DPDS: Assisting Data Science with Data Provenance

Adriane Chapman
University of Southampton, UK
Adriane.Chapman@soton.ac.uk

Luca Lauro
Università Roma Tre, Italy
luca.lauro@uniroma3.it

Paolo Missier
Newcastle University, UK
paolo.missier@ncl.ac.uk

Riccardo Torlone
Università Roma Tre, Italy
riccardo.torlone@uniroma3.it

ABSTRACT

Successful data-driven science requires a complex combination of data engineering pipelines and data modelling techniques. Robust and defensible results can only be achieved when each step in the pipeline that is designed to clean, transform and alter data in preparation for data modelling can be justified, and its effect on the data explained. The **DPDS** toolkit presented in this paper is designed to make such justification and explanation process an integral part of data science practice, adding value while remaining as un-intrusive as possible to the analyst. Catering to the broad community of python/pandas data engineers, **DPDS** implements an observer pattern that is able to capture the fine-grained provenance associated with each individual element of a dataframe, across multiple transformation steps. The resulting provenance graph is stored in Neo4j and queried through a UI, with the goal of helping engineers and analysts to justify and explain their choice of data operations, from raw data to model training, by highlighting the details of the changes through each transformation.

PVLDB Reference Format:
Adriane Chapman, Luca Lauro, Paolo Missier, and Riccardo Torlone. DPDS: Assisting Data Science with Data Provenance. PVLDB, 15(12): 3614 - 3617, 2022.
doi:10.14778/3554821.3554857

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/LucaLauro/Data_Provenance1.

1 INTRODUCTION

Explainability is a key requirement for data science. While the term is generally used to indicate the need to explain a prediction made by a machine learning model [1, 8], we extend it to include explainability of the data preparation pipelines that, starting from raw datasets, precede training. This is because such pipelines determine the robustness, generalisability, and accuracy of a model. Indeed, the data engineering decisions behind those pipelines may affect the ability to operationalize the model itself and defend its reliability. Thus, it ought to be possible to explain the effects that each step of a pipeline produces on each element of each of the datasets used in modelling, for instance as training sets.

The toolkit presented here, called **DPDS** (a shorthand for Data Provenance for Data Science), is designed to support data engineers and data scientists in such explanation tasks. **DPDS** caters to the large community of Python/Pandas programmers, is focused on tabular data, and is designed to collect, store, and present the provenance of each individual element in a dataframe, across a series of transformations, efficiently and in a very fine-grained fashion.

We consider transformations that apply across application domains, including feature selection; engineering of new features; imputation of missing values; downsampling or upsampling of data subsets in order to achieve better balance, typically on the class labels (for classification tasks) or on the distribution of the outcome variable (for regression tasks); outlier detection and removal; smoothing and normalization; de-duplication, as well as steps that preserve the original information but are required by some algorithms, such as "one-hot" encoding of categorical variables.

While these transformations can be encoded in many possible ways, they all fall into a limited number of categories, depending on the way in which data is changed, namely through reduction, augmentation, transformation, or merging. The tool attempts to make this distinction at runtime, by observing the program's execution and comparing dataframes before and after a particular command or function is applied, typically (but not necessarily) as part of a pipeline. For each of these steps, the observer, called *Provenance Tracker*, produces a provenance document, using the PROV data model [6], by instantiating one of several available templates, one for each of the above-mentioned data transformation categories. Each of such documents records the derivations of each of the affected elements in the before- and -after- dataframes, at the finest granularity possible. For instance, an imputation operation may *modify* entire columns, while a one-hot encoding operation will *invalidate* one column and *generate* a few new ones.

The set of all such documents produced during the program's execution form a larger provenance graph, where nodes representing either dataframe elements or operators are connected through usage/generation as well as derivation edges. The graph is stored in a Neo4J database and made available to a query layer. Through a GUI, analysts can inspect the provenance graph, zoom in on pairs of before/after dataframes of interest, and gain insight into their differences, at a level of granularity that is consistent with the operator type.

This demo paper is underpinned by our recent work [3], where details on the approach can be found. Here we provide a brief overview in Section 2, we describe components and features of

Proceedings of the VLDB Endowment, Vol. 15, No. 12 ISSN 2150-8097.
doi:10.14778/3554821.3554857

DPDS in Section 3, we illustrate how the system will be demonstrated in Section 4, and finally, in Section 5, we draw some conclusions.

