Fine-grained Provenance for High-quality Data Science

(Discussion Paper)

Adriane Chapman[1], Paolo Missier[2], Giulia Simonelli[3] and Riccardo Torlone[3]

[2]University of Southampton
[2]Newcastle University
[3]Università Roma Tre

Abstract

In this work we analyze the typical operations of data preparation within a machine learning process, and provide infrastructure for generating very granular provenance records from it, at the level of individual elements within a dataset. Our contributions include: (i) the formal definition of a core set of preprocessing operators, (ii) the definition of provenance patterns for each of them, and (iii) a prototype implementation of an application-level provenance capture library that works alongside Python.

1. Introduction

Data processing pipelines that are designed to clean, transform and alter data in preparation for learning predictive models, have an impact on those models' accuracy and performance, as well on other properties, such as model fairness.

However, while substantial recent research has produced techniques for model explanation that focus primarily on the model itself (e.g., [1, 2]) relatively little work has been done into trying to explain models in terms of the transformations that occur *before* the data is used for learning.

In this work, we enable the explanation on the effect of each transformation in a pre-processing pipeline on the data that is ultimately fed into a model. We consider transformations that apply to commonly used tabular or relational datasets and across application domains. These steps have been systematically enumerated in multiple reviews (see eg. [3]) and include, among others: feature selection, engineering of new features; imputation of missing values, or *listwise deletion* (excluding an entire record if data is missing on any variable for that record); 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 normalisation; de-duplication, as well as steps that preserve the original information but are required by some algorithms, such as "one-hot"

SEBD 2021: The 29th Italian Symposium on Advanced Database Systems, September 5-9, 2021, Pizzo Calabro (VV), Italy

adriane.chapman@soton.ac.uk (A. Chapman); paolo.missier@ncl.ac.uk (P. Missier); giulia.simonelli@uniroma3.it (G. Simonelli); giulia.simonelli@uniroma3.it (R. Torlone)

encoding of categorical variables. A complex pipeline may include some or all of these steps, and different techniques, algorithms, and choice of algorithm-specific parameters may be available for each of them. These are often grounded in established literature but variations can be created by data scientists to suit specific needs. We consider the space of all configured pipelines that can potentially be composed out of these operators, and we focus on relational datasets, which are arguably the most common data structures in popular analytics-friendly scripting languages like R, Spark, and Python (where they are called *dataframes*).

In this framework, we propose a formalisation and categorisation of a core set of these operators. Data derivations for such operators are expressed at the level of the atomic elements in the dataset using the PROV data model [4], a standard and a widely adopted ontology. Then, with each of the core operators we associate a *provenance pattern* that describes their effect on the data at the appropriate level of detail, i.e., on individual dataset elements, columns, rows, or collections of those. Provenance patterns defined in this work for data science operators play a similar role to that of *provenance polynomials* [5], i.e., annotations that are associated to relational algebra operators to describe the fine-grained provenance of the result of relational operators. Finally, we associate a provenance function $pf_{\mathbf{o}}()$ to each operator $\mathbf{o}$, which generates a provenance document $pf_{\mathbf{o}}(D)$ when a dataset D is processed using $\mathbf{o}$. The document is an instance of the pattern associated with $\mathbf{o}$. Provenance functions are implemented as part of a Python module. Collecting all the provenance documents from each operator's execution results in a seamless, end-to-end provenance document that contains the detailed history of each dataset element in the final training set, including their creation (e.g. as a new derived feature), transformation (value imputation, for example) and possibly deletion (e.g., by feature selection, removal of null values).

In the rest of this paper we illustrate the models we have adopted for describing data, operators, and provenance (Section 2) as well as the way in which provenance for a data preprocessing pipeline is captured (Section 3). Further details on the query capabilities of the resulting provenance and on the scalability of the overall approach can be found in the extended version of the paper [6].

