partial test processes, the recommendation algorithms were also provided with the information about the current context as described above, i.e., they received the list of operators that preceded the hidden one.

As evaluation measures, we use *Recall* and the *Mean Reciprocal Rank* (MRR). Since we only hide one element in each test process, Recall has the value of "1", in case the recommendation algorithm puts the hidden operator in the top-n list. Otherwise, no hit was made and the Recall is zero. 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 all examined data mining processes.

Dataset

The data that was used for the evaluation consisted of a collection of several thousand real-world data analysis workflows created with RapidMiner. The process definitions contained both processes that were publicly shared by users on the "myexperiment" web site[3], the example process sets from the RapidMiner framework, and data from other sources.

Since there were a number of duplicate processes stemming from, e.g., duplicate posts on the forums, we applied a comparably strict de-duplication strategy. Specifically, we considered process definitions to be duplicates when they contained the same operator sequences. Furthermore, we ignored all processes that consisted only of one operator. Table 1 shows some statistics of the final dataset.

Note that the average number of unique operators per process is lower than the average number of operators per process. This means that the same operator is used multiple times in several processes. The recommendation of an operator that is already used in the process might therefore be reasonable.

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

Number of processes	6,385		
Number of unique operators	833		
	Mean	*Median*	*St. dev.*
Operators per process	20.7	14	20.2
Unique operators per process	11.8	10	7.3
Longest connected path	6.5	5	4.4

Table 1. Dataset Characteristics

The average number of operators per process is also much lower than the average length of the longest path of connected operators. This indicates that processes are often not entirely linear but contain branches and subprocesses. Finally, the majority of the processes is fairly short and about 50% of the processes have a length between 6 and 15 operators. Some processes, however, comprise a comparably large number of operators. These processes are typically organized in subprocesses.

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, e.g., those for data import, sampling, or the computation of derived attributes, are part of a major fraction 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 3.

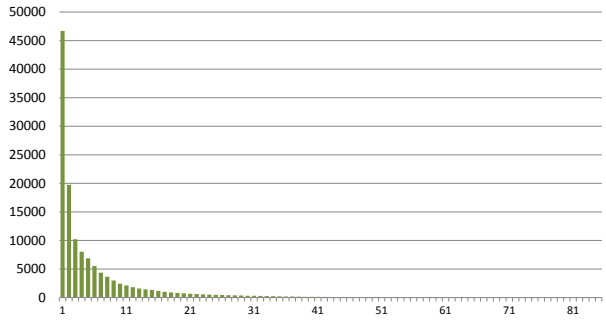

Figure 3. Distribution of the usage frequencies of the operators (each step on the x-axis represents a bin of 10 operators)

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 (left-most bar in the chart) therefore contains the most frequently used operators. Overall, about 40% of all operator usages in the processes correspond to these first ten very general operator types. On the other hand, the long-tail distribution shown in Figure 3 indicates that many of the operators are used less than 10 times in the several thousand process definitions.

Evaluation Results and Discussion

We compared the best performing algorithms from [12] with the context-aware ones proposed in the previous section using the above described protocols and evaluation measures.

Results for the Leave-One-Out setting

Table 2 shows the obtained Recall and MRR results using the best parameter settings for each algorithm, which we obtained through a manual fine-tuning process. We chose a list length of 10, assuming that this corresponds to a number users would probably inspect and 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 in Table 2 correspond to a setting in which we hid the second-to-last element in the process. Different hiding strategies – e.g., hiding a random element – led to different absolute values for the measures but not to a different algorithm ranking.

Beside the accuracy measures the average time required to compute one recommendation list[4] is shown. 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.