2 OVERVIEW OF DPDS

2.1 Data Model and Basic Operators.

Our approach relies on a small but representative set of basic operators for data preparation in machine learning. These operators operate over a generalization of dataframes [5] called *datasets*, which are just ordered collections of *rows* each of which has a unique *index* and includes values for a fixed set of *features*. The core operators that we have considered capture most of the data transformation operators that are available in packages for building data preprocessing pipelines (e.g., Orange [4] and SciKit [7]) and perform four main types of data manipulation over datasets.

Data reduction. These reduce the size of a dataset D by eliminating rows or columns. Two basic data reduction operators are defined over datasets. They are simple extensions of two well-known relational operators: the *(conditional) projection* π_C of D on a subset of its features and the *selection* σ_C of D with respect to a boolean condition C. Feature selection, data selection, and undersampling are examples of transformations that can be implemented by these operators.

Data augmentation. Augmentations increase the size of a dataset D by adding new rows or new features to D. Two basic data augmentation operators are defined over a dataset D: the *vertical augmentation* $\alpha^{\rightarrow}_{f(X):Y}$ of D, which adds to D a new set of features whose new values are obtained by applying a function f to values occurring in D, and the *horizontal augmentation* $\alpha^{\downarrow}_{X:f(Y)}$ of D, which adds one or more new rows to D by applying a (possibly aggregative) function f to values occurring in D. This category captures data transformations such as space transformation, oversampling, string indexer, and one-hot encoder.

Data transformation. Transformations just change the content of a dataset D. One basic data transformation operator $\tau_{f(X)}$ is defined over datasets: it replaces values of D using a function f. Data repair, binarization, discretization, and imputation are examples of data transformations of this type.

Data fusion. A data fusion combines different datasets with two basic operators: the *join* $\bowtie^t_C$, which applies the standard join operation based on a boolean condition C, and the *append* $\uplus$, which adds the rows of a dataset to another dataset possibly extending the result with nulls on their mismatching features. Operations aimed at collecting and combining data from different sources can be implemented by such operators.

In [3] we illustrate how a large variety of pre-processing operators that are used in data preparation pipelines can be suitably captured by the basic operators above or by a composition thereof.

2.2 Representing Data Provenance.

The purpose of data provenance is to extract relatively simple explanations for the existence (or the absence) of some piece of data in the result of complex data manipulations. Along this line, we adopt as the provenance model a subset of the PROV model [6] from the W3C, a widely adopted ontology that formalizes the notion of *provenance document* and which admits RDF and other serialization formats to facilitate interoperability. This model can be graphically described as shown in Figure 1.

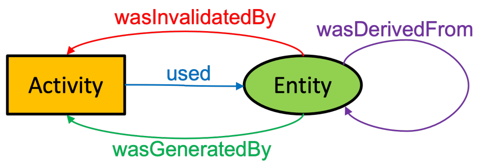

Figure 1: The core W3C PROV model.

In PROV an entity represents an element d of a dataset D and is uniquely identified by D and the coordinates of d in D (i.e., the corresponding row index and feature). An activity represents any pre-processing data manipulation that operates over datasets.

For each element d in a dataset D' generated by an operation $\mathbf{o}$ over a dataset D we represent the facts that: (i) d *wasGeneratedBy* $\mathbf{o}$, and (ii) d *wasDerivedFrom* a set of elements in D. In addition, we represent: (iii) all the elements d of D such that d was *used* by $\mathbf{o}$ and (iv) all the elements d of D such that d *wasInvalidatedBy* (i.e., deleted by) $\mathbf{o}$ (if any). Note that, in PROV, derivation implies usage, but the inverse is not true and this is why this notation is not redundant.

2.3 Capturing Provenance.

We associate a *prov-gen* function $pf_{\mathbf{o}}()$ to each of the core operators. This takes as inputs the sets of input and output datasets D, D' for the operator, and produces a PROV document that describes the transformation produced by the operator on each element of D, as reflected in D'. Note that for binary operators, namely join and append, D includes inputs from both operands.