2. Models for Data, Operators, and Provenance.

2.1. Data model

A *(dataset) schema* S is an array of distinct names called *features*: $S = [\mathbf{a}_1, \ldots, \mathbf{a}_n]$. Each feature is associated with a domain of atomic values (such as numbers, strings, and timestamps). A *dataset* D over a schema $S = [\mathbf{a}_1, \ldots, \mathbf{a}_n]$ is an ordered collection of *rows* (or *records*) of the form: $i : (d_{i1}, \ldots, d_{in})$ where i is the unique *index* of the row and each element d_{ij} (for $1 \leq j \leq n$) is either a value in the domain of the feature $\mathbf{a}_j$ or the special symbol $\perp$, denoting a missing value. Given a dataset D over a schema S we denote by $D_{i\mathbf{a}}$ the element for the feature $\mathbf{a}$ of S occurring the i-th row of D. We also denote by D_{i*} the i-th row of D, and by $D_{*\mathbf{a}}$ the column of D associated with the feature $\mathbf{a}$ of S.

Example 1. *A possible dataset D over the schema $S = [\textit{\textbf{CId}}, \textit{\textbf{Gender}}, \textit{\textbf{Age}}, \textit{\textbf{Zip}}]$ is as follows:*

	CId	***Gender***	***Age***	***Zip***
1	113	F	24	98567
2	241	M	28	$\perp$
3	375	C	$\perp$	32768
4	578	F	44	32768

$D_{*\textit{\textbf{Age}}}$ and D_{2*} denote the third column and the second row of D, respectively. $\square$

2.2. Data manipulation model

The data transformation operators that are available in packages for building data preprocessing pipelines (e.g., Orange [7] and SciKit [8]) can be classified in three main classes, according to the type of manipulation done on the input dataset D over a schema S, as follows.

Data reductions. They reduce the size of D by eliminating rows (without changing S) or columns (changing S to $S' \subset S$) from D. Two basic data reduction operators are defined over datasets. They are simple extensions of two well known relational operators.

π_C: the *(conditional) projection* of D on a set of features of S that satisfy a boolean condition C over S, denoted by $\pi_C(D)$, is the dataset obtained from D by including only the columns $D_{*\mathbf{a}}$ of D such that $\mathbf{a}$ is a feature of S that satisfy C;

σ_C: the *selection* of D with respect to a boolean condition C over S, denoted by $\sigma_C(D)$, is the dataset obtained from D by including the rows D_{i*} of D satisfying C.

The condition of both the projection and the selection operators can refer to the values in D, as shown in the following example that use an intuitive syntax for the condition.

Example 2. *Consider the dataset D in Example 1. The result of the expression $\pi_{\{features\ without\ nulls\}}(\sigma_{\textit{\textbf{Age}}<30}(D))$ is the following dataset:*

	CId	***Gender***	***Age***
1	113	F	24
2	241	M	28

$\square$

Data augmentations. They increase the size of D by adding rows (without changing S) or columns (changing S to $S' \supset S$) to D. Two basic data augmentation operators are defined over datasets. They allow the addition of columns and rows to a dataset, respectively.

$\alpha^{\rightarrow}_{f(X):Y}$: the *vertical augmentation* of D to Y using a function f over a subset of features $X = [\mathbf{a}_1 \ldots \mathbf{a}_k]$ of S is obtained by adding to D a new set of features $Y = [\mathbf{a}'_1 \ldots \mathbf{a}'_l]$ whose new values $d_{i\mathbf{a}'_1} \ldots d_{i\mathbf{a}'_l}$ for the i-th row are obtained by applying f to $d_{i\mathbf{a}_1} \ldots d_{i\mathbf{a}_k}$;

$\alpha^{\downarrow}_{X:f(Y)}$: the *horizontal augmentation* of D using an aggregative function f is obtained by adding one or more new rows to D obtained by first grouping over the features in X and then, for each group, by applying f to $\pi_Y(D)$ (extending the result to S with nulls if needed).

