**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。

1. **理论分析**

设为任意实矩阵，，则存在阶正交矩阵和阶正交矩阵，使得



其中



则称上式为的**奇异值分解，**为的**奇异值，**式中的被称为左奇异向量组成的标准正交基矩阵, 被称为特征值对角矩阵, 被称为右奇异向量组成的标准正交基

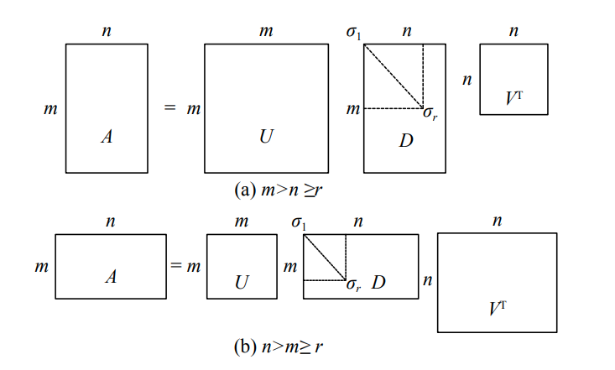
矩阵. 这一过程可用更为形象的图形来表述。

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, 并调动对应的和中的向量位置, 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，取前个特征值以及矩阵和中的前个向量构成矩阵A的近似矩阵



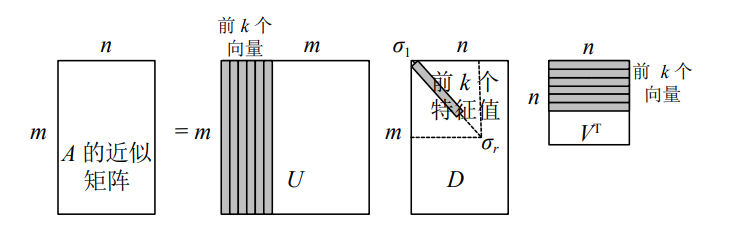


图 2 数据压缩原理

这样，在存储的时候便可以只存储矩阵的前列，矩阵的前k个特征值和矩阵的前行，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



和别是用户数和物品数，是一个小于等于的正整数，表示降维后的维度。

1. **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()

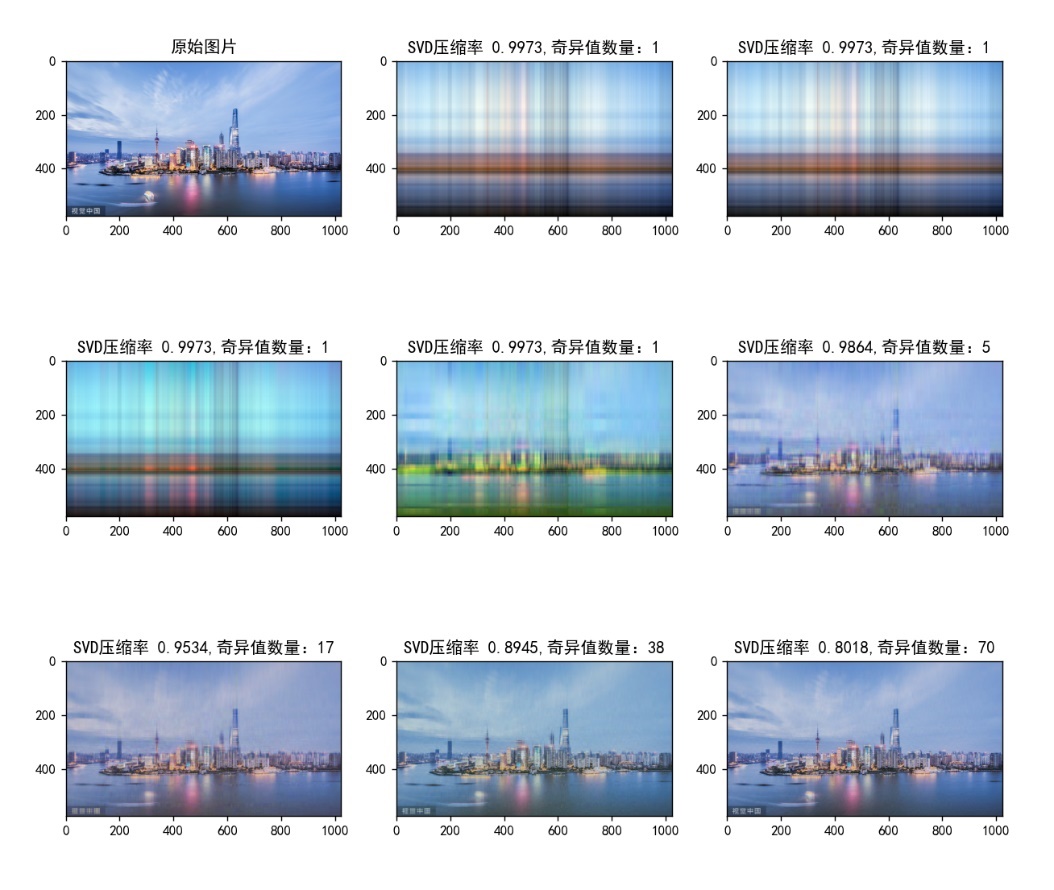
得到的图像如下：

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。
5. **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。