`qunfold`: Composable Quantification and Unfolding Methods in Python

Mirko Bunse[0000−0002−5515−6278]

Artificial Intelligence Unit, TU Dortmund University, 44227 Dortmund, Germany
mirko.bunse@cs.tu-dortmund.de

Abstract. We present `qunfold`, a Python package that implements several quantification and unfolding methods. A unique capability of `qunfold` is the composition of novel methods from existing and easily customized loss functions and data representations. Moreover, this package leverages a powerful optimization back-end to yield state-of-the-art performances for all compositions. This paper introduces the common usage patterns for `qunfold`, revisits the technical background of the package, and empirically demonstrates the resulting performance.

Keywords: Quantification · Unfolding · Unconstrained optimization · Multi-class classification · Software

1 Introduction

Many quantification methods, i.e., many methods for the supervised estimation of class prevalences [12], can be described as a combination of a loss function and a data representation [5,11]. This observation motivates implementations that make this combination explicit, to provide a high amount of code quality (separation of concerns, reusability) and to establish an opportunity of composing new methods from existing components.

The value of such compositions comes from the specific characteristics that each component introduces; unfolding methods, for instance, address an order among the classes through regularization [6]. Hence, the name `qunfold` mingles the terms "quantification" and "unfolding" to hint at the package's capability of composing new methods from existing loss functions and data representations. The diversity of quantification use cases, including political sciences, market research, epidemiology, and others [8] calls for this capability.

Implementations of quantification methods have to define another aspect in addition to the loss function and data representation: the numerical optimization algorithm through which the loss is minimized. This additional aspect is vital in the multi-class setting, where an exhaustive search of all class prevalences is not feasible. A recent proposal is to employ a soft-max operator to ensure legitimate and accurate results with unconstrained optimization algorithms [4], including those well-tested algorithms that a standard `numpy`/`scipy` stack provides.

Our Python package `qunfold`[1] leverages these recent developments to provide a highly performant and composable implementation of many existing quantification and unfolding methods. The package, which is released under an open-source license, is designed for meeting the following goals:

- focus on methods (disregarding data loading, evaluation protocols, etc.)
- easy composition of new methods
- high prediction performance due to a powerful optimization routine
- easy extendability through API design and through automatic differentiation
- compliance with the conventions established by `scikit-learn`
- detailed documentation and high test coverage

These goals partially differ from the goals of QuaPy [16], the current state-of-the-art implementation for all aspects of quantification, including the acquisition of data and the evaluation of methods. QuaPy provides a large collection of quantification methods, but does not allow to recompose them. We provide a thin wrapper for `qunfold` methods, which allows users to combine the functionalities of QuaPy and our package.

We introduce the usage of `qunfold` in Sect. 2 and revisit its conceptual background in Sect. 3. Sect. 4 demonstrates the performance of our package before Sect. 5 concludes with prospective extensions.

2 Usage

The package is easily installed via `pip` and its quantification methods are used like classifiers from `scikit-learn`. These design choices result in a seamless access for newcomers of quantification, as illustrated in Listing 1.

```
from qunfold import ACC # Adjusted Classify and Count
from sklearn.ensemble import RandomForestClassifier

acc = ACC( # use OOB predictions for training the quantifier
    RandomForestClassifier(oob_score=True)
)
acc.fit(X_trn, y_trn) # fit to training data
p_hat = acc.predict(X_tst) # estimate a prevalence vector p̂ ∈ ℝ^C
```

Listing 1: A minimal example where the quantification method Adjusted Classify and Count (ACC) [12] predicts the class prevalences of a testing sample.

Beyond the existing methods of quantification and unfolding, users have the opportunity to compose new methods from existing loss functions and data representations. This opportunity also includes the combination of multiple loss

[1] https://github.com/mirkobunse/qunfold

```
# the ACC loss, regularized with strength 0.01 for ordinality
loss = TikhonovRegularized(LeastSquaresLoss(), 0.01)

# the original data representation of ACC with 10-fold cross-validation
transformer = ClassTransformer(CVClassifier(LogisticRegression(), 10))

# the ordinal variant of ACC, wrapped for being used in QuaPy
ordinal_acc = QuaPyWrapper(GenericMethod(loss, transformer))
```

Listing 2: The ordinal variant [6] of ACC is composed of the original ACC loss, a regularization term, and the original data representation of ACC. Finally, this variant is wrapped for being used in QuaPy.

terms, like regularizers, through a `CombinedLoss` type. Listing 2 conveys, as an example of composition, the creation of an ordinal variant of ACC.

