Hey there's DALILA: a DictionAry LearnIng LibrAry

Veronica Tozzo[1], Vanessa D'Amario[2], and Annalisa Barla[3]

1 **Department of Informatics, Bioengineering, Robotics and System Engineering (DIBRIS), University of Genoa, Genoa, I-16146, Italy**
`veronica.tozzo@dibris.unige.it`

2 **Department of Informatics, Bioengineering, Robotics and System Engineering (DIBRIS), University of Genoa, Genoa, I-16146, Italy**
`vanessa.damario@dibris.unige.it`

3 **Department of Informatics, Bioengineering, Robotics and System Engineering (DIBRIS), University of Genoa, Genoa, I-16146, Italy**
`annalisa.barlao@unige.it`

Abstract

Dictionary Learning and Representation Learning are machine learning methods for decomposition, denoising and reconstruction of data with a wide range of applications such as text recognition, image processing and biological processes understanding. In this work we present DALILA, a scientific Python library for regularised dictionary learning and regularised representation learning that allows to impose prior knowledge, if available. DALILA, differently from the others available libraries for this purpose, is flexible and modular. DALILA is designed to be easily extended for custom needs. Moreover, it is compliant with the most widespread ML Python library and this allows for a straightforward usage and integration. We here present and discuss the theoretical aspects and discuss its strength points and implementation.

1998 ACM Subject Classification G.1.6 Optimization, D.1.3 Concurrent Programming, D.2.2 Design Tools and Techniques

Keywords and phrases Machine learning, dictionary learning, representation learning, alternating proximal gradient descent, parallel computing

Digital Object Identifier 10.4230/OASIcs.ICCSW.2017.6

1 Introduction

Nowadays many optimisation algorithms and libraries are freely available for the most disparate machine learning applications. For example some machine learning tasks, as classification, have hundreds of algorithms and libraries implementations. This is not the case for *Dictionary Learning* (DL) [11, 12, 13, 14, 17, 19, 23] and its specialisation *Representation Learning* (RL) [3, 25], which are designed for matrix decomposition and reconstruction. They can be applied on many application domains such as signal processing [23], image processing [18], bioinformatics [2] and text-recognition [1]. Even though dimensionality reduction methods such as PCA [10] and ICA [9] may be used to solve these tasks, dictionary learning is preferable when more emphasis on the interpretability of the dictionary is required [11].

In this paper we present DALILA, a Python library for DL and RL. The optimisation is based on *alternating proximal gradient descent* [4], which allows flexibility on the minimisation

2017 Imperial College Computing Student Workshop (ICCSW 2017).
Editors: Fergus Leahy and Juliana Franco; Article No. 6; pp. 6:1–6:14
Open Access Series in Informatics
Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

problem and therefore enables the imposition of prior knowledge. Designed to be extremely flexible and modular, DALILA can be easily extended differently from the other available libraries.

As regards the implementation, the library is compatible with the `scikit-learn` Python library and it supports the distribution of the most computationally heavy routines across different machines [6].

The remainder of this paper is organised as follows. In Section 2 we present DL, RL and the alternating proximal gradient descent algorithm. In Section 3 we give an overview on the library design and the implemented regularisers. In section 4 we comment the other available libraries on the topic. We finish with the conclusion and further work we intend to perform.

2 Theoretical background

In the following section a basic theoretical background on DL and RL problems is provided, together with the alternating proximal minimisation algorithm. The expert reader can feel free to skip it, if she/he is familiar with these mathematical concepts.

2.1 Dictionary Learning

Dictionary learning (DL) is a machine learning method that aims at finding a representation of the original data as a linear combination of basic patterns (atoms) and coefficients. The representation is completely data driven, in contrast to more generic and less adaptive methods such as Wavelet and Fourier transform [15, 26].

Given a dataset $\mathbf{X} \in \mathbb{R}^{n \times d}$ where n is the number of samples and d is the feature space dimension, the goal of DL is to decompose a dataset into two matrices:

- a dictionary $\mathbf{D} \in \mathbb{R}^{k \times d}$, a matrix of atoms that represent basic signals;
- the coefficients $\mathbf{C} \in \mathbb{R}^{n \times k}$, a matrix of weights for the atoms of the dictionary.

k represents the number of atoms composing the dictionary.

