xpandas - python data containers for structured types and structured machine learning tasks

Vitaly Davydov[*]
Department of Statistical Science University College London
Gower Street, London WC1E 6BT, United Kingdom

Franz J. Király[†]
Department of Statistical Science University College London
Gower Street, London WC1E 6BT, United Kingdom

Abstract

Data scientific tasks with structured data types, e.g., arrays, images, time series, text records, are one of the major challenge areas of contemporary machine learning and AI research beyond the "tabular" situation - that is, data that fits into a single classical data frame, and learning tasks on it such as the classical supervised learning task where one column is to be predicted from others.
With xpandas, we present a python package that extends the pandas data container functionality to cope with arbitrary structured types (such as time series, images) at its column/slice elements, and which provides a transformer interface to scikit-learn's pipeline and composition workflows.
We intend xpandas to be the first building block towards scikit-learn like toolbox interfaces for advanced learning tasks such as supervised learning with structured features, structured output prediction, image segmentation, time series forecasting and event risk modelling.

1 Introduction to XPandas

1.1 Motivation

Consider the three following advanced analytics scenarios:

(A) A probabilistic classifier for identifying audio signals as (human spoken) "yes" and "no" is to be developed.
(B) An image is to be segmented into multiple regions of interest.
(C) A medical AI is to be trained to determine likelihood of therapy success based on longitudinal electronic health records.

All of these scenarios have in common that the data has an element of hierarchical structure: in (A), there are multiple paired samples of time series (audio signals) and the label; in (B) there are multiple images, each containing multiple pixels, and for each image there may be multiple segments that in turn contain multiple pixels. Finally in (C), each patient may have individual characteristics as well as different time series of clinical measurements and unstructured text data associated with them, as well as a structured record of therapy success.

[*]`v.davydov@ucl.ac.uk`
[†]`f.kiraly@ucl.ac.uk`

Preprint. Work in progress.

What these scenarios unfortunately also have in common that at the current state-of-art, a data scientist and machine learner would spend considerable amount of time to convert the data into a tabular format, e.g., a data frame, for which toolbox modelling interfaces such as scikit-learn provide functionality. Or, they would spend considerable amount of time to convert the data into a hand-crafted combination of lists, arrays, dictionaries, and/or data frames, then hand-crafting a modelling pipeline.

Given the elegance with which contemporary modelling toolboxes (such as scikit-learn, mlr) provide modelling workflow interfaces for the tabular situation, it is natural to close this gap for the more advanced tasks, through designing, then implementing, formalism and respective functionality.

1.2 Features and use cases

The **XPandas** package for python offers a solution by extending the functionality of classical data frame style containers to that of a type-aware 2D-container for arbitrary structured entry types, together with an in-built interface for the scikit-learn pipeline formalism. Key features are:

(i) the **XPandas** data containers: 1D series-like, 2D data-frame-like and n-D data-array-like, with columns capable of storing or referencing objects of any type, compatible with pandas
(ii) column-wise transformation and summarization functionality
(iii) a scikit-learn-like transformation/pipeline interface, extensible by custom transformers, and compatible with scikit-learn transformers/pipelines
(iv) interface with specific feature extraction and transformation packages including tsfresh (time series) and scikit-image (images).

Supported core use cases are

(i) storage of complex hierarchical data, e.g., samples of images, series, custom objects, for convenient manipulation and easy access in-memory
(ii) data summarization and slice-wise transformation
(iii) building of scikit-learn pipelines involving initial feature extraction from complex input data types

We would anticipate support for complex output data types and scikit-learn-like learning tasks to be provided by separate packages using **XPandas** , rather than as extended **XPandas** core functionality.

1.3 Related and prior work

The R project's data frame is the seminal design for our work. For python, it is replicated with a pythonic interface in the Pandas [1] package, as a container design which includes attached summarization/transformation functionality. It is this design which we intend to extend with **XPandas** ("eXtended Pandas"). The R package S4vectors [2] already implements a type-generic generalization of the R data frame container, but without an abstracted transformer or pipeline interface for subsequent modelling workflows. scikit-learn[3] is the basis for our transformer/pipeline design, with the scikit-learn design, python's object orientation and python's duck typing greatly facilitating the interfacing.

2 Container and interface formalism

Intuitively, the **XPandas** containers are data frames with arbitrary entry type. We introduce some mathematical formalism from first-order type theory to make the container structure precise, as well as its interaction with the transformer/pipeline formalism of scikit-learn. For ease of notation (but by abuse of notation), we will use types and the sets of inhabitants interchangeably. The main difference between the two views should be noted, as it is important for implementation: unlike for mere set elements, the type is *queriable*, i.e., a typed object "knows" which type it is of (i.e., which domain set it came from), and the type may be queried in the computer.

