字典学习和稀疏表示

——数值计算方法大作业设计

组员 1：余鹏 学号：15352394

组员 2：袁子豪 学号：15352396

组员 3：詹巽霖 学号：15352399

组员 4：张海涛 学号：15352405

组员 5：张镓伟 学号：15352408

院系：数据科学与计算机

专业：软件工程(移动信息工程)

指导老师：纪庆革

组员合照：

从左到右依次为：张镓伟、余鹏、詹巽霖

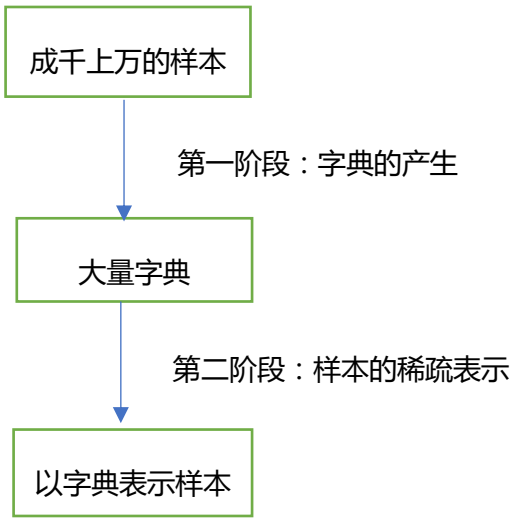
张海涛 

 袁子豪

2017 年 5 月 28 日

一、 引言

字典学习和稀疏表示一般被称为稀疏字典学习。该算法包括两个阶段：字典构建阶段和利用字典洗漱的表示样本阶段。如下图：



为什么要进行字典学习？

首先说一下字典的来源，我们知道人类社会的一切知识都必然要通过句子来表示出来，但无论有多少句子要被书写，对于每一个句子来说，它都有一个最本质上的特征，那就是构成这个句子字。可以这样说，无论人类的知识多么渊博，一本小小的字典就足以表达所有的知识，那些知识只不过是字典中的字的组合而已。所以我们看到字典学习的好处，它实质上是对于庞大数据集的一种降维表示。另外，正如同字是句子的最质朴的特征一样，字典学习总是尝试学习蕴含在样本背后最质朴的特征。所以我们要探讨字典学习。

为什么需要稀疏表示？

我想大家都有过这样的体会，在接触一个新的知识点时会有一种非常累的感觉，但是当我们把这些知识点完全掌握后，再次遇到相同的问题就会感觉到轻松，这是因为和初次接触知识点时相比，在完全掌握知识点后大脑可以调动尽可能少的脑区消耗尽可能少的能量进行同样有效的计算，此时大脑的计算速度也会变快，可以这样说，这是因为大脑学会了知识的

稀疏表示。因为稀疏表示的本质就是用尽可能少的资源表示尽可能多的知识，这种表示还能带来计算上的的加快。

二、字典学习的方法

字典学习称之为稀疏编码。从矩阵分解角度看字典学习过程：给定样本数据集 Y，Y 的每一列表示一个样本；字典学习的目标是把 Y 矩阵分解成 D、X 矩阵：

$$Y \approx D * X$$

同时满足约束条件：X 尽可能稀疏，同时 D 的每一列是一个归一化向量。

D 称之为字典，D 的每一列称之为原子；X 称之为编码矢量、特征、系数矩阵；字典学习可以有三种目标函数形式

(1)第一种形式：

$$D, X = argmin \frac{1}{2} \|Y - D \cdot X\|^2 + \lambda \cdot \|X\|_0$$

这种形式因为 L0 难以求解，所以很多时候用 L1 正则项替代近似。

(2)第二种形式：

$$D, X = \arg\min_{D,X} \{\|X\|_0\}$$
$$st. \quad \|Y - DX\|^2 \le \varepsilon$$

ε 是重构误差所允许的最大值。

(3)第三种形式：

$$D、X = \arg\min_{D,X} \|Y - DX\|^2$$
$$st. \quad \|X\|_0 \le L$$

L 是一个常数，稀疏度约束参数，上面三种形式相互等价。

SRC 基本模型：

1、对 x 进行编码，对系数加 l1 范数约束，取最小值。

$$\mathbf{a} = \underset{\mathbf{a}}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{D}\mathbf{a}\|_2^2 + \lambda\|\mathbf{a}\|_1$$

2、对 x 进行分类判决。

$$c = \underset{i}{\operatorname{argmin}} \|\mathbf{x} - \mathbf{X}_i\delta_i(\mathbf{a})\|_2^2$$

