Journal of Machine Learning Research 21 (2020) 1-6 Submitted 8/19; Revised 7/20; Published 7/20

metric-learn: Metric Learning Algorithms in Python

William de Vazelhes* WDEVAZELHES@GMAIL.COM
Paris Research Center, Huawei Technologies
92100 Boulogne-Billancourt, France

CJ Carey PERIMOSOCORDIAE@GMAIL.COM
Google LLC
111 8th Ave, New York, NY 10011, USA

Yuan Tang TERRYTANGYUAN@GMAIL.COM
Ant Group
525 Almanor Ave, Sunnyvale, CA 94085, USA

Nathalie Vauquier NATHALIE.VAUQUIER@INRIA.FR
Aurélien Bellet AURELIEN.BELLET@INRIA.FR
Magnet Team, INRIA Lille – Nord Europe
59650 Villeneuve d'Ascq, France

Editor: Balazs Kegl

arXiv:1908.04710v3 [cs.LG] 27 Jul 2020

Abstract

`metric-learn` is an open source Python package implementing supervised and weakly-supervised distance metric learning algorithms. As part of `scikit-learn-contrib`, it provides a unified interface compatible with `scikit-learn` which allows to easily perform cross-validation, model selection, and pipelining with other machine learning estimators. `metric-learn` is thoroughly tested and available on PyPi under the MIT license.

Keywords: machine learning, python, metric learning, scikit-learn

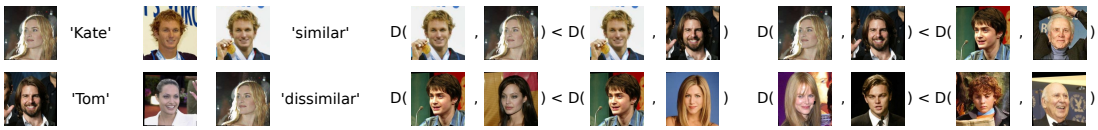
1. Introduction

Many approaches in machine learning require a measure of distance between data points. Traditionally, practitioners would choose a standard distance metric (Euclidean, City-Block, Cosine, etc.) using a priori knowledge of the domain. However, it is often difficult to design metrics that are well-suited to the particular data and task of interest. Distance metric learning, or simply metric learning (Bellet et al., 2015), aims at automatically constructing task-specific distance metrics from data. A key advantage of metric learning is that it can be applied beyond the standard supervised learning setting (data points associated with labels), in situations where only weaker forms of supervision are available (e.g., pairs of points that should be similar/dissimilar). The learned distance metric can be used to perform retrieval tasks such as finding elements (images, documents) of a database that are semantically closest to a query element. It can also be plugged into other machine learning algorithms, for instance to improve the accuracy of nearest neighbors models (for classification, regression, anomaly detection...) or to bias the clusters found by clustering

∗. Most of the work was carried out while the author was affiliated with INRIA, France.

©2020 William de Vazelhes and CJ Carey and Yuan Tang and Nathalie Vauquier and Aurélien Bellet.

License: CC-BY 4.0, see `https://creativecommons.org/licenses/by/4.0/`. Attribution requirements are provided at `http://jmlr.org/papers/v21/19-678.html`.



(a) classes (b) pairs (c) triplets (d) quadruplets

Figure 1: Different types of supervision for metric learning illustrated on face image data taken from the Labeled Faces in the Wild data set (Huang et al., 2012).

algorithms towards the intended semantics. Finally, metric learning can be used to perform dimensionality reduction. These use-cases highlight the importance of integrating metric learning with the rest of the machine learning pipeline and tools.

`metric-learn` is an open source package for metric learning in Python, which implements many popular metric-learning algorithms with different levels of supervision through a unified interface. Its API is compatible with `scikit-learn` (Pedregosa et al., 2011), a prominent machine learning library in Python. This allows for streamlined model selection, evaluation, and pipelining with other estimators.

Positioning with respect to other packages. Many metric learning algorithms were originally implemented by their authors in MATLAB without a common API convention.[1] In R, the package `dml` (Tang et al., 2018) implements several metric learning algorithms with a unified interface but is not tightly integrated with any general-purpose machine learning library. In Python, `pyDML` (Suárez et al., 2020) contains mainly fully supervised and unsupervised algorithms, while `pytorch-metric-learning`[2] focuses on deep metric learning using the `pytorch` framework (Paszke et al., 2019).