Algorithm	Recall	MRR	RecTime
MostFreq	0.414	0.143	0.01 ms
CoOccur	0.542	0.234	0.43 ms
kNN	0.669	0.228	3.77 ms
CoOccur-Ctx	0.540	0.262	0.27 ms
kNN-Ctx	0.843	0.567	3.94 ms
Link-Ctx	0.673	0.394	0.14 ms
Hybrid-Ctx	**0.852**	**0.573**	4.07 ms

Table 2. Recall, MRR (top-10) and recommenation time

Accuracy results: The results clearly show that all context-aware methods (except CoOccur-Ctx) by far outperform the non-contextualized versions from [12] and that the hybrid method leads to the overall best results. A t-test ($p < 0.05$ with Bonferroni correction for multiple testing) revealed that all pairwise differences between the algorithms are statistically significant except between (a) Link-Ctx and kNN (Recall), (b) CoOccur and CoOccur-Ctx (Recall), (c) CoOccur and kNN (MRR), and (d) Hybrid-CTX and kNN-CTX (Recall and MRR).

Computation times: Due to our internal bitset encoding of the processes, the computation times required to compute one recommendation list are very low even for the *kNN* approaches.

Parameter settings: We determined optimal neighborhood sizes of 5 for the kNN method and 9 neighbors for kNN-CTX, for which we used $\alpha = 0.5$ and $\beta = 0.001$. The optimal context-lengths were 2 (CoOccur-CTX) and 1 (Link-CTX). The value of 0.9 for γ led to the best results for the hybrid.

Results for the Given-n setting

Table 3 shows the results when using the Given-N procedure to simulate the incremental development process.

Precision and Recall: The results reveal that the proposed hybrid method Hybrid-Ctx performs best even in cold-start

[4]The computer used in our experiments was equipped with 16GB RAM and an Intel i5 CPU.

Algorithm	Given-1	Given-3	Given-5
MostFreq	0.378/0.158	0.412/0.177	0.390/0.148
CoOccur	0.504/0.221	0.557/0.247	0.504/0.217
kNN	0.228/0.088	0.542/0.170	0.586/0.175
CoOccur-Ctx	0.504/0.221	0.555/0.245	0.509/0.209
KNN-Ctx	0.293/0.167	0.639/0.386	0.684/0.439
Link-Ctx	**0.640/0.296**	0.713/0.385	0.718/0.383
Hybrid-Ctx	**0.640/0.296**	**0.765/0.444**	**0.803/ 0.468**

Table 3. Recall/MRR for different given-n configurations

situations, in which only the first few elements are known. The KNN methods alone perform better than the frequency-based technique as soon as three elements of the process are known. In the extreme *Given-1* case, the structure-based Link-Ctx method yields very good results. Most observed differences are again statistically significant using pairwise t-tests with Bonferroni correction, $p < 0.05$; in particular the difference between the hybrid method and Link-Ctx and all other techniques for the *Given-3* and *Given-5* situations are significant both in terms of Recall and MRR.

Overall, the results of the offline evaluation indicate that combining different aspects like process structure and neighborhood-based co-occurrence patterns – as done in Hybrid-Ctx – is the most promising approach to obtain high recommendation accuracy.

IMPLEMENTATION ARCHITECTURE

In addition to the offline analysis, 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 mining workflows more efficient. To be able to conduct such a test, we developed a fully-functional plug-in to RapidMiner with which (parts of) the participants of the study interacted. The developed software is not yet in productive use. An evaluation with real users is, however, planned in the near future.

Before we describe the details of the laboratory study, we will first outline the overall architecture of the recommendation service of RapidMiner (Figure 4).

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 4. 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[5] – is then presented in the designated area of RapidMiner. Figure 4 shows our 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 & drop the operator into the modeling window, which in turn leads to a request to the server for a new recommendation list. The purpose of the plug-in component is therefore to help the user find suitable operators more quickly than by scanning the entire hierarchical list of operators and their descriptions. In addition, the recommender will eventually make recommendations for operators that the user might have forgotten to include in the process otherwise.

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[6]. 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.[7] 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.

LABORATORY STUDY

In the following, we will describe the details of the laboratory study which we conducted to better understand the value of the proposed recommendation service within RapidMiner and get additional feedback from the users regarding, e.g., 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 for them).

Inspired by the work presented in [14], who made a comparable user study in the domain of business process modelling, 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.

[5]The confidence values presented to the user depend on the chosen algorithm.