其中δ(a) 是提取与第 i 个类别相关的元素的向量指示函数。

传统字典学习框架：

$$\begin{aligned}\{\mathbf{A}, \mathbf{D}\} &= \underset{\substack{\mathbf{D}\in\mathbb{R}^{d\times K}\\ \mathbf{A}\in\mathbb{R}^{K\times N}}}{\operatorname{argmin}} \sum_{i=1}^{N} \|\mathbf{x}_i - \mathbf{D}\mathbf{a}_i\|_2^2 + \lambda\|\mathbf{a}_i\| \\ &= \underset{\substack{\mathbf{D}\in\mathbb{R}^{d\times K}\\ \mathbf{A}\in\mathbb{R}^{K\times N}}}{\operatorname{argmin}} \|\mathbf{X} - \mathbf{D}\mathbf{A}\|_F^2 + \lambda\|\mathbf{A}\|_1 \\ &\text{s.t. } \|\mathbf{d}_i\|_2^2 \le 1, \text{ for } \forall i = 1, \ldots, N,\end{aligned}$$

其中，共 N 个信号。A=[a1,a2,...,aN]是编码系数矩阵。矩阵 A 的 1 范数等价于 A 的各个列向量的 1 范数之和。

但 SRC 方法是有缺点的：

1、预定字典包含冗余和琐碎的不利于人脸识别的信息。

2、训练数据增加，稀疏编码的计算量增加。

为了解决以上的问题，可以使用以下两类已经存在的基于 DL 的方法：一类是直接学习具有识别力的字典，另一类是稀疏化稀疏，是字典具有识别力。

1、直接学习具有识别力的字典

Meta-Face Learning 方法：

该方法是针对每一个类别学习得到一个自适应的字典。

$$\mathbf{D}_i = \arg\min_{\mathbf{D}_i} \|\mathbf{X}_i - \mathbf{D}_i\mathbf{A}_i\|_2^2 + \lambda\|\mathbf{A}_i\|_1,$$
$$\text{s.t.} \quad \|\mathbf{d}_j^i\|_2 \leq 1, \forall j = 1, \ldots, K,$$

其中矩阵 Xi 包含第 i 个类别的所有样本，Di 是第 i 个类别对应的字典。

Dictionary Learning with Structured Incoherence（DLSI)方法：

不同类型的子字典具有连贯性，则重构查询图像时的原子是可以互相代替的。这导致无法用重构误差进行判决，为了解决这一问题，Ramirez 等人增加了一个不连贯项的约束，使不同类型的子字典之间尽可能互相独立。

不连贯项：

$$\mathcal{Q}(\mathbf{D}_i, \mathbf{D}_j) = \|\mathbf{D}_i^T\mathbf{D}_j\|_F^2.$$

最终的字典学习算法为：

$$\min_{\{\mathbf{D}_i, \mathbf{A}_i\}_{i=1,\ldots,C}} \sum_{i=1}^{C} \left\{ \|\mathbf{X}_i - \mathbf{D}_i\mathbf{A}_i\|_F^2 + \lambda\|\mathbf{A}_i\|_1 \right\} + \eta \sum_{i \neq j} \|\mathbf{D}_i^T\mathbf{D}_j\|_F^2$$

不连贯项的含义是：在重构误差是，忽略与公共原子相关的系数的绝对值，一次提高系统的判决能力。

2、使系数具有识别力的方法

该类方法使稀疏系数具有识别力，间接使字典具有识别力，该类方法只需要学习一个整体的字典，不需要每个类别都学习一个相应的字典。

监督字典学习（SDL）方法：

Mairal 等人提出讲逻辑回归与传统字典学习框架结合。优化公式为：

$$(\mathbf{A},\mathbf{D}) = \underset{\substack{\boldsymbol{\theta}\\ \mathbf{D}\in\mathbb{R}^{d\times K}\\ \mathbf{A}\in\mathbb{R}^{K\times N}}}{\operatorname{argmin}} \sum_{i=1}^{N}(\mathcal{C}(y_i f(\mathbf{x}_i,\mathbf{a}_i,\boldsymbol{\theta})) + \lambda_0\|\mathbf{x}_i-\mathbf{D}\mathbf{a}_i\|_2^2 + \lambda_1\|\mathbf{a}_i\|_1) + \lambda_2\|\boldsymbol{\theta}\|_2^2$$
$$\text{s.t. } \|\mathbf{d}_i\|_2^2 \le 1, \text{ for } \forall i = 1,\ldots,N,$$

where $\mathcal{C}$ is the logistic loss function ($\mathcal{C}(x) = \log(1+e^{-x})$),

