基于奇异值分解的数据降维：图像压缩和推荐系统

1. 问题背景

奇异值分解（Singular Value Decomposition，SVD）是线性代数中的一个重要概念，将任意一个实矩阵分解成三个矩阵的乘积，实际上就是在寻找数据分布的主要维度，将原始的高维数据映射到低维子空间中实现数据降维。通过 SVD 分解，我们可以得到原始矩阵的主成分和它们的方差大小，这对于数据降维、特征提取等任务非常有用。此外，SVD 还被广泛应用于图像处理、推荐系统、自然语言处理等领域。本文主要是我在学习线性代数有关奇异值分解这一部分的有关深入学习、思考和实践，主要聚焦于利用奇异值分解进行图像压缩和推荐系统建立。

2. 理论分析

设A为任意$m\times n$实矩阵，$R(A)=r$，则存在m阶正交矩阵U和n阶正交矩阵V，使得

$$A=U_{m\times m}D_{m\times n}{V_{n\times n}}^{T}$$

其中

$$D=\begin{pmatrix}\Lambda & 0\\ 0 & 0\end{pmatrix},\quad \Lambda=\begin{pmatrix}\mu_1 & 0 & \cdots & 0\\ 0 & \mu_2 & \cdots & 0\\ \vdots & \vdots & & \vdots\\ 0 & 0 & \cdots & \mu_r\end{pmatrix}$$

则称上式为A的**奇异值分解**，$\mu_i(i=1,2,\cdots r)$为A的**奇异值**，式中的U被称为左奇异向量组成的标准正交基矩阵，D被称为特征值对角矩阵，V被称为右奇异向量组成的标准正交基矩阵. 这一过程可用更为形象的图形来表述。

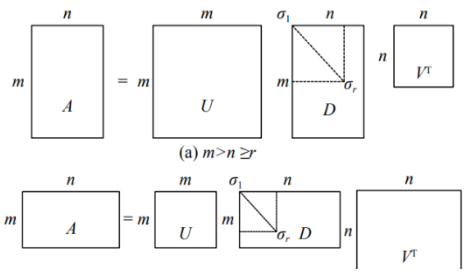

图 1 奇异值分解示意图

而如何用奇异值分解来进行图像压缩呢？

根据定义式, 如果将 r 个特征值从大到小排序, 并调动对应的U 和V 中的向量位置, 可以得到数据矩阵 A 的奇异值分解排序后的结果, 这个分解结果即可用于压缩数据矩阵 A，具体方法是确定一个奇异值数量k，取前k 个特征值以及矩阵U 和V 中的前k 个向量构成矩阵 A 的近似矩阵

$$Data_{m\times n}\approx U[:,0:k]D[0:k,0:k]V^T[0:k,:]$$

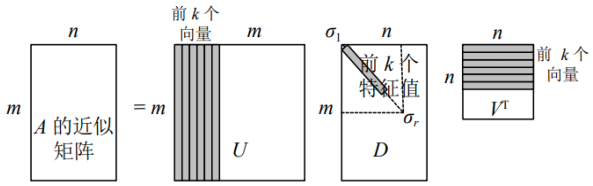

图 2 数据压缩原理

这样，在存储的时候便可以只存储U 矩阵的前k 列，矩阵D 的前 k 个特征值和矩阵V 的前k 行，然后再通过计算得到 Data 矩阵，这样存储的数据就要比直接存储矩阵 A 少的多。

同样，我们还可以根据奇异值分解的这一特性来应用于推荐系统。我们知道，推荐系统的协同过滤法， 其实是通过将用户和其他用户的评分数据进行比较实现推荐，而不关心物品的描述属性(如汉堡描述属性：高热量，碳水化合物等等)。在推荐系统中，用户数常常远大于物品数,每个用户只会对少量的物品进行打分,如果把用户和物品评分分构成一个矩阵，这会是一个较大的稀疏矩阵，因此在进行推荐之前，我们可以用奇异值分解对稀疏矩阵进行简化，从而大大提升推荐系统的效率和效果。

我们可以对用户-物品评分矩阵进行分解，得到三个矩阵：一个用户特征矩阵、一个物品特征矩阵和一个奇异值矩阵。其中，用户特征矩阵和物品特征矩阵表示了用户和物品在一些潜在特征上的相关性，而奇异值矩阵则反映了这些特征的重要程度。

