Unsupervised Learning: Dimensionality reduction

AAA-Python Edition

Plan

- 1- Dimensionality reduction
- 2- Some Math
- 3- PCA
- 4- PCA in scikit-learn
- 5- Manifold Learning
- 6- Manifold Examples

1- Dimensionality reduction

Objective

- **Dimensionality reduction** in **machine learning** is reducing the **number** of **features** of the training dataset.

- This reduction is necessary to:
  - Eliminate the **noise** from the data
  - Visualize the data in **2** or **3** dimensions
  - Speed up the learning process
  - Enhance the learning results by eliminating correlated features.
  - Eliminate unnecessary features.
  - Compress the data size.

- Two main approaches to dimensionality redcution are:
  - **Projection :** project the data into a lower dimensional space.
  - **Manifold:** suppose that the data in the higher dimension is just a manifold of a representation of the data in the lower dimension.

Projection

- Sometimes the degree of the variation of the data is different from one dimension to an other. So, for some features, the values can be very diverse, an for others, they can barely change.
- So we project the data into a lower dimension in order to keep only the most influential **information** ==> we define a mapping between the original data from the higher dimension to new data in a lower dimension.
- The most used technique to define this mapping, is **PCA** (**P**rincipal **C**omponent **A**nalysis) and its variations:
  - **Incremental** PCA
  - **Randomized** PCA
  - **Kernel** PCA

[By Amina Delali]

Manifold

1- Dimensionality reduction

- Like we said earlier, we make the hypothesis that our data is created from a **manifold** of a data in a **lower dimension**. So, reducing it to this low dimension is like **straightening up** this manifold (or **unrolling it**).

- The different techniques used, are:

  - **MDS**: Multidimensional Scaling. Tries to preserve the distances between instances.
  - **LLE**: Locally Linear Embedding. Tries to preserve the relationship between a sample and its closets points.
  - **Isomap:** the samples will represent nodes of a graph. These nodes are connected to their closets neighbors. The algorithm tries to preserve the number of nodes in the shortest path connecting two nodes.

[By Amina Delali]

Singular value decomposition

2- Some Math

AIM

- It is the the decomposition of a matrix **M** $_{(m,n)}$ into **3** matrices: **U**$_{(m,m)}$, **S**$_{(m,n)}$, and **V**$_{(n,n)}$. Considering only **real** values, we have the following characteristics:
  - $M = U . S . V^T$ (V^T is the transpose matrix of V : value at i,j becomes at j,i)
  - $U . U^T = U^T . U = I_{(m,m)}$ (the identity matrix)
  - $V . V^T = V^T . V^T = I_{(n,n)}$
  - The diagonal (values with the same row and column indices) of S are the **Singular values** of M
    - Singular values are the square roots of **eigenvalues**
    - The other values of S are **zeros.**
  - The columns of **U** are the **eigenvectors** of $\mathbf{M . M^T}$.
  - The columns of **V** are the **eigenvectors** of $\mathbf{M^T . M}$.

[By Amina Delali]

Eigenvectors, Eigenvalues

- Given **A** $_{(n,n)}$ a square matrix:
  - If A . $V_{(n)} = \lambda$. $V_{(n)}$ then: V is an **eigenvector** and λ is its corresponding **eigenvalue.**
  - The above equation can be rewritten as follow: (A- λI). V= 0
  - Several λ can solve the equation. For each λ lambda value, an eigenvector is computed.

- Example:
  - If $A = \begin{bmatrix} 2 & 1 \\ 1 & 2 \end{bmatrix}$
    - Its eigenvalues will be: 1 , 3
    - And their corresponding eigenvectors will be: $\begin{bmatrix} 1 \\ -1 \end{bmatrix}$ and $\begin{bmatrix} 1 \\ 1 \end{bmatrix}$

[By Amina Delali]

Standard Deviation

- The standard deviation σ measures how data is spread (or distant from the mean) . It is the **square root** of the **variance.**

- The variance is computed as follow:
  - $$variance = \frac{\sum_{i=1}^{N} (x_i - \mu)^2}{N}$$
- And the standard deviation:

$$\sigma = \sqrt{variance}$$

- To project data on new axis, we select the axis that preserve the maximum possible variance of the data. This way, most of the information is preserved.

