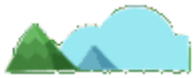
11 種數據降維算法，代碼已開源！

算法與數學之美 昨天


網上關於各種降維算法的資料參差不齊，同時大部分不提供源代碼。這裡有個GitHub 項目整理了使用Python 實現了11 種經典的數據抽取（數據降維）算法，包括：PCA、LDA、MDS、LLE、TSNE 等，並附有相關資料、展示效果；非常適合機器學習初學者和剛剛入坑數據挖掘的小伙伴。

> > > >

01 為什麼要進行數據降維？

所謂降維，即用一組個數為d 的向量Zi 來代表個數為D 的向量Xi 所包含的有用信息，其中d<D；通俗來講，即將高維度下降至低維度；將高維數據下降為低維數據。

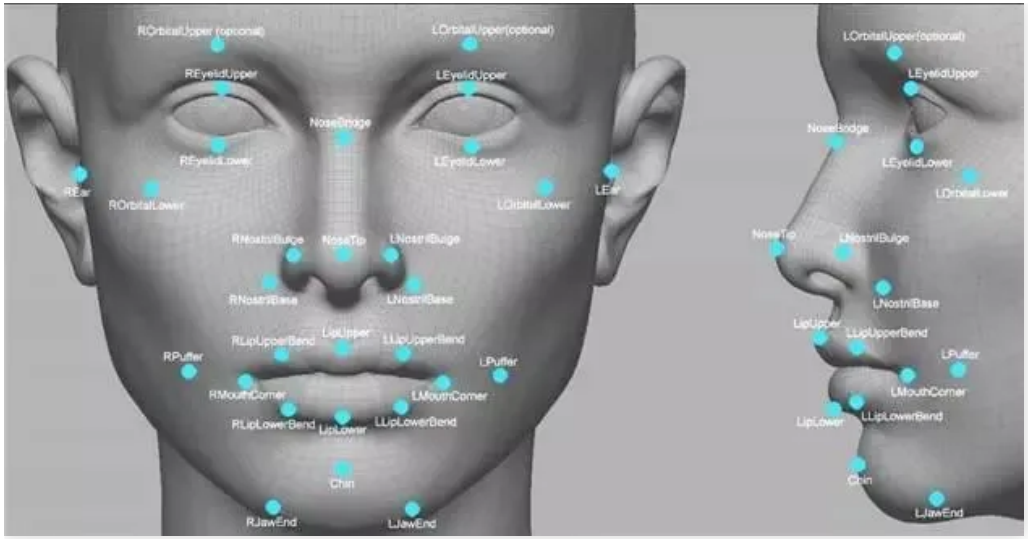
通常，我們會發現大部分數據集的維度都會高達成百乃至上千，而經典的MNIST，其維度都是64。

MNIST 手寫數字數據集

但在實際應用中，我們所用到的有用信息卻並不需要那麼高的維度，而且每增加一維所需的樣本個數呈指數級增長，這可能會直接帶來極大的「維數災難」；而數據降維就可以實現：

- 使得數據集更易使用
- 確保變量之間彼此獨立
- 降低算法計算運算成本
- 去除噪音

一旦我們能夠正確處理這些信息，正確有效地進行降維，這將大大有助於減少計算量，進而提高機器運作效率。而數據降維，也常應用於文本處理、人臉識別、圖片識別、自然語言處理等領域。



02 數據降維原理

往往高維空間的數據會出現分佈稀疏的情況，所以在降維處理的過程中，我們通常會做一些數據刪減，這些數據包括了冗餘的數據、無效信息、重複表達內容等。

例如：現有一張1024*1024 的圖，除去中心50*50 的區域其它位置均為零值，這些為零的信息就可以歸為無用信息；而對於對稱圖形而言，對稱部分的信息則可以歸為重複信息。

因此，大部分經典降維技術也是基於這一內容而展開，其中降維方法又分為線性和非線性降維，非線性降維又分為基於核函數和基於特徵值的方法。

- **線性降維方法：**

  PCA 、ICA LDA、LFA、LPP(LE 的線性表示)