Example 3. *Consider again the dataset D in Example 1 and the following functions: (i) f_1, which associates the string* young *to an age less than 25 and the string* adult *otherwise, and (ii) f_2, which computes the average of a set of numbers. Then, the expression $\alpha^{\rightarrow}_{f_1(\textbf{\textit{Age}}):\textbf{\textit{ageRange}}}(D)$ produces the following dataset:*

	CId	***Gender***	***Age***	***Zip***	***ageRange***
1	113	F	24	98567	*young*
2	241	M	28	$\perp$	*adult*
3	375	C	$\perp$	32768	$\perp$
4	578	F	44	32768	*adult*

whereas $E_2 = \alpha^{\downarrow}_{\textbf{\textit{Gender}}:f_2(\textbf{\textit{Age}})}(D)$ the dataset:

	CId	***Gender***	***Age***	***Zip***
1	113	F	24	98567
2	241	M	28	$\perp$
3	375	C	$\perp$	32768
4	578	F	44	32768
5	$\perp$	F	34	$\perp$
6	$\perp$	M	28	$\perp$

□

Data transformation. The goal is to transform (some of) the elements in D without changing its size or its schema. One basic data transformation operator is defined over datasets:

$\tau_{f(X)}$: the *transformation* of a set of features X of D using a function f is obtained by replacing each value $d_{i\mathbf{a}}$ with $f(d_{*\mathbf{a}})$, for each **a** occurring in X.

Example 4. *Let D be the dataset in Example 1 and f be an imputation function that associates with the $\perp$'s occurring in a feature* **a** *the most frequent value occurring in $D_{*\mathbf{a}}$. Then, the result of $\tau_{f(\textbf{\textit{Zip}})}(D)$ is the following dataset:*

	CId	***Gender***	***Age***	***Zip***
1	113	F	24	98567
2	241	M	28	32768
3	375	C	$\perp$	32768
4	578	F	44	32768

□

In [6] we illustrate how a large variety of pre-processing operators that are often used in data preparation workflows (including feature and instance selection, data repair, binarization, normalization, discretization, imputation, space transformation, and One-Hot encoder) can be suitably expressed as composition of the basic operators introduced in this section.

2.3. Data provenance model

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 [9] from the W3C.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).

Example 5. *Let E be the first expression in Example 3 and $D' = E(D)$. A fragment of the provenance generated by this operation is reported in Figure 1.* □

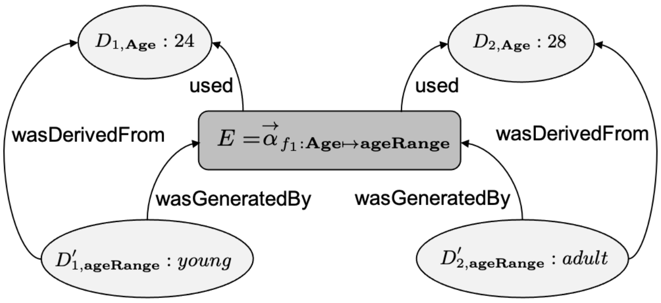
$D_{1,\mathbf{Age}} : 24$
used
$D_{2,\mathbf{Age}} : 28$
used
wasDerivedFrom
$E = \overrightarrow{\alpha}_{f_1:\mathbf{Age}\mapsto\mathbf{ageRange}}$
wasDerivedFrom
wasGeneratedBy
wasGeneratedBy
$D'_{1,\mathbf{ageRange}} : young$
$D'_{2,\mathbf{ageRange}} : adult$

Figure 1: A fragment of provenance data for the operation in Example 5.

3. Capturing Provenance

3.1. Provenance templates

In order to capture the provenance of a pipeline p of a sequence of preprocessing operations $\mathbf{o}_1, \ldots, \mathbf{o}_n$, we associate a provenance-generating function (*p-gen*) with each operation $\mathbf{o}_k$ occurring in p. Each such function generates a collection of provenance data whenever a dataset is processed using $\mathbf{o}_k$, which describes the effect of $\mathbf{o}_k$ on the data at the appropriate level of detail. As a large variety of preprocessing operators can be defined in terms of our five core pipeline operators [6], it is enough to define a p-gen function for each of these operators.

