Machine Learning Group
Fakulty IV, Technische Universität Berlin
Prof. Dr. Klaus-Robert Müller
Email: klaus-robert.mueller@tu-berlin.de

Problem Set 1: PCA, LLE, Outlier Detection

On the ISIS page you can download **code stubs**, **datasets** and a **test script** which is designed to help you debug your code. It contains test functions for each of the implementation assignments in Part 1. Make sure that your code passes all tests. Be aware that

- a passed test **does not guarantee correctness** for all possible inputs,
- if the test module produces a plot, you have to **check if the plot looks correct**, and
- your **code has to be efficient**, e.g. you can not use for-loops when matrix vector multiplications would give you a significant speed-up.

You have to submit **exactly two files**:

1. a **report** `sheet1.pdf` with your analysis (use the LaTeX template from ISIS, max. 20 pages) and
2. your **implementation** `sheet1.py`.

Part 1: Implementation

Assignment 1 (10 pts)

Write the class `PCA` with signature

```
class PCA():
    def __init__(self, Xtrain):
        # ...
        self.C = ...
        self.U = ...
        self.D = ...
    def project(self, Xtest, m):
        # ...
        return Z
    def denoise(self, Xtest, m):
        # ...
        return Y
```

which receives a $n \times d$ matrix `X` and computes the principle components `U` and `D`:

- `U` is a $d \times d$ matrix, which contains the principal directions (one direction per column), and
- `D` is a vector of length d, which contains the principal values sorted in descending order (i.e. $\mathtt{D}_1 \geq \mathtt{D}_2 \ldots$). The principle values should represent the sample variance along the principle component directions.

The `project` method returns the projected data points in a $n \times m$ matrix `Z` and the `denoise` method returns the denoised data in a $n \times d$ matrix `Y`.

Assignment 2 (5 pts)

Implement the γ-index (see Harmeling et al., 2006, on ISIS),

```
def gammaidx(X, k):
    # ...
    return y
```

which receives a $n \times d$ matrix `X` containing the data points and the number of neighbors k, and returns the γ-index for each datapoint in the vector `y` of length n.

Assignment 3 (10 pts)

Implement the AUC metric

```
def auc(y_true, y_pred, plot=False):
    # ...
    return c
```

that takes as input the true labels y_true $\in \{-1,+1\}^n$, and for each point a predicted value in y_pred where lower values tend to correspond to label -1 and higher values to label $+1$, and returns the AUC (area under the receiver operating curve). For example, y_pred could be the output of a learning algorithm (binary logistic regression, SVM before sign operation, …). If plot=True, the method should also plot the ROC curve.

Assignment 4 (15 pts)

Write a function lle with signature

```
def lle(X, m, tol, n_rule, k=None, epsilon=None):
    # ...
    return Y
```

which receives a $n \times d$ matrix X containing the data points, and which calculates an m-dimensional embedding Y $\in \mathbb{R}^{n\times m}$ using the LLE algorithm (Saul and Roweis, 2000, on ISIS). The parameter n_rule determines the method ('knn' or 'eps-ball') which is used to build the neighborhood graph where k is the number of neighbors for kNN and epsilon is the radius of the epsilon ball. tol is the regularization parameter.

Your function should be robust against malicious parameters. Use raise ValueError() for error reporting. In particular, you should check whether the resulting graph is connected.

Part 2: Application

Assignment 5 (15 pts)

In this assignment, we will analyze the usps dataset (available on ISIS) with PCA in the following manner:

1. Load the usps data set.
2. Analysis of PCA:
   (a) Calculate the PCA of this data.
   (b) Visualize (a) all principal values, (b) the largest 25 principal values both as a bar plots (see matplotlib.pyplot.bar) and (c) the first 5 principal directions as images (see matplotlib.pyplot.imshow).
3. Consider three noise scenarios (use numpy.random.randn):
   - low gaussian noise: add Gaussian noise to the images. Select an appropriate variance such that the resulting images are *moderately* noisy.
   - high gaussian noise: add Gaussian noise to the images. Select an appropriate variance such that the resulting images are *very* noisy.
   - outliers: add very strong Gaussian noise to *only five* images. Select an appropriate variance such that those five images are *extremely* noisy (beyond human recognition!) and that the noise is clearly reflected in the spectrum (plot of eigenvalues).

   For each of the scenarios

   (a) Calculate the PCA of this data and redo the plots of principal values. Explain the differences to the original spectrum.