在构建推荐系统时，我们可以用类似图像压缩的方法来对用户-物品评分矩阵进行降维，并根据用户特征矩阵和物品特征矩阵的乘积，计算出用户对未评价过的物品的预测评分。具体来说，在下列同样式子中

$$Data_{m\times n}\approx U[:,0:k]D[0:k,0:k]V^T[0:k,:]$$

m 和n 别是用户数和物品数，k 是一个小于等于$\min(m,n)$ 的正整数，表示降维后的维度。

3. 项目实现

3.1 基于奇异值分解的图像压缩实现

本文通过 python 来实现图像压缩，由于数字图片在计算机中以矩阵形式存储，彩色图片有 3 个图层，RGB（红、绿、蓝）也就是矩阵的一个位置上存储了 3 个基色的数值，由 3 个基色混合成不同的色彩。

通过对 3 个图层矩阵，分别进行 SVD 近似，可以取前 k 个最大的奇异值进行近似表达，最后再将 3 个图层的矩阵数据合并，用较少的数据去表达图片。

代码如下：

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

def zip_img_by_svd(img, plotId, rate=0.8):
    zip_img = np.zeros(img.shape)
    u_shape = 0
    sigma_shape = 0
    vT_shape = 0

    for chanel in range(3):   # 3 个图层
        u, sigma, v = np.linalg.svd(img[:, :, chanel])   # numpy svd 函数
        sigma_i = 0
        temp = 0

        while (temp / np.sum(sigma)) < rate:   # 选取的奇异值和需要达到设定的权重
            temp += sigma[sigma_i]
            sigma_i += 1
        SigmaMat = np.zeros((sigma_i, sigma_i))   # 选取了 sigma_i 最大的奇异值
        for i in range(sigma_i):
            SigmaMat[i, i] = sigma[i]   # 将奇异值填充到 Sigma 对角矩阵
        zip_img[:, :, chanel] = u[:, 0:sigma_i].dot(SigmaMat).dot(v[0:sigma_i, :])
        # 将分解得到的 3 个矩阵相乘，得到压缩后的近似矩阵
        u_shape = u[:, 0:sigma_i].shape
        sigma_shape = SigmaMat.shape
        vT_shape = v[0:sigma_i, :].shape

    for i in range(3):   # 对三个通道的矩阵数值进行归一化处理
        MAX = np.max(zip_img[:, :, i])
        MIN = np.min(zip_img[:, :, i])
        zip_img[:, :, i] = (zip_img[:, :, i] - MIN) / (MAX - MIN)

    zip_img = np.round(zip_img * 255).astype("uint8")
    # 不乘 255 图片是黑的（接近 0,0,0），数据类型 uint8
    plt.imsave("zip_svd_img.jpg", zip_img)   # 保存压缩后的图片
    zip_rate = (img.size - 3 * (
            u_shape[0] * u_shape[1] + sigma_shape[0] * sigma_shape[1] + vT_shape[0] * vT_shape[1])) /
(zip_img.size)

    f = plt.subplot(3, 3, plotId)
    f.imshow(zip_img)
    f.set_title("SVD 压缩率 %.4f,奇异值数量：%d" % (zip_rate, sigma_i))

    print("设置的压缩率：", rate)
    print("使用的奇异值数量：", sigma_i)
    print("原始图片大小：", img.shape)
    print("压缩后用到的矩阵大小：3x({}+{}+{})".format(u_shape, sigma_shape, vT_shape))
    print("压缩率为：", zip_rate)
```