- **非線性降維方法：**

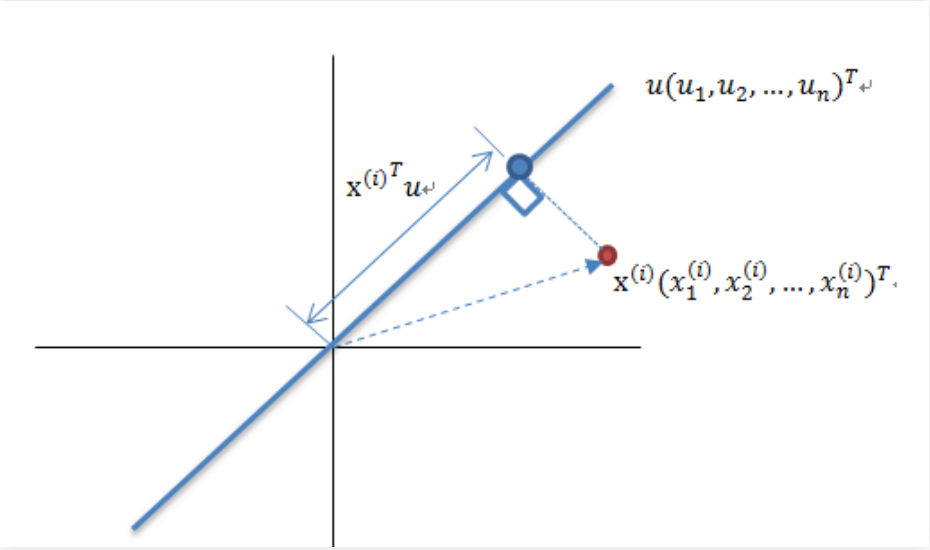
  基於核函數的非線性降維方法——KPCA 、KICA、KDA

  基於特徵值的非線性降維方法（流型學習）——ISOMAP、LLE、LE、LPP、LTSA、MVU

哈爾濱工業大學計算機技術專業的在讀碩士生Heucoder 則整理了PCA、KPCA、LDA、MDS、ISOMAP、LLE、TSNE、AutoEncoder、FastICA、SVD、LE、LPP 共12 種經典的降維算法，並提供了相關資料、代碼以及展示，下面將主要以PCA 算法為例介紹降維算法具體操作。

03 主成分分析（PCA）降維算法

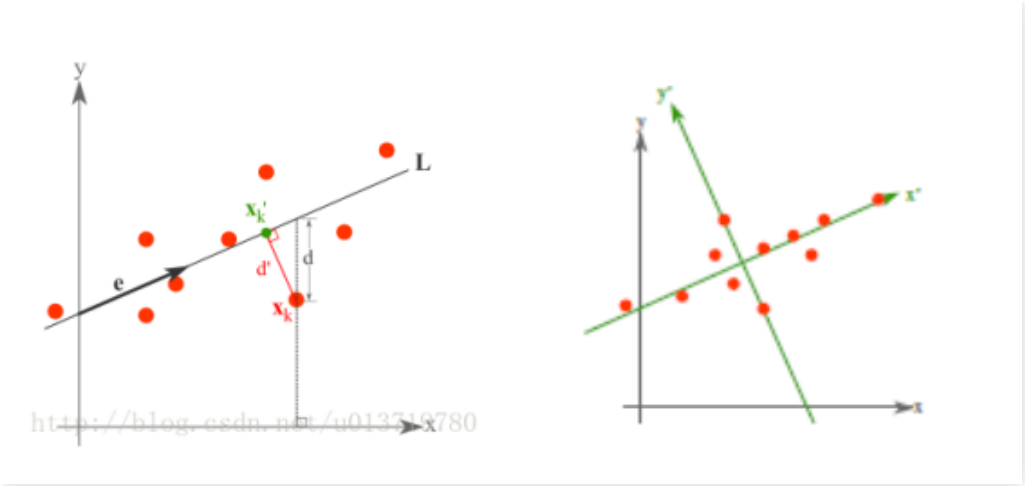
PCA 是一種基於從高維空間映射到低維空間的映射方法，也是最基礎的無監督降維算法，其目標是向數據變化最大的方向投影，或者說向重構誤差最小化的方向投影。它由Karl Pearson 在1901 年提出，屬於線性降維方法。與PCA 相關的原理通常被稱為最大方差理論或最小誤差理論。這兩者目標一致，但過程側重點則不同。



最大方差理論降維原理

將一組N 維向量降為K 維（K 大於0，小於N），其目標是選擇K 個單位正交基，各字段兩兩間COV(X,Y) 為0，而字段的方差則盡可能大。因此，最大方差即使得投影數據的方差被最大化，在這過程中，我們需要找到數據集Xmxn 的最佳的投影空間Wnxk、協方差矩陣等，其算法流程為：

- 算法輸入：數據集Xmxn；
- 按列計算數據集X 的均值Xmean，然後令Xnew=X−Xmean；
- 求解矩陣Xnew 的協方差矩陣，並將其記為Cov；
- 計算協方差矩陣COv 的特徵值和相應的特徵向量；
- 將特徵值按照從大到小的排序，選擇其中最大的k 個，然後將其對應的k 個特徵向量分別作為列向量組成特徵向量矩陣Wnxk；
- 計算XnewW，即將數據集Xnew 投影到選取的特徵向量上，這樣就得到了我們需要的已經降維的數據集XnewW。