3- PCA in scikit-learn

Definition

- It is a linear dimensionality reduction technique that project data using orthogonal axes (components) that preserve the maximum variance possible. One of the method used is singular value decomposition of the mean centered training data.

- As stated before the decomposition leads to **3** matrices. The vectors of the matrix V^T will be used to project the data. They are the “principal components”.

- Each component will conserve a certain amount of variance. The variance obtained after projection is the accumulation of the variances obtained by each component

- To project, we select a sufficient number of component to preserve the maximum of variance, then we apply the transformation (the projection), using only this number of vectors.
- The number of vectors will determine the dimension of the projection.

Example

```
# import necessary libraries
import numpy as np
from sklearn.datasets import load_iris
import sklearn.preprocessing as preprocess
```

```
# loading the data ( 4 feautres ==> 4 dimensions)
myIris = load_iris()
X= myIris.data
y=myIris.target
```

```
from numpy import mean
# centring the data
X_centred = preprocess.scale(X,with_std=False, axis=0)

X_centred = X - X.mean(axis=0)
# extracting the principal component

U, s, V = np.linalg.svd(X_centred)
# extracting  the 3 first principal commponets
c1 = V.T[:,0]# V.T is the transopose of V
c2 = V.T[:,1]
c3 = V.T[:,2]
```

Center the data to the mean, before applying the decomposition

The decomposition

```
# compute the projection in a 3D dimension
C3 = V.T[:, :3]
X3d = X_centred.dot(C3)
```

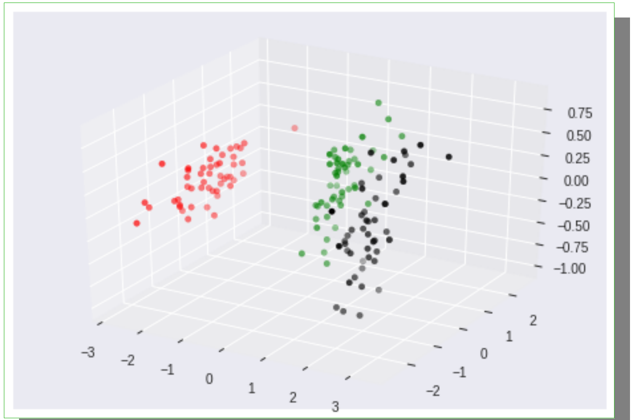
To project, we multiply the centered data by the first selected component==> we will have a 3 dimensions projection

[By Amina Delali]

3- PCA in scikit-learn

Results

3D projection



2D projection

Since our data was originally labeled (we don’t use those label for decomposition), we used them for colorizing the data.
And what is obvious, is that the data is clustered according to its classes.
Which proofs:

- that the clustering can in certain cases classify data.
- the decomposition preserved the most important amount of information.

[By Amina Delali]