[6]All specific pieces of information like parameters, data source names etc. are removed for privacy reasons before the transmission to the server.

[7]This feature was switched off for the user study.

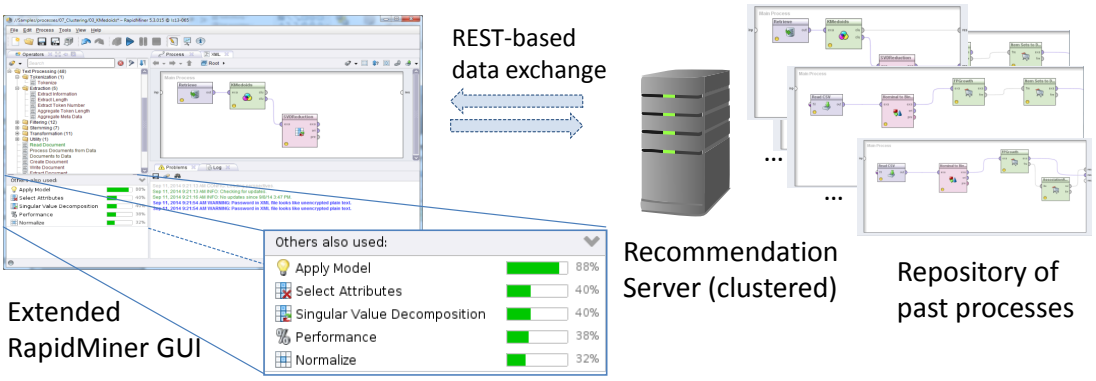

Figure 4. Architecture of the recommendation service for RapidMiner

- (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.

Expected differences within subgroups of plug-in users:

- (E4) Non-experienced users find the recommendations to be more helpful than experienced ones and have a higher tendency to recommend the tool to others.
- (E5) The plug-in leads to efficiency improvements both for experienced and inexperienced users.

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.

All subjects completed the task individually in the same office room at our department and used the 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.

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.

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 15 minutes was reached.

The problem to solve in the exercise itself consisted of a partial four-element data mining 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. Beside 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 7 operators which is a realistic and common size as mentioned above. Figure 5 shows a screenshot of a correctly completed process.

At the server side, we used the *kNN* method from [12] to compute the operator recommendations since this was the best performing method we had available at the time of the study.

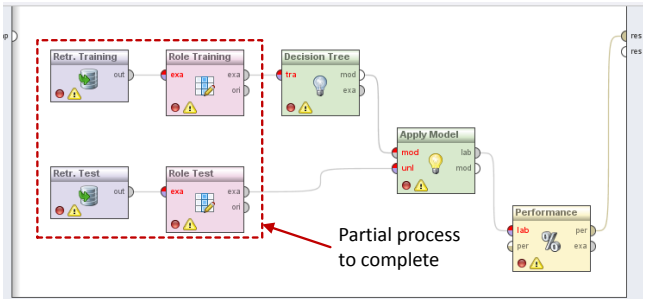

Figure 5. A correct solution to the modeling exercise

Questionnaire

In the post-experiment questionnaire, we asked several questions to validate our hypotheses. We asked questions around the following topics: (a) experience in data mining in general and with RapidMiner, (b) the perceived complexity of the modeling task, (c) the user's self-assessment of the quality of the final process and the level of encountered difficulties, and (d) their satisfaction with the tutorial material. The users that could use the recommendation plug-in were additionally asked (i) whether they found the tool helpful, (ii) if they felt well-supported by the tool, and (iii) if they would recommend the use of the plug-in to others. For all questions we used five-point or ten-point Likert scales.

Participants

In total, 28 subjects participated in the study. All of them had at least basic knowledge in the field of computer science (CS).

Some of them had some experience in data mining, workflow processes or 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.

Study results

Efficiency of modeling (E1)

As a measurement method for testing our expectation E1, i.e., 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 4 summarizes these observations.

Note that all subjects could successfully identify and insert the three missing operators within the 15 minute time limit. Not all subjects however 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 the time from the beginning of the task until the moment when the operators were successfully inserted[8].

