Journal of Machine Learning Research 22 (2021) 1-6 Submitted 1/21; Revised 6/21; Published 8/21

sklvq: Scikit Learning Vector Quantization

Rick van Veen[a] RICK.VAN.VEEN@RUG.NL
Michael Biehl[a,b] M.BIEHL@RUG.NL
[a]*Bernoulli Institute for Mathematics, Computer Science and Artificial Intelligence, University of Groningen, The Netherlands,*
[b]*SMQB, Institute of Metabolism and Systems Research, College of Medical and Dental Sciences, Birmingham, UK*

Gert-Jan de Vries GJ.DE.VRIES@PHILIPS.COM
Philips Research, Healthcare, Eindhoven, The Netherlands

Editor: Andreas Mueller

Abstract

The sklvq package is an open-source Python implementation of a set of learning vector quantization (LVQ) algorithms. In addition to providing the core functionality for the GLVQ, GMLVQ, and LGMLVQ algorithms, sklvq is distinctive by putting emphasis on its modular and customizable design. Not only resulting in a feature-rich implementation for users but enabling easy extensions of the algorithms for researchers. The theory behind this design is described in this paper. To facilitate adoptions and inspire future contributions, sklvq is publicly available on Github (under the BSD license) and can be installed through the Python package index (PyPI). Next to being well-covered by automated testing to ensure code quality, it is accompanied by detailed online documentation. The documentation covers usage examples and provides an in-depth API including theory and scientific references.

Keywords: Python, scikit-learn, learning vector quantization, matrix relevance learning

1. Introduction

Learning vector quantization (LVQ) has, since its introduction by Kohonen (1990), become an important family of supervised learning algorithms. In the training phase, the algorithms determine prototypes that represent the classes in the presented data. Predictions about novel samples are made based on the receptive fields of the prototypes. In other words, a novel sample is classified by computing the distance from the sample to all prototypes and assigning it to the label of the closest prototype. The computation of the prototypes and the definition of their receptive fields can be achieved in different ways. A comprehensive review of the most relevant LVQ algorithms is given by Nova and Estévez (2014).

Here we present "sklvq"[1], an open-source, Python based, and "scikit-learn" (Pedregosa et al., 2011) compatible[2] LVQ framework, including the following three variants: Generalized LVQ (GLVQ) by Sato and Yamada (1995), generalized matrix LVQ (GMLVQ), and localized GMLVQ (LGMLVQ) by Schneider et al. (2009); Bunte et al. (2012). Although

1. `https://github.com/rickvanveen/sklvq/releases/0.1.2`
2. `https://scikit-learn.org/stable/developers/develop.html`

other variants of LVQ exist (Nova and Estévez, 2014) the focus on these specific algorithms is motivated by our own research interests and their successful practical application, see e.g. van Veen et al. (2018, 2020); de Vries et al. (2015); Biehl (2017).

A key property of the LVQ family is that the prototypes are interpretable in feature space. However, even if the feature space is complex and unfeasible to understand, the prototypes can still be transformed into the original space under certain conditions (van Veen et al., 2020). In addition to the prototypes, the GMLVQ and LGMLVQ variants employ and construct a "relevance matrix" that can be used to generate low dimensional discriminant visualizations (Bunte et al., 2012; Biehl et al., 2012). These discriminant plots can help find potential sources of variation by visualizing the data within the decision space of the models (van Veen et al., 2020). These properties in combination with proved success, for instance, discussed by Nova and Estévez (2014); Biehl (2017) in numerous applications in the biomedical field, medicine, and industry[3], makes LVQ a valuable and popular tool.

2. Implementation

Here we discuss the theoretical concepts and link them with the reasoning behind the implementation in the code repository[2]. In the following, a dataset is referred to as $\mathcal{D} = \big\{(\vec{x}_i, y_i) \mid \vec{x}_i \in \mathbb{R}^N,\ y_i \in \{1, \ldots, C\}\big\}_{i=1}^{P}$, where P represents the number of samples $\vec{x}_i$, with labels y_i that represent C mutually exclusive classes. At the center of any LVQ model are the prototypes, which define the model. The definition of a set of Q prototypes is given by $\mathcal{W} = \big\{(\vec{w}_j, z_j) \ \in \ \mathbb{R}^N \times \{1, \ldots, C\}\big\}_{j=1}^{Q}$, with $Q \geq C$ and at least one prototype $\vec{w}_j$ with label z_j representing each class. Within the code, LVQ algorithms are referred to as *models*, each with an *objective function* quantifying how well it has been adapted to the data in terms of prototypes ($\mathcal{W}$) and single or multiple relevance matrices in GMLVQ and LGMLVQ, respectively. In order to explain the architecture and idea behind the code, we focus on GLVQ here and refer the reader to the work of Schneider et al. (2009); Bunte et al. (2012) for details about GMLVQ and LGMLVQ. Hence, the model parameter, denoted by $\vec{\theta}$ only contains prototypes. All algorithms in the sklvq package share the same objective function, i.e., the GLVQ objective function as introduced by Sato and Yamada (1995):

