模式识别与统计学习
实验五：*Singular Values Decomposition*实验报告

郭宇航 2016100104014

一 实验原理

1.1 问题引入

在线性代数中我们对矩阵的分解十分熟悉：即特征值分解。而特征值分解中的我们进行分解的对象是方阵。而如果是非方阵，我们应该如何对这个矩阵进行分解呢？这里就需要涉及到本次实验的奇异值分解(Singular Values Decompostion)在引入奇异值分解之前我们先简单回顾特征值分解。

首先给出目标矩阵$\boldsymbol{A}$，$\boldsymbol{A}$是一个$n \times n$的矩阵，特征值与特征向量的定义如下：

$$\boldsymbol{A}x = \lambda x \tag{1}$$

$\boldsymbol{x}$是一个n维的向量，如果我们能够找到这样一个$\boldsymbol{x}$向量，使得满足上式的条件，则我们认为λ为矩阵$\boldsymbol{A}$的一个特征值，$\boldsymbol{x}$为特征值λ所对应的特征向量。以相同的方式，我们可以得到矩阵$\boldsymbol{A}$的n个特征值可以写为：$\lambda_1, \lambda_2, \cdots, \lambda_n$,这$n$个特征值对应的特征向量可以表示为：$\mu_1, \mu_2, \cdots, \mu_n$,我们不难发现如果这$n$个特征向量是线性无关的，则矩阵$\boldsymbol{A}$可以使用特征分解为：

$$\boldsymbol{A} = \boldsymbol{U}\Sigma\boldsymbol{U}^{-1} \tag{2}$$

其中$\boldsymbol{U}$表示为这n个特征向量张成的矩阵，维度与矩阵$\boldsymbol{A}$一致，而$\boldsymbol{\Sigma}$表示为由n个特征值构成的对角阵。进一步的我们如果对$\boldsymbol{U}$中的特征向量进行标准正交化，即$\mu_i^T\mu_i = 1$同时$\mu_i^T\mu_j = 0$. 由此可以将之前的特征分解改写为：

$$\boldsymbol{A} = \boldsymbol{U'}\Sigma\boldsymbol{U'^T} \tag{3}$$

我们看到这是对$n \times n$的方阵进行的特征值分解，那如果我们得到的不是一个方阵，而是一个$m \times n$的一般化矩阵，我们的特征值将无法继续使用，而进一步的我们对这类矩阵使用奇异值分解(Singular Values Decomposition).

1.2 SVD定义

首先给出奇异值分解的定义[1]，假设存在一个一般化的矩阵或者是是我们的样本矩阵$\boldsymbol{A}$，这个矩阵的维度是$m \times n$，对矩阵$\boldsymbol{A}$奇异值分解可以表示为：

$$\boldsymbol{A}_{m\times n} = \boldsymbol{U}_{m\times m}\boldsymbol{\Sigma}_{m\times n}\boldsymbol{V}^T_{n\times n} \tag{4}$$

其中$\boldsymbol{U}$矩阵和$\boldsymbol{V}$矩阵均为酉矩阵，即其列空间内的各个向量是相互正交的；$\boldsymbol{\Sigma}$表示为对角阵，注意这里虽然表示为$m \times n$的维度，但实际上如果我们假设这里的$m > n$，则这个矩阵前n行构成的矩阵是一个对角阵，后面的$(m-n) \times n$行向量为全0向量，后面给出具体的说明，而这个对角阵上的值我们称为矩阵$\boldsymbol{A}$的奇异值。这里我们给出一张图片（图1）通过更加直观的方式理解：

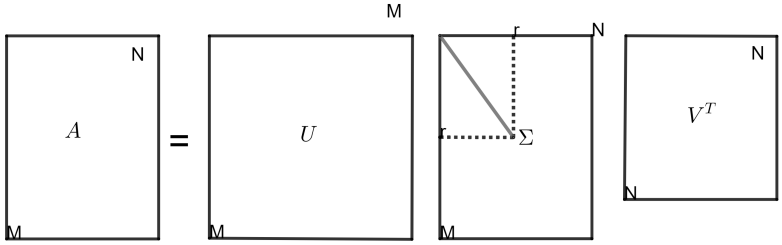

图 1: SVD分解示意图

直观上我们可以通过下面的式子理解奇异值的分解：

$$\boldsymbol{AV} = \boldsymbol{U\Sigma V^T V} = \boldsymbol{USigma} \Rightarrow \boldsymbol{A}v_i = \sigma_i u_i \Rightarrow \sigma_i = \frac{\boldsymbol{A}v_i}{u_i} \tag{5}$$

