MKLpy: a python-based framework for Multiple Kernel Learning

Ivano Lauriola   IVANO.LAURIOLA@PHD.UNIPD.IT
Fabio Aiolli   AIOLLI@MATH.UNIPD.IT
Department of Mathematics, University of Padova, 35121 Italy

Editor: ...

Abstract

Multiple Kernel Learning is a recent and powerful paradigm to learn the kernel function from data. In this paper, we introduce MKLpy, a python-based framework for Multiple Kernel Learning. The library provides Multiple Kernel Learning algorithms for classification tasks, mechanisms to compute kernel functions for different data types, and evaluation strategies. The library is meant to maximize the usability and to simplify the development of novel solutions.

Keywords: Multiple Kernel Learning, Deep kernels, Kernel methods

1. Introduction

Kernel machines (Shawe-Taylor et al., 2004) are a popular family of machine learning algorithms widely used in the literature to solve classification, regression, and clustering problems. These methods comprise two parts, (i) a function, named kernel, defining the similarities between pairwise examples and (ii) a learning algorithm leveraging these similarities. Recently, several methods have been proposed to directly learn the optimal kernel function from data, overcoming the limits of an expensive validation process. One of the most popular and effective kernel learning frameworks is the Multiple Kernel Learning (Gönen and Alpaydin, 2011) (MKL), whose purpose is to learn the kernel as a principled combination of several base (or weak) kernels. Nowadays, MKL is gaining popularity in different domains and, to the best of our knowledge, it is mainly applied to two categories of tasks, that are (i) information fusion, where users define a base kernel for each view or source (e.g. audio, video, and text) (Xu et al., 2013; Guo et al., 2014), and (ii) deep kernel learning, where the goal is to create kernels with the deep structure that characterizes neural networks (Donini and Aiolli, 2017; Lauriola et al., 2020).

Having said that, this paper introduces MKLpy: a python-based framework for MKL. Specifically, we hereby describe the most important features and characteristics of the library, focusing on the main use-cases and the computational challenges addressed.

2. Features and use-cases

MKLpy encapsulates *everything you need* to run MKL algorithms, to compute kernels, and to evaluate solutions, and it is specifically designed in accordance with two main principles, i.e. the usability and the extendability, covering different needs from user and developer
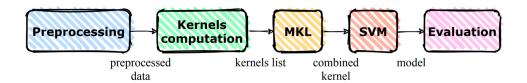
Figure 1: MKL pipeline. The intermediate output between states is also exposed.

perspectives. On one hand, the library provides high-level APIs to easily apply MKL algorithms to the user's specific task. Inspired by the scikit-learn (Pedregosa et al., 2011) project, the public interface of MKL algorithms consists of simple primitives to train an algorithm `mkl.fit(...)` and to perform predictions `mkl.predict(...)`. On the other hand, MKLpy is designed to dissect the entire MKL pipeline, depicted in Figure 1, from the computation of base kernels to the inference phase, allowing a modular, incremental, and targeted development. The main features of MKLpy can be divided into 3 categories:

Kernel functions : MKLpy natively supports several kernel functions for (i) vectorial data, such as the popular homogeneous polynomial kernel (HPK), (ii) binary valued data, such as the conjunctive and disjunctive boolean kernels, and (iii) structured data, such as the p-spectrum and string kernels. Additionally, MKLpy supports all the kernels from the scikit-learn package, which have the same public interface, and custom kernels. In the following snippet, we show a basic example of kernels computation. Specifically, we create a single monotone conjunctive kernel (`K`), and a list of kernels (`KL`) for MKL algorithms.

```
1  from MKLpy.metrics.pairwise import monotone_conjunctive_kernel as mck
2  K  = mck(X)   #X is a scikit-like training matrix or Torch.Tensor
3  KL = [mck(X, c=arity) for arity in range(1,6)]
```

MKL algorithms : The core of the library is the implementation of various MKL algorithms, such as AverageMKL[1], EasyMKL (Aiolli and Donini, 2015), R-MKL (Do et al., 2009), and GRAM (Lauriola et al., 2017). The algorithms are available with specialized wrappers exposing the same public interface of scikit-learn predictors. The wrappers automatically encapsulate (i) the kernels combination mechanism, (ii) the training leveraging the resulting kernel, and (ii) the prediction mechanism.

```
1  from MKLpy.algorithms import AverageMKL, EasyMKL
2  mkl = AverageMKL()  #create a MKL instance
3  mkl = mkl.fit(KLtr, Ytr)    #train the model
4  predictions = mkl.predict(KLte)
```

We also provide several customizations in the pipeline to have more control over execution. For instance, users may perform only the combination step, inspect the weights, and specify the kernel method dealing with the combination, such as the popular SVM, KOMD (Aiolli et al., 2008), or any scikit-compliant kernel machine.