λ2 是防止过拟合的正则项参数，f 是与系数 a 呈线性关系的函数：

$$f(x,a,\theta)=\theta^T a+b,\theta\in R^K$$

或者与 a 成双线性关系的函数：

$$f(x,a,\theta)=x^T Wa+b,\theta=\{W\in R^{d\times K},b\in R\}$$

用于字典学习的具有识别力的 K-SVD 方法：

D-KSVD 在传统 DL 框架上加了一个简单的线性回归作为惩罚项。

$$(\mathbf{D},\mathbf{W},\mathbf{A}) = \underset{\mathbf{D},\mathbf{W},\mathbf{A}}{\operatorname{argmin}} \|\mathbf{X}-\mathbf{D}\mathbf{A}\|_F^2 + \lambda_1\|\mathbf{Q}-\mathbf{G}\mathbf{A}\|_F^2 + \lambda_2\|\mathbf{H}-\mathbf{W}\mathbf{A}\|_F^2 + \lambda_3\|\mathbf{A}\|_1$$
$$\text{s.t. } \|\mathbf{d}_i\|_2^2 \le 1, \text{ for } \forall i = 1,\ldots,N,$$

H=[h1,h2,……hN]是训练图像的标签。hn=[0,……0,1,0……0],非零元素的位置即为所属的类别。W 是分类器的参数。由上式可是，前两项可以混合为一项，最后一项根据 KSVD 可以去掉。最终优化得到 D 和 W 后，即可迅速对查询图像进行分类。

接下来详细介绍 K-SVD 方法

K-SVD 是字典学习的一种经典算法，其求解方法是固定其中一个，然后更新另外一个变量，交替迭代更新。字典 D 的每一列就相当于一个聚类中心，稀疏编码 X 的每一列允许有几个非零元素，给定训练数据 Y,Y 的每一列表示一个样本，我们的目标是求解字典 D 的每一列。

1.随机初始化字典 D：

从样本集 Y 中随机挑选 k 个样本作为 D 的原子，并且初始化编码　矩阵 X 为 0 矩阵。

2.固定字典，求取每个样本的稀疏编码：

编码过程可以采用如下公式：

$$D, X = \arg\min_{D,X}\{\|X\|_0\}$$
$$st. \quad \|Y - DX\|^2 \le \varepsilon$$

其中 ε 是重构误差所允许的最大值。

假设我们的单个样本是向量 y，为了简单起见我们就假设原子只有这 4 个，也就是字典 D=[α1、α2、α3、α4]，且 D 是已经知道的；我们的目标是计算 y 的编码 x，使得 x 尽量的稀疏。

(1)首先从 α1、α2、α3、α4 中找出与向量 y 最近的那个向量，也就是分别计算点乘：

α1*y、α2*y、α3*y、α4*y

然后求取最大值对应的原子 α。

(2)假设 α2*y 最大，那么我们就用 α2，作为我们的第一个原子，然后我们的初次编码向量就为：

x1=（0,b,0,0）

b 是一个未知参数。

(3)求解系数 b：y-b*α2=0

方程只有一个未知参数 b，是一个高度超静定方程，求解最小二乘问题。

(4)然后我们用 x1 与 α2 相乘重构出数据，然后计算残差向量：y’ =y-b*a2

如果残差向量 y’ 满足重构误差阈值范围 ε，那么就结束，否则进入下一步。

(5)计算剩余的字典 α1、α3、α4 与残差向量 y′ 的最近的向量，也就是计算

α1*y′ 、α3*y′ 、α4*y′

然后求取最大值对应的向量 α，假设 α3*y′ 为最大值，那么就令新的编码向量为：

x2=（0,b,c,0）

b、c 是未知参数。

(6)求解系数 b、c,于是我们可以列出方程：

y-b*α2-c*α3=0

方程中有两个未知参数 b、c，我们可以进行求解最小二乘方程，求得 b、c。

(7)更新残差向量 y′ ：

y′ =y-b*α2-c*α3

如果 y′ 的模长满足阈值范围，那么就结束，否则就继续循环，就这样一直循环下去。

3、逐列更新字典，并更新对应的非零编码

通过上一步我们已经知道样本的编码 ,接着我们的目标是更新字典 ,同时还要更新编码，K-SVD 采用的方法是更新字典，就是更新第 k 列原子的时候其他的院子固定不变，假设我们当前更新第 k 个原子 ak，矩阵 X 对应的第 k 行为 xk，则目标函数是：

$$\left\|Y-DX\right\|^2=\left\|Y-\sum_{j=1}^{K}\alpha_j\cdot x_j\right\|^2=\left\|(Y-\sum_{j\neq k}\alpha_j\cdot x_j)-\alpha_k\cdot x_k\right\|^2=\left\|E_k-\alpha_k\cdot x_k\right\|^2$$

