Generating Scientific Documentation for Computational Experiments Using Provenance

Adianto Wibisono[1,2], Peter Bloem[1], Gerben K.D. de Vries[1], Paul Groth[2], Adam Belloum[1], Marian Bubak[1,3]

[1] System and Network Engineering Group, Informatics Institute, University of Amsterdam, The Netherlands
{a.wibisono, p.bloem, g.k.d.devries, a.z.s.belloum}@uva.nl
[2] VU University Amsterdam, The Netherlands
pgroth@vu.nl
[3] Department of Computer Science, AGH Krakow, Poland
bubak@agh.edu.pl

Abstract. Electronic notebooks are a common mechanism for scientists to document and investigate their work. With the advent of tools such as IPython Notebooks and Knitr, these notebooks allow code and data to be mixed together and published online. However, these approaches assume that all work is done in the same notebook environment. In this work, we look at generating notebook documentation from multi-environment workflows by using provenance represented in the W3C PROV model. Specifically, using PROV generated from the Ducktape workflow system, we are able to generate IPython notebooks that include results tables, provenance visualizations as well as references to the software and datasets used. The notebooks are interactive and editable, so that the user can explore and analyze the results of the experiment without re-running the workflow.
We identify specific extensions to PROV necessary for facilitating documentation generation. To evaluate, we recreate the documentation website for a paper which won the Open Science Award at the ECML/PKDD 2013 machine learning conference. We show that the documentation produced automatically by our system provides more detail and greater experimental insight than the original hand-crafted documentation. Our approach bridges the gap between user friendly notebook documentation and provenance generated by distributed heterogeneous components.

1 Introduction

Common approaches to computational experimentation[1] span a spectrum. On one side, we find quick, informative experiments intended for fast iteration. These often involve a single researcher, working on consumer-scale hardware, and can take as little as a few minutes to run. The aim is to get quick results to inform further experiments and to build towards larger results in an iterative manner. The environment that is used for this type of experimentation is usually designed around quick iteration, and quick inspection of results: MATLAB, R, or a simple

UNIX command line. More recently, this is often done within interactive notebook environments such as IPython notebooks [1], Knitr [2] or Mathematica [3].

On the other side of the spectrum we find large-scale experimentation. Well-prepared, thoroughly designed experiments, intended to run for long amounts of time on powerful hardware. These experiments are often implemented by scientific programmers, separate from the researchers designing the experiment. The chosen environment is often a workflow system [4], providing features like monitoring of execution, robustness against hardware failure and provenance tracking. The downside is that each experiment must be carefully prepared, and purpose-written for the workflow system.

Experimentation usually starts with quick iterations in an interactive system, and progresses towards the more robust environments as the experiments become more involved, often at the expense of a re-implementation step as the code is ported to a more robust environment. At the larger scales, iterations invariably become slower.

Finally, once results have been produced that are expected to be fit for publication, the researchers must translate and summarize their approach to allow for peer-review, reproduction and reuse. The ideal is to publish the datasets, the code and to provide instructions for reproducing the experiment. In the small-scale iterative end of the spectrum, this can be very cumbersome: gathering unversioned code, unstructured datasets and documenting all idiosyncratic steps required to execute it. In the large-scale end, experiments tend to be more structured, as enforced by the workflow system, but the description of the workflow is still tied to the workflow platform. Even a provenance trace, which is intended to illustrate the source of the results, can be difficult to interpret in its raw form.

1.1 Main Idea

In this paper, we present a concept for generating notebook documentation for computational experiments from provenance information. Our approach aims to retain some of the iteration speed of the small-scale experimentation at the large-scale end of the spectrum. This documentation generation process is built on three ideas.

1. After a large-scale experiment has finished, many questions raised by its output can, theoretically, be answered without re-running the experiment. Unfortunately, these questions were not the ones which the experiment was originally designed to answer, so the required data was not collected during the run. Output representing as much information about the original run as possible can help to postpone the need for a new run of a (redesigned) experiment.