Hence, the original matrix $\mathbf{X}$ can be retrieved as a linear combination of dictionary and coefficients as in Equation (1).

$$\mathbf{X} \approx \mathbf{CD} \tag{1}$$

The recovering of the matrices $\mathbf{C}$ and $\mathbf{D}$ is solved by minimising a loss term ℓ: a a positive differentiable function that quantifies how well the multiplication of the two matrices approximates the signal.

$$\underset{\mathbf{C},\mathbf{D}}{\operatorname{argmin}} \left[\ell(\mathbf{X}, \mathbf{CD}) \right] \tag{2}$$

The minimisation problem in Equation (2) not always gives us the best possible solution. In presence of noisy data, ill-posed problems or when we have prior knowledge on the problem one of the usual tricks is to add terms or constraints to the functional in order to obtain a regularised problem (Equation (3)).

$$\underset{\mathbf{C},\mathbf{D}}{\operatorname{argmin}} \left[\ell(\mathbf{X}, \mathbf{CD}) + \Phi(\mathbf{C}) + \Psi(\mathbf{D}) \right] \tag{3}$$

$\Phi(\mathbf{C})$ and $\Psi(\mathbf{D})$ are two penalty functions that act respectively on the coefficients and on the dictionary. The functional in Equation (3) can be further specialised in order to include

Algorithm 1 Alternating proximal gradient descent

1: Random initialization of the matrices C and D
2: **for** $i = 0 : \text{max\_iters}$ **do**
3: $\quad\gamma_D \leftarrow \text{lipschitz\_step}_\ell(D)$
4: $\quad\gamma_C \leftarrow \text{lipschitz\_step}_\ell(C)$
5: $\quad D_{t+1} = \text{prox}_{\gamma_D \Psi}(D_t - \gamma_D \nabla_D (\ell_t))$
6: $\quad C_{t+1} = \text{prox}_{\gamma_C \Phi}(C_t - \gamma_C \nabla_C (\ell_t))$
7: $\quad$**if** difference between iterates $< \epsilon$ **and**
8: $\qquad$different between previous and current objective function $< \epsilon$ **then**
9: $\qquad$**break**

constraints sets that reduce the space in which a solution is admissible. A very common example is when we impose the involved matrices to be non-negative, a problem known as Non-negative Matrix Factorization (NMF) [11].

2.2 Alternating proximal gradient descent

It is worth noting that the optimisation of Equation (3) poses some issues. In fact, due to the multiplication present in the loss function ℓ, Equation (3) is jointly non-convex. Moreover the generality of the penalty terms requires the use of a minimisation algorithm that deals with different choices of penalties without a substantial change in its flow.

All taken into account, our optimisation choice is *alternating proximal gradient descent* [4] which assures the convergence to a local minima under the following assumptions: 1) the loss function $\ell(\mathbf{X}, \mathbf{CD})$ is differentiable and partially Lipschitz continuous; 2) it is possible to compute the proximal mapping of the penalty terms in closed form or at least to approximate it. See [4] for mathematical details and theoretical proofs.

The proximal mapping [16] of a function g for a point $\mathbf{u}$ is defined as Equation (4).

$$\text{prox}_{\mu g}(\mathbf{u}) = \underset{\mathbf{v}}{\text{argmin}} \left(g(\mathbf{v}) + \frac{1}{2\mu} \parallel \mathbf{v} - \mathbf{u} \parallel_2^2 \right) \tag{4}$$

When the computation of the prox is not available in a closed form, it can be approximated via a iterative algorithm.

A general overview on the optimization algorithm is given in Algorithm 1. The algorithm alternates the steps 3, 4, 5 and 6 until convergence. In step 4 the computation of the gradient descent is performed on the dictionary keeping the coefficients fixed. On the result the proximal mapping of the dictionary learning penalty Ψ is applied. The same is done in step 6 on the coefficients.

We want to remark that the gradient is computed w.r.t. the previous iteration in both cases [17]. This allows to perform the two steps in parallel if needed.

2.3 Representation learning (sparse coding)

Representation learning is a more general form of sparse coding (SC) that similarly to DL aims at finding the best approximation of a signal $\mathbf{X}$ (Equation (1)), when the dictionary $\mathbf{D}$ is given. This leads to a problem which is easier to minimise and, given convex loss function and penalty, becomes convex. The convexity property guarantees that a global optimum can be always reached.

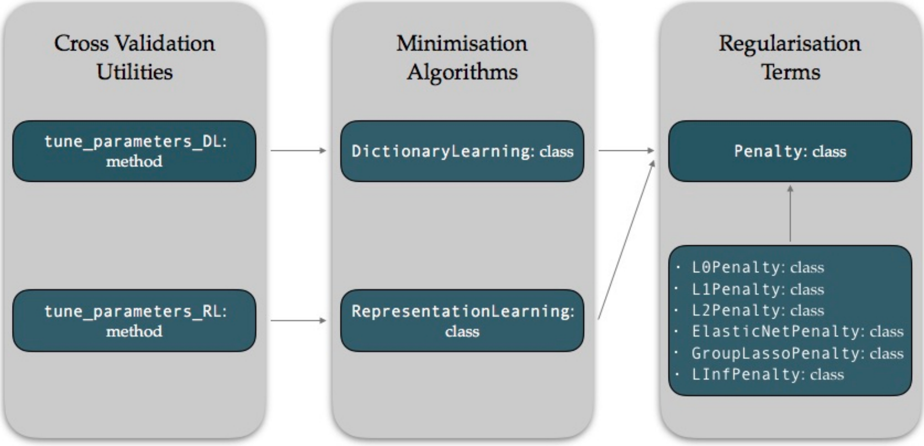

Figure 1 Diagram of DALILA library structure. The main core consists of the two classes which address the minimization problem. This depends on the class Penalty which represents a generic penalty term and that can be specialised by declaring subclasses. The library also offers cross validation utilities for the free parameters tuning.