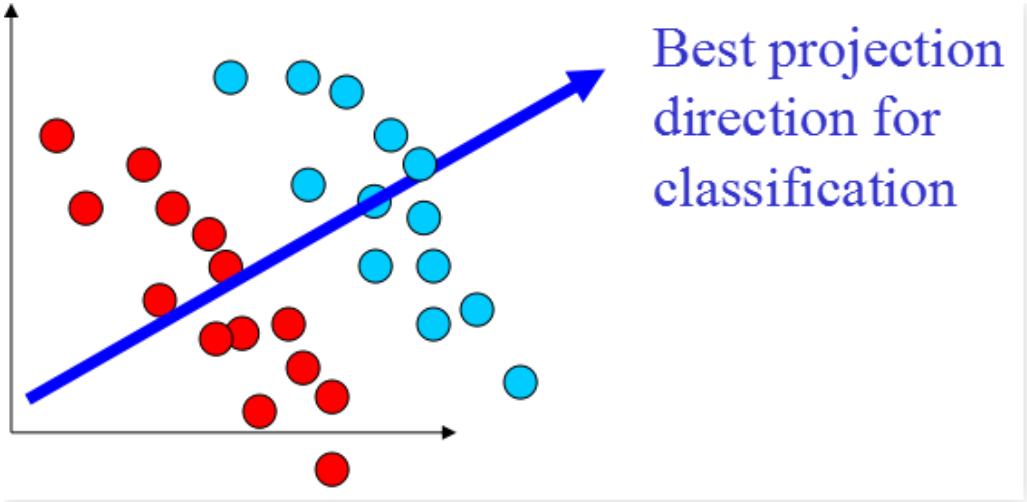


最小誤差理論降維原理

而最小誤差則是使得平均投影代價最小的線性投影，這一過程中，我們則需要找到的是平方錯誤評價函數J0(x0) 等參數。

> 詳細步驟可參考《從零開始實現主成分分析(PCA) 算法》：
>
> https://blog.csdn.net/u013719780/article/details/78352262

04 主成分分析（PCA）代碼實現



關於PCA 算法的代碼如下：

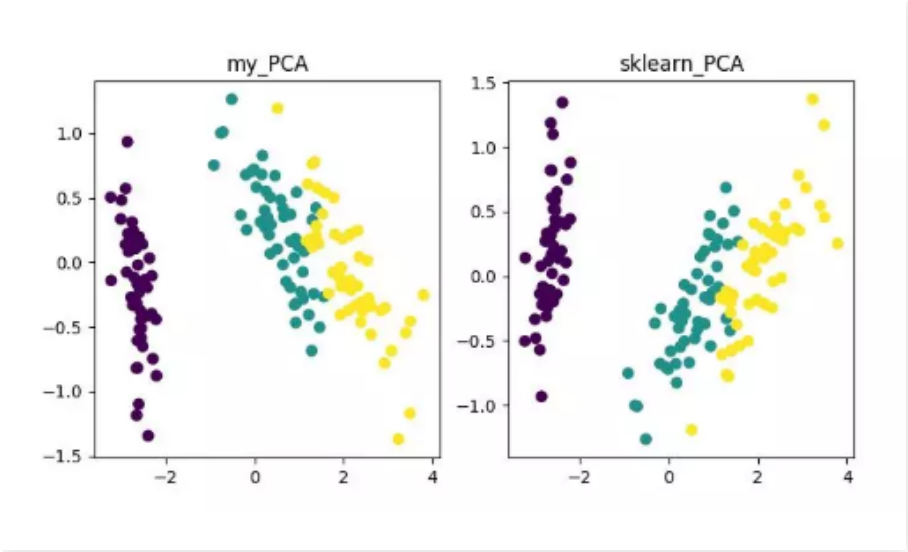
```
from __future__ import print_function
from sklearn import datasets
import matplotlib.pyplot as plt
import matplotlib.cm as cmx
import matplotlib.colors as colors
import numpy as np
%matplotlib inline

def  shuffle_data (X, y, seed=None) :
   if seed:
     np. random.seed(seed)

   idx = np.arange(X.shape[ 0 ])
   np.random.shuffle(idx)

   return X[idx], y[idx]

#正規化數據集X
def  normalize (X, axis=- 1 , p= 2 ) :
    lp_norm = np.atleast_1d(np.linalg.norm(X, p, axis))
   lp_norm[lp_norm == 0 ] = 1
   return X / np.expand_dims( lp_norm, axis)
```

```
 #標準化數據集X
def  standardize (X) :
    X_std = np.zeros(X.shape)
   mean = X.mean(axis= 0 )
   std = X.std(axis= 0 )

    #做除法運算時請永遠記住分母不能等於0的情形
   # X_std = (X - X.mean(axis=0)) / X.std(axis=0)
   for col in range(np.shape(X)[ 1 ]):
      if std[col]:
       X_std[:, col] = (X_std[:, col] - mean[col]) / std[col]
   return X_std
#劃分數據集為訓練集和測試集
def  train_test_split (X, y, test_size= 0.2 , shuffle= True, seed=None) :
   if shuffle:
     X, y = shuffle_data(X, y, seed)
   n_train_samples = int(X.shape[ 0 ] * ( 1 -test_size))
   x_train, x_test = X[:n_train_samples], X [n_train_samples:]
   y_train, y_test = y[:n_train_samples], y[n_train_samples:]

   return x_train, x_test, y_train, y_test

#計算矩陣X的協方差矩陣
def  calculate_covariance_matrix (X, Y=np.empty ( ( 0, 0 ) ) ):
   if notY.any():
      Y = X
   n_samples = np.shape(X)[0]
   covariance_matrix = (1/ (n_samples-1)) * (X - X.mean(axis=0)).T.dot(Y - Y.mean(axis=0))
   returnnp.array(covariance_matrix, dtype=float)
#計算數據集X每列的方差
def calculate_variance(X):
   n_samples = np.shape( X)[0]
   variance = (1/ n_samples) * np.diag((X - X.mean(axis=0)).T.dot(X - X.mean(axis=0)))
   return variance
#計算數據集X每列的標準差
def  calculate_std_dev (X) :
    std_dev = np.sqrt(calculate_variance(X))
   return std_dev

#計算相關係數矩陣
def  calculate_correlation_matrix (X, Y=np.empty ( [ 0 ]) ) :
   #先計算協方差矩陣
```

```
    covariance_matrix = calculate_covariance_matrix(X, Y)
    #計算X, Y的標準差
    std_dev_X = np.expand_dims(calculate_std_dev(X), 1 )
    std_dev_y = np.expand_dims(calculate_std_dev(Y ), 1 )
    correlation_matrix = np.divide(covariance_matrix, std_dev_X.dot(std_dev_y.T))

    returnnp.array(correlation_matrix, dtype=float)

class  PCA() :
    """
    主成份分析算法PCA，非監督學習算法.
    """
    def  __init__ (self) :
        self.eigen_values = None
        self.eigen_vectors = None
        self. k = 2

    def  transform (self, X) :
      """
      將原始數據集X通過PCA進行降維
      """
       covariance = calculate_covariance_matrix(X)

      #求解特徵值和特徵向量
      self.eigen_values, self.eigen_vectors = np. linalg.eig(covariance)

      #將特徵值從大到小進行排序，注意特徵向量是按列排的，即self.eigen_vectors第k列是self.eigen_values中第k個特徵值對應的特徵向量
      idx = self.eigen_values.argsort()[::- 1 ]
      eigenvalues = self.eigen_values[idx][:self.k]
      eigenvectors = self.eigen_vectors[:, idx][:, :self.k]
      #將原始數據集X映射到低維空間
      X_transformed = X.dot(eigenvectors)

      return X_transformed

def  main () :
    # Load the dataset
     data = datasets.load_iris()
    X = data.data
    y = data.target

    #將數據集X映射到低維空間
```

```
    X_trans = PCA().transform(X)

    x1 = X_trans[:, 0 ]
    x2 = X_trans[:, 1 ]

    cmap = plt.get_cmap( 'viridis' )
    colors = [cmap(i) fori in np.linspace( 0 , 1 , len(np.unique(y)))]

    class_distr = []
    # Plot the different class distributions
    for i, l in enumerate(np.unique(y)):
        _x1 = x1[ y == l]
        _x2 = x2[y == l]
        _y = y[y == l]
        class_distr.append(plt.scatter(_x1, _x2, color=colors[i]))

    # Add a legend
     plt.legend (class_distr, y, loc= 1 )

    # Axis labels
     plt.xlabel( 'Principal Component 1' )
    plt.ylabel( 'Principal Component 2' )
    plt.show()

if __name__ =="__main__" :
    main()
```