[1] In this paper, we will call an experiment which can be run entirely *in silico* (ie. as a computer program) a *computational experiment*.

2. While provenance is often seen as a kind of semantically annotated log file—helping for keeping track of the origins of data, and for finding answers in the case of unforeseen errors—a complete provenance trace will actually contain all information about a run of a computational experiment: all data produced, and the semantic links between them [5]. Any output required from the experiment, such as tables, graphs and statistical analysis, can be reconstructed from the provenance trace.
3. A semantically annotated representation of a run of an experiment (such as a provenance trace) allows us to make intelligent guesses at default modes of reporting. Thus, we can automatically create reasonable scientific documentation; reporting not only the results of the experiment, but a human-readable representation of how the results emerged: which datasets were used, where they can be found, what code was used, using which versions and in what configuration. An interactive environment allows the researcher to tweak this documentation to filter out less relevant information.

In short, we propose to put provenance at the heart of computational experimentation, rather than the sidelines, to combine the best of both worlds. A large-scale experiment is run on a workflow system, producing mainly a provenance trace. This trace is then loaded into an interactive environment, allowing a researcher to investigate the questions that inspired the experiment, and any further questions that these results raise. The researcher can filter, plot and analyze the results at length, with much greater depth than a non-semantic output, such as a CSV file, could offer. Only when all information produced by the original run is exhausted, does a new experiment need to be started.

When the time comes for the results to be shared, e.g. via a publication, the provenance trace provides all required information. All that is needed is a means to convert it to human readable form. The semantic annotations allow us to create reasonable default documentation, while anybody interested in the experiment can load the provenance trace into an interactive system and study the details.

1.2 Contributions

Interactive notebooks provide both a good format for presenting default documentation and an interactive environment to study experimental results. The proof-of-concept implementation presented in this paper uses provenance, in the W3C PROV-O [6] format, generated by our own workflow system Ducktape[2], to automatically create IPython notebooks. We chose IPython Notebooks as this system is becoming widely used in data processing. Additionally, they provide a web-based environment, independent of the underlying language. This means that future versions of our system could also support R, Julia and other programming languages. Our notebooks have result tables and graphs, visualization of the provenance and links to the software and datasets used. Furthermore, they

[2] http://github.com/Data2semantics/ducktape

are interactive and editable, so that the user can explore and analyze the results of the experiment without re-running the workflow. As a running example use-case, we take the documentation web-page that won the Open Science Award at the ECML/PKDD 2013 machine learning conference.

The rest of this paper is structured as follows. In the next section, we discuss related work. Section 3 describes our proof-of-concept implementation. The final section contains conclusions and directions for future research.

2 Related Work

A key part of related work is in the area of workflow systems. Often, these systems provide accessible documentation to the end user through graphical representations of the workflow. Additionally, they attach detailed provenance information to those workflows [7]. Our work is different in that we build a notebook style representation directly from the provenance.

Other existing papers also explore and derive insight from scientific workflow provenance, with different goals than ours. Work by Biton et al. [8] lets users define views based on relevant workflow parts that determines how a possibly large workflow provenance graph can be explored. The high level query languages for provenance: QLP [9] and OPQL [10], can be used for interactive querying and visualization. Both views simplify provenance results and allow exploration of scientific workflow provenance at the graph level.

Close to our work is that of Gibson et al. [11], on creating an interactive environment where provenance is stored. We see our work as complementary as one can see the generation of the workflows as similar to generating a notebook. Deep [12], an executable document environment that generates scientific results dynamically and interactively, also records the provenance for these results in the document. In this system, provenance is exposed to users via an interface that provides them with an alternative way of navigating the executable document.

Burrito[13] is a system that uses a combination of provenance tracking and user interface constructs for notes to help generate a lab notebook. Our approach shares their motivation but focuses instead documenting distributed computational workflows using provenance. Similarly, Scientific Application Middleware [14] combines information coming from both lab notebooks but also distributed computational components to create documentation for experiments. Our work adds to this vision by connecting to widely used interactive (computational) notebook environments.

