skpro: A domain-agnostic modelling framework for probabilistic supervised learning

Frithjof Gressmann[*]
Department of Statistical Science
University College London
Gower Street, London WC1E 6BT, UK
frithjof.gressmann.16@ucl.ac.uk

Franz J. Király
Department of Statistical Science
University College London
Gower Street, London WC1E 6BT, UK
f.kiraly@ucl.ac.uk

Abstract

We present `skpro`, a Python framework for domain-agnostic probabilistic supervised learning. It features a *scikit-learn*-like general API that supports the implementation and fair comparison of both Bayesian and frequentist prediction strategies that produce conditional predictive distributions for each individual test data point. The `skpro` interface also supports strategy optimization through hyperparamter tuning, model composition, ensemble methods like bagging, and workflow automation. The package and documentation are released under the BSD-3 open source license and available at GitHub.com/alan-turing-institute/skpro.

1 Introduction

Probabilistic supervised learning endeavours to make predictions in settings where even perfect prior knowledge does not allow for an exact label prediction. Probabilistic predictions model the uncertainty inherent in the prediction by specifying a full predictive distribution. In a supervised context, probabilistic prediction problems have been tackled through various strategies in both the frequentist and the Bayesian domain. While a variety of learning algorithms are available that make predictions in the form of probability distributions, they are difficult to instantiate together in a single workflow, e.g., for fair comparison, or higher-order meta-modelling (tuning, ensembling).

The `skpro` package provides a unified, domain-agnostic interface for probabilistic supervised learning with these use cases in mind. Its design is heavily based on the scikit-learn library [11]. `skpro` also provides experimental integration for supervised methods constructed in popular Bayesian toolboxes such as PyMC3 [12] as it aspires to connect a variety of projects.

2 Data scientific use case: probabilistic supervised learning

The principal use cases that skpro tries to address are the probabilistic supervised learning task - most prominently, probabilistic supervised regression - and the associated training, prediction, and validation workflows.

Mathematically, we are in the supervised learning setting of i.i.d. training data $(X_1, Y_1), \ldots, (X_N, Y_N) \sim (X, Y)$, taking values in $\mathcal{X} \times \mathcal{Y}$. Univariate regression is the case where $\mathcal{Y} = \mathbb{R}$. Probabilistic supervised learning is the endeavour to produce a "good" probabilistic prediction functional $f : \mathcal{X} \to \mathcal{P}$, where $\mathcal{P}$ is a family of distributons over $\mathcal{Y}$, e.g., absolutely continuous distributions. The functional f is considered "good" if $f(x)$ is close to the conditional law of $Y|X = x$, also known as the conditional distribution of Y given $X = x$. Closeness is measured by certain probabilistic loss functions, also known as proper scoring rules.

[*]current affiliation: Graphcore Research

32nd Conference on Neural Information Processing Systems (NIPS 2018), Montréal, Canada.

Note that $f(x)$ is an *explicit* representation of the conditional predictive distribution, hence at the same time a prediction, as well as an explicit representation of (estimated) knowledge about the uncertainty in the prediction.

A full theoretical discussion of the probabilistic prediction setting which motivates the skpro package is given in Gressmann et al. [5]; this also includes an extensive discussion of the theoretical setting in the context of prior literature (frequentist and Bayesian, statistical and machine learning). The primary purpose of this manuscript, however, is introducing the skpro package.

3 Key features

Key features of skpro are:

Fit-predict interface for probabilistic prediction strategies: Following a common interface design in the supervised setting, skpro encapsulates probabilistic prediction strategies in a base object `ProbabilisticEstimator` from which all models inherit. The base class implements the `Estimator` API of the scikit-learn project [2] and stays close to its syntax and logic whenever possible. In particular, it provides the training method `fit`(training features X, corresponding training labels y) and the prediction interface `predict`(test features X) which returns a predicted distribution for each test data point.