4- Processing Data

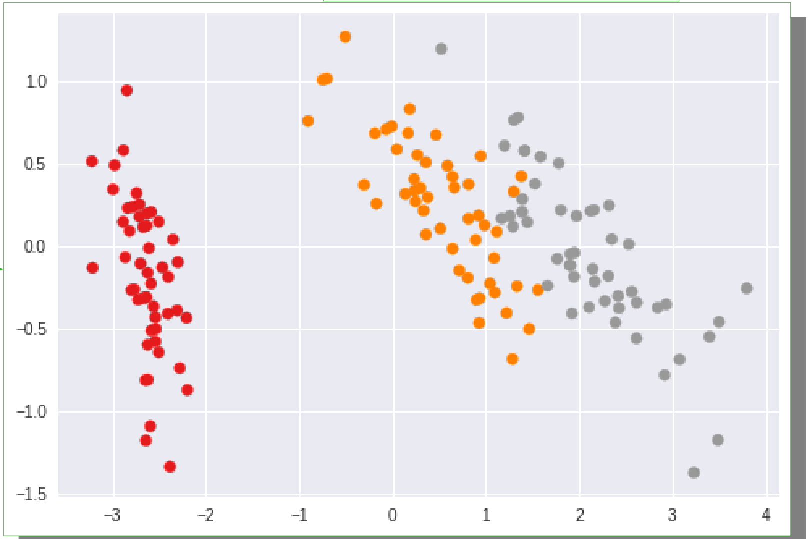
With matplotlib

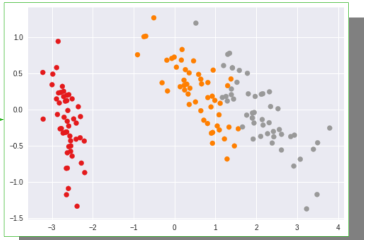
```python
# we are using matplotlib version 2.1.2
# it will be removed in version 3.1
from matplotlib.mlab import PCA as PCA2
my2PCA = PCA2(X, standardize=False)
results = my2PCA.project(X, minfrac=0.02)
fig = plt.figure()
plt.scatter(results[:,0],results[:,1],c=y, cmap = plt.cm.Set1)
```

It tells to only center the data, and to not standardize

It will drop all the axis with variance ratio **< minfrac.** In this case, it will only keep 2 axis.

Same results as in our previous implementation





[By Amina Delali]

4- Processing Data

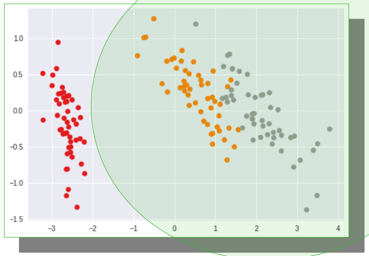
With sklearn

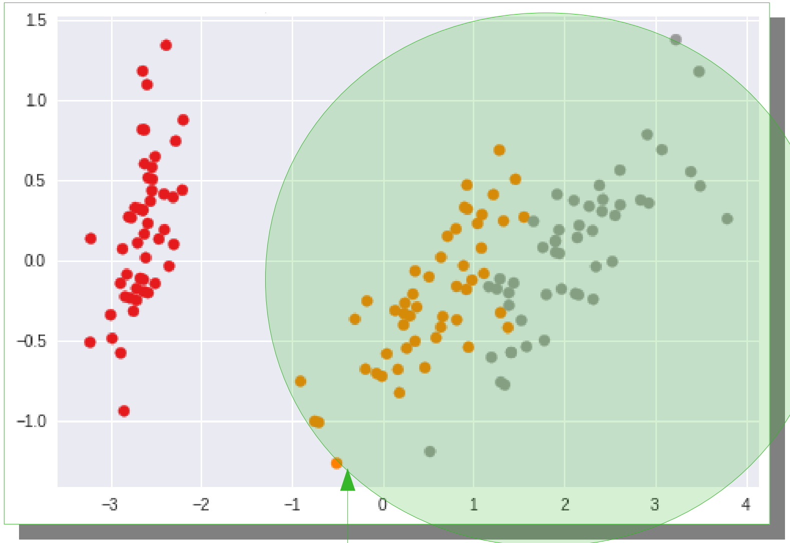
```
from sklearn.decomposition import PCA

fig = plt.figure()
pca = PCA(n_components = 2)
pca.fit(X)
X2d_2= pca.transform(X)


plt.scatter(X2d_2[:,0],X2d_2[:,1],c=y, cmap = plt.cm.Set1)
```

We have to select the number of components before transforming the data

As in matplotlib, we don't have to center the data

The reason of this inversion is that **sklearn flip** the **eigenvector's sign** before the projection : it apply the method **svd_flip** on the vectors U and V in the fitting methods





Comparing with matplotlib we see that the directions are inverted

[By Amina Delali]