The idea of using provenance as a singular result of workflow execution shares some aims with the idea of *Research Objects* [15]. This is a construct that aims to replace the traditional paper article as the main unit of scientific publication. A research object is a package of not just the research results, but also all artifacts used to create them, such as datasets, code and provenance. Within the research object, the provenance is seen as a feature to facilitate auditing. In our approach, we see the provenance as the key entry point: it should not just be used to audit the experiments, but also to aggregate results and to perform statistical

analyses. Our perspective does not change or replace the use of Research Objects, but suggests that the provenance could be used as its central component, tying together the other contents of the package.

3 Proof-of-Concept

The proof-of-concept implementation for our documentation generation approach consists of three components: a workflow system, workflow provenance and generating notebooks from provenance. We first introduce a running example that will illustrate these three components and then we describe the components themselves.

3.1 Running Example

The webpage[3] for the paper *A fast approximation of the Weisfeiler Lehman graph kernel for RDF data* [16] won one of the two Open Science Awards at ECML/PKDD 2013, the conference where it was published. On the page, links to software libraries, datasets and the original source code are provided, as well as instructions on how to run the experiments using the provided material. The datasets are available online, via `figshare.com`, and the code is stored in a git repository, at `github.com`. We have recreated two partial experiments in the ECML/PKDD 2013 paper [16] for our proof-of-concept. We use these experiments as running examples below. Note that we do not recreate the full set of experiments in the paper. However, the recreated parts are a representative subset, since we cover both a classification experiment and a runtime experiment.

In the classification experiment a number of graph kernels for RDF data are tested on an affiliation prediction task. The goal in this task is to predict affiliations for persons in the dataset. Three different kernels are tested, each for a number of parameter settings. These kernels are combined with a Support Vector Machine (SVM) to perform prediction. To reduce randomness, the experiment is repeated 10 times, with different random seeds.

The runtime experiment uses the same graph kernels and dataset, but this time the kernels are computed for different fractions of that dataset to investigate the runtime performance of the different kernels. The most computationally intensive settings for the kernels are used. For each dataset fraction, the computation is performed 10 times (on 10 random subsets).

3.2 Workflow System: Ducktape

Ducktape is a light-weight workflow system developed in the context of the Data2Semantics[4] project. This project provides essential semantic infrastructure for e-science and focuses on how to share, publish, access, analyze, interpret

[3] `http://www.data2semantics.org/publications/ecmlpkdd-2013/`
[4] `http://www.data2semantics.org`

and reuse scientific data. Ducktape is designed to compose experiments using components developed within the project. By using an annotation approach, we keep the system light-weight and impose little additional effort for a scientist to use his existing code in our environment.

Ducktape uses computational modules, which are annotated pieces of codes, typically classes. The annotations indicate what the inputs and outputs of the module are and what the main computation routine is. Currently, Java, Python and command line scripts are supported.

A Ducktape workflow is described in a simple data flow format represented in YAML (YAML Ain't Markup Language) [17], which contains a list of modules and specifications of each of the modules' input data. Figure 1 shows part of the workflow description for the affiliation prediction experiment. Module inputs can either be raw data type values, i.e. integers, doubles and strings, or data produced by other modules within the same workflow (e.g. Fig. 1, line 17, 20, 22).

Module input fields in the YAML workflow description can be supplied with lists of inputs of the same type, to allow for parameter sweeps (Fig. 1, line 23). Ducktape allows users to specify whether they want input lists to be consumed in a pair-wise manner or whether the full Cartesian product between the lists should be used in the parameter sweep. Furthermore, there are keywords to indicate whether certain inputs represent datasets (Fig. 1, line 10), what module outputs should be considered experimental results (Fig. 1, line 25) and for which input parameter we want to aggregate results (Fig. 1, line 26).