The creation of novel data representations only requires implementing the `fit_transform` and `transform` methods of the `AbstractTransformer` type. Novel loss functions are easily implemented through `jax` [13], a package which automatically differentiates the loss while complying to the well-known `numpy` API. All of the above aspects are thoroughly documented, illustrated through examples, and tested in a continuous integration pipeline.

3 Background

We intend to predict $\mathbf{p} \in \mathbb{R}^C$, the class prevalences of an unlabeled data sample $\mathrm{D} \in \mathcal{X}^m$. For this purpose, we have a labeled training set $\bigcup_{i=1}^{C} \mathrm{D}_i$ where D_i contains the data items of the i-th class. The composition of quantification methods is enabled through the observation [5,11] that many methods estimate $\mathbf{p}$ by solving a system of linear equations

$$\begin{aligned} \mathbf{q} &= \mathbf{M}\mathbf{p} \\ \text{where} \quad [\mathbf{q}]_i &= \frac{1}{|\mathrm{D}|} \sum_{\mathbf{x} \in \mathrm{D}} [f(\mathbf{x})]_i \end{aligned} \tag{1}$$

is a mean embedding of D, which employs a feature transformation $f : \mathcal{X} \to \mathbb{R}^F$. Here, the matrix $\mathbf{M} \in \mathbb{R}^{F \times C}$ with entries

$$[\mathbf{M}]_{ji} = \frac{1}{|\mathrm{D}_j|} \sum_{\mathbf{x} \in \mathrm{D}_j} [f(\mathbf{x})]_i$$

represents the mean embedding of each class in the training set.

Finding a solution $\hat{\mathbf{p}}$ for Eq. 1 requires the minimization of a loss function $\mathcal{L} : \mathbb{R}^C \to \mathbb{R}$, which reflects the goodness of $\hat{\mathbf{p}}$. Hence, quantification methods of the above kind are defined through a loss function and a feature transformation.

3.1 Unconstrained Soft-Max Optimization

Given a loss function and a feature transformation, a recent proposal [4] for solving Eq. 1 is

$$\hat{\mathbf{p}} \;=\; \mathrm{softmax}\big(\mathbf{l}^*\big) \qquad (2)$$
$$\text{where} \quad \mathbf{l}^* = \operatorname*{arg\,min}_{\mathbf{l}\in\mathbb{R}^C} \mathcal{L}\big(\,\mathrm{softmax}(\mathbf{l}); \mathbf{q}, \mathbf{M}\,\big)$$

is a vector of latent quantities. Here, the output of the soft-max operator is $[\mathrm{softmax}(\mathbf{l})]_i = \exp([\mathbf{l}]_i)/(\sum_{j=1}^{C}\exp([\mathbf{l}]_j))$, which ensures that any $\hat{\mathbf{p}}$ is a legitimate estimate of class prevalences. We regard an estimate as being legitimate if it represents a probability density function, i.e., if $[\hat{\mathbf{p}}]_i \geq 0\ \forall\, i$ and $\sum_i [\hat{\mathbf{p}}]_i = 1$. The latent quantities can be interpreted as scaled log-probabilities of the classes.

In `qunfold`, we establish the uniqueness of $\mathbf{l}^*$ by fixing the first dimension to the constant value $[\mathbf{l}]_1 = 0$. Thereby, we minimize $\mathcal{L}$ only over $(n-1)$ actual variables in $\mathbf{l}$. This reduction of dimensionality comes without sacrificing the optimality of $\mathbf{l}^*$; it only defines the scaling of the latent quantities.

3.2 Out-of-Bag Training of Quantifiers

The estimation of $\mathbf{M}$ (see Eq. 1) requires a labeled training set. In case of a supervised feature transformation f, like the `ClassTransformer` of ACC, this estimation should not use the same training data as f; otherwise, biases of f will hardly be corrected by the quantification method. One possibility of diversifying the training data of f and $\mathbf{M}$ is through cross-validation [12]. Here, f is trained with the training folds and $\mathbf{M}$ is trained with the test predictions. We implement this training strategy in the `CVClassifier` class (revisit Listing 2).

In addition, we implement a similar technique which builds on bagging estimators [3]. Here, f is trained with the training folds and $\mathbf{M}$ is trained with the out-of-bag predictions of the estimator. The advantage of bagging over cross-validation is that bagging ensembles, like random forests, can be trained at no extra cost. For this strategy, the bagging classifier can be used directly, without the need for a meta-classifier (revisit Listing 1).