我们需要注意的是 xk 不是把 X 一整行都拿出来更新（因为 Xk 中既有另元素又有非零元素，如果全部取出来的话那么计算的时候 xk 就不再保持以前的稀疏性），所以我们只能抽取出非零的向量。然后 Ek 只保留 xk 对应的非零元素项。

对于上面的方程，我们可以通过求解最小二乘的方法来求解 αk，不过这样有存在一个问题，我们求解的 αk 不是一个单位向量，因此我们需要采用 svd 分解，才能得到单位向量 αk。进过 svd 分解后，我们以最大奇异值所对应的正交单位向量，作为新的 αk，同时我们还需要更新系数编码 xk 中的非零元素(根据 svd 分解)。然后算法就在 1 和 2 之间一直迭代更新，直到收敛。

三、字典学习和稀疏表示的演示和实现

下面使用基于 python3.5 实现的代码演示，测试平台为 Anaconda 的 spyder IDE。代码及意义如下：

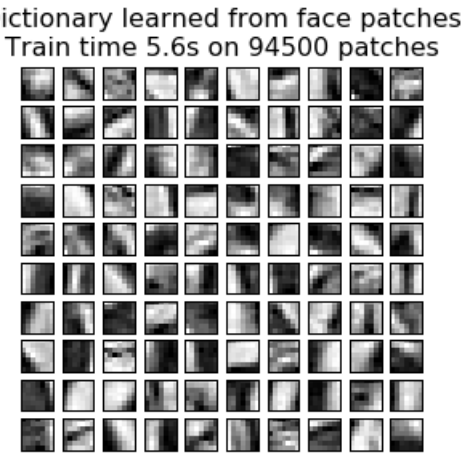
第一步先导入各种工具包和测试样例，此测试样例为 scipy 库自带的图片

```
# -*- coding: utf-8 -*-
"""
step1：首先是各种工具包的导入和测试样例的导入
"""
from time import time  #导入time模块，用于测算一些步骤的时间消耗
import matplotlib.pyplot as plt #导入画图模块
import numpy as np #导入矩阵计算模块
import scipy as sp #导入科学计算模块
from sklearn.decomposition import MiniBatchDictionaryLearning #导入一种字典学习方法
from sklearn.feature_extraction.image import extract_patches_2d
#导入碎片提取函数，碎片提取函数将图片分割成一个一个小块(pitch)
from sklearn.feature_extraction.image import reconstruct_from_patches_2d
#导入图片复原函数，可通过pitch复原一整张图片
from sklearn.utils.testing import SkipTest
#导入测试工具nose下的异常抛出函数SkipTest
from sklearn.utils.fixes import sp_version
#导入SciPy版本检测函数sp_version用于检测版本高低，版本低于0.12的SciPy没有我们需要的样本测试
if sp_version < (0, 12):
    raise SkipTest("Skipping because SciPy version earlier than 0.12.0 and "
                   "thus does not include the scipy.misc.face() image.")
#下面是测试样例导入，我们使用的是scipy库自带的测试图片
try:
    from scipy import misc
    face = misc.face(gray=True)
except AttributeError:
    # Old versions of scipy have face in the top level package
    face = sp.face(gray=True)
```