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

Table 4. Efficiency results

The observations show that the users that were supported by the plug-in needed significantly ($p < 0.05$) 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 5 minutes. The provision of recommendations helped to reduce the average time to about 2 minutes. The comparably large variance 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 30 seconds.

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 5). 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 are marginally significant ($p < 0.1$). Nonetheless, we see this as some indication that the participants assumed that the recommendations made by the system were correct.

[8]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.

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

Table 5. Confidence in results

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, i.e., find and connect the right operators and complete the workflow. The mean values of the provided answers in the questionnaire are shown in Table 6.

Group	Ease of task	Comprehens. of task
no plug-in	3.42	3.29
with plug-in	3.72	4.29

Table 6. Perceived ease/comprehension of modeling task

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 at $p < 0.05$ and our expectation was therefore not fulfilled. However, plug-in users found the overall task significantly clearer (4.29 vs. 3.29, $p < 0.05$) 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.

Utility of plug-in for different user groups (E4)

We differentiate between experienced and non-experienced users based on their self-reported assessment at the beginning of the experiment. We expect that non-experienced users find the tool more helpful than 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 ten-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.

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 non-experienced 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 217 seconds and inexperienced about 301 seconds. *With* the plug-in, the average time needed by experienced users went down to about 52 seconds. Inexperienced users needed about 111 seconds. As a result, the relative efficiency gain was about three fourths for experienced users and more than two thirds for inexperienced users. Table 7 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.

Group	Experienced	Inexperienced
no plug-in	214.6	301.2
with plug-in	52.5	111.25
rel. improvement	75.5%	63.0%

Table 7. Rel. efficiency improvement (experienced vs. inexperienced)

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. 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 e-commerce sites. 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[9]. Further experiments are therefore required to determine the optimal default position and a layout of the recommendations that does not overwhelm the users.

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

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, 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 (new starters) 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.

RELATED WORK

The use of recommendation techniques to support the user in the specific area of modeling data mining 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 [13], our work would fall into the category of *single-element, structural recommendations*. Our Given-n simulation corresponds to a "forward-completion" approach.

In [14], Koschmieder et al. 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. Similar 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 [14] 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.

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 iden-

tify the user's modeling *intention*, see [10] or [14]. 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 analysis and explore more complex *structural* patterns in the currently developed process and the process fragments in the repository. Such analyses were done, e.g., in [16] through an efficient encoding of the graph matching problem or can rely on other process similarity measures as proposed in [7].

Several approaches in the literature like the ones mentioned above or the Bayesian Networks one proposed in [2] 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, e.g., by a case-based reasoning approach as proposed in [19].

Finally, some other modeling support approaches in the literature in the domain of business processes aim to the suggestion of appropriate textual names [15] or present recommendations based on the syntactical rules of the used modeling language [18]. 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 [20]). 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, e.g., 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.

During the last decade, inter-organizational 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, e.g., for process completion, the recommendation of alternatives for broken services, or web service discovery [3, 5, 21]. The discovery approach presented in [3], 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 [3] 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 [5]. Instead of relying on text-based (content-based) matching as done, e.g., in [8], they try to apply historical usage data and classical collaborative and content-based filtering techniques to determine suitable recommendations, e.g., by comparing user profiles. Our kNN method has some similarity with the process-similarity based approach in [5]. 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 [4], 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. Similar to our work, the authors of [4] propose to take the current modeling context into account and to generate recommendations depending on the selected activity in the workflow. The evaluation in [4] 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 [17], 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. Similar to our work, *CommunityCommands* in some form recommends user actions[10]. 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.

CONCLUSIONS

We have presented an extension to the wide-spread RapidMiner software environment for modeling data mining 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. 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 (skeletons) in addition to the current single-element recommendations.

[10]See also [6] for an earlier work on predicting user actions.