最終，我們將得到降維結果如下。其中，如果得到當特徵數(D) 遠大於樣本數(N) 時，可以使用一點小技巧實現PCA 算法的複雜度轉換。



PCA 降維算法展示

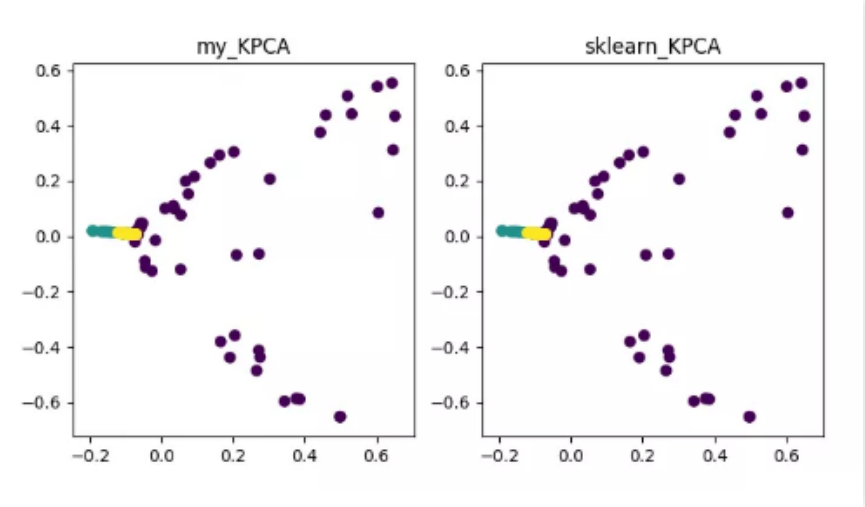
當然，這一算法雖然經典且較為常用，其不足之處也非常明顯。它可以很好的解除線性相關，但是面對高階相關性時，效果則較差；同時，PCA 實現的前提是假設數據各主特徵是分佈在正交方向上，因此對於在非正交方向上存在幾個方差較大的方向，PCA 的效果也會大打折扣。