2. Background on Metric Learning

Metric learning is generally formulated as an optimization problem where one seeks to find the parameters of a distance function that minimize some objective function over the input data. All algorithms currently implemented in `metric-learn` learn so-called Mahalanobis distances. Given a real-valued parameter matrix L of shape `(n_components, n_features)` where `n_features` is the number of features describing the data, the associated Mahalanobis distance between two points x and x' is defined as $D_L(x, x') = \sqrt{(Lx - Lx')^\top (Lx - Lx')}$. This is equivalent to Euclidean distance after linear transformation of the feature space defined by L. Thus, if L is the identity matrix, standard Euclidean distance is recovered. Mahalanobis distance metric learning can thus be seen as learning a new embedding space, with potentially reduced dimension `n_components`. Note that D_L can also be written as $D_L(x, x') = \sqrt{(x - x')^\top M (x - x')}$, where we refer to $M = L^\top L$ as the Mahalanobis matrix.

Metric learning algorithms can be categorized according to the form of data supervision they require to learn a metric. `metric-learn` currently implements algorithms that fall into the following categories. *Supervised learners* learn from a data set with one label per training example, aiming to bring together points from the same class while spreading

1. See `https://www.cs.cmu.edu/~liuy/distlearn.htm` for a list of MATLAB implementations.
2. `http://github.com/KevinMusgrave/pytorch-metric-learning`

points from different classes. For instance, data points could be face images and the class could be the identity of the person (see Figure 1a). *Pair learners* require a set of pairs of points, with each pair labeled to indicate whether the two points are similar or not. These methods aim to learn a metric that brings pairs of similar points closer together and pushes pairs of dissimilar points further away from each other. Such supervision is often simpler to collect than class labels in applications when there are many labels. For instance, a human annotator can often quickly decide whether two face images correspond to the same person (Figure 1b) while matching a face to its identity among many possible people may be difficult. *Triplet learners* consider 3-tuples of points and learn a metric that brings the first (*anchor*) point of each triplet closer to the second point than to the third one. Finally, *quadruplet learners* consider 4-tuples of points and aim to learn a metric that brings the two first points of each quadruplet closer than the two last points. Both triplet and quadruplets learners can be used to learn a metric space where closer points are more similar with respect to an attribute of interest, in particular when this attribute is continuous and/or difficult to annotate accurately (e.g., the hair color of a person on an image, see Figure 1c, or the age of a person, see Figure 1d). Triplet and quadruplet supervision can also be used in problems with a class hierarchy.

3. Overview of the Package

The current release of `metric-learn` (v0.6.2) can be installed from the Python Package Index (PyPI) and conda-forge, for Python 3.6 or later.[3] The source code is available on GitHub at `http://github.com/scikit-learn-contrib/metric-learn` and is free to use, provided under the MIT license. `metric-learn` depends on core libraries from the SciPy ecosystem: `numpy`, `scipy`, and `scikit-learn`. Detailed documentation (including installation guidelines, the description of the algorithms and the API, as well as examples) is available at `http://contrib.scikit-learn.org/metric-learn`. The development is collaborative and open to all contributors through the usual GitHub workflow of issues and pull requests. Community interest for the package has been demonstrated by its recent inclusion in the `scikit-learn-contrib` organization which hosts high-quality `scikit-learn`-compatible projects,[4] and by its more than 1000 stars and 200 forks on GitHub at the time of writing. The quality of the code is ensured by a thorough test coverage (97% as of June 2020). Every new contribution is automatically checked by a continuous integration platform to enforce sufficient test coverage as well as syntax formatting with `flake8`.

Currently, `metric-learn` implements 10 popular metric learning algorithms. Supervised learners include Neighborhood Components Analysis (NCA, Goldberger et al., 2004), Large Margin Nearest Neighbors (LMNN, Weinberger and Saul, 2009), Relative Components Analysis (RCA, Shental et al., 2002),[5] Local Fisher Discriminant Analysis (LFDA, Sugiyama, 2007) and Metric Learning for Kernel Regression (MLKR, Weinberger and Tesauro, 2007). The latter is designed for regression problems with continuous labels. Pair learners include Mahalanobis Metric for Clustering (MMC, Xing et al., 2002), Information Theoretic Metric Learning (ITML, Davis et al., 2007) and Sparse High-Dimensional Metric

3. Support for Python 2.7 and 3.5 was dropped in v0.6.0.
4. `https://github.com/scikit-learn-contrib/scikit-learn-contrib`
5. RCA takes as input slightly weaker supervision in the form of *chunklets* (groups of points of same class).

