PySEF: A Python Library for Similarity-based Dimensionality Reduction

Nikolaos Passalis and Anastasios Tefas
passalis@csd.auth.gr, tefas@aiia.csd.auth.gr

Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki 54124, Greece

Abstract

PySEF is an efficient and modular implementation of the Similarity Embedding Framework (SEF) in Python that allows for easily performing similarity-based dimensionality reduction (DR) as well as defining custom similarity targets and embedding functions. *PySEF* contains a collection of predefined target functions that can be used to perform DR using various existing techniques, ranging from Principal Component Analysis (PCA) to providing out-of-sample extensions for the t-Distributed Stochastic Neighbor Embedding (t-SNE). Furthermore, developing novel DR techniques within *PySEF* becomes a matter of just defining a new similarity target function using a few lines of code. *PySEF* also allows for transparently switching between the CPU and the GPU for the optimization, follows the *scikit-learn* calling conventions, and it is optimized to efficiently handle large-scale datasets.

Keywords: Dimensionality Reduction, Similarity Embedding Framework

1. Introduction

Dimensionality reduction (DR) methods are among the fundamental preprocessing steps for a wide range of knowledge-based systems, ranging from medical diagnosis systems [1], to recommendation systems [2]. Most DR techniques rely on second-order statistics to define their optimization objective. However, using unbounded distance metrics comes with several drawbacks. Most methods cannot effectively handle outliers, carefully designed regularizers are needed and it is not always clear how to manipulate the distances to derive new DR techniques [3].

2. Background

The aforementioned drawbacks are addressed in a recently proposed DR framework that builds upon the notion of similarity, the Similarity Embedding Framework (SEF) [3]. SEF defines a generic optimization objective that uses the pairwise similarities between the data samples instead of their distances. This allows for expressing different DR techniques by simply setting the appropriate target similarity matrix. Let $f : \mathbb{R}^d \rightarrow \mathbb{R}^m$ be an embedding function that projects the high dimensional data samples $\mathbf{x}_i \in \mathbb{R}^d$ to a lower dimensional space $\mathbb{R}^m$, where $\mathbf{y}_i \in \mathbb{R}^m$ is the low dimensional representation of $\mathbf{x}_i$. Also, let $S(\mathbf{x}_i, \mathbf{x}_j)$ be a function that measures the similarity between two data points $\mathbf{x}_i$ and $\mathbf{x}_j$. Any differentiable similarity function can be used to define $S(\mathbf{x}_i, \mathbf{x}_j)$. In [3], the Gaussian kernel is used to define the similarity measure:

$$S(\mathbf{x}_i, \mathbf{x}_j) = exp(-||\mathbf{x}_i - \mathbf{x}_j||_2^2/\sigma_P^2), \tag{1}$$

where σ_P is the scaling factor of the Gaussian kernel. SEF aims to learn an embedding function f that projects the data into a lower dimensional space where the similarities between the data are transformed according to a given target. To this end, the following loss function is used during the optimization:

$$J_s = \frac{1}{2||\mathbf{M}||_1} \sum_{i=1}^{N} \sum_{j=1}^{N} [\mathbf{M}]_{ij}([\mathbf{P}]_{ij} - [\mathbf{T}]_{ij})^2, \tag{2}$$

where N is the number of training samples, $[\mathbf{P}]_{ij} = S(f(\mathbf{x}_i), f(\mathbf{x}_j))$ denotes the similarity between two data samples in the low-dimensional space, $[\mathbf{T}]_{ij}$ is the target similarity between the i-th and the j-th sample and $\mathbf{M} \in \mathbb{R}^{N \times N}$ is a weighting mask that defines the importance of achieving the target similarity between two points in the low-dimensional space. That way, SEF can perform different types of dimensionality reduction just by defining a different target similarity matrix $\mathbf{T}$.

3. The PySEF Library

PySEF is an efficient and modular implementation of the SEF in Python that allows for performing similarity-based dimensionality reduction without dealing with the implementation details of the SEF. *PySEF* contains a collection of predefined target functions that can be used to perform DR using various existing techniques. Furthermore, developing novel DR techniques within the *PySEF* becomes a matter of just writing a few lines of code. That way, novel DR techniques can be easily implemented and evaluated, assisting research on similarity-based DR methods, as well as providing a practical DR tool.

PySEF is built on top of the *PyTorch* library [4], allowing for transparently switching between the CPU and the GPU for the optimization. Using the *PyTorch* library also significantly simplifies the process of the developing custom embedding functions. Also, the implementation was optimized towards handling large-scale datasets, e.g., optimization in batches using h5py is supported [5]. Finally, motivated by the fact that many machine learning researchers are familiar with the *scikit-learn* library [6], we follow the standard *scikit-learn* calling conventions. That way, existing users of *scikit-learn* should be able to get familiar with *PySEF* in a matter of minutes, while providing a transparent way to interact with the *PySEF* library and hiding the complexities of the underlying implementation.

PySEF currently supports both linear and kernel embedding functions. The following four similarity targets (DR methods) are already implemented in *PySEF* (extensive examples on how to use them are also included in the documentation): a) *copy target*, which can be used to provide out-of-sample extensions and fast linear approximations of complex DR techniques, such as t-SNE, etc., b) *supervised target*, which can be used to derive similarity-based LDA-like techniques, c) *SVM-based target*, which can be used to perform SVM-based analysis (more details are given in [3]), and d) *fixed target*, which can be used to perform similarity-based PCA.