In representation learning the choice of the penalty on the coefficients is arbitrary and dictated by the problem while in SC we assume the use of L^0 or L^1 norms. The formalisation is given in Equation (5).

$$\underset{\mathbf{C}}{\text{argmin}} \left[\ell(\mathbf{X}, \mathbf{CD}) + \Phi(\mathbf{C}) \right] \tag{5}$$

This optimisation problem, since involves the minimisation on only one variable, can be solved with proximal gradient descent that acts similarly to what explained in Algorithm 1 without steps 3 and 5.

3 DALILA

DALILA is a library for signal decomposition and reconstruction. Its first focus is Dictionary Learning (DL) described in Section 2. The fact that both the dictionary and the coefficients are learned from data allows for a more complete analysis of the results extracting useful information about the original signals. Moreover the possibility to impose prior knowledge on the problem using penalty terms grants that the final matrices respect certain constraints. Examples for regularised DL are: 1) image denoising where sparsity imposition forces the most important atoms to be used and the noisy ones to be discarded; 2) pattern recognition, where the atoms of the dictionary are seen as latent patterns from which the original signals are generated.

DALILA second focus is Representation Learning whose purpose is to represent the original data matrix on a new space defined by the atoms of the dictionary $\mathbf{D}$. Penalty terms can be added to impose a structure on this new representation.

The learned coefficients may be used as a new representation for further tasks such as: 1) compressed sensing that exploits sparsity reducing the size of the original signal; 2) classification where, rather than considering the original signal, the coefficients are used as new features.

3.1 Implementation

DALILA is implemented in Python. It supports different versions of Python and it is `scikit-learn` compatible. See `https://slipguru.github.io/dalila` for the full documentation and a quick start.

DALILA has a modular and easily extendible design (see Figure 1). The core of the library consists of two classes, `DictionaryLearning` and `RepresentationLearning` which respectively solve the two minimisation problems in Equation (3) and (5). These classes depend on a generic penalty term (`Penalty`) which can be specialised into different regularisers by declaring a subclass. The library is therefore easily extendible with new regularisers and flexible in the choice of the model. Cross validation utilities to tune the parameters of the model are provided, the two methods shown in Figure 1, `tune_parameters_DL` and `tune_parameters_RL`, can perform the tuning by parallel or distributed computation using `dask` library [6].

The loss function ℓ introduced in a generic form in Equation (3) and (5) is common to both `DictionaryLearning` and `RepresentationLearning` classes. It is implemented as the Frobenius norm of the difference between the original signal and its reconstruction, defined as Equation (6).

$$\ell(\mathbf{X}, \mathbf{CD}) = \|\mathbf{X} - \mathbf{CD}\|_F^2 \tag{6}$$

As regards the regularisation terms, called Φ and Ψ in Equation (3) and (5), DALILA offers many possibilities.

In this way a proper regulariser, dependent from the task, can be chosen, during the initialisation of the `DictionaryLearning/RepresentationLearning` instances. In fact `DictionaryLearning/RepresentationLearning` minimisation algorithms do not depend on the penalties chosen, as long as the penalty classes inherit from the superclass `Penalty` and reimplement the same methods (Figure 1). For a better understanding see Appendices B, C.

Available regularisation terms

The penalty terms Φ and Ψ are the product between a regularisation parameter and, typically, a norm. The norm is used to impose a structure on the matrix, while the regularisation parameter, a positive scalar, weights the regularisation term influence on the solution.

In the following we show the possible choices for Φ and Ψ applied on a generic matrix $\mathbf{M}$ whose i-th row is indicated as $\mathbf{M}_{i,:}$ and j-th column as $\mathbf{M}_{:,j}$. Its generic element is denoted by m_{ij}. With the notation $\Phi|\Psi$ we indicate that the penalty can be applied or on the dictionary or on the coefficients or on both.

- **`L1Penalty` - ℓ_1 norm**

$$\Phi|\Psi(\mathbf{M}) = \lambda \sum_i \|\mathbf{M}_{i,:}\|_1 = \lambda \sum_i \sum_j |m_{ij}| \tag{7}$$

  Regularisation terms of this form, due to the geometrical meaning of the ℓ_1 norm, force the solution to be sparse and, therefore, highly interpretable [21]. If the penalty is applied on the dictionary it promotes a dictionary whose atoms have a low number of non-null components. For the coefficients, the penalisation promotes a reconstruction based only on few atoms of the dictionary, discarding the ones which give minor contribution to the original signal.