Explicit representation of distribution properties at distributional level: Notably, since the predicted distribution at a feature point is not a random variable but an explicit distribution, the object that represents the distribution provides a direct and self-explanatory interface to its distributional properties (e.g. the mean or the predicted density function etc.). This explicit representation of distributional properties implies that predictions need to be readily represented on a distributional level and not, say, as a sample from the associated random variable.

Vectorized implementation: As prediction is usually required for a number of points, learning algorithms often allow for concurrent and efficient parallel processing, e.g. Gaussian process models. Hence, the API allows for fast vectorized implementation at the discretion of the user. Furthermore, for test feature points, the resulting vector of predicted distributions represented in a vector-like interface allows both point-wise and vector-wise access to the predicted distribution and its accessible properties.

Model selection and performance metrics: To evaluate the prediction accuracy, the package provides a number of probabilistic loss metrics such as the log-loss, integrated squared loss and the continuous rank probability score (CRPS) for mixed predictions.

Meta-modelling and composition: Hyper-parameter tuning, pipelining and ensembling are standard meta-motifs which are also supported in the form of meta-estimators that combine probabilistic modelling strategies into a higher-level probabilistic modelling strategy (for example, hyper-parameter optimization or model ensembling). Since skpro's probablisitic estimator stays compatible with the API of an sklearn estimator, it is possible to directly use sklearn's meta-estimators for hyper-parameter tuning and pipelining of the probabilistic prediction strategies. Furthermore, skpro comes with experimental support for ensemble methods. Currently, this includes bagging in a regression setting. The meta-estimator fits base regressors (i.e. probabilistic estimators) on random subsets of the original dataset and then aggregates their individual predictions in a distribution interface to form a final prediction.

Automated validation workflow: The framework aims to not only enable meaningful and domain-agnostic model comparison, but also to standardize and simplify the workflow that leads to such comparisons. The package therefore includes a workflow module that allows for a flexible implementation of prediction experiments and helps automating repetitive tasks as much as possible.

Simplified extension and model integration: Skpro treats third party models as first-class citizens. The API should help the integration of existing prediction algorithms into the framework without rewriting them to enable seamless interoperability between model of whatever toolbox.

Off-shelf probabilistic prediction strategies: The package comes with ready-to-use prediction strategies that are built using the described API and documented on the projects website. Currently, the implemented algorithms include a baseline that estimates a density from the training labels, an estimator that simplifies the integration of PyMC3 models [12], as well as composite strategies

that leverage arbitrary scikit-learn models for probabilistic prediction by predicting parameters of common parametric distributions.

Extensiblity and model integration To interface existing prediction strategies from the framework, users can implement their own sub-classes of the probabilistic estimator base class. The API, however, also offers default semi-automated strategy integration. Users can define `on_fit` and `on_predict` events that are invoked automatically to construct the probabilistic estimator interface. If a learning algorithm, such as Bayesian inference via Markov chain Monte Carlo (MCMC) sampling, does not yield an explicit interface to a predictive density, an `Adaptor` can be used to automatically estimate the missing distributional properties. The `DensityAdapter`, for example, transforms a given input into an estimated density function, e.g. a posterior sample into kernel estimated density.

4 Relation to prior work

Major machine learning toolboxes for classical supervised learning such as Weka [6, 7], scikit-learn [2, 11], caret [8, 9] and mlr [1] share the idea of object-oriented encapsulation of models or prediction strategies, and sometimes aspects of the validation process, with a consistent, unified and modular interface. While they usually do not implement interfaces for probabilistic prediction (except classification or variance predictions), they do provide the method encapsulation and workflow modularization that is necessary for model-agnostic strategy development and comparison.

On the other hand, there are a number of abstract model-building environments for probabilistic modelling such as WinBUGS [10], Stan [3], Church [15], Anglican [16], or Edward [13, 14], and more generally a wealth of distinct approaches often subsumed more recently under the term *probabilistic programming* [3, 4]. These have in common that they usually implement a generic interface for Bayesian algorithms of a specific family (usually but not always: Markov Chain Monte Carlo type), thus are neither model nor domain agnostic. Also, by algorithm and interface design, they are usually not geared towards prediction, nor do they implement model validation and model comparison functionality. Nonetheless, they usually include a high-level interface for fitting and sampling from some structured and hierarchical probability distribution models, which contemporary supervised learning toolboxes are lacking.