$$E(\vec{\theta}, \mathcal{D}) = \textstyle\sum_{i=1}^{P} e(\vec{\theta}, \vec{x}_i), \text{ with } e(\vec{\theta}, \vec{x}_i) = f\big[\mu(d_L(\vec{x}_i), d_K(\vec{x}_i))\big]. \tag{1}$$

The sklvq package extracts the components found in (Equation 1) and locates them in their own sub-packages, with the *activation function* $f(\cdot)$ and the *discriminant function* $\mu(\cdot)$. The discriminant function takes the two *distances* $d_L(\cdot)$ and $d_K(\cdot)$ as arguments. If we define $\vec{w}_L(\vec{x}_i)$ to be the prototype closest to $\vec{x}_i$ with the same label then the distance between this prototype and the sample would be given by $d_L(\vec{x}_i) = d(\vec{x}_i, \vec{w}_L(\vec{x}_i))$. Similarly, $d_K(\vec{x}_i) = d(\vec{x}_i, \vec{w}_K(\vec{x}_i))$ denotes the distance between $\vec{x}_i$ and the prototype closest to it $\vec{w}_K(\vec{x}_i)$ that has a different label. The objective function in Equation (1) quantifies how close prototypes are to data samples with the same label compared to how far away they are from samples with a different label. Lower values imply a better model fit than higher values. Hence, the goal is to change prototypes ($\mathcal{W}$) in such a way that the objective value

3. `http://www.cis.hut.fi/research/som-bibl/`

is minimized. The minimization is based on gradients as obtained by the chain rule:

$$\partial e(\vec{\theta}, \vec{x}_i) \Big/ \partial \vec{w}_L(\vec{x}_i) = \partial f / \partial \mu \cdot \partial \mu / \partial d_L(\vec{x}_i) \cdot \partial d_L(\vec{x}_i) / \partial \vec{w}_L(\vec{x}_i). \tag{2}$$

The structure of the objective function shows that a "modular" code design is possible: Substituting the activation, discriminant, or distance functions with another variant does not alter the form of the objective function (Equation 1) or its derivatives (Equation 2). The realization of the objective function and the resulting models follow this structure where each component can be provided as a hyper-parameter[4]. This ability to implement each component separately, with the support of "base" classes to ensure the right interface[4], is the key design difference distinguishing sklvq from other LVQ packages.

How the model parameters are exactly updated depends on the solver, which comes into play whenever the model's fit method is called and the objective function and its components are initialized. The general structure of the implementation can be understood by looking at a single example, i.e., stochastic steepest gradient descent. The stochastic gradient descent updates the model parameter $\vec{\theta}$ in the following way

$$\vec{\theta} = \vec{\theta} - \eta(t) \cdot \nabla e(\vec{\theta}, \vec{x}_i), \tag{3}$$

where $\nabla e_i(\vec{\theta}, \vec{x}_i)$ denotes the gradient of the objective function with respect to the model parameters given a single random sample $\vec{x}_i$ from the dataset $\mathcal{D}$. The step size $\eta(t)$ controls the size of the update step taken and decreases after each epoch t, see the work by (LeKander et al., 2017) for details. The process is repeated for all other samples in the dataset after which the epoch (t) is incremented. One can see that this structure extends to other solvers without having to adapt the objective function or its components. Thus, in sklvq solvers and objectives are separated from each other, making it easier to add multiple solvers (see Table 1b). For instance, batch gradient descent replaces the single sample update in (Equation 3) with $\nabla E(\vec{\theta}) = \sum_i^P \nabla e_i(\vec{\theta}, \vec{x}_i)$, i.e., the full gradient based on all the samples.

3. Comparison

In the previous section, we have shown the theory behind the design and implementation of sklvq and the resulting advantages. This section, particularly Table 1, provides a comparison of the resulting functionality with that of other LVQ toolboxes.