3.3 Existing Loss Functions and Feature Representations

Our package implements several existing methods in terms of their loss functions and feature representations, which are listed in Tab. 1. The modular design of our package enables compositions of novel methods from the existing components.

In case of HDx and HDy [14], we have replaced the original loss function with a surrogate loss that is better suited for numerical optimization. The original loss, which is the average of feature-wise (or class-wise) Hellinger distances, is problematic because it lacks twice differentiability and, hence, complicates numerical optimizations. As a twice differentiable surrogate, we therefore employ the average of squared Hellinger distances. This `HellingerSurrogateLoss` behaves similar to the original loss, while facilitating numerical optimizations.

Table 1. Existing methods in terms of their loss functions and feature transformations.

method	loss function	feature transformation
ACC [12]	`LeastSquaresLoss()`	`ClassTransformer(*, is_probabilistic=False)`
PACC [1]	`LeastSquaresLoss()`	`ClassTransformer(*, is_probabilistic=True)`
HDx [14]	`HellingerSurrogateLoss()`	`HistogramTransformer(*)`
HDy [14]	`HellingerSurrogateLoss()`	`HistogramTransformer(*, preprocessor=ClassTransformer(*))`
EDx [15]	`EnergyLoss(*)`	`DistanceTransformer(*)`
EDy [7]	`EnergyLoss(*)`	`DistanceTransformer(*, preprocessor=ClassTransformer(*))`
RUN [2]	`TikhonovRegularized(BlobelLoss(), *)`	any `AbstractTransformer`
CC [12]	`None`	`ClassTransformer(*, is_probabilistic=False)`
PCC [1]	`None`	`ClassTransformer(*, is_probabilistic=True)`

4 Performance

We evaluate the performance of our package on the public data set [9] of the LeQua2022 competition [10]. This dataset, which contains 28 classes, constitutes a gold-standard benchmark for multi-class text quantification. We employ the vectorial representation of the data and a logistic regression classifier with $C \in \{10^{-3}, 10^{-2}, 10^{-1}, 10^{0}, 10^{1}\}$, where the best C is chosen separately for each quantification method on hold-out validation samples. A classifier of this kind obtained winning performances during the competition [17]. For HDy, we optimize the number of bins per class on hold-out data over $B \in \{2, 4, 6\}$. We compare the results of `qunfold` and QuaPy in terms of the mean absolute error (AE) and in terms of the mean relative absolute error (RAE). We omit HDx, EDx, CC, and PCC because we deem these methods unsuitable for text quantification. We also omit several methods that are available in QuaPy but not (yet) in `qunfold`.

Table 2. Performance comparison between `qunfold` and QuaPy. The performances are measured in terms of two error metrics, AE and RAE. The performance of the best implementation, for each method and metric, is printed in **boldface**. An asterisk (∗) indicates that a method is missing from QuaPy v.0.1.7.

method	AE (↓)		RAE (↓)	
	QuaPy	`qunfold`	QuaPy	`qunfold`
ACC	0.0190±0.0045	**0.0164±0.0046**	1.5380±1.4460	**1.2553±1.1763**
PACC	0.0197±0.0050	**0.0119±0.0034**	1.7070±2.0091	**0.9594±0.8342**
HDy	0.0163±0.0042	**0.0143±0.0041**	1.3634±1.2062	**1.1319±1.0803**
EDy	∗	**0.0125±0.0036**	∗	**1.1856±1.1080**
RUN	∗	**0.0165±0.0046**	∗	**1.2305±1.1478**

The results of our evaluation are displayed in Tab. 2. They convey that the methods from `qunfold` beat the corresponding implementations from QuaPy, which is the state-of-the-art package for quantification. We attribute this outcome to the powerful soft-max optimization technique that our package leverages. The current version of Quapy[2], in contrast, employs the pseudo-inversion method for ACC and PACC and constrained optimization for HDy, all of which have been shown to yield inferior performances [4]. We note, however, that our soft-max optimization is computationally more expensive than the pseudo-inverse method.

5 Conclusion

We have presented `qunfold`, a highly performant and composable implementation of several quantification and unfolding methods. This implementation leverages two recent findings: first, that many quantification methods consist of a loss function and a data representation, which can be reassembled in arbitrary ways; second, that these methods can be optimized through a soft-max operator. Further improvements of our implementation are a surrogate loss for HDx and HDy and an optional out-of-bag training of quantifiers. These features lead to performances that beat QuaPy, the current state-of-the-art implementation for quantification methods. We recommend `qunfold` to anyone who is looking for composability or for strong baseline methods.

In the future, we are planning to extend our package with additional loss functions and data representations. We also conceive novel features, like ensembling and automatic compositions of methods, as valuable extensions.