3.3 Provenance: W3C PROV

Whenever a workflow is executed, Ducktape automatically generates the provenance that captures this execution in the W3C PROV-O [6] format.[5] Table 1 shows how the different elements of a Ducktape workflow map to the concepts in W3C PROV. The main concepts from W3C PROV that we use are prov:Activity and prov:Entity and their connecting relations: prov:used and prov:wasGeneratedBy. Essentially, a workflow leads to a bipartite graph with alternating nodes of prov:Activity and prov:Entity.

Modules are prov:Activitys and inputs and outputs are prov:Entitys. We model this by creating a class dt-rsc:ModuleName[6] with the name of the module for all modules. Each dt-rsc:ModuleName is rdfs:subClassOf of prov:Activity. Every instance of a module executed during the run of the workflow is an rdf:type of its corresponding dt-rsc:ModuleName. We do the same for the inputs and outputs, introducing a dt-rsc:InputName or dt-rsc:OutputName for each input and output, which are rdfs:subClassOf of prov:Entity. Each input/output instance is an rdf:type of its corresponding dt-rsc:InputName/OutputName. Outputs that are inputs of another module have one unique URI. For example, the specific instance of 'seed'

[5] We note other serializations of PROV [18] can also be supported.

[6] dt-rsc is a shorthand for: http://prov.data2semantics.org/resource/ducktape/

Fig. 1. Example of YAML workflow description from the Affiliation Prediction experiment. The full workflow is not shown.

```
1  workflow:
2     name: "Affiliation Prediction Experiment IPAW 2014"
3     modules:
4     - module:
5        name: RDFDataSet
6        source: d2s.RDFDataSetModule
7        inputs:
8           filename: "http://.../aifb_fixed_complete.n3"
9           ...
10       datasets: filename
11    ...
12    - module:
13       name: Experiment
14       source: d2s.SingleGraphKernelExperimentModule
15       inputs:
16          matrix:
17               - reference: RDFWLSubTreeKernel.matrix
18               ...
19          target:
20             reference: AffiliationDataSet.target
21          parms:
22             reference: LibSVMParms.parameters
23          seed: [1,2,3,4,5,6,7,8,9,10]
24          folds: 5
25       results: [accuracy, f1]
26       aggregators: seed
```

with value '1' in the module 'Experiment' in Fig. 1, line 23, would be of type dt-rsc:Experiment/seed/[7] which is an rdfs:subClassOf of prov:Entity.

Each module (dt-rsc:ModuleName) is associated with a prov:Agent, which represent the specific Ducktape engine used for execution (i.e. the machine(s) and version), and a prov:Plan, the specific YAML workflow file.

Optionally, inputs can also be a dt-voc:Dataset[8], if they refer to a dataset (e.g. by a URL) or a dt-voc:Aggregator, if they determine how to aggregate experiment outputs based on this input. Outputs can have the dt-voc:resultOf predicate that links them to the workflow (i.e. prov:Plan), if they should be considered the results of that workflow. These optional concepts are added when they are specified in the YAML workflow file.

Furthermore, we also add the software artifact dependencies that we know that are used during execution to the provenance. This is done by creating

[7] There can be multiple inputs/outputs with the same name, so the module name is also included in this URI.

[8] dt-voc is a shorthand for: http://prov.data2semantics.org/vocab/ducktape/

Table 1. Mapping of Ducktape elements to W3C PROV

Ducktape	W3C PROV	Optional
Ducktape Engine	prov:Agent	
Workflow Description	prov:Plan	
Module Instance	prov:Activity	
Input	prov:Entity	dt-voc:Dataset, dt-voc:Aggregator
Output	prov:Entity	dt-voc:resultOf

URI for each artifact and adding it to the prov:Plan via a new property dt-voc:usesArtifact. Currently, we manage our dependencies and execute our workflows using Maven[9], thus each artifact furthermore has the properties: dt-voc:hasArtifactId, dt-voc:hasGroupId and dt-voc:hasVersion.