4- Processing Data

Explained variance ratio

- The **correct number** of **components** can be defined by the **explained variance ratio** of each component.
- It is computed by the value of **explained variance** divided by the **sum of all variances.**
- The ratio of each component are **summed up** until a certain percentage is obtained.
- The **variances** can be computed from the **square** of the **singular values** in **S**

```
# the explained variance ratio (our implementation, 2D)

explained_ratio_2FirstC = (np.square(s[0])+ np.square(s[1]))/np.sum(np.square(s))
```

```
# explained variance ratio (with  matplotlib, 2D)
EVR2= my2PCA.fracs[0] + my2PCA.fracs[1]
```

```
# explained variance ratio (with sklearn, 2D)
EVR2= np.sum(pca.explained_variance_ratio_)
```

```
0.977685206318795
```


[By Amina Delali]

Algorithm

- **LLE** for **L**ocally **L**inear **E**mbeeding. The algorithm consist of **3** major steps:
- **Step 1 - identifying the neighbors for each sample x_i from the data $X_{(N,D)}$**(for N samples and D features) **:**
  - Compute the distances of the other samples from x_i
  - Select the **k** smallest distances.
- **Step 2 - for each sample x_i compute its neighbors weights:**
  - Create the matrix $Z_{(k,D)}$ with the k samples rows from $X_{(N,D)}$ corresponding to the neighbors of x_i
  - Subtract x_i values from each row of $Z_{(k,D)}$
  - Compute $C_{(k,k)} = Z_{(k,D)} . Z^T_{(D,k)}$(in the original page it is inverted because of X and Z are transposed)
  - Compute the row **i** of the matrix $\mathbf{W}_{(N,N)}$ with:
    - Compute the weights in the one column vector $\mathbf{w}_{(k,1)}$ that solve the equation $C_{(k,k)} . w_{(k,1)} = 1_{(k,1)}$ (1 is a column vector with only **1** as values)

5- Manifold Learning: LLE

[By Amina Delali]

Algorithm (Suite)

- For the samples **j** that do not belong to each $x_{i,}$ neighbors, set the weights to **0**.
- For each neighbor **b** of x_i set the weight to: **w(p) /sum($w_{(k,1)}$).** Where **p** is the indices in **w** corresponding to the **b** neighbor of $x_{i.}$

- **Step 3 – reduce the dimensionality to d < D in a new matrix $Y_{(N,d)}$:**
  - Compute the matrix $\mathbf{M}_{(N,N)} = (I_{(N,N)} - W_{(N,N)})^T . (I_{(N,N)} - W_{(N,N)})$
  - Select the **d+1** eigenvectors of $M_{(N,N)}$ corresponding to the d+1 smallest eigenvalues. Order these eigenvectors according to the corresponding eigenvalues sorted in a **decreasing order**.
  - For each **column q** in Y set the values equal to the values of the q+1 smallest eigenvector counting from the bottom (to discard the last eigenvector corresponding to the eigenvalue 0)

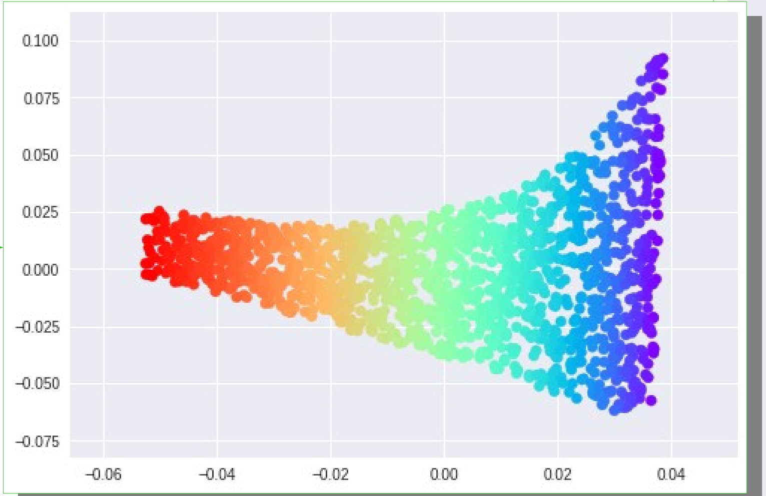
[By Amina Delali]