The proximal operator related to this regulariser is

$$\text{prox}_{\Phi|\Psi}(m_{ij}) = \begin{cases} m_{ij} - \lambda & \text{if } m_{ij} > \lambda \\ 0 & \text{if } m_{ij} \in [-\lambda, \lambda] \\ m_{ij} + \lambda & \text{if } m_{ij} < -\lambda \end{cases} \tag{8}$$

- `L2Penalty` - **ℓ_2 norm**

$$\Phi|\Psi(\mathbf{M}) = \lambda \sum_i \left(\sum_j m_{ij}^2 \right)^{\frac{1}{2}} \tag{9}$$

Penalties of this form, as in the previous case, can be applied to both matrices $\mathbf{C}$ and $\mathbf{D}$. The ℓ_2 regularisation term leads to the shrinkage of the components of each row, but, differently from the ℓ_1 norm, it does not lead to a sparse solution [22].
The proximal operator is

$$\text{prox}_{\Phi|\Psi}(\mathbf{M}_{i,:}) = \max(1 - \lambda/\|\mathbf{M}_{i,:}\|_2, 0)\ \mathbf{M}_{i,:} \tag{10}$$

- `ElasticNetPenalty`

$$\Phi|\Psi(\mathbf{M}) = \sum_i \Big[\alpha\lambda_1 \|\mathbf{M}_{i,:}\|_1 + (1-\alpha)\lambda_2 \|\mathbf{M}_{i,:}\|_2 \Big] \tag{11}$$

Elastic Net can be preferable to ℓ_1 norm, in the case of highly correlated variables, and also to ℓ_2 norm since it inherits the possibility of finding a sparse solution [27].
Here λ_1 and λ_2 weight the two norms separately while $\alpha \in [0, 1]$ balances the contribution of the two terms. The proximal operator is

$$\text{prox}_{\Phi|\Psi}(\mathbf{M}_{i,:}) = \left(\frac{1}{1 + \alpha\lambda_2} \right) \text{prox}_{\lambda_1 \|\cdot\|_1}(\mathbf{M}_{i,:}) \tag{12}$$

- `L0Penalty` - **ℓ_0 pseudo-norm**

$$\Phi|\Psi(\mathbf{M}) : \forall i \quad \|\mathbf{M}_{i,:}\|_0 \le s \tag{13}$$

where $\|\mathbf{M}_{i,:}\|_0$ counts the number of non-zero elements in the row. The regularisation parameter $s \in \mathbb{N}$ impose the maximum number of non-null elements in $\mathbf{M}_{i,:}$, naturally leading to sparse results. The proximal operator is

$$\text{prox}_{\Phi|\Psi}(\mathbf{M}_{i,:}) = \begin{cases} m_{ij}, \ \text{if } m_{ij} \in \mathcal{S} \\ 0, \ \text{otherwise} \end{cases} \tag{14}$$

where $\mathcal{S}$ is the set containing the first s biggest components of $\mathbf{M}_{i,:}$.

- `LInfPenalty` - **ℓ_∞ norm**

$$\Phi(\mathbf{M}) = \lambda \sum_j \|\mathbf{M}_{:,j}\|_\infty \tag{15}$$

where $\|\mathbf{M}_{:,j}\|_\infty$ returns the maximum element in the column. This regularisation term acts column-wise only on the coefficients and it is useful in presence of a redundant dictionary [24].

The effect of this regulariser is to discard the atoms that overall have a low impact in the reconstruction while emphasising the atoms that, even if only in few samples, contribute largely.
The proximal operator is

$$\mathrm{prox}_{\Phi}(\mathbf{m}_j) = \mathbf{m}_j - \lambda\Pi_1(\mathbf{m}_j/\lambda) \tag{16}$$

The algorithm for the projection on the ℓ_1 ball is explained in [7].

- `GroupLassoPenalty` - **$\ell_{1,2}$ norm**

$$\Phi|\Psi(\mathbf{M}) = \lambda \sum_i \sum_{g\in\mathcal{G}} \|\mathbf{M}_{i,g}\|_2 \tag{17}$$

where $\mathcal{G}$ is the set of the groups (i.e. the indices of the columns) defined by the user. For each row of the matrix $\mathbf{M}$ the penalty enforces all the values of a group to be selected or discarded together (i.e. all of them set to zero). The groups cannot be overlapping and they have to cover all the columns indices. Its proximal mapping is

$$\mathrm{prox}_{\Phi|\Psi}(\mathbf{M}_{i,:})_{\mathcal{G}} = \max(1 - \lambda/\|\mathbf{M}_{i,g}\|_2, 0)\ \mathbf{M}_{i,g} \quad \text{for all } g \in \mathcal{G} \tag{18}$$

- **Additional user-implemented penalties** As introduces before DALILA is flexible in the sense that it allows to use different penalties without changing the minimisation flow and it further allows the user to declare new non-considered penalties. More details are given in Appendix C.

