# PySEF: A Python Library for Similarity-based Dimensionality Reduction

This package provides an implementation of the Similarity Embedding Framework (SEF) on top of the *theano* library. The implementation exposes a *scikit-learn*-like interface.

## What is the Similarity Embedding Framework?

The vast majority of Dimensionality Reduction techniques rely on second-order statistics to define their optimization objective. Even though this provides adequate results in most cases, it comes with several shortcomings. The methods require carefully designed regularizers and they are usually prone to outliers. The Similarity Embedding Framework can overcome the aforementioned limitations and provides a conceptually simpler way to express optimization targets similar to existing DR techniques. Deriving a new DR technique using the Similarity Embedding Framework becomes simply a matter of choosing an appropriate target similarity matrix. A variety of classical tasks, such as performing supervised dimensionality reduction and providing out-of-of-sample extensions, as well as, new novel techniques, such as providing fast linear embeddings for complex techniques, are demonstrated.

## How to use PySEF?

Simply import sef_dr, create a SEF object, fit it and transform your data!

```
import sef_dr
proj = sef_dr.LinearSEF(input_dimensionality=784, output_dimensionality=9)
proj.fit(data=data, target_labels=data, target='supervised', iters=10)
transformed_data = proj.transform(data)
```

## What PySEF can do?

## 1. Recreate the geometry of a high dimensional space into a space with less dimensions!

In examples/unsupervised_approximation.py we recreate the 20-d PCA using just 10 dimensions:

| Method | Accuracy |
|---|---|
| PCA 10-d | 82.88% |
| Linear SEF mimics PCA-20d | 84.68% |

## 2. Re-derive similarity-based versions of well-known techniques!

In examples/supervised_reduction.py we derive a similarity-based LDA:

| Method | Accuracy |
|---|---|
| LDA 9d | 85.66% |
| Linear SEF 9d | 88.19% |
| Linear SEF 18d | 88.45% |

Note that LDA is limited to 9-d projections for classifications problems with 10 classes.

## 3. Provide out-of-sample extensions!

In examples/linear_outofsample.py and examples/kernel_outofsample.py we use the SEF to provide (linear and kernel) out-of-sample extensions for the ISOMAP technique. Note that the SEF, unlike the regression-based method, is not limited by the number of dimensions of the original technique.

| Method | Accuracy |
|---|---|
| Linear Regression 10d | 85.25% |
| Linear SEF 10d | 85.94% |
| Linear SEF 20d | 89.03% |

| Method | Accuracy |
|---|---|
| Kernel Regression 10d | 89.48% |
| Kernel SEF 10d | 87.74% |
| Kernel SEF 20d | 90.24% |

## 4. Perform SVM-based analysis!

In examples/svm_approximation.py an SVM-based analysis technique that mimics the similarity induced by the hyperplanes of the 1-vs-1 SVMs is used to perform DR. This method allows for using a light-weight classifier, such as the NCC, to perform fast classification.

| Method | Accuracy |
|---|---|
| NCC - Original | 80.84% |
| NCC - Linear SEF 10d | 85.90% |
| NCC - Linear SEF 20d | 86.36% |

## 5. Deriving a new DR method becomes simply a matter of defining a custom similarity target!

The SEF allows for easily deriving novel DR techniques by simply defining the target similarity matrix. For the current implementation this can be done by defining a function that adheres to the following signature:

```
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 *target_data*, *target_labels*, *sigma*, and *target_params* are passed to the *.fit()* function. During the training this function is called with a different set of indices *idx* and it is expected to return the target similarity matrix for the data that corresponds indices defined by *idx*.

For example, let's define a function that sets a target similarity of 0.8 for the samples that belong to the same class, and 0.1 for the samples that belong to different classes:

```
def sim_target_supervised(target_data, target_labels, sigma, idx, target_params):

    cur_labels = target_labels[idx]
    N = cur_labels.shape[0]

    N_labels = len(np.unique(cur_labels))