3.4 Notebook Generation

Based on the generated provenance, draft IPython notebooks are created. There are two types of notebook drafts: an overview notebook with general workflow execution information and a more detailed notebook at the workflow module level.

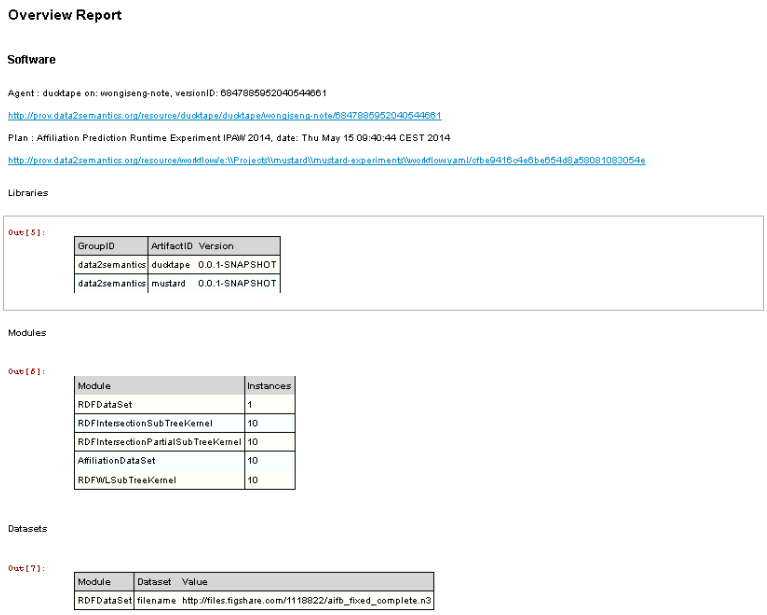
The overview notebook contains general information about the workflow plan, software artifacts and datasets used. A summary of the Ducktape modules instantiated during the experiment and inline provenance visualization generated using Prov-O-Viz [19] [10] is also included in this overview notebook to give intuitive insight into the overall workflow execution. This notebook is illustrated in Fig. 2 and 3.

The detailed notebook draft describes individual module execution results. Users have access to the module input parameters and execution results through default Python code snippets injected into the notebook. The code snippets are generated by performing SPARQL queries on the workflow provenance graph. By using these snippets, users can manipulate how they view the module parameters and execution results.

We use the existing Python Data Analysis library (Pandas)[11] in the code snippets, to allow users to play with and change the view on their results. Essentially, what the user has here is a data analysis view of each individual module in workflow execution. By default we provide tables of relevant input and outputs for each individual module which users can change by tweaking the injected Python code.

For modules that have input data marked as dt-voc:Aggregator, we provide a pivot table, which aggregates the outputs that are dt-voc:resultOf, grouping by the other input parameters. The default form of aggregation is computing the mean value, however this can be easily changed by editing the code snippet. An

[9] http://maven.apache.org/
[10] http://provoviz.org
[11] pandas.pydata.org



Overview Report

Software

Agent : ducktape on: wongiseng-note, versionID: 6847885952040544661

http://prov.data2semantics.org/resource/ducktape/ducktape/wongiseng-note/6847885952040544661

Plan : Affiliation Prediction Runtime Experiment IPAW 2014, date: Thu May 15 09:40:44 CEST 2014

http://prov.data2semantics.org/resource/workflow/e:\\Projects\\mustard\\mustard-experiments\\workflow.yaml/cfbe9416c4e6be654d8af58081083054e

Libraries

Out[5]:

GroupID	ArtifactID	Version
data2semantics	ducktape	0.0.1-SNAPSHOT
data2semantics	mustard	0.0.1-SNAPSHOT

Modules

Out[6]:

Module	Instances
RDFDataSet	1
RDFIntersectionSubTreeKernel	10
RDFIntersectionPartialSubTreeKernel	10
AffiliationDataSet	10
RDFWLSubTreeKernel	10

Datasets

Out[7]:

Module	Dataset	Value
RDFDataSet	filename	http://files.figshare.com/1118822/aifb_fixed_complete.n3

Fig. 2. Overview Report for the Runtime Experiment, part 1.

example of this aggregation is given in Fig. 4, where the results accuracy and F1 are aggregated over the seed input parameter.