Both for `DictionaryLearning` and `RepresentationLearning` the user can impose non-negativity constraints on the involved matrices. When this requirement is applied both on the dictionary and the coefficients it is called Non-negative Matrix Factorization [11]. The non-negativity condition can, moreover, be imposed only on the coefficients in order to obtain a more interpretable contribution of the dictionary elements to the reconstruction of the original signal [19]. The projection is performed by setting to zero all the negative elements in the considered matrix.

Furthermore, in the `DictionaryLearning` class the user can impose the normalization condition on the dictionary matrix, which is equivalent to set the euclidean norm of each row equal to 1.

$$\|\mathbf{D}_{i,:}\|_2 = 1 \quad \text{for all } i \in \{1, \ldots, k\} \tag{19}$$

Model selection

A critical aspect of these reconstruction techniques is constituted by the choice of the free parameters, which are the number of atoms k that define the dictionary and the regularisation values that weight the penalty terms. This choice depends on the dataset given as input $\mathbf{X}$ and it can varies depending on different factors, as the high level of noise in the measurements, the redundancy of the founded dictionary and the interpretability of the solution. Given the fact that there is an infinite set of possible values for each parameter and no theoretical formulation that guides to the best solution exists, the only feasible approach is to empirically solve a searching problem over the parameters space.

DALILA allows for a fine tuning of the free parameters of the model on the dataset by performing a grid search based on cross validation. The best combination of parameters is selected as the one that returns the best mean score over multiple iterations. As score we use BIC (Bayesian Information Criterion) [20], shown in Equation (20).

$$\mathrm{BIC} = -\log(n) \cdot k - c \cdot \ell(\mathbf{X}, \mathbf{CD}) \tag{20}$$

where c is a positive constant. The highest value of the BIC corresponds to the best model in the search space. This procedure is available both for `DictionaryLearning` and `RepresentationLearning`.

The two procedures, `tune_parameters_DL` and `tune_parameters_RL`, allow the user to specify different search modalities. In `tune_parameters_DL` the user can choose among different configurations.

- tuning the number of atoms together with the dictionary penalty and after searching the regularisation parameter on the coefficients;
- fixing the number of atoms in the estimator and tuning the penalties together;
- fixing the penalties values and tuning the best number of atoms;
- tuning all the possible value together, number of atoms and regularisation parameters, analysing every possible combination in the grid.

4 Related work

As of today other libraries addressing similar tasks are available, SPAMS[1] (SPArse Modeling Software) [13] and the Decomposition modules of scikit-learn [5, 12].

SPAMS, implemented in C++, performs the decomposition tasks through dictionary learning, non negative matrix factorization and sparse PCA. It offers a good set of options, but, even if it is interfaceable with Python, it is not `scikit-learn` compatible. Therefore, it cannot be integrated in `scikit-learn` pipelines. Moreover, it is non trivial to customise or extend it.

The other main competitor, the decomposition module of `scikit-learn` library, implements dictionary learning and NMF but it only has few fixed penalty terms.

5 Conclusions and further work

In this work we introduced DALILA, a library for dictionary learning and representation learning. We presented its main features: the wide variety of penalties, the possibility to customise the library on specific problems, its compatibility with `scikit-learn` library, its high flexibility and its scalable architecture which allows to perform parallel parameter searching procedures.

The wide variety of penalties applicable on the matrices allow the user to solve a broad range of problems. Moreover, since scientific problems can introduce more specific and new needs, the possibility to customise and adapt the library is essential.

DALILA is fully compliant with one of the most complete machine learning Python libraries that is `scikit-learn`. This makes almost effortless its integration with the majority of machine learning Python pipelines.

The possibility to parallelise or distribute computationally heavy routines [6] greatly reduce the wall-clock time. Nevertheless our implementation is still basic and therefore the time performance are worse compared to the other presented libraries. In the future we plan to replace the use of `dask` with an hybrid parallelised system which will take advantage both of MPI tasks distribution and the computational acceleration given by GPUs.

Given DALILA flexibility and the existence of other Dictionary Learning related problems, we aim at extending it. The planned expansions are: 1) Discriminative Dictionary Learning [14], a variant of the dictionary learning problem which includes the classification task; 2)

[1] `http://spams-devel.gforge.inria.fr/doc/html/`

Shift Invariant Dictionary Learning [8] that allows the reconstruction of signals using atoms with smaller support than the original signal and 3) Total Variation penalty in combination with Lasso [19].