```
if __name__ == '__main__':
    imgfile = "svd_img.jpg"
    plt.figure(figsize=(12, 12))
    plt.rcParams['font.sans-serif'] = 'SimHei'   # 消除中文乱码
    img = plt.imread(imgfile)
    f1 = plt.subplot(331)   # 绘制子图，3 行3 列，3*3 个子图，现在画第1 幅
    f1.imshow(img)
    f1.set_title("原始图片")
    for i in range(8):   # 再画8 个子图
        rate = (i + 1) / 10.0   # 压缩率 10% - 80%
        zip_img_by_svd(img, i + 2, rate)
    plt.suptitle('图片 SVD 效果对比', fontsize=17, y=0.02)   # y 偏移距离
    plt.show()
```

得到的图像如下：

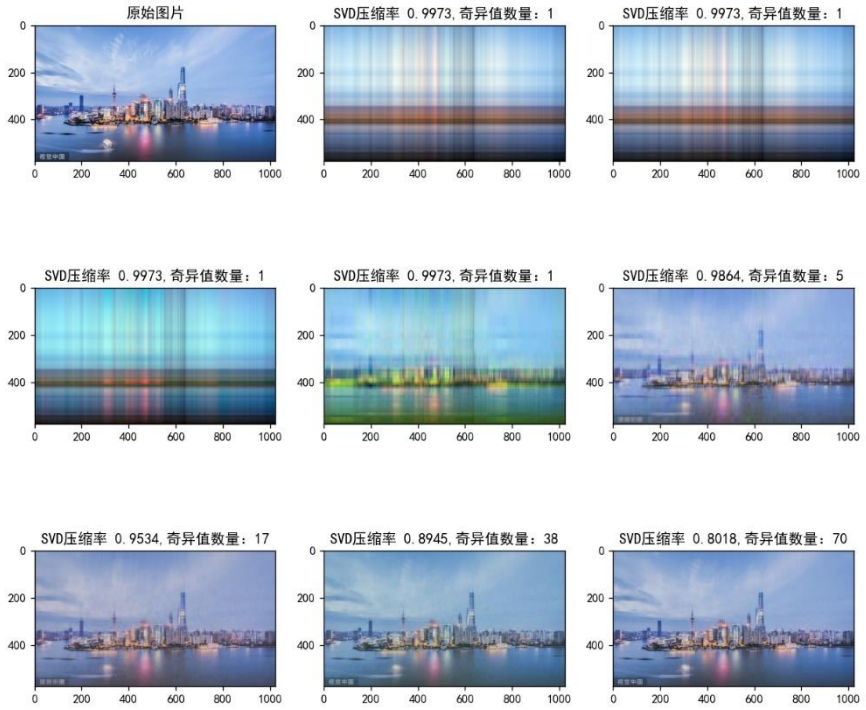

图 3 通过不同 SVD 压缩率得到的图片

原图尺寸为 1024 x 576，可以看到压缩后大概使用 70 个奇异值的 SVD 压缩的情况下，便可以得到跟原图效果类似的图片，此时 SVD 压缩率为 0.8018，这意味着只需要大约 20%的数据量就可以近似表达原始数据，压缩效果是十分可观的。

在实际应用中，一些网络传输图片的场景中，如果用户点击，则传输原图，如果用户不点击，则可以利用 SVD 压缩图片矩阵数据，比如微信收到的图片、相册缩略图等都可以利用这种原理来节约数据空间和网络流量。

3.2 基于奇异值分解的推荐系统实现

具体步骤是首先加载测试数据集，随后根据推荐系统的知识，定义三种计算相似度的方法；随后比较重要的一点事，我们这里通过计算奇异值平方和的百分比来确定将数据降到多少维才合适，返回需要降到的维度；

随后，在已经降维的数据中，基于 SVD 对用户未打分的物品进行评分预测，返回未打分物品的预测评分值；

最后，产生前 N 个评分值高的物品，返回物品编号以及预测评分值。

```
from numpy import *
from numpy import linalg as la

'''加载测试数据集'''


def loadExData():
    return mat([[0, 0, 0, 0, 0, 4, 0, 0, 0, 0, 5],
                [0, 0, 0, 3, 0, 4, 0, 0, 0, 0, 3],
                [0, 0, 0, 0, 4, 0, 0, 1, 0, 4, 0],
                [3, 3, 4, 0, 0, 0, 0, 2, 2, 0, 0],
                [5, 4, 5, 0, 0, 0, 0, 5, 5, 0, 0],
                [0, 0, 0, 0, 5, 0, 1, 0, 0, 5, 0],
                [4, 3, 4, 0, 0, 0, 0, 5, 5, 0, 1],
                [0, 0, 0, 4, 0, 4, 0, 0, 0, 0, 4],
                [0, 0, 0, 2, 0, 2, 5, 0, 0, 1, 2],
                [0, 0, 0, 0, 5, 0, 0, 0, 0, 4, 0],
                [1, 0, 0, 0, 0, 0, 0, 1, 2, 0, 0]])


'''以下是三种计算相似度的算法，分别是欧式距离、皮尔逊相关系数和余弦相似度,
注意三种计算方式的参数 inA 和 inB 都是列向量'''


def ecludSim(inA, inB):
    return 1.0 / (1.0 + la.norm(inA - inB))  # 范数的计算方法 linalg.norm()，

def pearsSim(inA, inB):
    if len(inA) < 3: return 1.0
    return 0.5 + 0.5 * corrcoef(inA, inB, rowvar=0)[0][
        1]  # 皮尔逊相关系数的计算方法 corrcoef()，

def cosSim(inA, inB):
    num = float(inA.T * inB)
    denom = la.norm(inA) * la.norm(inB)
    return 0.5 + 0.5 * (num / denom)  # 将相似度归一到 0 与 1 之间

'''按照前 k 个奇异值的平方和占总奇异值的平方和的百分比 percentage 来确定 k 的值,
后续计算 SVD 时需要将原始矩阵压缩到 k 维'''

def sigmaPct(sigma, percentage):
    sigma2 = sigma ** 2  # 对 sigma 求平方
    sumsgm2 = sum(sigma2)  # 求所有奇异值 sigma 的平方和
    sumsgm3 = 0  # sumsgm3 是前 k 个奇异值的平方和
    k = 0
    for i in sigma:
        sumsgm3 += i ** 2
        k += 1
        if sumsgm3 >= sumsgm2 * percentage:
            return k
```