Due to space constraints, here we illustrate *prov-gen* for only one operator type, namely Vertical Augmentation (VA): $\alpha^{\rightarrow}_{f(\mathbf{Age}):\mathbf{ageRange}}(D)$, where attribute **Age** is binarised into *{young, adult}* based on a pre-defined cutoff, defined as part of $f()$. The reader is referred to [3] for full details on all *prov-gen* functions.

The prov-gen function for VA will have to produce a collection of small PROV documents, one for each input-output pair $\langle D_{i,\mathbf{Age}}, D'_{i,\mathbf{AgeRange}}\rangle$. As these documents all share the same structure, we define a common *PROV template* which is then instantiated multiple times, once for each inputs/output pair. A template is a PROV document that may contain variables, indicated by the namespace *var:*, which are used as placeholders for values. Here templates are designed to capture the transformation at the level of individual elements of D, or its rows or columns, as appropriate. Thus a template will have a *used* set of entities, which refer to the subset of data items in D which have been used by an operator $\mathbf{o}$, and a *generated* set of new entities, corresponding to new elements in D' (for projection and selection, it will have an *invalidated* set of entities instead, as these operators remove data from D).

The PROV template for VA is shown in Figure 2 (top), where we use the generic attribute names X, Y to indicate the old and new feature names. One or more *binding generators* are associated with each template, each of which determines how values found in D,

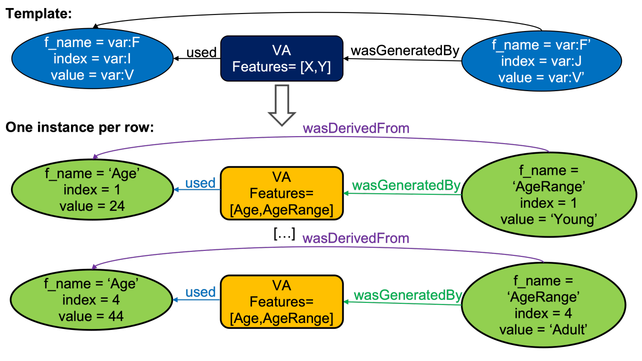

Figure 2: Example of PROV template for Vertical Augmentation and corresponding instances.

D' upon execution of the operator are substituted for the variables. Each variable substitution results in a PROV document, which we refer to as a *provlet*.

In the VA example, the transformation between D and D' is 1:1 and thus a new provlet is created from each value of column **Age** and the corresponding value in **AgeRange**.

Two sample instance provlets obtained from the template for VA are shown in the bottom part of Figure 2.

3 SYSTEM DESCRIPTION

3.1 System Architecture

The main components of **DPDS** are shown in Figure 3. The Provenance-Tracker automates the process of detecting and tracking the provenance of a user-defined pipeline of data preparation. It includes a Prov-generator that produces the provenance of each operator in the pipeline by analyzing its effect on the underlying dataset. This is done at execution time by:

(a) identifying, at each step of the pipeline, the operator under execution on the basis of a series of increasingly complex comparisons between the input and the output datasets,
(b) executing the prov-gen function of the core operation that captures the identified operator by suitably instantiating the function template, and
(c) storing the provenance data produced by the prov-gen function on an underlying repository.

Since provenance data have a natural graphical representation, Neo4j, a popular graph database management system, is used for this purpose.

The Query-generator allows the user to perform several types of analyses of the data provenance collected for a given data preparation pipeline, by translating a specific data-provenance exploration chosen from a menu of a graphical interface into a query expressed in Cypher, the query language of Neo4j.

3.2 Main Features of DPDS

Unlike many provenance visualization tools, which focus on the presentation and navigation of the provenance graph, in **DPDS** the provenance graph is mainly used as a backbone to support