References

1 Charu C Aggarwal and ChengXiang Zhai. *Mining text data*. Springer Science & Business Media, 2012.
2 Ludmil B Alexandrov, Serena Nik-Zainal, David C Wedge, Peter J Campbell, and Michael R Stratton. *Deciphering signatures of mutational processes operative in human cancer*. *Cell reports*, 3(1):246–259, 2013.
3 Yoshua Bengio, Aaron C. Courville, and Pascal Vincent. Representation learning: A review and new perspectives. *IEEE Trans. Pattern Anal. Mach. Intell.*, 35(8):1798–1828, 2013. doi:10.1109/TPAMI.2013.50.
4 Jérôme Bolte, Shoham Sabach, and Marc Teboulle. *Proximal alternating linearized minimization for nonconvex and nonsmooth problems*. *Mathematical Programming*, 146(1-2):459–494, 2014.
5 Andrzej Cichocki and PHAN Anh-Huy. *Fast local algorithms for large scale nonnegative matrix and tensor factorizations*. *IEICE transactions on fundamentals of electronics, communications and computer sciences*, 92(3):708–721, 2009.
6 Dask Development Team. *Dask: Library for dynamic task scheduling*, 2016. URL: http://dask.pydata.org.
7 John Duchi, Shai Shalev-Shwartz, Yoram Singer, and Tushar Chandra. *Efficient projections onto the l 1-ball for learning in high dimensions*. In *Proceedings of the 25th international conference on Machine learning*, pages 272–279. ACM, 2008.
8 Roger Grosse, Rajat Raina, Helen Kwong, and Andrew Y Ng. *Shift-invariance sparse coding for audio classification*. *arXiv preprint arXiv:1206.5241*, 2012.
9 Aapo Hyvärinen, Juha Karhunen, and Erkki Oja. *Independent component analysis*, volume 46. John Wiley & Sons, 2004.
10 Ian Jolliffe. *Principal component analysis*. Wiley Online Library, 2002.
11 Daniel D Lee and H Sebastian Seung. *Learning the parts of objects by non-negative matrix factorization*. *Nature*, 401(6755):788–791, 1999.
12 Chih-Jen Lin. Projected gradient methods for nonnegative matrix factorization. *Neural Computation*, 19(10):2756–2779, 2007. doi:10.1162/neco.2007.19.10.2756.
13 Julien Mairal, Francis Bach, Jean Ponce, and Guillermo Sapiro. *Online Learning for Matrix Factorization and Sparse Coding*. *Journal of Machine Learning Research*, 11:19–60, 2010. URL: http://dl.acm.org/citation.cfm?id=1756006.1756008.
14 Julien Mairal, Jean Ponce, Guillermo Sapiro, Andrew Zisserman, and Francis R Bach. *Supervised dictionary learning*. In *Advances in neural information processing systems*, pages 1033–1040, 2009.
15 Stephane Mallat. *A wavelet tour of signal processing: the sparse way*. Academic press, 2008.
16 Neal Parikh, Stephen Boyd, et al. *Proximal algorithms*. *Foundations and Trends® in Optimization*, 1(3):127–239, 2014.
17 Alain Rakotomamonjy. *Direct optimization of the dictionary learning problem*. *IEEE Transactions on Signal Processing*, 61(22):5495–5506, 2013.
18 Saiprasad Ravishankar and Yoram Bresler. *MR image reconstruction from highly undersampled k-space data by dictionary learning*. *IEEE transactions on medical imaging*, 30(5):1028–1041, 2011.
19 Saverio Salzo, Salvatore Masecchia, Alessandro Verri, and Annalisa Barla. *Alternating proximal regularized dictionary learning*. *Neural computation*, 26(12):2855–2895, 2014.

20 Gideon Schwarz et al. *Estimating the dimension of a model*. *The annals of statistics*, 6(2):461–464, 1978.
21 Robert Tibshirani. *Regression shrinkage and selection via the lasso*. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.
22 Andreĭ Nikolaevich Tikhonov, Vasiliĭ Iakovlevich Arsenin, and Fritz John. *Solutions of ill-posed problems*, volume 14. Winston Washington, DC, 1977.
23 Ivana Tosic and Pascal Frossard. *Dictionary learning*. *IEEE Signal Processing Magazine*, 28(2):27–38, 2011.
24 Joel A Tropp. *Just relax: Convex programming methods for identifying sparse signals in noise*. *IEEE transactions on information theory*, 52(3):1030–1051, 2006.
25 Joel A. Tropp and Anna C. Gilbert. Signal recovery from random measurements via orthogonal matching pursuit. *IEEE Trans. Information Theory*, 53(12):4655–4666, 2007. doi:10.1109/TIT.2007.909108.
26 Martin Vetterli, Jelena Kovacevic, and Vivek K Goyal. *Fourier and wavelet signal processing*. *Book site*, 2013.
27 Hui Zou and Trevor Hastie. *Regularization and variable selection via the elastic net*. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 67(2):301–320, 2005.