05 其它降維算法及代碼地址

- KPCA（kernel PCA）

KPCA 是核技術與PCA 結合的產物，它與PCA 主要差別在於計算協方差矩陣時使用了核函數，即是經過核函數映射之後的協方差矩陣。

引入核函數可以很好的解決非線性數據映射問題。kPCA 可以將非線性數據映射到高維空間，在高維空間下使用標準PCA 將其映射到另一個低維空間。

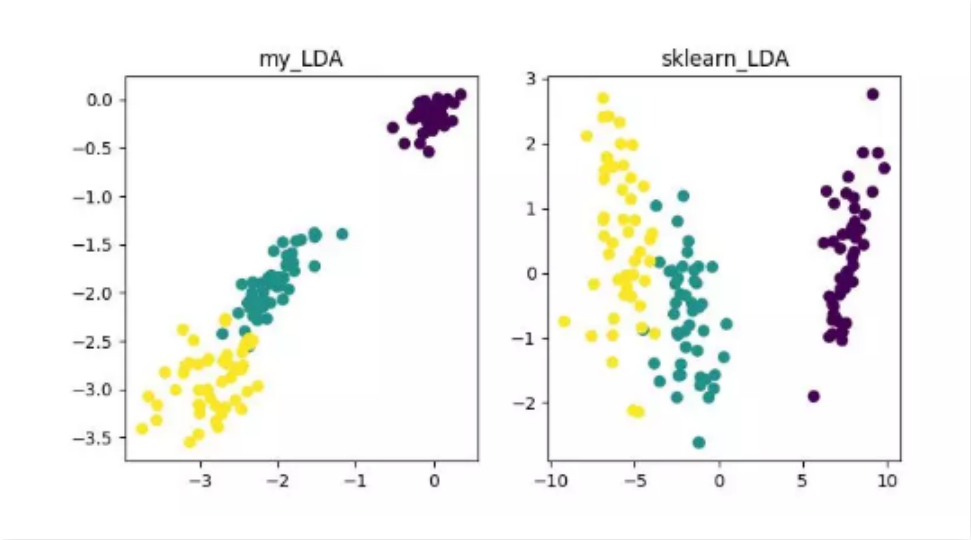


KPCA 降維算法展示

> 詳細內容可參見《Python 機器學習》之特徵抽取——kPCA：
>
> https://blog.csdn.net/weixin_40604987/article/details/79632888
>
> 代碼地址：
>
> https://github.com/heucoder/dimensionality_reduction_alo_codes/blob/master/codes/PCA/KPCA.py

- LDA（Linear Discriminant Analysis）

LDA 是一種可作為特徵抽取的技術，其目標是向最大化類間差異，最小化類內差異的方向投影，以利於分類等任務即將不同類的樣本有效的分開。LDA 可以提高數據分析過程中的計算效率，對於未能正則化的模型，可以降低維度災難帶來的過擬合。



LDA 降維算法展示

> 詳細內容可參見《數據降維—線性判別分析（LDA）》：
>
> https://blog.csdn.net/ChenVast/article/details/79227945
>
> 代碼地址：
>
> https://github.com/heucoder/dimensionality_reduction_alo_codes/tree/master/codes/LDA

- MDS（multidimensional scaling）

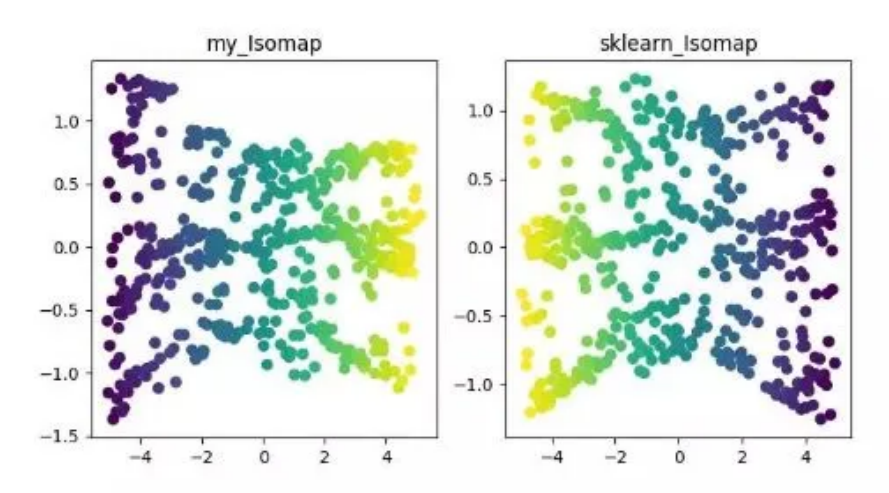
MDS 即多維標度分析，它是一種通過直觀空間圖表示研究對象的感知和偏好的傳統降維方法。該方法會計算任意兩個樣本點之間的距離，使得投影到低維空間之後能夠保持這種相對距離從而實現投影。
由於sklearn 中MDS 是採用迭代優化方式，下面實現了迭代和非迭代的兩種。