Acknowledgements This work was partly funded by the Federal Ministry of Education and Research of Germany and the state of North Rhine-Westphalia as part of the Lamarr Institute for Machine Learning and Artificial Intelligence.

We also thank the reviewers of our LQ 2022 publication [4] for pointing out that the solution of Eq. 2 is unique if $[\mathbf{l}]_1 = 0$ is fixed. This observation has substantially improved our implementation.

References

1. Bella, A., Ferri, C., Hernández-Orallo, J., Ramírez-Quintana, M.J.: Quantification via probability estimators. In: Int. Conf. on Data Mining. pp. 737–742. IEEE (2010). https://doi.org/10.1109/ICDM.2010.75
2. Blobel, V.: Unfolding methods in high-energy physics experiments. Tech. rep., CERN (1985). https://doi.org/10.5170/CERN-1985-009.88, https://cds.cern.ch/record/157405/files/p88.pdf
3. Breiman, L.: Bagging predictors. Mach. Learn. **24**(2), 123–140 (1996). https://doi.org/10.1007/BF00058655

[2] https://github.com/HLT-ISTI/QuaPy/releases/tag/0.1.7

4. Bunse, M.: On multi-class extensions of adjusted classify and count. In: Int. Worksh. on Learn. to Quantify: Meth. and Appl. pp. 43–50 (2022), https://lq-2022.github.io/proceedings/CompleteVolume.pdf
5. Bunse, M.: Unification of algorithms for quantification and unfolding. In: Worksh. on Mach. Learn. for Astropart. Phys. and Astron. pp. 459–468. Gesellschaft für Informatik e.V. (2022). https://doi.org/10.18420/INF2022_37
6. Bunse, M., Moreo, A., Sebastiani, F., Senz, M.: Ordinal quantification through regularization. In: Europ. Conf. on Mach. Learn. and Knowl. Discov. in Databases. pp. 36–52. Springer (2023). https://doi.org/10.1007/978-3-031-26419-1_3
7. Castaño, A., González, P., González, J.A., del Coz, J.J.: Matching distributions algorithms based on the earth mover's distance for ordinal quantification. IEEE Trans. on Neur. Netw. and Learn. Syst. pp. 1–12 (2022). https://doi.org/10.1109/tnnls.2022.3179355
8. Esuli, A., Fabris, A., Moreo, A., Sebastiani, F.: Learning to Quantify, The Inform. Retr. Series, vol. 47. Springer (2023). https://doi.org/10.1007/978-3-031-20467-8
9. Esuli, A., Moreo, A., Sebastiani, F.: Learning to quantify: LeQua 2022 datasets (2021). https://doi.org/10.5281/zenodo.6546188
10. Esuli, A., Moreo, A., Sebastiani, F.: LeQua@CLEF2022: Learning to quantify. In: Adv. in Inform. Retr. pp. 374–381. Springer (2022). https://doi.org/10.1007/978-3-030-99739-7_47
11. Firat, A.: Unified framework for quantification (2016), http://arxiv.org/abs/1606.00868
12. Forman, G.: Quantifying counts and costs via classification. Data Mining and Knowl. Discov. **17**(2), 164–206 (2008). https://doi.org/10.1007/s10618-008-0097-y
13. Frostig, R., Johnson, M.J., Leary, C.: Compiling machine learning programs via high-level tracing. Syst. for Mach. Learn. **4**(9) (2018), http://github.com/google/jax
14. González-Castro, V., Alaíz-Rodríguez, R., Alegre, E.: Class distribution estimation based on the Hellinger distance. Inform. Sci. **218**, 146–164 (2013). https://doi.org/10.1016/j.ins.2012.05.028
15. Kawakubo, H., du Plessis, M.C., Sugiyama, M.: Computationally efficient class-prior estimation under class balance change using energy distance. IEICE Trans. Inform. Syst. **99-D**(1), 176–186 (2016). https://doi.org/10.1587/transinf.2015EDP7212
16. Moreo, A., Esuli, A., Sebastiani, F.: QuaPy: A Python-based framework for quantification. In: Int.Conf. on Inform. and Knowl. Management. pp. 4534–4543. ACM, New York, NY, USA (2021). https://doi.org/10.1145/3459637.3482015
17. Senz, M., Bunse, M.: DortmundAI at LeQua 2022: Regularized SLD. In: Conf. and Labs of the Eval. Forum. vol. 3180, pp. 1911–1915. CEUR (2022), http://ceur-ws.org/Vol-3180/paper-152.pdf