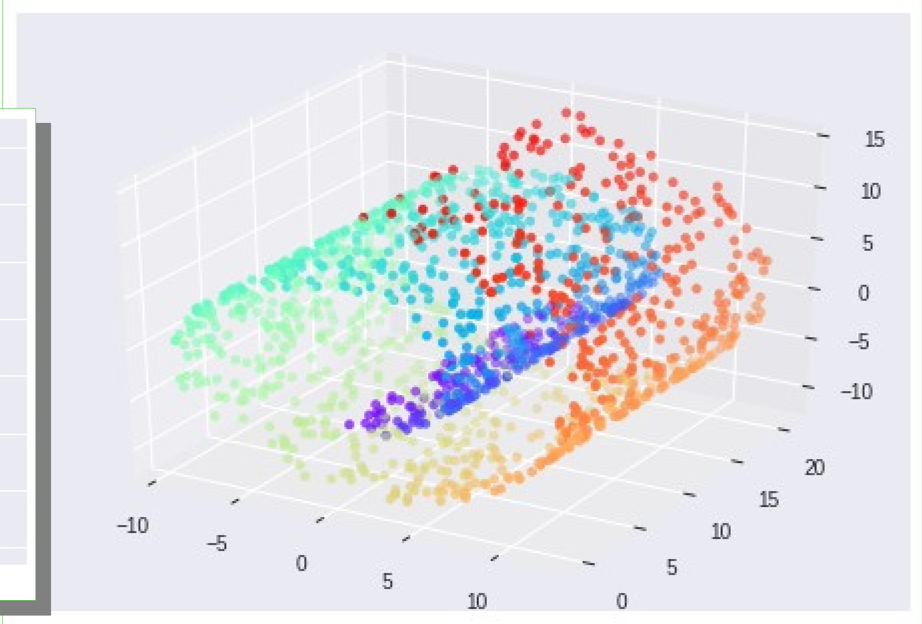
5- Manifold Learning: LLE

Example

```
from sklearn.datasets import make_swiss_roll
#from sklearn.datasets.samples_generator import make_swiss_roll
X_swiss, color = make_swiss_roll(n_samples=1500)
```

N== 1500, D == 3






```
from sklearn import manifold
X_r, err = manifold.locally_linear_embedding(X_swiss, n_neighbors=12,
                                             n_components= 2)
```

LLE : k == 12, d == 2

[By Amina Delali]

Algorithm

- There are two types of Multidimensional Scaling: classical (or metric) that tries to reproduce the original distances. The second one is non-metric (**NMDS**) that tries to reproduces only the rank of the distances.
- We will describe the algorithm of the classical method using the euclidean distance:
  - Compute the distances between all points, and form a matrix of those distances in a matrix **D**.
  - Compute the matrix **A** as follow: $A(i,j) = -1/2 * D(i,j)^2$
  - Compute the matrix **B** as follow: $B(i,j)= A(i,j)- A(i,.) - A(.,j) +A(.,.)$

    where: $A(i,.)$ is the average of all $A(i,j)$ for a selected i

    $A(.,j)$ is the average of all $A(.,j)$ for a selected j

    $A(.,.)$ is the average of all values of A
  - Find the **p** (the **new dimension, lesser** than the original dimension) largest eigenvalues of B: $\lambda_1>\lambda_2>...>\lambda_p$ and their corresponding normalized eigenvectors $L_1, L_2, ..., L_p$ so that $L_i^T . L_i = \lambda_i$

[By Amina Delali]

Algorithm (suite)

- Form the matrix **L** as follow: $L = (L_1, L_2, \ldots, L_p)$. The new values (coordinates) are the **rows** of L.