Evaluation : An important step in kernel learning is the evaluation of the learned kernels. MKLpy offers several criteria and tools to evaluate the quality of a kernel, such as the

---

1. AverageMKL is a simple wrapper computing the average of base kernels.

radius of the Minimum Enclosing Ball (Gai et al., 2010; Chung et al., 2003) (MEB) containig data in the kernel space, the margin, the alignment between kernels (Kandola et al., 2002), or the empirical complexity (Spectral Ratio) (Donini and Aiolli, 2017).

```
1  from MKLpy import metrics
2  margin = metrics.margin(K,Y)     #margin between classes
3  radius = metrics.radius(K)       #radius of MEB
4  SR     = metrics.spectral_ratio(K, norm=True)    #empirical complexity
```

Additionally, the user may leverage tools from the scikit-learn package to evaluate the performances in terms of accuracy, AUC, and other metrics.

3. Computational efficiency

Besides scikit-learn, MKLpy relies on multiple scientific libraries, namely numpy (Walt et al., 2011), CVXOPT (Andersen et al., 2013), and PyTorch (Paszke et al., 2019). These high-level libraries leverage, in turn, low-level routines, provided by BLAS (Blackford et al., 2002) and LAPACK (Anderson et al., 1999), that execute the most of operations. To speed up the computation, we strongly suggest to correctly install, configure, and link the most appropriate configuration and external libraries before installing MKLpy, according to the system architecture. Furthermore, we recommend running the code on a multi-core machine with the AVX2 instructions set.

The main bottleneck of MKL algorithms preventing to scale with large datasets is memory consumption. In order to improve the efficiency, most of the optimization engines used in this library require to keep the whole kernels list in memory during training. In order to make the resource allocation tractable and comparable to a single SVM, we provide specialized kernels generators that perform a lazy computation of kernels matrices, making them available on-the-fly only when they are needed during optimization. The generators are designed to reduce as much as possible the required memory and to speed up the kernels computation, when possible, with specific caching mechanisms.

```
1  from MKLpy.generators import HPK_generator #homogeneous polynomial kernels
2  KLtr = HPK_generator(Xtr, degrees=range(1,21))
3  mkl  = EasyMKL.fit(KLtr, Ytr)
```

In the Table 1 we show the time and memory consumption required by EasyMKL with 20 HPKs applied to two datasets, Madelon and Phishing[2]. Specifically, we show the resources usage when dealing with explicit lists of kernels and MKLpy generators. The values include the kernels computation and model training. What is striking from the table is the huge reduction of memory consumption, with the average of 3.7 times.

Dataset	examples	features	list	generator w. cache	generator
Madelon	6000	5000	24.4 s / 7.3 GB	28.0 s / 2.6 GB	60.5 s / 2.3 GB
Phishing	11055	68	115.7 s / 23.9 GB	120.4 s / 6.2 GB	126.8 s / 5.7 GB

Table 1: Computational time and memory usage of EasyMKL when trained with explicit kernels list and MKLpy generators.

---

2. The datasets are freely available on https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/

4. Guidelines and best practices

The development of novel MKL algorithms may often be tricky and brimful of smokescreens, and there are several details and practices that need to be taken into account during the development and evaluation processes. To this end, we hereby describe a useful set of guidelines and best practices to develop and to properly evaluate novel MKL algorithms leveraging this library. These guidelines are mainly inspired by our research experience and by the numerous problems we encountered during the development of MKLpy, and they are listed in the following:

- MKL algorithms leverage the concept for which the combination of multiple representations improve the single representation. This aspect is fundamental but not obvious. Indeed, different representations may introduce noise or useless information. To this end, a comparison of the combined kernel against the single base kernels may certify the quality of the MKL solution.

- Despite its simplicity, several existing MKL algorithms do not even perform better than the simple average of base kernels. However, this baseline is rarely considered in the assessment process. Thus, we strongly encourage to use this baseline when evaluating MKL solutions.

- The combination weights provide useful insights concerning the sparsity of the solution and the contribution of base kernels, and their analysis prevents the development a trivial solution. For instance, the algorithm may assign 0 weights to all kernels but one. In this case, the MKL can be used for feature selection or hyper-parameter tuning (Massimo et al., 2016).