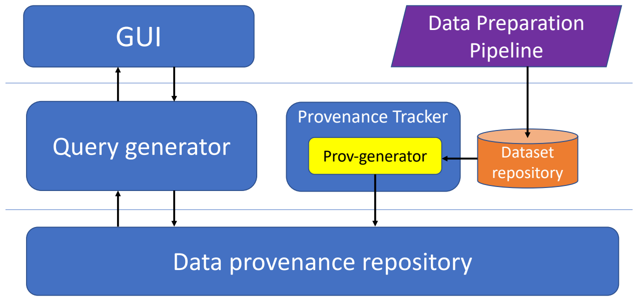

Figure 3: DPDS architecture

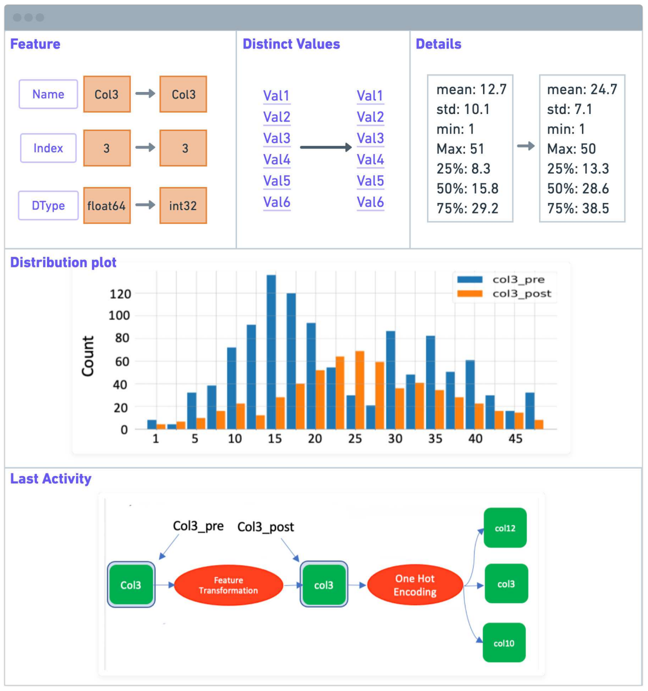

Figure 4: DPDS interface example showing how data changes after a transformation process that operates locally, at the feature level.

the exploration of the dataset characteristics and to highlight the changes through the transformations.

Specifically, visual exploration proceeds in two steps. First, users can query the provenance trace and thus "zoom in" to the "before/after" transformations of interest. Depending on the type of operator that was applied, this may be at the level of values within a column, in the case of a *local* transformation, or at the level of the entire framework, in the case of a *global* transformation. A local type of transformation is illustrated in Figure 4, where a dropna() operation, which modifies values in col3, is represented in the provenance fragment at the bottom. The user is then able to navigate through the retrieved provenance, identify the pre- and post- states

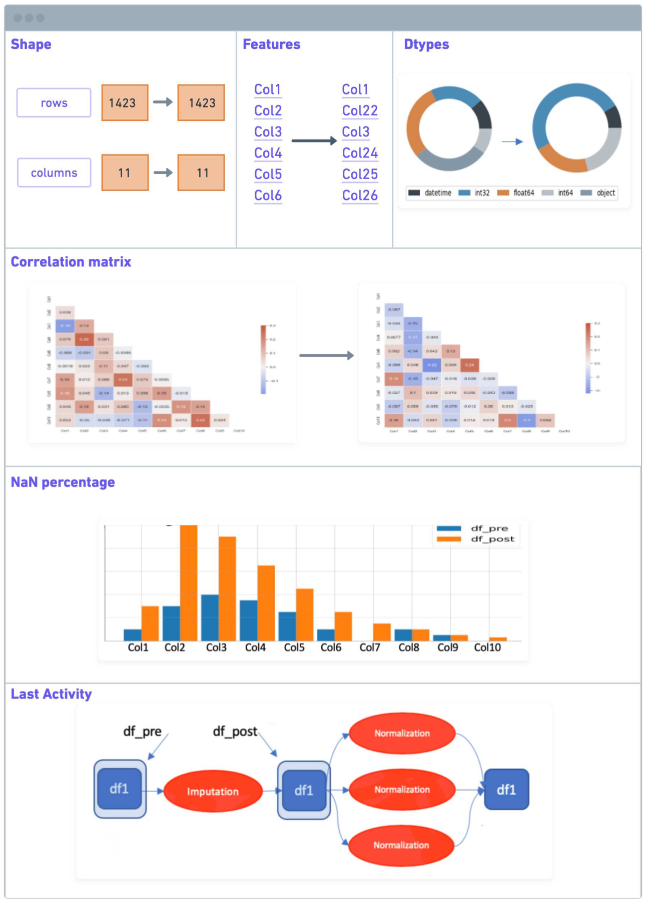

Figure 5: DPDS interface example showing how data changes at the dataset level, after an imputation operation over several columns.

for col3, and visualise their differences in terms of summary statistics (top right), values distribution (center), and optionally each value can be inspected (top middle).