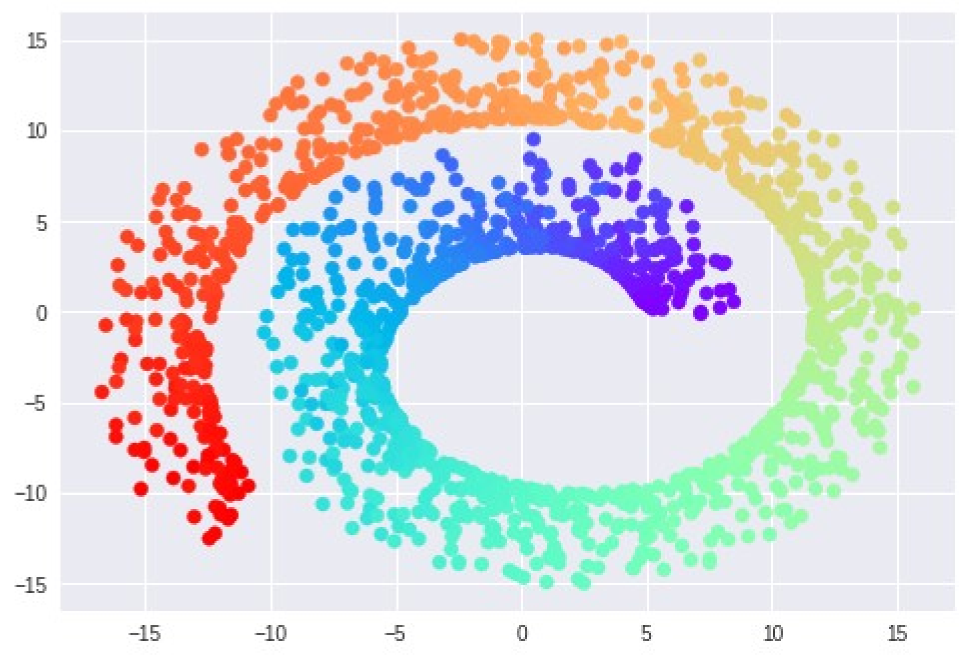
- This method minimizes the value of the **Stress**
- The **stress** is a measure that can be used to find the optimal lower dimension. It is computed as follow:
  - stress = $\sqrt{\dfrac{\sum_{i<j} \left(D(i,j) - \Delta(i,j)\right)^2}{\sum_{i<j} D(i,j)^2}}$
  - where: Δ is the matrix of the distances of the new matrix L
- A stress with a value < 0.05 is acceptable, below 0.01 is considered to be good.

6-Polynomial Regression

Example in Scikit-learn

```
from sklearn.manifold import MDS
embedding = MDS(n_components=2)
X_transformed_mds = embedding.fit_transform(X_swiss)
```

The results are completely different from the previous manifold technique. We see here, the goal is to keep the same original distances values as much as possible.



[By Amina Delali]

References

- Aurélien Géron. Hands-on machine learning with Scikit-Learn and Tensor-Flow: concepts, tools, and techniques to build intelligent systems. O'Reilly Media, Inc, 2017.
- J. D. Hunter. Matplotlib: A 2d graphics environment. Computing In Science & Engineering, 9(3):90–95, 2007.
- NCSS Statistical Software. Multidimensional Scaling, ncss, llc edition.
- Scikit-learn.org. scikit-learn, machine learning in python. On-line at https://scikit-learn.org/stable/. Accessed on 03-11-2018.
- Jake VanderPlas. Python data science handbook: essential tools for working with data. O'Reilly Media, Inc, 2017.
- web.mit.edu. Singular value decomposition (svd) tutorial. On-line at https://web.mit.edu/be.400/www/SVD/Singular_Value_Decomposition.htm. Accessed on 28-12-2018.
- wikipedia.org. Wikipedia, the free encyclopedia. On-line at https://www.wikipedia.org/. Accessed on 25-12-2018.

Thank you!

FOR ALL YOUR TIME