    Gt, mask = np.zeros((N, N)), np.zeros((N, N))

    for i in range(N):
        for j in range(N):
            if cur_labels[i] == cur_labels[j]:
                Gt[i, j] = 0.8
                mask[i, j] = 1
            else:
                Gt[i, j] = 0.1
                mask[i, j] = 0.8 / (N_labels - 1)

    return np.float32(Gt), np.float32(mask)
```

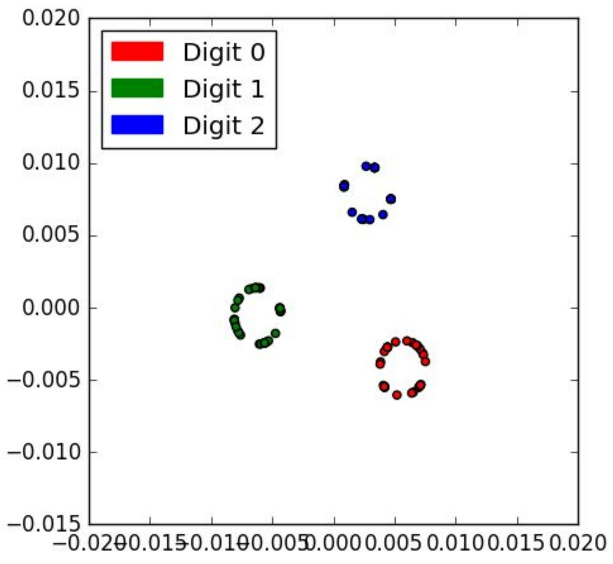
Note that we also appropriately set the weighting mask to account for the imbalance between the intra-class and inter-class samples. It is important to remember to work only with the current batch (using the *idx*) and not the whole training set (that is always passed to *target_data*/*target_labels*). You can find more target function examples in sef_dr/targets.py

The target that we just defined tries to place the samples of the same class close together (but not to collapse them into the same point), as well as to repel samples of different classes (but still maintain as small similarity between them). Of course this problem is ill-posed in the 2-D space (when more than 3 points per class are used), but let's see what happens!

Let's overfit the projection:

```
   proj = KernelSEF(train_data, train_data.shape[0], 2, sigma=1,
learning_rate=0.0001, regularizer_weight=0)
    proj.fit(train_data, target_labels=train_labels, target=sim_target_supervised,
iters=500, verbose=True)
    train_data = proj.transform(train_data)
```

and visualize the results:



Close enough! The samples of the same class have been arranged in circles, while the circles of different classes are almost equidistant to each other!

(The MNIST dataset was used for all the conducted experiments)

## Are there any tutorials?

Yes! Check out the notebooks in the tutorials folder (Python 2.7 only)!

## How to install PySEF?

You can clone the repository and then install the package:

```
git clone https://github.com/passalis/sef
cd sef
python setup.py install --user
```

Currently only *float32* are supported, so set *floatX=float32* (.theanorc).

PySEF is developed and tested on Linux (both Python 2.7 and Python 3 are supported). However, it is expected to run on Windows/Mac OSX as well, since all of its components are cross-platform.

## Issues

If the installed version of *lasagne* does not match the installed *theano* version you may receive the following error: "ImportError: cannot import name 'downsample'" (*lasagne* does not frequently freeze the versions, so it is difficult to specify the version that you should install ). Installing the bleeding edge versions of both *lasagna* and *theano* should resolve the issue:

```
pip install --upgrade https://github.com/Theano/Theano/archive/master.zip
pip install --upgrade https://github.com/Lasagne/Lasagne/archive/master.zip
```

## Further reading

You can find more details about the Similarity Embedding Framework in our paper.

If you use PySEF for research, please cite the following paper:

```
@article{passalis2017dimensionality,
  title={Dimensionality Reduction using Similarity-induced Embeddings},
  author={Passalis, Nikolaos and Tefas, Anastasios},
  journal={IEEE Transactions on Neural Networks and Learning Systems},
  year={2017},
  volume={(to appear)}
}
```