第二步计算样例图片的字典 V

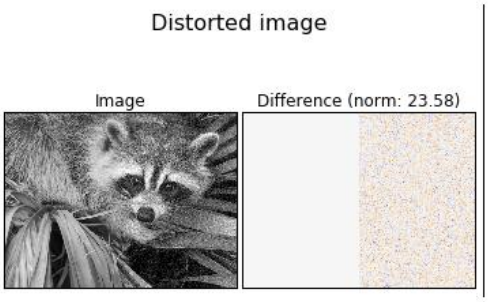
```
"""
step2：通过测试样例计算字典V
Convert from uint8 representation with values between 0 and 255 to
a floating point representation with values between 0 and 1.
"""
face = face / 255.0
#读入的face大小在0~255之间，所以通过除以255将face的大小映射到0~1上去
face = face[::2, ::2] + face[1::2, ::2] + face[::2, 1::2] + face[1::2, 1::2]
face = face / 4.0
#以上两行对图形进行采样，把图片的长和宽各缩小一般。
#记住array矩阵的访问方式 array[起始点：终结点（不包括）：步长]
height, width = face.shape
print('Distorting image...')
distorted = face.copy()
#将face的内容复制给distorted，这里不用等号因为等号在python中其实是地址的引用
distorted[:, width // 2:] += 0.075 * np.random.randn(height, width // 2)
#对照片的右半部分加上噪声，之所以左半部分不加是因为想要产生一个对比的效果
print('Extracting reference patches...')
t0 = time()#开始计时，并保存在t0中
patch_size = (7, 7)#tuple格式的pitch大小
data = extract_patches_2d(distorted[:, :width // 2], patch_size) #对图片的左半部分（未加噪声的部分）提取pitch
data = data.reshape(data.shape[0], -1)
#用reshape函数对data(94500,7,7)进行整形，reshape中如果某一位是-1，则这一维会根据
#（元素个数/已指明的维度）来计算这里经过整形后data变成（94500，49）
data -= np.mean(data, axis=0)
data /= np.std(data, axis=0)
#以上两行作用在于每一行的data减去均值除以方差，这是zscore标准化的方法
print('done in %.2fs.' % (time() - t0))
print('Learning the dictionary...')
t0 = time() #开始计时，并保存在t0中
dico = MiniBatchDictionaryLearning(n_components=100, alpha=1, n_iter=500)
#上面这行初始化MiniBatchDictionaryLearning类，并按照初始参数初始化类的属性
V = dico.fit(data).components_
#调用fit方法对传入的样本集data进行字典提取，components_返回该类fit方法的运算结果，也就是我们想要的字典V
dt = time() - t0
print('done in %.2fs.' % dt)
plt.figure(figsize=(4.2, 4))#figsize方法指明图片的大小，4.2英寸宽，4英寸高。其中一英寸的定义是80个像素点
for i, comp in enumerate(V[:100]):#：循环画出100个字典V中的字
    plt.subplot(10, 10, i + 1)
    plt.imshow(comp.reshape(patch_size), cmap=plt.cm.gray_r,
               interpolation='nearest')
    plt.xticks(())
    plt.yticks(())
plt.suptitle('Dictionary learned from face patches\n' +
             'Train time %.1fs on %d patches' % (dt, len(data)),
             fontsize=16)
plt.subplots_adjust(0.08, 0.02, 0.92, 0.85, 0.08, 0.23)#left, right, bottom, top, wspace, hspace

```

本步骤之后可以看到字典 V 如下：



第三步我们画出标准图像和真正的噪声，方便同之后字典学习学到的噪声相比较

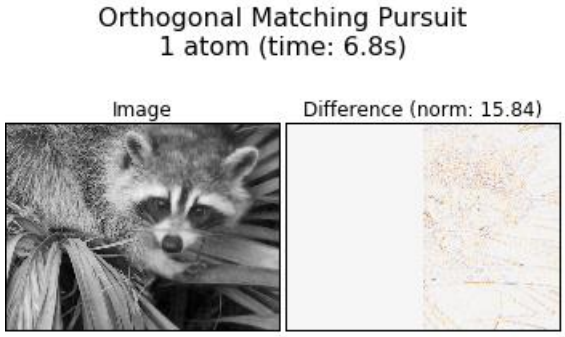
```
'''
step3：画出标准图像和真正的噪声，方便同之后字典学习学到的噪声相比较
'''
def show_with_diff(image, reference, title):
    """Helper function to display denoising"""
    plt.figure(figsize=(5, 3.3))
    plt.subplot(1, 2, 1)
    plt.title('Image')
    plt.imshow(image, vmin=0, vmax=1, cmap=plt.cm.gray,
               interpolation='nearest')
    plt.xticks(())
    plt.yticks(())
    plt.subplot(1, 2, 2)
    difference = image - reference
    plt.title('Difference (norm: %.2f)' % np.sqrt(np.sum(difference ** 2)))
    plt.imshow(difference, vmin=-0.5, vmax=0.5, cmap=plt.cm.PuOr,
               interpolation='nearest')
    plt.xticks(())
    plt.yticks(())
    plt.suptitle(title, size=16)
    plt.subplots_adjust(0.02, 0.02, 0.98, 0.79, 0.02, 0.2)
show_with_diff(distorted, face, 'Distorted image')
```

