奇异值分解算法（SVD）简述

电计1601班 李英平 201685040

1. SVD算法概述

   在我们的线性代数课程里学过方阵的特征值分解，对于一个方阵，可以通过求解特征方程得到的一组特征值构成的对角矩阵表示成$A = Q\Lambda Q^{-1}$的形式，这里面的Q包含了矩阵A的一组特征向量。通过这样的方法，我们能够把一个矩阵通过他的特征值和特征向量表示出来，特征向量描述了决定矩阵的特征，特征值描述了对应的特征对矩阵影响的大小。通过特征值分解，我们能将一个大规模矩阵的问题，转化为三个小规模的矩阵。并可以根据特征值的大小舍掉一些影响不大的特征，从而起到降维的作用。然而，特征值分解需要原矩阵为方阵，而一般情况下，现实数据并不能完全是方阵，因此对于非方阵，引入了奇异值分解。

   奇异值分解的形式和特征值分解类似，对于非方阵$A_{m\times n}$有分解形式$A_{m\times n} = U_{m\times m}\Sigma_{m\times n}V^T_{n\times n}$。这里面的$U$和$V$分别称左奇异向量阵和右奇异向量阵，$\Sigma$为一对角矩阵，其各对角元为原矩阵的奇异值。奇异值分解的求解如下：

$$(A^TA)\overrightarrow{v_i} = \lambda_i\overrightarrow{v_i}$$

$$\overrightarrow{\sigma_i} = \sqrt{\lambda_i}$$

$$V = [\overrightarrow{v_i}]$$

$$\overrightarrow{u_i} = \frac{1}{\sigma_i}A\overrightarrow{v_i}$$

$$U = [\overrightarrow{u_i}]$$

   同时，奇异值有一个特点，他的衰减很快，所以同样可以用较少的几个奇异值和奇异向量来近似描述原矩阵的性质而保证很高的数据近似程度,即$A_{m\times n} = U_{m\times r}\Sigma_{r\times r}V^T_{r\times n}$, $r \ll m$, $r \ll n$。而这里面的三个矩阵的运算规模是远小于对原矩阵的运算的。

2. SVD算法应用场景

   吴军博士在他的作品《数学之美》中介绍了SVD算法在潜在语义索引中的应用。假设有一系列文章，共M篇，每篇文章N个词，那么对这个文章系列的分析可以逐文章逐词地将其加权词频保存在一个$M \times N$的矩阵中。将其进行奇异值分解之后，左奇异向量矩阵描述了每一个词语与其一个相近的语义类的关系，右奇异向量矩阵描述了每一篇文章和其主题类的相关关系，对角阵则描述了语义类和主题类的相关关系。在大幅降低运算量的同时，很好地抽象出了文本的特征。

   2017年，Google中国团队提出了基于Map Reduce的SVD并行算法，进一步提高了大规模数据下SVD的运算效率，使得SVD降维算法的应用效果更加优异。

3. SVD算法应用示例

- 运行环境：Python 3.6.4 numpy 1.14.0 matplotlib 2.1.2
- 功能描述：使用SVD降维的图像的压缩存储
- 源代码：

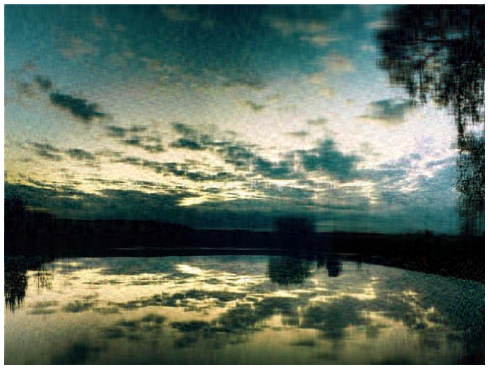
```
import numpy as np
from matplotlib import pyplot as plt

def SVDImageCompress(filename,rate):
    original=plt.imread(filename)
    # 提取RGB通道原始数据
    R0=np.array(original[:,:,0])
    G0=np.array(original[:,:,1])
    B0=np.array(original[:,:,2])
    # 各通道进行SVD分解（调用Numpy的线性工具包linalg）
    U_R,Sigma_R,V_R=np.linalg.svd(R0)
    U_G,Sigma_G,V_G=np.linalg.svd(G0)
    U_B,Sigma_B,V_B=np.linalg.svd(B0)
    # 用零矩阵初始化SVD还原后的图像图像
    R1=np.zeros(R0.shape)
    G1=np.zeros(G0.shape)
    B1=np.zeros(B0.shape)
    # 使用SVD分解后的矩阵重建各通道图像
    for i in range(int(rate*len(Sigma_R))+1):
        R1+=Sigma_R[i]*np.dot(U_R[:,i].reshape(-1,1),V_R[i,:].reshape(1,-1))
    for i in range(int(rate*len(Sigma_G))+1):
        G1+=Sigma_G[i]*np.dot(U_G[:,i].reshape(-1,1),V_G[i,:].reshape(1,-1))
    for i in range(int(rate*len(Sigma_B))+1):
        B1+=Sigma_B[i]*np.dot(U_B[:,i].reshape(-1,1),V_B[i,:].reshape(1,-1))
    # 合并RGB通道
    final=np.stack((R1,G1,B1),2)
    final[final>255]=255
    final[final<0]=0
    final=np.rint(final).astype('uint8')
    return final

if __name__=='__main__':
    # 测试代码，分别给出0.1，0.3，0.5的压缩率下的结果
    file='test.jpg'
    for rate in [0.1, 0.3, 0.5]:
        plt.imsave("SVD_Out_With_"+str(rate)+".jpg", SVDImageCompress(file, rate))
```

- 运行结果及分析：

  以下以此为压缩率为0.1，0.3，0.5时依据SVD分解重建的结果，第四张图片为原图。可见，在保留50%的奇异值的情况下，重建图片已经非常逼真原图片。可见，SVD是一种非常有效的降维算法。




4. SVD算法总结：

   SVD可以有效地对数据降维并消噪。其生成的三个矩阵在不同的具体应用中，代表着不同的数据特征，能够有效地提取出数据特征并反映它们的相关关系。SVD在信息检索、推荐算法等领域应用广泛，并行SVD难关的攻破更是使SVD可以在大数据环境下借助Map Reduce等并行工具高效运行。可以说，SVD是一个强大有力的降维工具。