MDS 降維算法展示

> 詳細內容可參見《MDS 算法》
>
> https://blog.csdn.net/zhangweiguo_717/article/details/69663452
>
> 代碼地址：
>
> https://github.com/heucoder/dimensionality_reduction_alo_codes/tree/master/codes/MDS

- ISOMAP

Isomap 即等度量映射算法，該算法可以很好地解決MDS 算法在非線性結構數據集上的弊端。
MDS 算法是保持降維後的樣本間距離不變，Isomap 算法則引進了鄰域圖，樣本只與其相鄰的樣本連接，計算出近鄰點之間的距離，然後在此基礎上進行降維保距。

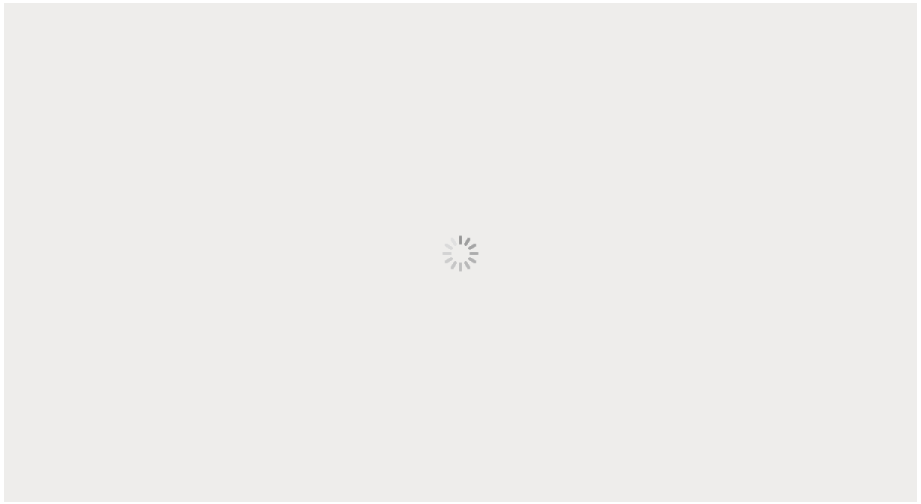


ISOMAP 降維算法展示

> 詳細內容可參見《Isomap》
>
> https://blog.csdn.net/zhangweiguo_717/article/details/69802312
>
> 代碼地址：
>
> https://github.com/heucoder/dimensionality_reduction_alo_codes/tree/master/codes/ISOMAP

- LLE（locally linear embedding）

LLE 即局部线性嵌入算法，它是一种非线性降维算法。该算法核心思想为每个点可以由与它相邻的多个点的线性组合而近似重构，然后将高维数据投影到低维空间中，使其保持数据点之间的局部线性重构关系，即有相同的重构系数。在处理所谓的流形降维的时候，效果比 PCA 要好很多。

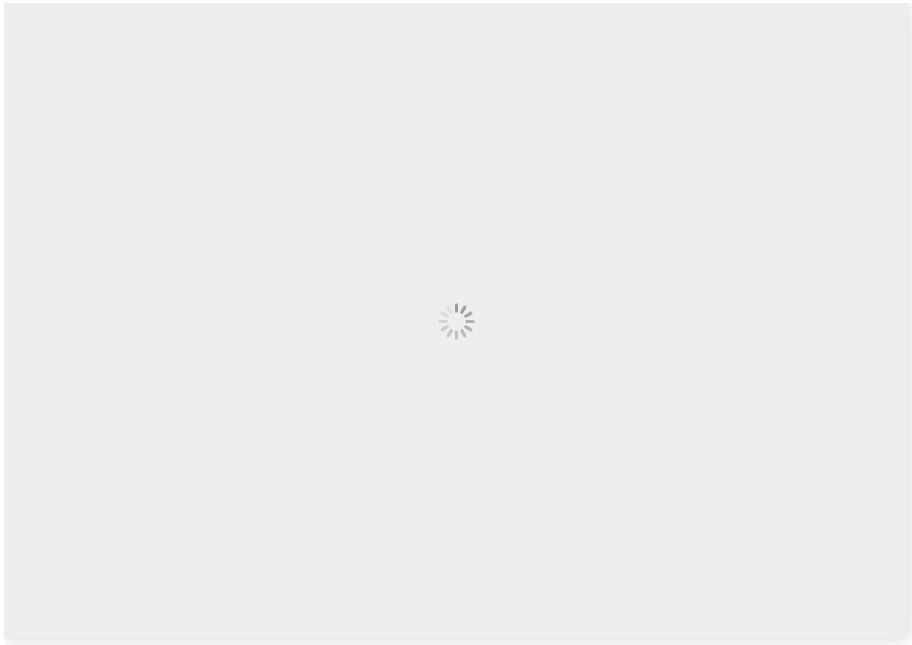


LLE 降维算法展示

> 详细内容可参见《LLE 原理及推导过程》
>
> https://blog.csdn.net/scott198510/article/details/76099630
>
> 代码地址：
>
> https://github.com/heucoder/dimensionality_reduction_alo_codes/tree/master/codes/LLE