对于一个任意的矩阵来说，我们总可以找到一组单位正交基，使得该矩阵对其进行变换后得到的向量组仍然是一组正交的基底。

1.3 SVD推导和几何解释

顺着上面关于SVD的定义内容，在这一部分我们给出SVD的具体推导过程。仍然是假设我们的基本矩阵是$\boldsymbol{A}(m \times n)$，考虑虽然我们无法对矩阵$\boldsymbol{A}$本身进行特征值分解，但是我们可以构造出方阵，而且是有着良好性质的方阵。如果我们将矩阵$\boldsymbol{A}$的转置和它自身进行矩阵的乘法，我们就可以得到$n \times n$的一个方阵$\boldsymbol{A^T A}$，既然$\boldsymbol{A^T A}$是方阵，那么我们就可以进行特征值的分解，得到特征值和特征向量满足下式：

$$(\boldsymbol{A^T A})\vec{v_i} = \lambda_i \vec{v_i} \tag{6}$$

以同样的方式，我们可以进行$\boldsymbol{A}$右乘它的转置$\boldsymbol{A}^T$，得到一个维度为$m \times m$的方阵，通过这个方阵我们也可以进行特征值的分解：

$$(\boldsymbol{A A^T})\vec{u_i} = \lambda_i \vec{u_i} \tag{7}$$

而前面我们已经做过了假设，假设给定的任意的矩阵$\boldsymbol{A}$都可以分解为$\boldsymbol{A} = \boldsymbol{U\Sigma V}^T$的形式，对这个式子进行变换：

$$\boldsymbol{A A^T} = \boldsymbol{U\Sigma V^T V \Sigma^T U^T} = \boldsymbol{U\Sigma^2 U^T} \tag{8}$$

同理有：

$$\boldsymbol{A^T A} = \boldsymbol{V\Sigma^2 V^T} \tag{9}$$

由上述的两个式子和之前的结论，我们发现$\boldsymbol{\Sigma}$对角阵中的所有元素满足：$\sigma_i = \sqrt{\lambda_i}$.
有了上面的一系列的结论，我们从几何直观的角度给出一个更好理解的方式[2]，考虑二维的情况：矩阵$\boldsymbol{A}_{m\times n}$，对一组正交基底$\vec{v_1}$和$\vec{v_2}$进行变换，变换之后有这样的性质：

$$\boldsymbol{A}\vec{v_1} = \sigma_1 \vec{u_1} \qquad \boldsymbol{A}\vec{v_2} = \sigma_2 \vec{u_2} \tag{10}$$

其中$\vec{u_1}, \vec{u_2}$仍然为一组二维空间中的正交基底。

下面假设使用$\boldsymbol{A}$对$\vec{x}$进行空间的线性变换：

$$\begin{aligned}\boldsymbol{A}\vec{x} &= \boldsymbol{A}\boldsymbol{I}\vec{x} = \boldsymbol{A}\begin{bmatrix}\vec{v_1}, \vec{v_2}\end{bmatrix}\begin{bmatrix}\vec{v_1}^T \\ \vec{v_2}^T\end{bmatrix}\vec{x} = \boldsymbol{A}\begin{bmatrix}\vec{v_1}, \vec{v_2}\end{bmatrix}\begin{bmatrix}\vec{v_1}^T\vec{x} \\ \vec{v_2}^T\vec{x}\end{bmatrix} \\ &= \begin{bmatrix}\boldsymbol{A}\vec{v_1}, \boldsymbol{A}\vec{v_2}\end{bmatrix}\begin{bmatrix}\vec{v_1}^T\vec{x} \\ \vec{v_2}^T\vec{x}\end{bmatrix} = \begin{bmatrix}\sigma_1\vec{u_1}, \sigma_2\vec{u_2}\end{bmatrix}\begin{bmatrix}\vec{v_1}^T\vec{x} \\ \vec{v_2}^T\vec{x}\end{bmatrix} \\ &= \begin{bmatrix}\vec{u_1}, \vec{u_2}\end{bmatrix}\begin{bmatrix}\sigma_1 & 0 \\ 0 & \sigma_2\end{bmatrix}\begin{bmatrix}\vec{v_1}^T \\ \vec{v_2}^T\end{bmatrix}\vec{x} \\ &= \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\boldsymbol{T}}\vec{x}\end{aligned} \tag{11}$$