Learning (SDML, Qi et al., 2009). Finally, the package implements one triplet learner and one quadruplet learner: Sparse Compositional Metric Learning (SCML, Shi et al., 2014) and Metric Learning from Relative Comparisons by Minimizing Squared Residual (LSML, Liu et al., 2012). Detailed descriptions of these algorithms can be found in the package documentation.

4. Software Architecture and API

`metric-learn` provides a unified interface to all metric learning algorithms. It is designed to be fully compatible with the functionality of `scikit-learn`. All metric learners inherit from an abstract `BaseMetricLearner` class, which itself inherits from `scikit-learn`'s `BaseEstimator`. All classes inheriting from `BaseMetricLearner` should implement two methods: `get_metric` (returning a function that computes the distance, which can be plugged into `scikit-learn` estimators like `KMeansClustering`) and `score_pairs` (returning the distances between a set of pairs of points passed as a 3D array). Mahalanobis distance learning algorithms also inherit from a `MahalanobisMixin` interface, which has an attribute `components_` corresponding to the transformation matrix L of the Mahalanobis distance. `MahalanobisMixin` implements `get_metric` and `score_pairs` accordingly as well as a few additional methods. In particular, `transform` allows to transform data using `components_`, and `get_mahalanobis_matrix` returns the Mahalanobis matrix $M = L^T L$.

Supervised metric learners inherit from `scikit-learn`'s base class `TransformerMixin`, the same base class used by `sklearn.LinearDiscriminantAnalysis` and others. As such, they are compatible for pipelining with other estimators via `sklearn.pipeline.Pipeline`. To illustrate, the following code snippet trains a `Pipeline` composed of LMNN followed by a k-nearest neighbors classifier on the UCI Wine data set, with the hyperparameters selected with a grid-search. Any other supervised metric learner can be used in place of LMNN.

```
from sklearn.datasets import load_wine
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.pipeline import Pipeline
from metric_learn import LMNN
X_train, X_test, y_train, y_test = train_test_split(*load_wine(return_X_y=True))
lmnn_knn = Pipeline(steps=[('lmnn', LMNN()), ('knn', KNeighborsClassifier())])
parameters = {'lmnn__k':[1, 2], 'knn__n_neighbors':[1, 2]}
grid_lmnn_knn = GridSearchCV(lmnn_knn, parameters, cv=3, n_jobs=-1, verbose=True)
grid_lmnn_knn.fit(X_train, y_train)
grid_lmnn_knn.score(X_test, y_test)
```

Weakly supervised algorithms (pair, triplet and quadruplet learners) `fit` and `predict` on a set of tuples passed as a 3-dimensional array. Tuples can be pairs, triplets, or quadruplets depending on the algorithm. Pair learners take as input an array-like `pairs` of shape `(n_pairs, 2, n_features)`, as well as an array-like `y_pairs` of shape `(n_pairs,)` giving labels (similar or dissimilar) for each pair. In order to `predict` the labels of new pairs, one needs to set a threshold on the distance value. This threshold can be set manually or automatically calibrated (at fit time or afterwards on a validation set) to optimize a given score such as accuracy or F1-score using the method `calibrate_threshold`. Triplet learners work on array-like of shape `(n_triplets, 3, n_features)`, where for each triplet we