Each p-gen function takes inputs D, D' (the inputs and outputs of their associated operator) along with a description of the operator itself, and produces a PROV document that describes the transformation produced by the operator on each element of D. The output PROV document is obtained by instantiating an appropriate provenance template [10], which is designed to capture the transformation at the most granular level, i.e., at the level of individual elements of D, or its rows or columns, as appropriate. A template is simply a PROV document where: (i) variables, indicated by the namespace **var:**, are used as placeholders for values and (ii) a set of rules is used to specify how the "used" and the "generated" sides of the template are repeatedly instantiated, by binding the variables to each of the data items involved in the transformation. We refer to each instantiated template produced by a p-gen function as a *provlet.*

Take for example the case of Vertical Augmentation (VA): $\alpha^{\rightarrow}_{f_1(\mathbf{Age}):\mathbf{ageRange}}(D)$ which we used in Example 3, where attribute **Age** is binarised into *{young, adult}* based on a pre-defined cutoff, defined as part of $f()$. The p-gen function for VA will produce a collection of small PROV documents, one for each input-output pair $\langle D_{i,\mathbf{Age}}, D'_{i,\mathbf{AgeRange}}\rangle$ as shown in the example. As these documents all share the same structure, we specify p-gen by giving two elements. First, a single PROV template for (VA) as shown in Figure 2, where we use the generic attribute names X, Y to indicate the old and new feature names, as per the operator's general definition in Section 2.2.

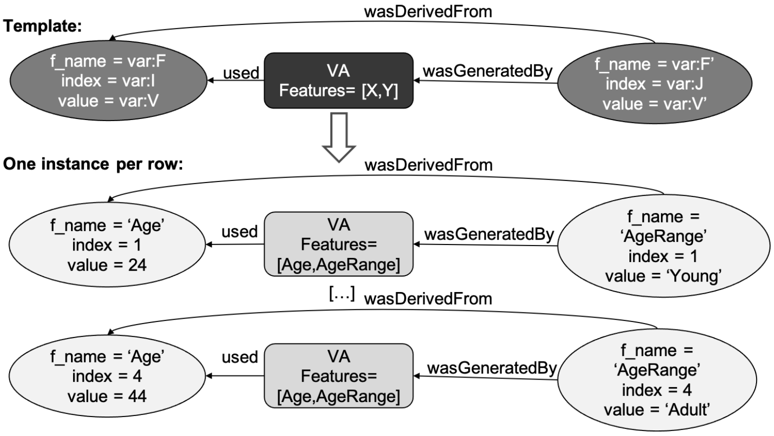

Figure 2: PROV template for Vertical Augmentation and corresponding instances.

3.2. Code instrumentation

Approaches for automated provenance capture, such as by using the python call stack fail to capture data provenance at the level of the individual element within a dataframe. To accomplish this, in our initial prototype, we opted for explicit and analyst-controlled instrumentation at the script level. We have packaged the implementation of the p-gen functions described in the previous section as a python library that analysts can add to their code where provenace capture is desired. Note also that it may be possible to automate function call injection, at least in part, by leveraging mature code annotation tools. While this does not completely eliminate the need for manual intervention, this is now a simple comment/ annotation effort (which can be driven by a smart UI) rather than requiring additional programming.

3.3. Generating provenance documents

A complete provenance document is produced by combining the collection of provlets that results from calling p-gen functions. Specifically, one provlet is generated for every transformation and every element in the dataframe that are affected by that transformation. The document is represented by such collection of provlets, where entity identifiers match across provlets, and never needs to be fully materialised, as explained shortly. To illustrate how provlets are generated, consider the following pipeline: $\sigma_C(\alpha^{\rightarrow}_{f_1(\mathbf{Age}):\mathbf{ageRange}}(D))$ where $C = \{\mathbf{AgeRange} \neq \text{'Young'}\}$ and D is the dataset of Ex. 3. The corresponding provenance document is represented in Figure 3

Applying vertical augmentation produces one provlet for each record in the input dataframe, showing the derivation from **Age** to **AgeRange**. The second step, selecting records for 'not

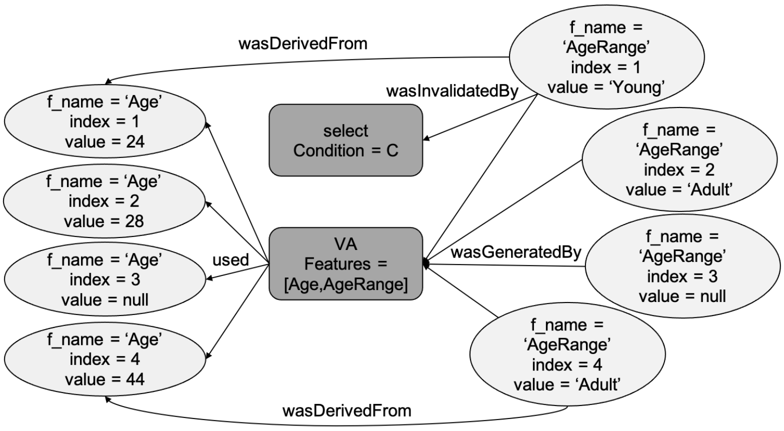

Figure 3: Provlet composition.