再进一步直观的理解：

$$\begin{aligned}\boldsymbol{A}\vec{x} &= \begin{bmatrix}\vec{u_1}, \vec{u_2}\end{bmatrix}\begin{bmatrix}\sigma_1 & 0 \\ 0 & \sigma_2\end{bmatrix}\begin{bmatrix}\vec{v_1}^T \\ \vec{v_2}^T\end{bmatrix}\vec{x} \\ &= \begin{bmatrix}\vec{u_1}, \vec{u_2}\end{bmatrix}\begin{bmatrix}\sigma_1 & 0 \\ 0 & \sigma_2\end{bmatrix}\begin{bmatrix}\vec{v_1}^T \\ \vec{v_2}^T\end{bmatrix}\begin{bmatrix}\vec{v_1}, \vec{v_2}\end{bmatrix}\begin{bmatrix}\xi_1 \\ \xi_2\end{bmatrix}\end{aligned} \tag{12}$$

其中的ξ_1, ξ_2表示为$\vec{x}$在以$\vec{v_1}, \vec{v_2}$为基底的坐标系下的坐标值，给出一幅可以直观理解的图片（图2）：

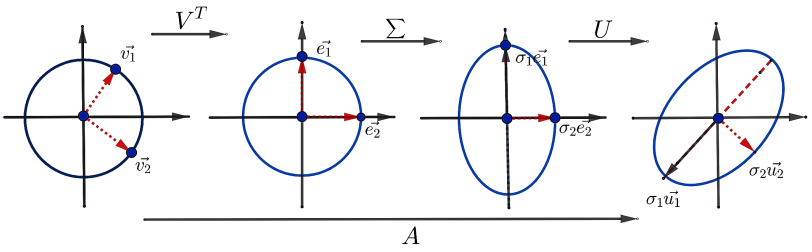

图 2: SVD分解几何直观

不难发现，我们进行了如下的一些变换：首先进行旋转，然后拉伸，再旋转的过程。我们使用二维正交基乘以长度表示了$\vec{x}$.

二 Python代码实现

本次实际试验中使用到了一张蝴蝶的图片如下所示（图3）：

图 3: Butterfly Experiment Figure

2.1 图片读入

上一节给出了本次实验的图片，首先我们对图片的像素点数据进行读入，读入图片像素点矩阵的方式有很多种，这里采用了openCV库中的imread函数进行处理，由于这里我们的图片是一个RGB三通道的图，而本次试验的主要目的不在这一方面我们就直接对图片进行灰度化的处理，最终得到了像素点矩阵，矩阵的维度是243×437.实现的代码如下：

```
def load_data(self,filepath):
    # PCA we use PIL.Image and now we use cv2 to load data and gray
    image = cv2.imread(filepath,cv2.IMREAD_GRAYSCALE)
    #print(image.shape)
    #print(type(image))
    #print(image)
    return image
```

2.2 数据预处理

SVD实验中的数据预处理只需要对标准化处理，这里就是对像素点矩阵中的每一个元素都除以255以标准化，同时方便后面的计算：

```
def data_preprocessing(self,pixel_matrix):
    pixel_matrix_1 = pixel_matrix / 255.0
    return pixel_matrix_1
```

2.3 SVD主函数

前面给出了SVD的理论推导和几何上的直观理解，下面给出我们在具体进行计算机仿真时的算法流程（Algorithm1）：

Algorithm 1: Singular Values Decomposition Algorithm

输入:

数据样本矩阵：$\boldsymbol{A}_{m\times n}$;

输出:

左奇异矩阵：$\boldsymbol{U}_{m\times k}$;

奇异值矩阵：$\boldsymbol{\Sigma}_{k\times k}$;

右奇异矩阵：$\boldsymbol{V}_{k\times n}$;

初始化数据，对输入的样本矩阵进行标准化;

计算$(\boldsymbol{A}\boldsymbol{A}^{\boldsymbol{T}})_{m\times m}$和$(\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A})_{n\times n}$;

对矩阵$(\boldsymbol{A}\boldsymbol{A}^{\boldsymbol{T}})_{m\times m}$和$(\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A})_{n\times n}$分别进行特征值分解;