Table 1a shows that a user has multiple choices depending on the required programming language and LVQ variant. The key differentiator between sklvq and the single other choice when one needs a scikit-learn compatible Python implementation (Jensen and Paassen, 2018) is its modular design, as described in Section 2. Together with the inclusion of the work of LeKander et al. (2017) on different solvers for LVQ and Villmann et al. (2020) for the comparison of activation functions, this results in a more feature-rich (Table 1b) and easier to customize implementation[4] of, currently, GLVQ, GMLVQ, and LGMLVQ.

4. Conclusion and Future Work

We have provided an overview of the resulting benefit (Table 1b) and reasoning behind sklvq's modular and customizable design (Section 2). Together with the open and online

4. `https://sklvq.readthedocs.io/en/0.1.2/`

Algorithms	(Jensen and Paassen, 2018)	(Bunte, 2012)	(Biehl and Westermann, 2019)	(Leberecht et al., 2018)	sklvq
GLVQ	✓	✓	✓	✗	✓
GRLVQ	✓	✓	✓	✗	✗
GMLVQ	✓	✓	✓	✓	✓
LGMLVQ	✓	✓	✗	✗	✓
RSLVQ	✓	✗	✗	✗	✗
MRSLVQ	✓	✗	✗	✗	✗
LMRSLVQ	✓	✗	✗	✗	✗
Language	Python	Matlab	Matlab	Java	Python

(a) Overview of implemented algorithms in sklvq and other toolboxes.

Distance Functions					
Euclidean[a]	✗	✗	✓	✗	✓
Squared Euclidean[a]	✓	✓	✗	✗	✓
Adaptive Squared Euclidean[b]	✓	✓	✓	✓	✓
Local Squared Euclidean[c]	✓	✓	✓	✓	✓
Objective Functions					
Generalized Learning Objective	✓	✓	✓	✓	✓
Discriminant Functions					
Relative Distance	✓	✓	✓	✓	✓
Activation Functions					
Identity	✗	✗	✓	✗	✓
Sigmoid	✓	✓	✗	✓	✓
Soft+	✗	✗	✗	✗	✓
Swish	✗	✗	✗	✗	✓
Solvers					
Steepest Gradient Descent	✗	✓	✗	✓	✓
Waypoint Gradient Descent	✗	✗	✓	✗	✓
Adaptive Moment Estimation	✗	✗	✗	✗	✓
LBFGS	✓	✓	✗	✗	✓
BFGS	✗	✗	✗	✗	✓

(b) Overview of GLVQ, GMLVQ, LGMLVQ specific functionality compared to other LVQ toolboxes. Note that small differences in the implementations may exist. Distance functions compatibility: **a**: GLVQ. **b**: GMLVQ. **c**: LGMLVQ. All other functions are compatible with all three algorithms.

Table 1: Algorithm and component level functionality overview. The ✓ and ✗ indicate if an algorithm or component has or has not been implemented, respectively.

setup, we expect to facilitate high-quality contributions. Future work will focus on LVQ variants not yet available in sklvq (Table 1a). In particular, variants that provide probability estimates such as RSLVQ (Seo and Obermayer, 2003; Schneider, 2010) require different objective and discriminant functions. In addition, the inclusion of reject options (Fischer et al., 2014; Brinkrolf and Hammer, 2017) will be considered.

Acknowledgments

This work was supported by the Michael J. Fox Foundation through grant ID 17081.

References

M. Biehl. Biomedical applications of prototype based classifiers and relevance learning. In *International Conference on Algorithms for Computational Biology*, pages 3–23. Springer, 2017.

M. Biehl and F. Westermann. A collection of no-nonsense GMLVQ demo code, March 2019. URL `https://www.cs.rug.nl/~biehl/gmlvq`. Version 3.0.

M. Biehl, K. Bunte, F. Schleif, P. Schneider, and T. Villmann. Large margin linear discriminative visualization by Matrix Relevance Learning. In *International Joint Conference on Neural Networks*, pages 1–8, 2012. doi: 10.1109/IJCNN.2012.6252627.

J. Brinkrolf and B. Hammer. Probabilistic extension and reject options for pairwise LVQ. In *Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering and Data Visualization*, pages 1–8, 2017. doi: 10.1109/WSOM.2017.8020028.

K. Bunte. Matrix Relevance LVQ, 2012. URL `http://matlabserver.cs.rug.nl/gmlvqweb/web/`. Direct download: `https://www.cs.rug.nl/~biehl/LVQ%5Ftoolbox.tar.gz`.