   (b) Denoise the images by reconstruction from projections on the m largest principal components: the reconstruction $\widehat{x}$ of a data point x by the m largest eigenvectors $\mathbf{u}_1,\ldots,\mathbf{u}_m$ of the covariance matrix is given by

   $$\widehat{x} = \mu + \sum_{i=1}^{m} \mathbf{u}_i(x^\top \mathbf{u}_i),$$

   where μ is the mean of the original data set. The reconstruction error of x is $\|x - \widehat{x}\|$.
   The number of components m should be tuned by hand such that it is low but the reconstructed (denoised) images are fairly good. Can the spectrum be used to determine a good value for m?

   (c) For 10 examples of your choice (including the noisy ones in the outliers setting), plot the original image, the noisy image and its denoised reconstruction (using imshow).

Remark: use subplot, gridspec or axes from the matplotlib library to arrange multiple plots/images in a single figure.

Assignment 6 (15 pts)

In this exercise, we use the positive class of the banana dataset (available on ISIS) as "inliers" to which we add outliers generated from a random process. We will investigate the performance of outlier detection algorithms for outlier contamination rates (i.e. percentage of outliers in the data set) of 1%, 10%, 50% and 100% relative to the positive class. For each of these rates repeat the following procedure 100 times:

data set	file name	dim	data	reference
fishbowl	`fishbowl_dense.npz`	3	`X.T`	`X.T[:, 2]`
swissroll	`swissroll_data.npz`	3	`x_noisefree.T`	`z.T[:, 0]`
flatroll	`flatroll_data.npz`	2	`Xflat.T`	`true_embedding`

Table 1: the data sets to be embedded (available on ISIS)

1. Sample a random set of outliers uniformly from the box that contains the data, $x_{\text{out}} \sim \begin{bmatrix} \mathcal{U}(-4,4) \\ \mathcal{U}(-4,4) \end{bmatrix}$ of the respective size (depending on the contamination rate).
2. Add the outliers to the positive class, and compute (a) the γ-index with $k = 3$, (b) the γ-index with $k = 10$ and (c) the distance to the mean for each data point.
3. Compute the AUC (area under the ROC) for each method.

For each contamination rate and method we thus obtain 100 AUC values.

Compare the performance of all methods by using a boxplot or violine plot to visualize the distribution of the 100 AUC values for each contamination rate / method combination. Also show the results from one exemplary run: plot the dataset three times (for the three conditions (a), (b) and (c)) with a contamination rate of your choice, show original data and the sampled outliers in two different colors and scale the data points by their outlier scores.

Assignment 7 (15 pts)

Apply LLE to the data sets fishbowl, swissroll and flatroll (see Table 1), using appropriate values for the hyperparameters. LLE is sensitive with respect to the parameters. You have to fine-tune `n_rule`, `epsilon`, `k`, and `tol`.

For 3D datasets, plot (1) the dataset in 3D and (2) a 2D embedding (found by LLE). Color the datapoints according to the reference value. For 2D datasets, plot (1) the dataset in 2D and (2) the reference values versus a 1D embedding (found by LLE) in a scatter plot. How can we find *good* hyperparameters? How would you asses the quality of an embedding when no ground truth reference was available?

Assignment 8 (15 pts)

In this exercise, we study the influence of noise on LLE, using the example of the `flatroll` data set. Do the following:

1. Load the data set.
2. Add Gaussian noise with variance 0.2 and 1.8 to the data set (this results in 2 noisy data sets).
3. Apply LLE on both data sets, where the neighborhood graph should be constructed using kNN. For both noise levels, try to find (a) a good value for k which unrolls the flat roll and (b) a value which is obviously too large.
4. For each of the four combinations of low/high noise level good/too large k, plot the neighborhood graph and the resulting embedding. The neighborhood graph shows the connections between neighbors in the original scatter plot.

References

S. Harmeling, G. Dornhege, D. M. J. Tax, F. C. Meinecke, and K.-R. Müller. From outliers to prototypes: Ordering data. *Neurocomputing*, 69(13-15):1608–1618, 2006. URL `https://doi.org/10.1016/j.neucom.2005.05.015`.

L. K. Saul and S. T. Roweis. An introduction to locally linear embedding. *unpublished*, 2000. URL `http://www.cs.toronto.edu/~roweis/lle/publications.html`.