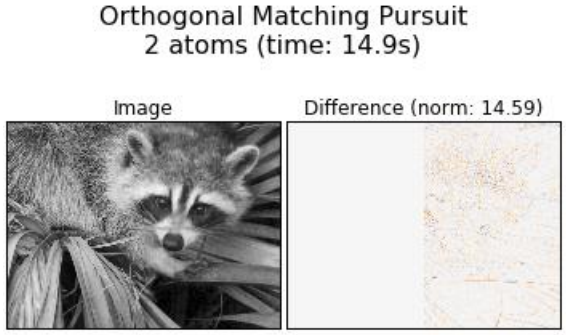


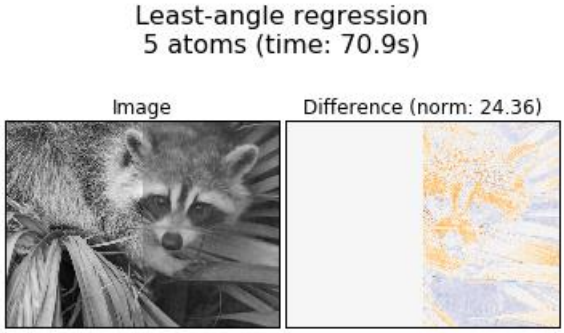
第四步我们测试不同字典学习方法对字典学习的影响

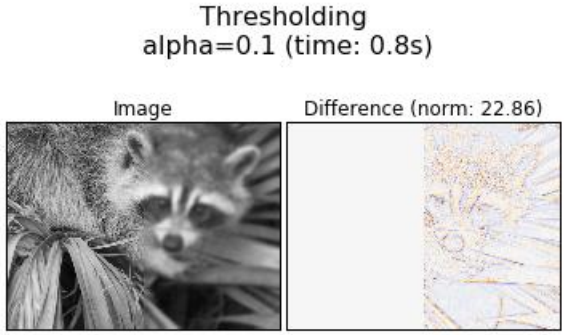
```
'''
step4：测试不同的字典学习方法和参数对字典学习的影响
'''
print('Extracting noisy patches... ')
t0 = time()
data = extract_patches_2d(distorted[:, width // 2:], patch_size)#提取照片中被污染过的右半部进行字典学习。
data = data.reshape(data.shape[0], -1)
intercept = np.mean(data, axis=0)
data -= intercept
print('done in %.2fs.' % (time() - t0))
transform_algorithms = [  #这里是四中不同字典的表示策略
    ('Orthogonal Matching Pursuit\n1 atom', 'omp',
     {'transform_n_nonzero_coefs': 1}),
    ('Orthogonal Matching Pursuit\n2 atoms', 'omp',
     {'transform_n_nonzero_coefs': 2}),
    ('Least-angle regression\n5 atoms', 'lars',
     {'transform_n_nonzero_coefs': 5}),
    ('Thresholding\n alpha=0.1', 'threshold', {'transform_alpha': .1})]
reconstructions = {}
for title, transform_algorithm, kwargs in transform_algorithms:
    print(title + '...')
    reconstructions[title] = face.copy()
    t0 = time()
    dico.set_params(transform_algorithm=transform_algorithm, **kwargs)#通过set_params对第二阶段的参数进行设置
    code = dico.transform(data)
    #transform根据set_params对设完参数的模型进行字典表示，表示结果放在code中。
    #code总共有100列，每一列对应着V中的一个字典元素，所谓稀疏性就是code中每一行的大部分元素都是0，
    #这样就可以用尽可能少的字典元素表示回去。
    patches = np.dot(code, V)#code矩阵乘V得到复原后的矩阵patches
    patches += intercept
    patches = patches.reshape(len(data), *patch_size)#将patches从（94500，49）变回（94500，7，7）
    if transform_algorithm == 'threshold':#
        patches -= patches.min()
        patches /= patches.max()
    reconstructions[title][:, width // 2:] = reconstruct_from_patches_2d(
        patches, (height, width // 2))#通过reconstruct_from_patches_2d函数将patches重新拼接回图片
    dt = time() - t0
    print('done in %.2fs.' % dt)
    show_with_diff(reconstructions[title], face,
                   title + ' (time: %.1fs)' % dt)
plt.show()
```

下面是 4 种不同的字典学习方法的结果：









提交附件

test.py　该算法的实现代码