In summary, the notebooks for the classification[12] and the runtime[13] experiments contain the following information: a list of datasets, a list of software artifacts, provenance visualization and detailed result tables. This is significantly more information than the original webpage and the notebooks can easily be extended by hand, both by changing the tables and adding more explanatory text[14]. Currently, the notebooks lack instructions on how to re-execute the experiments, this can be partly solved by adding instructions that explain how to use the datasets and artifacts. However, in future work we would like to add automatic re-execution of the workflow from the notebook, all the ingredients are already there.

4 Conclusions and Future Work

We have described an approach for automatic generation of scientific documentation for computational experiments. This is approach is based on the idea of

[12] Available here: `http://j.mp/ecml-notebook`.

[13] Available here: `http://j.mp/runtime-notebook`.

[14] Note that the used artifacts are different from the original version, and that the samples above are static views requiring a local IPython environment to edit.

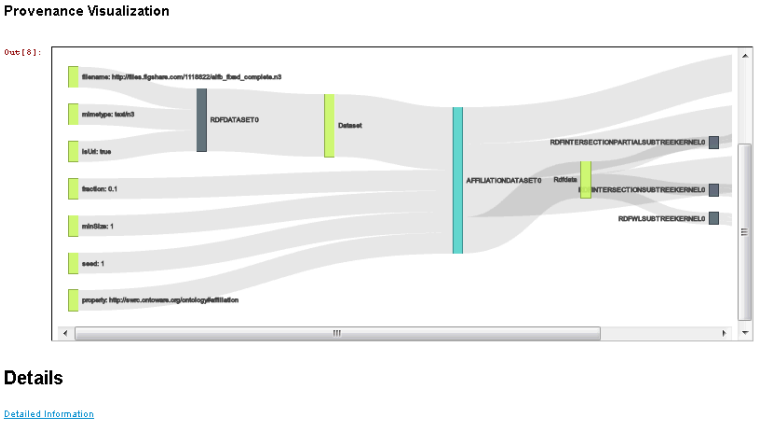


Fig. 3. Overview Report for the Runtime Experiment, part 2.

placing provenance at the heart of such experiments, using it as the main output, not just as a way to trace the execution of a workflow. Interactive notebooks provide a way to explore the results and its provenance and are an ideal starting point for creating documentation for the experiments.

We have created a proof-of-concept implementation to automatically generate IPython notebooks from provenance created by workflows run using our Ducktape platform. These notebooks aggregate the main results and components of an experiment. This automatically generated draft documentation provides more information and insight then a hand-crafted documentation page for a machine learning paper that won an Open Science Award.

While our proof-of-concept uses a specific workflow system and a specific interactive platform to load and analyze the provenance, the approach is transferable to other workflow systems and interactive environments. Indeed, most PROV serializations can be represented as a more human-friendly notebook. Central to this conception is the notion that provenance can be a true interface between the execution of an experiment and the analysis of its results.

Another outcome of this work is confirmation of the importance of connecting interactive notebook environments and provenance. By using the IPython Notebook environment, we were able to benefit significantly from the variety of tools within that community, including notebook visualization (using the nbviewer app) and analytics. We believe that the connection between notebooks in general and distributed provenance generation is an area that the community should look at in more detail as there are a number of areas of interest. For instance, one may investigate the issue of maintaining the provenance of live results streamed to notebook environment, encapsulating provenance within a notebook or tracking provenance of interactive sessions.