```
'''函数 svdEst()的参数包含：数据矩阵、用户编号、物品编号和奇异值占比的阈值，
数据矩阵的行对应用户，列对应物品，函数的作用是基于 item 的相似性对用户未评过分的物品进行预测评分'''

def svdEst(dataMat, user, simMeas, item, percentage):
    n = shape(dataMat)[1]
    simTotal = 0.0;
    ratSimTotal = 0.0
    u, sigma, vt = la.svd(dataMat)
    k = sigmaPct(sigma, percentage)   # 确定了 k 的值
    sigmaK = mat(eye(k) * sigma[:k])   # 构建对角矩阵
    xformedItems = dataMat.T * u[:, :k] * sigmaK.I   # 根据 k 的值将原始数据压缩到 k 维空间
    for j in range(n):
        userRating = dataMat[user, j]
        if userRating == 0 or j == item: continue
        similarity = simMeas(xformedItems[item, :].T, xformedItems[j, :].T)   # 计算物品 item 与物品 j 之间的相似度
        simTotal += similarity   # 对所有相似度求和
        ratSimTotal += similarity * userRating   # 用"物品 item 和物品 j 的相似度"乘以"用户对物品 j 的评分"，并求和
    if simTotal == 0:
        return 0
    else:
        return ratSimTotal / simTotal   # 得到对物品 item 的预测评分


'''函数 recommend()产生预测评分最高的 N 个推荐结果，默认返回 5 个；
参数包括：数据矩阵、用户编号、相似度衡量的方法、预测评分的方法、以及奇异值占比的阈值；

def recommend(dataMat, user, N=5, simMeas=cosSim, estMethod=svdEst, percentage=0.9):
    unratedItems = nonzero(dataMat[user, :].A == 0)[1]   # 建立一个用户未评分 item 的列表
    if len(unratedItems) == 0: return 'you rated everything'   # 如果都已经评过分，则退出
    itemScores = []
    for item in unratedItems:   # 对于每个未评分的 item，都计算其预测评分
        estimatedScore = estMethod(dataMat, user, simMeas, item, percentage)
        itemScores.append((item, estimatedScore))
    itemScores = sorted(itemScores, key=lambda x: x[1], reverse=True)   # 按照 item 的得分进行从大到小排序
    return itemScores[:N]   # 返回前 N 大评分值的 item 名，及其预测评分值

if __name__ == '__main__':
    testdata = loadExData()
    print(recommend(testdata, 1, N=5, percentage=0.8) ) # 对编号为 1 的用户推荐评分较高的 3 件商品



    输出为：[(6, 3.3329499901459845), (9, 3.331544717872839), (4, 3.331447487712862), (8,
3.3268848098453243), (0, 3.326828341851847)]
```

利用 SVD 的推荐系统优势如下：

1. 降维：SVD 可以将原始数据降维，从而减少计算量和存储空间。在推荐系统中，用户和物品之间的评分矩阵通常非常稀疏，使用 SVD 可以将其转换为一个低维矩阵，使得只需要保存少量的主要特征向量即可表示整个矩阵。
2. 特征提取：SVD 可以从原始数据中提取出最重要的特征，这些特征通常包含了数据的大部分信息。在推荐系统中，这些特征可以表示用户和物品之间的潜在关系，从而能够更好地预测用户对未评分物品的喜好程度。
3. 个性化推荐：SVD 可以为每个用户生成一个独特的特征向量，该向量表示该用户的偏好和行为模式。这些向量可以与物品的特征向量进行匹配，以确定哪些物品最符合用户的兴趣。

4. 可扩展性：由于 SVD 可以将数据降维，因此可以更快地处理大型数据集。此外，由于 SVD 是一种矩阵分解技术，因此它还可以通过并行计算来加速处理。

装

订

线

4. 总结与收获

本次大作业是通过高代课本上的奇异值分解知识来拓展出的其作为一种数学方法在计算机视觉与机器学习领域展现出的魅力。在这次实践中，我从课本出发，由于之前了解奇异值分解在机器学习方面有一些应用，但是还不是特别熟悉，于是正好趁着此次机会参考了一些网课和资料对这方面进行学习，同时关注到了图像压缩这一领域，我于是自己准备了一些图片和推荐系统稀疏矩阵数据，根据原理编写 python 程序进行实现。我原本认为在目前阶段我们用不到太多数学知识，但是随着更加深入的学习，我发现不论是线性代数还是概率论，都是学术路上迈向更高台阶必不可少的部分，所以我以后也会继续加强数理基础的学习，提高自己的学术水平。