- t-SNE

t-SNE 也是一种非线性降维算法，非常适用于高维数据降维到 2 维或者 3 维进行可视化。它是一种以数据原有的趋势为基础，重建其在低纬度（二维或三维）下数据趋势的无监督机器学习算法。
下面的结果展示参考了源代码，同时也可用 tensorflow 实现（无需手动更新参数）。

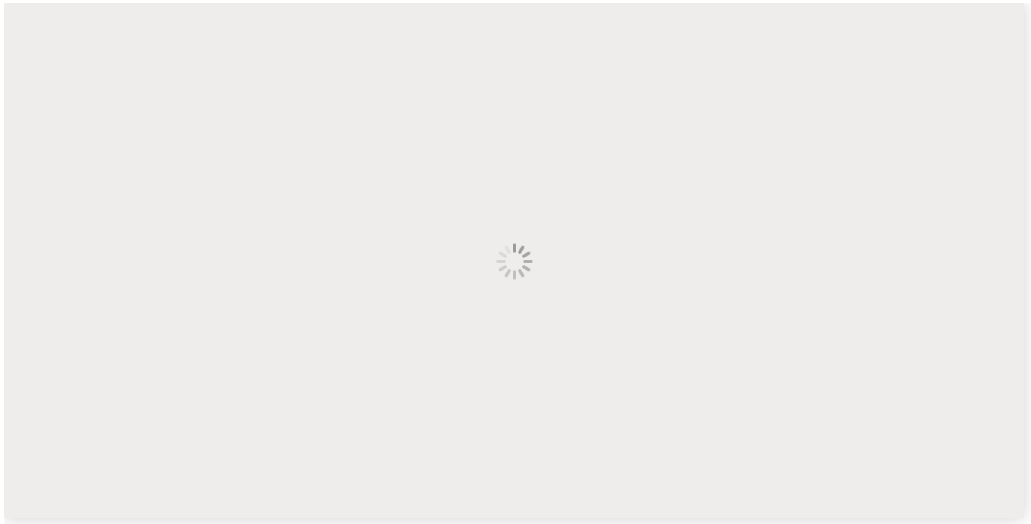


t-SNE 降维算法展示

> 详细内容可参见《t-SNE 使用过程中的一些坑》：
>
> http://bindog.github.io/blog/2018/07/31/t-sne-tips/
>
> 代码地址：
>
> https://github.com/heucoder/dimensionality_reduction_alo_codes/tree/master/codes/T-SNE

- LE（Laplacian Eigenmaps）

LE 即拉普拉斯特征映射，它与 LLE 算法有些相似，也是以局部的角度去构建数据之间的关系。它的直观思想是希望相互间有关系的点（在图中相连的点）在降维后的空间中尽可能的靠近；以这种方式，可以得到一个能反映流形的几何结构的解。

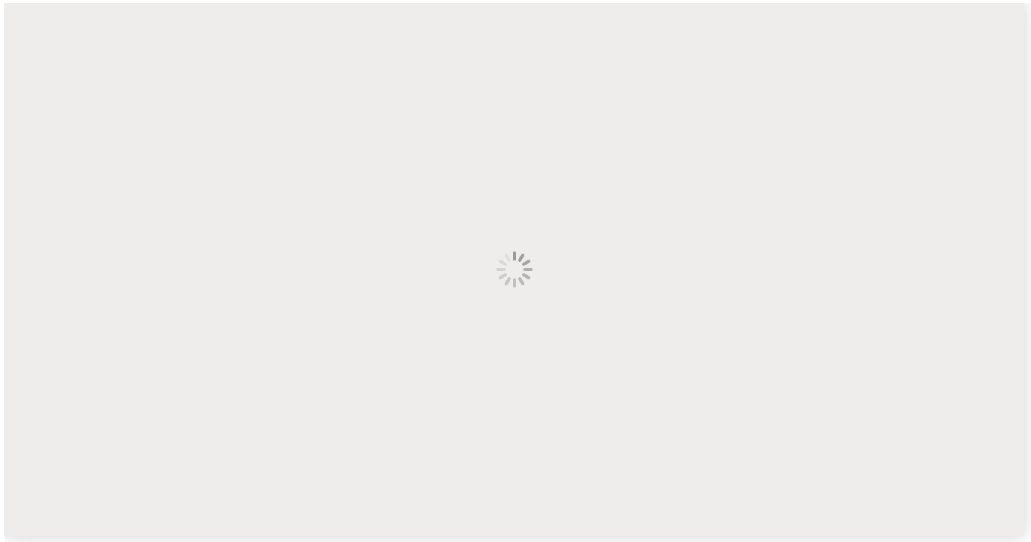


LE 降维算法展示

> 详细内容可参见《拉普拉斯特征图降维及其 python 实现》：
>
> https://blog.csdn.net/HUSTLX/article/details/50850342
>
> 代码地址：
>
> https://github.com/heucoder/dimensionality_reduction_alo_codes/tree/master/codes/LE