K. Bunte, P. Schneider, B. Hammer, F.M. Schleif, T. Villmann, and M. Biehl. Limited rank matrix learning, discriminative dimension reduction and visualization. *Neural Networks*, 26:159–173, February 2012. ISSN 08936080. doi: 10.1016/j.neunet.2011.10.001.

J.J.G. de Vries, S.C. Pauws, and M. Biehl. Insightful stress detection from physiology modalities using Learning Vector Quantization. *Neurocomputing*, 151:873–882, 2015. ISSN 0925-2312. doi: 10.1016/j.neucom.2014.10.008.

L. Fischer, D. Nebel, T. Villmann, B. Hammer, and H. Wersing. Rejection strategies for learning vector quantization – A comparison of probabilistic and deterministic approaches. In T. Villmann, F.M. Schleif, M. Kaden, and M. Lange, editors, *Advances in Self-Organizing Maps and Learning Vector Quantization*, pages 109–118. Springer International Publishing, 2014. ISBN 978-3-319-07695-9.

J. Jensen and B. Paassen. Sklearn-lvq, 2018. URL `https://github.com/MrNuggelz/sklearn-lvq`. Version v1.1.0.

T. Kohonen. The Self-Organizing Map. *Proceedings of the IEEE*, 78(9):1464–1480, 1990. doi: 10.1109/5.58325.

C. Leberecht, S. Bittrich, and F. Kaiser. GMLVQ WEKA plug-in, August 2018. URL `https://doi.org/10.5281/zenodo.1326272`. Version v0.1.0.

M. LeKander, M. Biehl, and H. de Vries. Empirical evaluation of gradient methods for matrix learning vector quantization. In *Workshop on Self-Organizing Maps and Learning Vector Quantization, Clustering, and Data Visualization*, pages 1–8, 2017. doi: 10.1109/wsom.2017.8020027.

D. Nova and P.A. Estévez. A review of learning vector quantization classifiers. *Neural Computing and Applications*, 25(3-4):511–524, 2014. doi: 10.1007/s00521-013-1535-3.

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.

A. Sato and K. Yamada. Generalized learning vector quantization. In *Conference on Neural Information Processing Systems*, NIPS'95, page 423–429, Cambridge, MA, USA, 1995. MIT Press.

P. Schneider. *Advanced methods for prototype-based classification*. PhD thesis, University of Groningen, 2010. URL `http://hdl.handle.net/11370/71c6861f-edc6-4030-908e-19e87e3fc2ad`.

P. Schneider, M. Biehl, and B. Hammer. Adaptive relevance matrices in learning vector quantization. *Neural Computation*, 21(12):3532–3561, December 2009. ISSN 0899-7667. doi: 10.1162/neco.2009.11-08-908.

S. Seo and K. Obermayer. Soft learning vector quantization. *Neural Computation*, 15(7): 1589–1604, 2003. doi: 10.1162/089976603321891819.

R. van Veen, L. Talavera Martinez, R.V. Kogan, S.K. Meles, D. Mudali, J.B.T.M. Roerdink, F. Massa, M. Grazzini, J.A. Obeso, M.C. Rodriguez-Oroz, K.L. Leenders, R.J. Renken, J.J.G. de Vries, and M. Biehl. *Machine learning based analysis of FDG-PET image data for the diagnosis of neurodegenerative diseases*, volume 310 of *Frontiers in Artificial Intelligence and Applications*, pages 280–289. Amsterdam, Netherlands: IOS Press, 2018. ISBN 978-1-61499-928-7. doi: 10.3233/978-1-61499-929-4-280.

R. van Veen, V. Gurvits, R.V. Kogan, S.K. Meles, J.J.G. de Vries, R.J. Renken, M.C. Rodriguez-Oroz, R. Rodriguez-Rojas, D. Arnaldi, S. Raffa, B.M. de Jong, K.L. Leenders, and M. Biehl. An application of generalized matrix learning vector quantization in neuroimaging. *Computer Methods and Programs in Biomedicine*, 197:105708, 2020. ISSN 0169-2607. doi: 10.1016/j.cmpb.2020.105708.

T. Villmann, J. Ravichandran, A. Villmann, D. Nebel, and M. Kaden. Investigation of activation functions for generalized learning vector quantization. In A. Vellido, K. Gibert, C. Angulo, and J.D. Martín Guerrero, editors, *Advances in Self-Organizing Maps, Learning Vector Quantization, Clustering and Data Visualization*, pages 179–188. Springer International Publishing, 2020. ISBN 978-3-030-19642-4. doi: 10.1007/978-3-030-19642-4_18.