Thus, connecting these resources with a domain-overarching API bears the great portential to reach a more complete supervised framework for probabilistic prediction making. Given its enormous practical importance one also has to note that from a scientific perspective, a solid workflow and modelling API is a necessary practical precondition for fair quantitative assessment and comparison, be it frequentist, Bayesian or otherwise. We hope that skpro's API can enable a more integrated modelling experience that provides the foundation for the discovery of improved prediction strategies.

5 Full paper and repository

The package, including a full usage tutorial and a reference manual, is available at GitHub.com/alan-turing-institute/skpro and ready for installation via the Python Package Index (PyPI).
A detailed discussion of the skpro API, as well as a benchmarking study using skpro may be found in Section 8 and 9 of the paper [5] which also provides a mathematical formalization of the probabilistic supervised learning setting, algorithmic primitives and meta-learning building blocks, as well as related model validation workflows. The exposition above is heavily based on (and in parts taken verbatim from) the introductory section of the cited source [5].

References

[1] Bernd Bischl et al. "mlr: Machine Learning in R". In: *Journal of Machine Learning Research* 17.170 (2016), pp. 1–5. URL: `http://jmlr.org/papers/v17/15-066.html`.

[2] Lars Buitinck et al. "API design for machine learning software: experiences from the scikit-learn project". In: *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*. 2013, pp. 108–122.

[3] Bob Carpenter et al. "Stan: A probabilistic programming language". In: *Journal of Statistical Software* 20 (2016), pp. 1–37.

[4] Andrew D Gordon et al. “Probabilistic programming”. In: *Proceedings of the on Future of Software Engineering*. ACM. 2014, pp. 167–181.

[5] Frithjof Gressmann et al. “Probabilistic Supervised Learning”. In: (Jan. 2, 2018). arXiv: 1801.00753 [cs, math, stat]. URL: http://arxiv.org/abs/1801.00753.

[6] Mark Hall et al. “The WEKA data mining software: an update”. In: *ACM SIGKDD explorations newsletter* 11.1 (2009), pp. 10–18.

[7] Geoffrey Holmes, Andrew Donkin, and Ian H Witten. “Weka: A machine learning workbench”. In: *Intelligent Information Systems, 1994. Proceedings of the 1994 Second Australian and New Zealand Conference on*. IEEE. 1994, pp. 357–361.

[8] Max Kuhn. Contributions from Jed Wing et al. *caret: Classification and Regression Training*. R package version 6.0-70. 2016. URL: https://CRAN.R-project.org/package=caret.

[9] Max Kuhn. “Caret package”. In: *Journal of Statistical Software* 28.5 (2008).

[10] David J Lunn et al. “WinBUGS-a Bayesian modelling framework: concepts, structure, and extensibility”. In: *Statistics and computing* 10.4 (2000), pp. 325–337.

[11] Fabian Pedregosa et al. “Scikit-learn: Machine learning in Python”. In: *Journal of Machine Learning Research* 12.Oct (2011), pp. 2825–2830.

[12] John Salvatier, Thomas V Wiecki, and Christopher Fonnesbeck. “Probabilistic programming in Python using PyMC3”. In: *PeerJ Computer Science* 2 (2016), e55.

[13] Dustin Tran et al. “Deep probabilistic programming”. In: *International Conference on Learning Representations*. 2017.

[14] Dustin Tran et al. “Edward: A library for probabilistic modeling, inference, and criticism”. In: *arXiv preprint arXiv:1610.09787* (2016).

[15] David Wingate, Andreas Stuhlmueller, and Noah Goodman. “Lightweight implementations of probabilistic programming languages via transformational compilation”. In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. 2011, pp. 770–778.

[16] Frank Wood, Jan Willem Meent, and Vikash Mansinghka. “A new approach to probabilistic programming inference”. In: *Artificial Intelligence and Statistics*. 2014, pp. 1024–1032.