young' people, produces the new set of provlets on the right, to indicate invalidation of the first record. Note that the "used" side on the left refers to existing entities, which are created either into the pipeline from the input dataset, or by an upstream data generation operator.

Provlet composition requires looking up the set of entities already produced, whenever a new provlet is added to the document. For this, we have built a bespoke architecture that allows lazy provenance composition. Each p-gen function generates a set of provlets, one for each element in the dataframe (in the worst case), constructs a partial document, and stores it to a persistent MongoDB back end. This allows the provenance to be collected quickly at execution of each script, and be assembled later, minimizing execution dependencies and possible bottlenecks during the actual execution of the pipeline.

Concretely, each *p-gen* function creates, *at query time*, a provenance object containing all provlets, and an input json file representing the input dataframe. By capturing provlets from each p-gen function, it is possible to compose these provlets into a complete graph, which can be traversed as a bipartite graph for any d_{ij}. The process for composing provlets, and tracing the influence (either direct of indirect) of data and operations on d_{ij} can then support "why provenance" [11].

4. Conclusions

In this work, we have illustrated an approach for producing fine-grained data provenance in machine learning pipelines, irrespective of the pipeline tool used. Indeed, because a substantial effort goes into selecting and preparing data for use in modelling, and because changes made during preparation can affect the ultimate model, it is important to be able to trace what is happening to the data at a such level of detail. Using an implementation of this system, we have demonstrated the utility and performance of our approach over real-world ML benchmark pipelines. In order to investigate scalability issues with our design, we also used the TPC-DI generator and apply several operators over that data at scale. Our results indicate that we can collect fine-grained provenance that is both useful and performant [6].

References

[1] A. M. Alaa, M. van der Schaar, Demystifying black-box models with symbolic metamodels, in: Advances in Neural Information Processing Systems, 2019, pp. 11301–11311.

[2] M. T. Ribeiro, S. Singh, C. Guestrin, “Why should i trust you?”: Explaining the predictions of any classifier, in: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, 2016, pp. 1135–1144.

[3] S. García, et.al., Big data preprocessing: methods and prospects, Big Data Analytics 1 (2016).

[4] L. Moreau, P. Missier, K. Belhajjame, R. B’Far, J. Cheney, S. Coppens, S. Cresswell, Y. Gil, P. Groth, G. Klyne, et al., Prov-dm: The prov data model. w3c recommendation rec-prov-dm-20130430, WWW Consortium (2013).

[5] V. Green, Todd J., Karvounarakis, G., Tannen, Provenance Semirings, in: PODS, 2007, pp. 31–40.

[6] A. Chapman, P. Missier, G. Simonelli, R. Torlone, Capturing and querying fine-grained provenance of preprocessing pipelines in data science, Proc. VLDB Endow. 14 (2020) 507–520.

[7] J. Demšar, et al., Orange: Data mining toolbox in python, Journal of Machine Learning Research 14 (2013) 2349–2353.

[8] Pedregosa, et al., Scikit-learn: Machine learning in Python 12 (2011) 2825–2830.

[9] L. Moreau, J. Cheney, P. Missier, Constraints of the prov data model, 2013. URL: http://www.w3.org/TR/2013/REC-prov-constraints-20130430/.

[10] L. Moreau, B. V. Batlajery, T. D. Huynh, D. T. Michaelides, H. S. Packer, A templating system to generate provenance, IEEE Transactions on Software Engineering 44 (2018) 103–121.

[11] P. Buneman, S. Khanna, T. Wang-Chiew, Why and where: A characterization of data provenance, in: ICDT, Springer, 2001, pp. 316–330.