REFERENCES

1. Agrawal, R., and Srikant, R. Fast algorithms for mining association rules in large databases. In *Proceedings of VLDB '94* (1994), 487–499.
2. Bobek, S., Baran, M., Kluza, K., and Nalepa, G. J. Application of bayesian networks to recommendations in business process modeling. In *Proceedings of AIBP '13* (2013), 41–50.
3. Chan, N., Gaaloul, W., and Tata, S. Composition context matching for web service recommendation. In *Proceedings of SCC '11* (2011), 624–631.
4. Chan, N., Yongsiriwit, K., Gaaloul, W., and Mendling, J. Mining event logs to assist the development of executable process variants. In *Proceedings of CAiSE '14*. 2014, 548–563.
5. Chan, N. N., Gaaloul, W., and Tata, S. A recommender system based on historical usage data for web service discovery. *Service Oriented Computing and Applications 6*, 1 (2012), 51–63.
6. Davison, B. D., and Hirsh, H. Predicting sequences of user actions. In *Proceedings of AAAI '98 Workshop on Predicting the Future: AI Approaches to Time Series Analysis* (1998), 5–12.
7. Dijkman, R., Dumas, M., van Dongen, B., Käärik, R., and Mendling, J. Similarity of business process models: Metrics and evaluation. *Information Systems 36*, 2 (2011), 498–516.
8. Dong, X., Halevy, A., Madhavan, J., Nemes, E., and Zhang, J. Similarity search for web services. In *Proceedings of VLDB '04* (2004), 372–383.
9. Fournier-Viger, P., Faghihi, U., Nkambou, R., and Nguifo, E. M. CMRules: Mining sequential rules common to several sequences. *Knowledge-Based Systems 25*, 1 (2012), 63–76.
10. Hornung, T., Koschmider, A., and Lausen, G. Recommendation based process modeling support: Method and user experience. In *Proceedings of ER '08* (2008), 265–278.
11. Hu, Y.-H., and Chen, Y.-L. Mining association rules with multiple minimum supports: a new mining algorithm and a support tuning mechanism. *Decision Support Systems 42*, 1 (2006), 1–24.
12. Jannach, D., and Fischer, S. Recommendation-based modeling support for data mining processes. In *Proceedings of RecSys '14* (2014), 334–340.
13. Kluza, K., Baran, M., Bobek, S., and Nalepa, G. J. Overview of recommendation techniques in business process modeling. In *Proceedings of KESE9* (2013), 46–57.
14. Koschmider, A., Hornung, T., and Oberweis, A. Recommendation-based editor for business process modeling. *Data and Knowledge Engineering 70*, 6 (2011), 483–503.
15. Leopold, H., Mendling, J., and Reijers, H. A. On the automatic labeling of process models. In *Advanced Information Systems Engineering*, vol. 6741 of *Lecture Notes in Computer Science*. 2011, 512–520.
16. Li, Y., Cao, B., Xu, L., Yin, J., Deng, S., Yin, Y., and Wu, Z. An efficient recommendation method for improving business process modeling. *IEEE Transactions on Industrial Informatics 10*, 1 (2014), 502–513.
17. Matejka, J., Li, W., Grossman, T., and Fitzmaurice, G. W. CommunityCommands: Command recommendations for software applications. In *Proceedings of UIST '09* (2009), 193–202.
18. Mazanek, S., and Minas, M. Business process models as a showcase for syntax-based assistance in diagram editors. In *Model Driven Engineering Languages and Systems*, vol. 5795 of *Lecture Notes in Computer Science*. 2009, 322–336.
19. Minor, M., Bergmann, R., Görg, S., and Walter, K. Towards case-based adaptation of workflows. In *Case-Based Reasoning. Research and Development*, vol. 6176 of *Lecture Notes in Computer Science*. 2010, 421–435.
20. Piorkowski, D., Fleming, S., Scaffidi, C., Bogart, C., Burnett, M., John, B., Bellamy, R., and Swart, C. Reactive information foraging: An empirical investigation of theory-based recommender systems for programmers. In *Proceedings of CHI '12*, ACM (New York, NY, USA, 2012), 1471–1480.
21. Sellami, M., Tata, S., Maamar, Z., and Defude, B. A recommender system for web services discovery in a distributed registry environment. In *Proceedings of ICIW '09* (2009), 418–423.