want the first element to be closer to the second than to the third one. Quadruplet learners work on array-like of shape `(n_quadruplets, 4, n_features)`, where for each quadruplet we want the two first elements to be closer together than the two last ones. Both triplet and quadruplet learners can naturally `predict` whether a new triplet/quadruplet is in the right order by comparing the two pairwise distances. To illustrate the weakly-supervised learning API, the following code snippet computes cross validation scores for MMC on pairs from Labeled Faces in the Wild (Huang et al., 2012). Thanks to our unified interface, MMC can be switched for another pair learner without changing the rest of the code below.

```
from sklearn.datasets import fetch_lfw_pairs
from sklearn.model_selection import cross_validate, train_test_split
from metric_learn import MMC
ds = fetch_lfw_pairs()
pairs = ds.pairs.reshape(*ds.pairs.shape[:2], -1)  # we transform 2D images into 1D vectors
y_pairs = 2 * ds.target - 1  # we need the labels to be in {+1, -1}
pairs, _, y_pairs, _ = train_test_split(pairs, y_pairs)
cross_validate(MMC(diagonal=True), pairs, y_pairs, scoring='roc_auc',
               return_train_score=True, cv=3, n_jobs=-1, verbose=True)
```

5. Future Work

`metric-learn` is under active development. We list here some promising directions to further improve the package. To scale to large data sets, we would like to implement stochastic solvers (SGD and its variants), forming batches of tuples on the fly to avoid loading all data in memory at once. We also plan to incorporate recent algorithms that provide added value to the package, such as those that can deal with multi-label (Liu and Tsang, 2015) and high-dimensional problems (Liu and Bellet, 2019), or learn other forms of metrics like bilinear similarities, nonlinear and local metrics (see Bellet et al., 2015, for a survey).

Acknowledgments

We are thankful to Inria for funding 2 years of development. We also thank `scikit-learn` developers from the Inria Parietal team (in particular Gaël Varoquaux, Alexandre Gramfort and Olivier Grisel) for fruitful discussions on the design of the API and funding to attend SciPy 2019, as well as `scikit-learn-contrib` reviewers for their valuable feedback.

References

A. Bellet, A. Habrard, and M. Sebban. *Metric Learning*. Morgan & Claypool Publishers, 2015.

J. V. Davis, B. Kulis, P. Jain, S. Sra, and I. S. Dhillon. Information-Theoretic Metric Learning. In *ICML*, 2007.

J. Goldberger, S. Roweis, G. Hinton, and R. Salakhutdinov. Neighbourhood Components Analysis. In *NIPS*, 2004.

G. B. Huang, M. Mattar, H. Lee, and E. Learned-Miller. Learning to Align from Scratch. In *NIPS*, 2012.

E. Y. Liu, Z. Guo, X. Zhang, V. Jojic, and W. Wang. Metric Learning from Relative Comparisons by Minimizing Squared Residual. In *ICDM*, 2012.

K. Liu and A. Bellet. Escaping the Curse of Dimensionality in Similarity Learning: Efficient Frank-Wolfe Algorithm and Generalization Bounds. *Neurocomputing*, 333:185–199, 2019.

W. Liu and I. W. Tsang. Large Margin Metric Learning for Multi-Label Prediction. In *AAAI*, 2015.

A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*, 2019.

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.

G.-J. Qi, J. Tang, Z.-J. Zha, T.-S. Chua, and H.-J. Zhang. An Efficient Sparse Metric Learning in High-dimensional Space via L1-penalized Log-determinant Regularization. In *ICML*, 2009.

N. Shental, T. Hertz, D. Weinshall, and M. Pavel. Adjustment Learning and Relevant Component Analysis. In *ECCV*, 2002.

Y. Shi, A. Bellet, and F. Sha. Sparse Compositional Metric Learning. In *AAAI*, 2014.

M. Sugiyama. Dimensionality Reduction of Multimodal Labeled Data by Local Fisher Discriminant Analysis. *Journal of Machine Learning Research*, 8:1027–1061, 2007.

J. L. Suárez, S. García, and F. Herrera. pyDML: A Python Library for Distance Metric Learning. *Journal of Machine Learning Research*, 96:1–7, 2020.

Y. Tang, T. Gao, and N. Xiao. dml: Distance metric learning in R. *Journal of Open Source Software*, 3(30):1036, 2018.

K. Q. Weinberger and L. K. Saul. Distance Metric Learning for Large Margin Nearest Neighbor Classification. *Journal of Machine Learning Research*, 10:207–244, 2009.

K. Q. Weinberger and G. Tesauro. Metric Learning for Kernel Regression. In *AISTATS*, 2007.

E. P. Xing, A. Y. Ng, M. I. Jordan, and S. J. Russell. Distance Metric Learning with Application to Clustering with Side-Information. In *NIPS*, 2002.