- The weight assigned to each kernel depends on the norm of the latter. If we scale up the norm of a base kernel, its contribution in the combination may scale up as well[3] and the kernel may receive a high weight just because it has a high norm. This aspect may provide a sub-optimal solution in some cases, and the resulting weights vector may be misleading. To reduce this problem, we suggest normalizing base kernels.

To this end, MKLpy provides multiple callbacks and tools to visualize the weights vector over iterations, to monitor the training objective, and to evaluate validation metrics.

5. Conclusions

This paper introduces MKLpy, a python-based framework for Multiple Kernel Learning (MKL). The framework encapsulates everything you need to run and to evaluate MKL algorithms. MKLpy is specifically designed to maximize the usability, and to support the development of novel algorithms and functionalities. Here, we only exposed the main concepts of the library, without providing a deep dive into specific functionalities. Exhaustive examples covering a plethora of use-cases and implementation details for developers are available on our documentation[4]. The code of MKLpy is currently hosted on GitHub[5].

---

3. This happens, for instance, with EasyMKL.
4. Documentation: `https://mklpy.readthedocs.io/en/latest/`
5. GitHub: `https://github.com/IvanoLauriola/MKLpy`

References

Fabio Aiolli and Michele Donini. Easymkl: a scalable multiple kernel learning algorithm. *Neurocomputing*, 169:215–224, 2015.

Fabio Aiolli, Giovanni Da San Martino, and Alessandro Sperduti. A kernel method for the optimization of the margin distribution. In *International Conference on Artificial Neural Networks*, pages 305–314. Springer, 2008.

Martin S Andersen, Joachim Dahl, and Lieven Vandenberghe. Cvxopt: A python package for convex optimization, version 1.1. 6. *Available at cvxopt. org*, 54, 2013.

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. *LAPACK Users' Guide*. Society for Industrial and Applied Mathematics, Philadelphia, PA, third edition, 1999. ISBN 0-89871-447-8 (paperback).

L Susan Blackford, Antoine Petitet, Roldan Pozo, Karin Remington, R Clint Whaley, James Demmel, Jack Dongarra, Iain Duff, Sven Hammarling, Greg Henry, et al. An updated set of basic linear algebra subprograms (blas). *ACM Transactions on Mathematical Software*, 28(2):135–151, 2002.

Kai-Min Chung, Wei-Chun Kao, Chia-Liang Sun, Li-Lun Wang, and Chih-Jen Lin. Radius margin bounds for support vector machines with the rbf kernel. *Neural computation*, 15 (11):2643–2681, 2003.

Huyen Do, Alexandros Kalousis, Adam Woznica, and Melanie Hilario. Margin and radius based multiple kernel learning. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 330–343. Springer, 2009.

Michele Donini and Fabio Aiolli. Learning deep kernels in the space of dot product polynomials. *Machine Learning*, 106(9-10):1245–1269, 2017.

Kun Gai, Guangyun Chen, and Chang-shui Zhang. Learning kernels with radiuses of minimum enclosing balls. In *Advances in neural information processing systems*, pages 649–657, 2010.

Mehmet Gönen and Ethem Alpaydin. Multiple kernel learning algorithms. *Journal of machine learning research*, 12(64):2211–2268, 2011.

D. Guo, J. Zhang, X. Liu, Y. Cui, and C. Zhao. Multiple kernel learning based multi-view spectral clustering. In *2014 22nd International Conference on Pattern Recognition*, pages 3774–3779, 2014.

Jaz Kandola, John Shawe-Taylor, and Nello Cristianini. Optimizing kernel alignment over combinations of kernel. 2002.

Ivano Lauriola, Mirko Polato, and Fabio Aiolli. Radius-margin ratio optimization for dot-product boolean kernel learning. In *International conference on artificial neural networks*, pages 183–191. Springer, 2017.

Ivano Lauriola, Claudio Gallicchio, and Fabio Aiolli. Enhancing deep neural networks via multiple kernel learning. *Pattern Recognition*, page 107194, 2020.

Carlo M Massimo, Nicolò Navarin, and Alessandro Sperduti. Hyper-parameter tuning for graph kernels via multiple kernel learning. In *International Conference on Neural Information Processing*, pages 214–223. Springer, 2016.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dÁlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL `http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf`.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *the Journal of machine Learning research*, 12:2825–2830, 2011.

John Shawe-Taylor, Nello Cristianini, et al. *Kernel methods for pattern analysis*. Cambridge university press, 2004.

Stéfan van der Walt, S Chris Colbert, and Gael Varoquaux. The numpy array: a structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

Chang Xu, Dacheng Tao, and Chao Xu. A survey on multi-view learning. *arXiv preprint arXiv:1304.5634*, 2013.