将得到的特征值进行降序排列，并对应排序特征向量张成的矩阵$\boldsymbol{U}_{m\times m}$和$\boldsymbol{V}_{n\ timesn}$;

取前k个特征值，并对应取出特征向量张成的矩阵$\boldsymbol{U}_{m\times k}$和$\boldsymbol{V}_{n\ timesk}$;

对特征值开方求解奇异值并构造对角阵$\boldsymbol{\Sigma}_{k\times k}$;

输出左右奇异矩阵和奇异值对角阵;

实现代码如下：

```
def SVD_main_function(self,pixel_matrix,k):
    data_1 = np.dot(pixel_matrix,pixel_matrix.T)
    data_2 = np.dot(pixel_matrix.T,pixel_matrix)
```

```
        eigenvalues_1,eigenvectors_1 = np.linalg.eigh(data_1)
        #eigenvalues_2,eigenvectors_2 = np.linalg.eigh(data_2)
        #eigenvectors_2.transpose()
        eig_sort_1 = np.argsort(eigenvalues_1)
        top_k_eigenvalues = eigenvalues_1[eig_sort_1[:-k - 1:-1]]
        #eig_sort_2 = np.argsort(eigenvalues_2)
        top_k_eigenvector_1 = np.array(eigenvectors_1[:,eig_sort_1[:-k-1:-1]])
        print(top_k_eigenvector_1.shape)
        #top_k_eigenvector_2 = np.array(eigenvectors_2[:,eig_sort_2[:-k-1:-1]]).real    #V
        singular_values = np.sqrt(top_k_eigenvalues)  # sigma
        singular_matrix = np.mat(np.diag(singular_values))
        singular_inverse = singular_matrix.I    #V' = s^{-1} U.T A
        singular_v = singular_inverse.dot(top_k_eigenvector_1.T.dot(pixel_matrix))
        print(singular_v.shape)
        print(singular_matrix.shape)
        result_Matrix = np.dot((np.dot(top_k_eigenvector_1,singular_matrix)),singular_v)
        print(result_Matrix.shape)
        return result_Matrix
```

2.4 图像重构

做完SVD分解之后我们需要通过图像的重构来观察图像压缩的效果，这个实现比较容易，给出实现代码如下：

```
    def reconstruct_figure(self,init_data,pixel_matrix):
        restored_figure = pixel_matrix * 255.0
        #restored_figure_1 = restored_figure.real
        print(restored_figure)
        image_init = Image.fromarray(init_data)
        image_restored = Image.fromarray(restored_figure)
        fig = plt.figure()
        fig.suptitle('1')
        fig_1 = fig.add_subplot(1,2,1)
        fig_2 = fig.add_subplot(1,2,2)
        fig_1.imshow(init_data)
        fig_2.imshow(restored_figure)
        fig.show()
        pylab.show()
```

三 结果及图形展示

首先给出本次实验中的代码主函数:

```
if __name__ == '__main__':
    Singular_vd = Singular_value_decomposition
    image_pixel = Singular_vd.load_data(Singular_vd,'Butterfly.bmp')
    image_pixel_preprocessing = Singular_vd.data_preprocessing(Singular_vd,image_pixel)
    print(image_pixel_preprocessing)
    result = Singular_vd.SVD_main_function(Singular_vd,image_pixel_preprocessing,20)
    #Singular_vd.reconstruct_figure(Singular_vd,image_pixel,result)
    u,sigma,v = np.linalg.svd(image_pixel_preprocessing)
    print(u.shape,sigma.shape,v.shape)
    u_1 = np.array(u[:,:50])
    sigma_1 = np.diag(sigma[:50])
    v_1 = v[:50]
    result_1 = np.dot(u_1,np.dot(sigma_1,v_1))
```

```
Singular_vd.reconstruct_figure(Singular_vd,image_pixel,result)
```

3.1 图形绘制

这里给出提取了1,10,20,50个特征值后重构的图像(图4)图片左上右上左下右下分别为取前1个，10个，20个和50个特征值，特征向量重构出来的图片： 不难发现，当特征值和特征向量取到前50个