A Appendix: example of usage and related output

In this appendix we want to offer some insights of how DALILA code works and the possible outcomes we can obtain. All the experiments are performed on the ORL database of faces [2]. This database is saved as a matrix within the library with 400 samples and 112×92 features.

In order to perform Dictionary Learning on this dataset we firstly need to flatten the matrices into arrays and then apply the fitting procedure. In this estimator we are using an arbitrary number of atoms that we choose randomly and we impose non-negativity on both the matrices since we are dealing with gray scale images that have values in the range of [0,255].

```python
import numpy as np

from dalila.dictionary_learning import DictionaryLearning
from dalila.penalty import L1Penalty

dataset = np.load("/path/to/dalila/folder/dalila/databases/
    ORL_database.npy")
d1, d2, n = dataset.shape
data = np.empty((n, d1*d2))
for i in range(n):
data[i,:] = np.ravel(dataset[:,:,i])

estimator = DictionaryLearning(k=60, non_negativity="both")
estimator.fit(data, n_iter=1000)
C, D = estimator.decomposition()
```

Some of the results obtained with this code are depicted in Figure 2. Here the most used atoms in the reconstructions are showed. In Figure 3, instead, we show some reconstructions and their original signals to perform a qualitative comparison.

[2] The database is available for download at the link http://www.cl.cam.ac.uk/research/dtg/attarchive/facedatabase.html

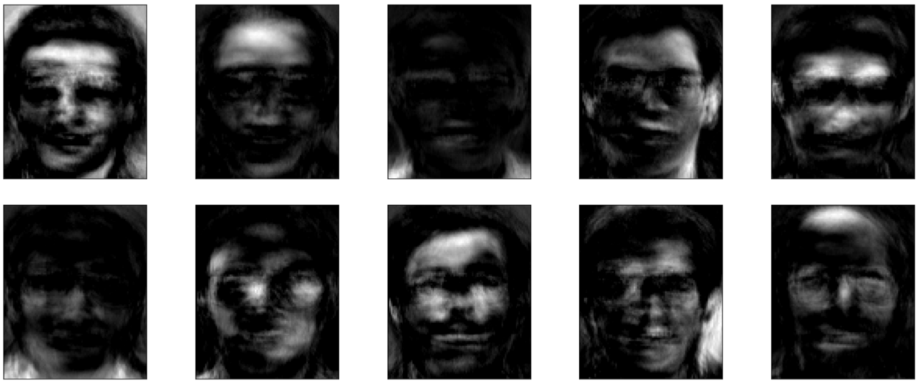

Figure 2 The ten more recurrent atoms for the reconstruction of the samples in ORL dataset.

In this example we tune only the right number of atoms we should use for this particular dataset. Given the dimensionality of the dataset we picked as range $k \in [5, 10, 15, 20, 30, 40, 50]$.

Moreover we did not use the basic BIC score but a normalised one since the values within the BIC computation, with this dataset, are not comparable in their magnitude (see `scoring_function`). The results are showed in Figure 4 where we can see that the best number of atoms is 15.

```
import matplotlib.pyplot as plt
import numpy as np
from dalila.parameters_research import tune_parameters_DL
from dalila.dictionary_learning import DictionaryLearning

dataset = np.load("/path/to/dalila/folder/dalila/databases/ORL_database.npy")
d1, d2, n = dataset.shape
data = np.empty((n, d1*d2))
for i in range(n):
data[i,:] = np.ravel(dataset[:,:,i])
```

```
def scoring_function(estimator, X, y=None):
C, D = estimator.decomposition()
r_error = (np.linalg.norm(estimator.X - C.dot(D))/
np.linalg.norm(estimator.X))
n = estimator.X.shape[0]
return -(2.3*np.array(r_error) + 0.001*estimator.k*np.log(n))

possible_ks = [5,10,15,20,30, 40,50]
estimator = DictionaryLearning(k=5, non_negativity="both")
gscv = tune_parameters_DL(data, estimator, analysis=2,
range_k=possible_ks,  fit_params={'n_iter':500},
scoring_function=scoring_function)
```

Figure 3 Examples of the reconstruction obtained with DL decomposition. In the first row the reconstructions are shown and beneath them their original picture.