Beyond investigating these larger themes, there are a number of concrete extensions to the environment we intend to make. First, the current configuration

Result for module Experiment

```
In [23]: pt = pivot_table(df, rows= ['Aggregator','parent0'])
         pt
```

Out[23]:

		accuracy	f1
Aggregator	**parent0**		
0	**AffiliationDataSet0 LibSVMParms0 RDFWLSubTreeKernel0**	0.865537	0.811191
1	**AffiliationDataSet0 LibSVMParms0 RDFWLSubTreeKernel1**	0.811299	0.747912
2	**AffiliationDataSet0 LibSVMParms0 RDFWLSubTreeKernel10**	0.863277	0.824467
3	**AffiliationDataSet0 LibSVMParms0 RDFWLSubTreeKernel11**	0.822599	0.766933
4	**AffiliationDataSet0 LibSVMParms0 RDFIntersectionSubTreeKernel0**	0.829379	0.747343
5	**AffiliationDataSet0 LibSVMParms0 RDFIntersectionSubTreeKernel1**	0.825989	0.752534
6	**AffiliationDataSet0 LibSVMParms0 RDFIntersectionSubTreeKernel2**	0.809040	0.730443
7	**AffiliationDataSet0 LibSVMParms0 RDFIntersectionSubTreeKernel3**	0.787571	0.703818
8	**AffiliationDataSet0 LibSVMParms0 RDFIntersectionPartialSubTreeKernel0**	0.762712	0.661025
9	**AffiliationDataSet0 LibSVMParms0 RDFIntersectionPartialSubTreeKernel1**	0.748023	0.655392
10	**AffiliationDataSet0 LibSVMParms0 RDFIntersectionPartialSubTreeKernel2**	0.685876	0.541812
11	**AffiliationDataSet0 LibSVMParms0 RDFIntersectionPartialSubTreeKernel3**	0.560452	0.366949
12	**AffiliationDataSet0 LibSVMParms0 RDFWLSubTreeKernel2**	0.853107	0.810756
13	**AffiliationDataSet0 LibSVMParms0 RDFWLSubTreeKernel3**	0.850847	0.818110
14	**AffiliationDataSet0 LibSVMParms0 RDFWLSubTreeKernel4**	0.865537	0.811191
15	**AffiliationDataSet0 LibSVMParms0 RDFWLSubTreeKernel5**	0.811299	0.747912
16	**AffiliationDataSet0 LibSVMParms0 RDFWLSubTreeKernel6**	0.861017	0.824743
17	**AffiliationDataSet0 LibSVMParms0 RDFWLSubTreeKernel7**	0.838418	0.788040
18	**AffiliationDataSet0 LibSVMParms0 RDFWLSubTreeKernel8**	0.865537	0.811191
19	**AffiliationDataSet0 LibSVMParms0 RDFWLSubTreeKernel9**	0.811299	0.747912

20 rows × 2 columns

Fig. 4. Part of the detailed notebook for the Affiliation Prediction Experiment which shows a table for the Experiment module.

does not allow us to directly re-run the experiments from within the notebooks. We aim to implement such a feature to further improve reproducibility. Furthermore, while we can create links to software artifacts that were used, it would be even nicer to link to the actual source code for these artifacts, if that is available. Therefore, we plan to investigate how to integrate with methods such as GIT2Prov [20] to connect from execution to the source code. Furthermore, we are also investigating what additional visualizations we can embed to make the documentation richer.

Acknowledgments We thank the reviewers and Rinke Hoekstra for their useful feedbacks and discussion. This publication was supported by the Dutch national program COMMIT.