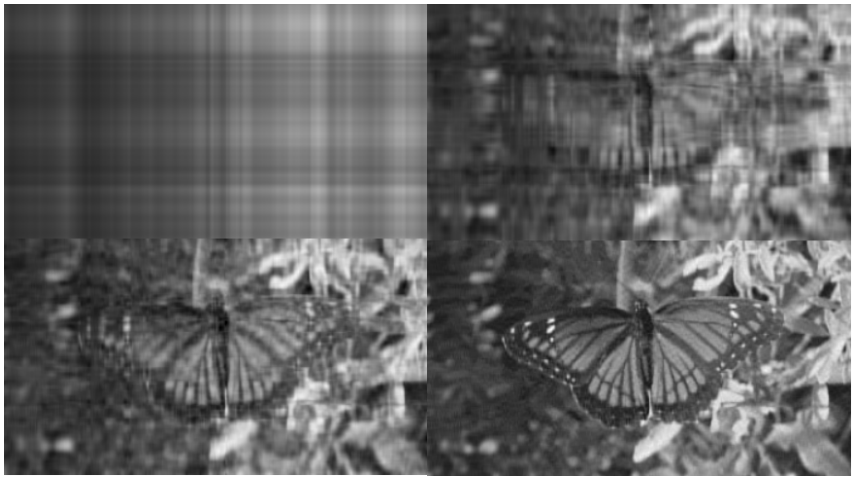

图 4: Butterfly Experiment Figure

的时候重构出来的图像的效果已经相当好了，下面我们通过分析重构的误差和压缩比之间的关系分析一下如何进行前k个特征值的选取更加合理。以此问题为例，我们不难发现压缩比和压缩后的误差可以表示为：

$$\text{Compression Ratio} = \frac{243 \times 437}{k^2 + 680k} \qquad \text{Compression Error} = \sum_{i=1}^{243}\sum_{j=1}^{437}(x_{ij} - \hat{x}_{ij})^2 \tag{13}$$

我们不难将这两个图像绘制出来(图5)：

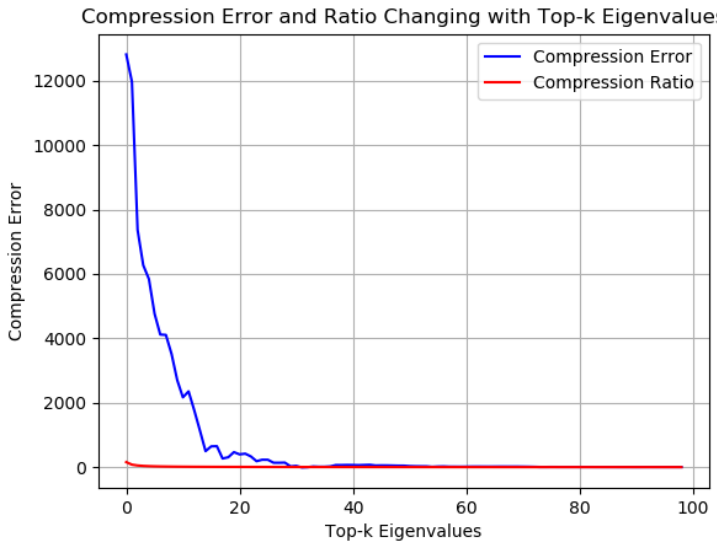

图 5: Compression Error and Ratio Changing with Top-k Eigenvalues

通过图像我们不难发现，在我们选取的特征值的数量大概在30个左右的时候，压缩误差可以达到较小的值，同时还保证了一定的压缩比例。

四 总结体会

这一次关于SVD的实验最大的一个收获在于做实验的过程中发现，有一个小问题，就是如果我们在计算的过程中分别求解左奇异矩阵和右奇异矩阵进行图像的恢复时发现无法复原，主要问题在于通过这种方法求解出来的$\boldsymbol{U},\boldsymbol{V}$不能满足：$\boldsymbol{A}=\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\boldsymbol{T}}$，所以在实际的求解过程中，我们先求解左奇异矩阵$\boldsymbol{U}$，再通过$\boldsymbol{V}^T=\boldsymbol{A}(\boldsymbol{U}\boldsymbol{\Sigma})^{-1}$间接求解右奇异矩阵。这样做出来才能满足：$\boldsymbol{A}=\boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^{\boldsymbol{T}}$，才能完整的实现图像的重构。

参考文献

[1] 刘建平Pinard, “奇异值分解(svd)原理与在降维中的应用,” https://www.cnblogs.com/pinard/p/6251584.html.

[2] “We recommend a singular value decomposition,” http://www.ams.org/publicoutreach/feature-column/fcarc-svd.