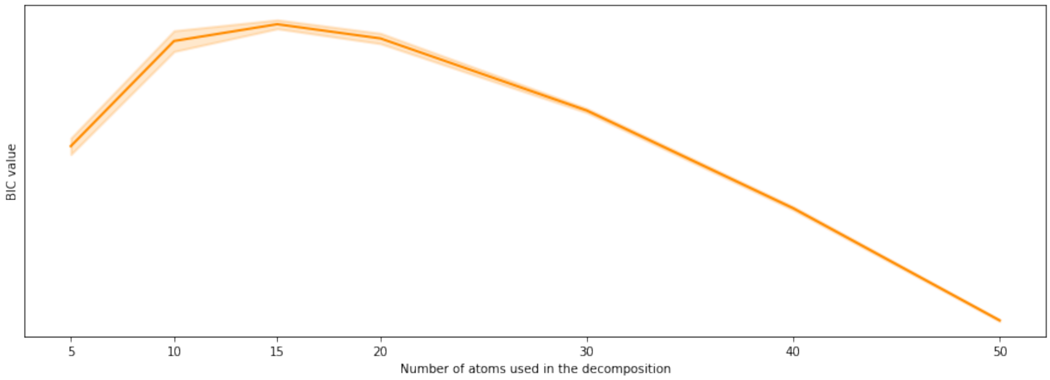

Figure 4 Curve of the `scoring_function` values (refined BIC) w.r.t. the number of atoms used for the decomposition. To an higher value corresponds a better model for the dataset. In this case the best model is the one with 15 atoms in the dictionary.

B Appendix: interchangeability of the penalties

With DALILA trying different penalties on the same dataset requires only few lines of code.

For example suppose we have the right number of atoms (with the dataset we use is 7) in which to decompose the dataset and that, some oracle, told us the perfect regularisation parameters for each penalty on that dataset. Then we may want to try different sparsifications on the coefficients to see which one better approximates the original signal. We tried `L1Penalty`, `L0Penalty` and `ElasticNetPenalty`.

```
import numpy as np

from dalila.dictionary_learning import DictionaryLearning
from dalila.dataset_generator import synthetic_data_non_negative
from dalila.penalty import L1Penalty, L0Penalty, ElasticNetPenalty

X, D, C = synthetic_data_non_negative()

estimator = DictionaryLearning(k=7, coeff_penalty=L1Penalty(0.01),
    non_negativity="both")
estimator.fit(X)
C_l1, D_l1 = estimator.decomposition()
error_l1 = estimator.reconstruction_error()

estimator = DictionaryLearning(k=7, coeff_penalty=L0Penalty(3),
    non_negativity="both")
estimator.fit(X)
C_l0, D_l0 = estimator.decomposition()
error_l0 = estimator.reconstruction_error()

estimator = DictionaryLearning(k=7, coeff_penalty=ElasticNetPenalty
    (0.01, 0.1, 0.7), non_negativity="both")
estimator.fit(X)
C_en, D_en = estimator.decomposition()
error_en = estimator.reconstruction_error()
```

After the execution of this piece of code the comparison between the results is straight-forward and you can notice that the effort is minimal.

C Appendix: addition of a new penalty

DALILA allows to easily introduce new customised penalties for the optimisation of dictionary learning or representation learning. We want to underline that, even if the implementation steps are easy, the function that computes the proximal mapping has to be correct and no theoretical inconsistencies should be present. The behaviour is otherwise unpredictable.

The first step is the import of the super-class Penalty that our new penalty has to extend. We also import other things that we need later.

```
from dalila.representation_learning import RepresentationLearning
from dalila.penalty import Penalty
import numpy as np
```

The implementation of the new class, besides the construction, has to expose the method `apply_prox_operator` that is the one called during the minimisation. In this method the prox operator is implemented.

```
class NewPenalty(Penalty):

def __init__(self, regularization_parameter):
self.regularization_parameter = regularization_parameter

# x is the matrix on which apply the prox
```

```
# gamma is the gradient descent step
def apply_prox_operator(self, x, gamma):
# if you are declaring a real penalty
# change the implementation and
# transform x according to your penalty
return x
```

Once we have declared the penalty, in this case a penalty that does nothing, we put it into the representation learning procedure.

```
fake_data = np.random.rand(50,50)
fake_dictionary = np.random.rand(5, 50)
estimator = RepresentationLearning(fake_dictionary, penalty=
    NewPenalty(5))
estimator.fit(fake_data)
```

It is also possible to use the parameter research procedures with the new penalty provided that we also overwrite the method `make_grid` since the searching function assumes it exists. We here show a basic example of how it can be implemented, one may want to vary the interval or the sampling procedure.

```
class NewPenalty(Penalty):

# ..as above..

def make_grid(self, low=0.001, high=1, number=10):
# possible regularization parameters to analyse
values = np.linspace(low, high, number)
l = []
# the list has to be composed of NewPenalty objects
for (i, v) in enumerate(values):
l.append(NewPenalty(v))
return l
```

Again we show that it is usable right away without further code.

```
from dalila.parameters_research import tune_parameters_RL
estimator = RepresentationLearning(fake_dictionary, penalty=
    NewPenalty(5))
gscv = tune_parameters_RL(fake_data, estimator, coeff_penalty_range
    =(0.1, 1, 3))
```