4. Using the PySEF

PySEF is readily available in the Python Package Index (PyPI) and it can be easily installed just by executing the following command (all the dependencies, except of the PyTorch library, will be automatically installed):

```
pip install pysef
```

Then, a linear embedding function can learned using less than 5 lines of code:

```python
import sef_dr
proj=sef_dr.LinearSEF(input_dimensionality=784,
 output_dimensionality=9)
```

Table 1: Using *PySEF* to perform various types of DR

Unsupervised DR:

Method	**Acc.**
PCA(10d)	82.88%
SEF(10d)	**84.87**%

Supervised DR:

Method	**Accuracy**
LDA(9d)	85.66%
SEF(9d)	88.89%
SEF(18d)	**89.48**%

Out-of-sample extensions (ISOMAP):

Method	**Accuracy**
Regression (10d)	85.25%
SEF(10d)	85.76%
SEF(20d)	**89.48**%

```python
proj.fit(data=data, target_labels=labels, target='supervised',
epochs=10, batch_size=128)
transformed_data = proj.transform(data)
```

When the `LinearSEF` object is created the input dimensionality (dimensions of the original space), as well as the output dimensionality (dimensions of the target space) must be provided. Then, the embedding function is learned by calling the `.fit()` method and the data are projected into the learned space using the `.transform()` method. For non-linear projections the `KernelSEF` class can be similarly used. Table 1 summarizes some experimental results using the *PySEF* library and the MNIST dataset (`http://yann.lecun.com/exdb/mnist`). A dataset loader is also provided to easily load any of the six datasets that were originally used for evaluating the proposed technique along with detailed examples that allow for reproducing the results reported in [3]. Please refer to project's documentation `http://pysef.readthedocs.io` for more details.

PySEF can be also easily extended by defining custom similarity targets and/or embedding functions. To define a custom similarity target, a function that adheres to the following signature must be defined:

```python
def custom_similarity_function(target_data, target_labels, sigma,
idx, target_params):
  Gt = np.zeros((len(idx), len(idx)))
  Gt_mask = np.zeros((len(idx), len(idx)))
  # Calculate the similarity target here
  return np.float32(Gt), np.float32(Gt_mask)
```

The defined custom similarity target function must be passed to the `target` argument of the `.fit()` function. During the optimization the custom similarity function is called and the arguments `target_data`, `target_labels`, `sigma` (scaling factor of the similarity function) and `target_params` (optional arguments) are passed to the similarity function. Note that the `target_data` can be an *h5py* array stored in the disk, allowing the method to easily scale to larger datasets. Finally, the defined function must return both the target similarity mask (as calculated between the batch samples), as well as the optimization mask for the corresponding target ($\mathbf{M}$ in Equation 2). An extensive tutorial on how to define custom target functions is provided in the documentation of *PySEF*. New embedding functions can be defined by extending the `SEF_Base` class. The subclass is expected to implement a set of functions (defined in `SEF_Base`). Most reusable functions have been already implemented in `SEF_Base` to reduce code duplication and simplify the implementation.

5. Conclusions

An efficient and modular implementation of the Similarity Embedding Framework (SEF) in Python, called *PySEF*, that allows for easily performing similarity-based dimensionality reduction (DR) as well as defining custom similarity targets and embedding functions was presented in this paper.

Acknowledgements

Nikolaos Passalis was supported by the General Secretariat for Research and Technology (GSRT) and the Hellenic Foundation for Research and Innovation (HFRI) (PhD Scholarship No. 1215).

References

[1] Y. Liu, Dimensionality reduction and main component extraction of mass spectrometry cancer data, Knowledge-Based Systems 26 (2012) 207–215.

[2] C.-X. Yin, Q.-K. Peng, A careful assessment of recommendation algorithms related to dimension reduction techniques, Knowledge-Based Systems 27 (2012) 407–423.

[3] N. Passalis, A. Tefas, Dimensionality reduction using similarity-induced embeddings, IEEE Transactions on Neural Networks and Learning Systems.

[4] Pytorch: Tensors and Dynamic neural networks in Python with strong GPU acceleration, `https://github.com/pytorch/pytorch` (2017).

[5] A. Collette, Python and HDF5, O'Reilly, 2013.

[6] scikit-learn: machine learning in Python, `https://github.com/scikit-learn/scikit-learn` (2017).

Required Metadata

Current executable software version

Nr.	(executable) Software metadata description	Please fill in this column
S1	Current software version	v0.2.9
S2	Permanent link to executables of this version	`https://github.com/passalis/sef/releases/download/v0.2.9/PySEF-0.2.9-py2.py3-none-any.whl`
S3	Legal Software License	MIT License
S4	Computing platform/Operating System	Linux, OS X, Microsoft Windows
S5	Installation requirements & dependencies	Python 2.7 (or Python 3.5), PyTorch, numpy, scikit-learn, scipy
S6	If available, link to user manual - if formally published include a reference to the publication in the reference list	`http://pysef.readthedocs.io`
S7	Support email for questions	`passalis@csd.auth.gr`

Table 2: Software metadata (optional)

Current code version

Nr.	Code metadata description	Please fill in this column
C1	Current code version	v0.2.9
C2	Permanent link to code/repository used of this code version	`https://github.com/passalis/sef`
C3	Legal Code License	MIT License
C4	Code versioning system used	git
C5	Software code languages, tools, and services used	Python 2.7 (or Python 3.5)
C6	Compilation requirements, operating environments & dependencies	PyTorch, numpy, scikit-learn, scipy
C7	If available Link to developer documentation/manual	`http://pysef.readthedocs.io`
C8	Support email for questions	`passalis@csd.auth.gr`

Table 3: Code metadata (mandatory)