Semi-formally, a classical data frame may be obtained as a list of typed lists of equal length, where the inner list types are primitive. As data scientific primitive types, one usually considers numbers (integers, floats, dates/durations, etc, e.g., length of hospital stay), categories (values in a pre-specified finite set, e.g., $\{$male, female, other$\}$), and strings. Formally:

Definition 1 *Let T be a type (identified with its inhabitant set).*
A typed list of length $N \in$, with type T, has first-order type $(T; N)$ and inhabitant set T^N, i.e., the ordered tuples in T of length N.
A data frame of $N \in$ samples and $M \in$ variables with variable types $T_1, \dots, T_M$, has first-order type $(T_1, \dots, T_M; N)$ and inhabitant set $(T_1) \times \dots \times (T_M)$.

For classical implementations of data frames - such as in Pandas and R - all variable types in Definition 2 are primitive. The expectation that this is the case is usually mirrored in the implementation: operations with the data container that include non-primitive types either directly cause errors, or lead to unexpected and unsupported behaviour in standard workflows.

XPandas provides a robust implementation of typed lists and data frames (as well as higher-dimensional generalizations not discussed due to space constraints), where the type T in Definition 2 may be anything: a decorated 2D or 3D array (e.g., x-ray image), a time series (e.g., heartbeat rate), or any abstract class, including a Pandas or **XPandas** data frame.

The transformer formalism - found in scikit-learn as the eponymous class, or in mlr in the form of wrappers - abstracts operations on data frames such as variable transformations and feature extraction algorithms:

Definition 2 *Let $T := (T_1, \dots, T_M)$ be a tuple of (input) types, let $T' := (T'_1, \dots, T'_{M'})$ be a tuple of (output) types. A transformer is a first-order type $(T \rightarrow T')$ of composite/object kind.*
Usually, it has a fitting sub-method $. : [(T; N) \rightarrow]$, and a transformation method $. : [(T; N) \rightarrow (T'; N)]$.
(for expository convenience, we do not formalize the otherwise important hyper-parameter interface)

Transformers may be composed in pipelines with modelling composite types, such as estimator/predictors which may also be viewed as first-order composite types, e.g., $(T' \rightarrow T'')$. Pipelines are built by composition, such as $(T \rightarrow T') \times (T' \rightarrow T'') \rightarrow (T \rightarrow T'')$. We refrain from giving an exhaustive exposition due to space constraints.

Common examples of transformers are, for example, variable normalization of number types, one-hot encoding of categorical types, or computation of principal component scores. In these examples, both input types and output types are primitive. For non-primitive types, the simplest case is feature extraction, where T may contain non-primitive types, but T' contains only primitive types. One example is extraction of numerical features from individual time series $(() \rightarrow^{n})$, as done by the tsfresh package. Another example is extraction of numerical features from greyscale images, $(^{m\times n}\rightarrow^{n})$, as done by some methods in scikit-image. Advanced examples create non-primitive types from non-primitive types, such as image filters, or smoothing/binning of time series.

For ease of use, the **XPandas** interface distinguishes transformers arising directly from a by-sample transformation $f : T \rightarrow T'$ applied to all rows, from the more general transformers.

3 Interface design and implementation

Following Definition 2, **XPandas** implements, as core components, typed lists in its *XSeries* class, and type-generic data frame containers as *XDataFrame* . The API is fully compatible with pandas and identical if primitive types are stored.

The lower-order type schema of an *XDataFrame* can be queried by the $data\_type$ attribute which returns the type tuple.

XPandas provides two ways to construct transformers: the *CustomTransformer* class allows specification of an *XSeries* transformer which may transform *XSeries* to *XSeries* or *XDataFrame* .

DataFrameTransformer supports a full transformer interface of transforming *XDataFrame* to *XDataFrame* .

All transformers can be used as components of scikit-learn pipelines - directly, if output types are primitive, or otherwise through the **XPandas** *PipelineChainTransformer* interface.

A full tutorial with usage examples and source code is available at `https://github.com/alan-turing-institute/xpandas/blob/master/examples/ExampleUsage.ipynb`.

References

[1] W. McKinney. pandas: a foundational python library for data analysis and statistics.

[2] H. Pagès, M. Lawrence, and P. Aboyoun. S4vectors: S4 implementation of vectors and lists. *R package version 0.13*, 15, 2017.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.