References

1. Pérez, F., Granger, B.E.: IPython: a System for Interactive Scientific Computing. Computing in Science and Engineering **9**(3) (May 2007) 21–29
2. Xie, Y.: Knitr: A general-purpose package for dynamic report generation in R. R package version **1**(7) (2013)
3. Wolfram, S.: The mathematica book. Volume 221. Wolfram Media Champaign, Illinois (1996)

4. Gil, Y., Deelman, E., Ellisman, M., Fahringer, T., Fox, G., Gannon, D., Goble, C., Livny, M., Moreau, L., Myers, J.: Examining the challenges of scientific workflows. IEEE Computer **40**(12) (December 2007) 26–34
5. Moreau, L.: Provenance-based reproducibility in the semantic web. Web Semantics: Science, Services and Agents on the World Wide Web **9**(2) (2011)
6. Lebo, T., Sahoo, S., McGuinness, D., Belhajjame, K., Corsar, D., et al.: Prov-o: The prov ontology. W3C Recommendation. World Wide Web Consortium (2013)
7. Davidson, S., Ludaescher, B., McPhillips, T., Freire, J.: Provenance in scientific workflow systems. Bulletin of the Technical Committee on Data Engineering **30**(4) (December 2007) 44–50
8. Biton, O., Cohen-Boulakia, S., Davidson, S.B., Hara, C.S.: Querying and managing provenance through user views in scientific workflows. In: Proceedings of the 24th International Conference on Data Engineering, ICDE 2008. (2008) 1072–1081
9. Anand, M., Bowers, S., Ludascher, B.: Provenance browser: Displaying and querying scientific workflow provenance graphs. In: Data Engineering (ICDE), 2010 IEEE 26th International Conference on. (March 2010) 1201–1204
10. Lim, C., Lu, S., Chebotko, A., Fotouhi, F., Kashlev, A.: OPQL: Querying scientific workflow provenance at the graph level. Data and Knowledge Engineering **88**(0) (2013) 37 – 59
11. Gibson, A., Gamble, M., Wolstencroft, K., Oinn, T., Goble, C., Belhajjame, K., Missier, P.: The data playground: An intuitive workflow specification environment. Future Generation Computer Systems **25**(4) (2009) 453–459
12. Yang, H., Michaelides, D.T., Charlton, C., Browne, W.J., Moreau, L.: Deep: A provenance-aware executable document system. In: Provenance and Annotation of Data and Processes: 4th International Provenance and Annotation Workshop, IPAW 2012. Springer (2012) 24–38
13. Guo, P.J., Seltzer, M.: Burrito: Wrapping your lab notebook in computational infrastructure. In: Proceedings of the 4th USENIX Workshop on the Theory and Practice of Provenance. TaPP'12, Berkeley, CA, USA, USENIX Association (2012)
14. Myers, J.D., Chappell, A., Elder, M., Geist, A., Schwidder, J.: Re-integrating the research record. Computing in Science and Engg. **5**(3) (May 2003) 44–50
15. Bechhofer, S., De Roure, D., Gamble, M., Goble, C., Buchan, I.: Research objects: Towards exchange and reuse of digital knowledge. The Future of the Web for Collaborative Science (2010)
16. de Vries, G.K.D.: A fast approximation of the Weisfeiler-Lehman graph kernel for RDF data. In Blockeel, H., Kersting, K., Nijssen, S., Železný, F., eds.: ECML/PKDD (1). Volume 8188 of Lecture Notes in Computer Science., Springer (2013) 606–621
17. Ben-Kiki, O., Evans, C., Ingerson, B.: Yaml ain't markup language (yaml) version 1.1. Working Draft 2008-05 **11** (2001)
18. Moreau, L., Groth, P.: Provenance: An introduction to prov. Synthesis Lectures on the Semantic Web: Theory and Technology **3**(4) (2013) 1–129
19. Hoekstra, R., Groth, P.: PROV-O-Viz - Understanding the Role of Activities in Provenance. In: Provenance and Annotation of Data and Processes: 5th International Provenance and Annotation Workshop, IPAW 2014. IPAW (2014)
20. De Nies, T., Magliacane, S., Verborgh, R., Coppens, S., Groth, P., Mannens, E., Van de Walle, R.: Git2prov: Exposing version control system content as w3c prov. In: Posters & Demonstrations Track within the 12th International Semantic Web Conference (ISWC-2013), CEUR-WS (2013) 125–128