In contrast Figure 5 shows a global transformation which is the effect of an imputation step. As this may change more than one column at the time (for instance, using Multiple Inference, as in MICE), in this case the dashboard displays salient differences at the dataframe level. We can see for instance that the operation has not changed the number of rows and columns of the dataframe (top left), but the imputation has updated the values of several columns (col2, col4, col5, col6, see top middle), and has altered the percentage of null values (bar chart). We can also see the changes in the correlation between each pair of columns in the dataframes before and after the operation is performed.

4 DEMO OUTLINE

In our demonstration, we will simulate typical scenarios of data preprocessing in data science, which are aimed at giving a comprehensive view of **DPDS** and leading to discussions on supporting data science with provenance data. We highlight the minimum impact provenance collection using our technique has on the fundamental data science development and the utility of the provenance information collected

Scenario A: provenance capture. In this scenario, we show how **DPDS** can be used within minimal overhead for a data science developer. The audience will be provided with a working data science pipeline, COMPAS, which has been used to predict recidivism and has been shown to be unfair to minority groups [2]. The audience will be asked to *capture* provenance across the data science pipeline to showcase how simple it is, and how little it impacts the data scientists' actions.

Scenario B: provenance modelling choices. Using the provenance captured from Scenario A, we then show how relationships are generated among the provenance collected. The audience can change the configuration of **DPDS** and review how the relationships in the provenance store are generated offline and stored.

Scenario C: data diff. In this scenario, we highlight the utility of the provenance to the data scientist. The audience is invited to explore the interface that allows data scientists to drill down on the datasets, their content, and the changes that were introduced through the transformations in the data science pipeline. The audience will be able to iterate through Scenarios A, B, and C to explore the impact the capture choices in Scenario A have on the relationships in the provenance from Scenario B and their detailed effect on the values of the dataframes, in Scenario C.

5 CONCLUSION

DPDS is a tool supporting data specialists to collect, store, and investigate the provenance of each individual element in a dataset across a data preparation process, efficiently and in a very fine-grained fashion.

Using **DPDS**, the data scientist can understand how preprocessing steps change the data profile of any dataset used within the data science process. The provenance captured facilitates explanations each component of a pipeline produces on each element of each of the datasets used in modelling, for instance as training sets.

This demonstration aims at showing the distinctive tools that **DPDS** offers to data scientists, their minimal impact on the data science process, and the ability they provide to explore and introspect into development choices.

REFERENCES

[1] Ahmed M Alaa and Mihaela van der Schaar. 2019. Demystifying Black-box Models with Symbolic Metamodels. In *Proc. of Neural Information Processing Systems.* 11301–11311.
[2] Julia Angwin, Jeff Larson, Surya Mattu, and Lauren Kirchner. 2016. Machine bias. *ProPublica, May* 23 (2016), 139–159.
[3] Adriane Chapman, Paolo Missier, Giulia Simonelli, and Riccardo Torlone. 2020. Capturing and querying fine-grained provenance of preprocessing pipelines in data science. *Proc. of VLDB Endowment* 14, 4 (2020), 507–520.
[4] Janez Demšar et al. 2013. Orange: Data Mining Toolbox in Python. *The Journal of Machine Learning Research* 14, 1 (2013), 2349–2353.
[5] Devin Petersohn et al. 2020. Towards Scalable Dataframe Systems. *Proc. of VLDB Endowment* 13, 11 (2020), 2033–2046.
[6] Moreau et. al. 2012. *PROV-DM: The PROV Data Model.* Technical Report. World Wide Web Consortium. http://www.w3.org/TR/prov-dm/ (last accessed: June 2022).
[7] Pedregosa et al. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011), 2825–2830.
[8] Marco Tulio Ribeiro, Sameer Singh, and Carlos Guestrin. 2016. "Why should i trust you?": Explaining the predictions of any classifier. In *Proc. of KDD.* 1135–1144.