- LPP（Locality Preserving Projections）

LPP 即局部保留投影算法，其思路和拉普拉斯特征映射类似，核心思想为通过最好的保持一个数据集的邻居结构信息来构造投影映射，但 LPP 不同于 LE 的直接得到投影结果，它需要求解投影矩阵。

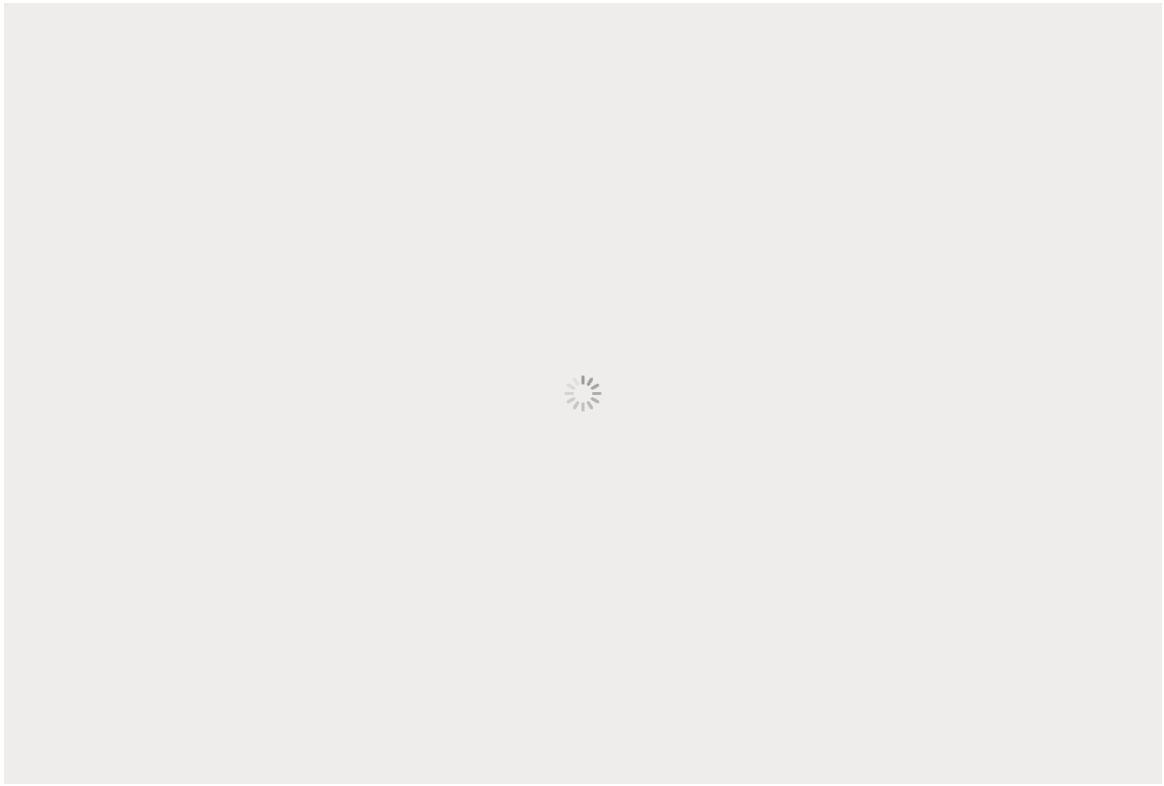


LPP 降维算法展示

05 算法刷题神器：解题PDF

如果刷题还是建议去Leetcode刷。除此之外，这里再跟大家推荐一本前不久火爆 GitHub 的 LeetCode 中文刷题手册：LeetCode Cookbook。

GitHub：https://github.com/halfrost/LeetCode-Go



> 详情请参见《局部保留投影算法 (LPP) 详解》：
>
> https://blog.csdn.net/qq_39187538/article/details/90402961
>
> 代码地址：
>
> https://github.com/heucoder/dimensionality_reduction_alo_codes/tree/master/codes/LPP

> Github 项目地址:
>
> https://github.com/heucoder/dimensionality_reduction_alo_codes

–THE END–

文章推荐

☞10部关于数学的顶级纪录片

☞假如地球变成甜甜圈形状，世界会变成什么样子？

☞應用數學的強大威力

☞丘成桐：本科教育怪事多

☞浙大一位數學教授火了！聯手哈佛劍橋學者破解數學界幾十年的謎題，為人很低調，曾說勤能補拙

☞青年博士離職高校被索賠10.5萬違約金，後博士上訴至法院，判決來了！

喜歡此內容的人還喜歡

深度學習領域最常用的10個激活函數，一文詳解數學原理及優缺點

算法與數學之美