专业训练第五次学习报告

数试 002 梁玮同

2022 年 9 月 19 日

1 主成分分析与奇异值分解

1.1 PCA 基本原理

PCA 作为现在使用最广泛的数据处理算法，PCA 的主要思想是将 n 维特征映射到 k 维上，这 k 维是全新的正交特征也被称为主成分，是在原有 n 维特征的基础上重新构造出来的 k 维特征。

PCA 在实现降维的过程中的主要目的使分解的数据尽可能离散，同时使新选取的 k 个基尽可能不相关，而协方差矩阵则可以很好的刻画“离散”与“不相关”。“离散”可以用方差来衡量，“相关”可以用协方差来衡量，而以包含两维数据的矩阵 X 为例：

$$\begin{pmatrix} a_1 & a_2 & \dots & a_n \\ b_1 & b_2 & \dots & b_n \end{pmatrix}$$

则对于其协方差矩阵 $\frac{1}{n}XX^T$，其对角线上的元素即为字段 a,b 的方差，非对角线上的元素即为字段 a,b 的协方差。因此实现 PCA 既可以通过计算其协方差矩阵最大的 k 个特征值，取其对应的 k 个特征向量作为新的基向量实现。其中 k 的选取依赖于所需要降维后保留数据程度的要求，设 $x^{(i)}_{approx}$ 为映射后的向量，则可以用下式衡量经过压缩的信息保留程度：

$$\frac{\frac{1}{m}\sum_{i=1}^{m}||x^{(i)}-x^{(i)}_{approx}||^2}{\frac{1}{m}||x^{(i)}||^2}$$

由于在实现的过程中考虑的主要是协方差矩阵，PCA 和 SVD 也有了相当的联系，一般通过 SVD 实现 PCA 的求解。

1.2 SVD 应用于 PCA 求解

在进行 PCA 的过程中，涉及到计算需要降维数据的协方差矩阵的特征值的过程，而当其维数较大时，特征值的计算就会时一件困难且时间复杂度极高的过程，因此在一般进行 PCA

的实现过程中，都是通过 SVD 实现对于协方差矩阵特征值的计算。

设 A 是 $m \times n$ 阶矩阵，则定义 A 的奇异值分解为：

$$A = U\Sigma V^T$$

其中 U 和 V 分别为 m,n 阶酉矩阵，Σ 的主对角元素元素为 A 的奇异值，其他元素为 0。假设已经得到了 A 的奇异值分解，则我们有:

$$AA^T = U\Sigma^2 U^T$$

对于 PCR 的求解问题我们可以表示为以下求解投影矩阵 W 形式：

$$\max tr(WXX^TW^T)$$

$$s.t. WW^T = 1$$

因此在奇异值分解的基础上，我们可以直接得到左奇异矩阵 U 即为所求的投影矩阵，进而大大减少了 PCA 的计算难度。同时，奇异值分解中的右奇异矩阵在需要的情况下可以实现字段本身长度的降维。

需要注意的是, 求解 SVD 的过程虽然可以绕开对于原本矩阵的协方差矩阵特征值求解的过程，但其目前常用的 SVD 求解算法如随机 SVD 算法，其本质上是通过将求解大矩阵的奇异值分解问题转化为求解小矩阵的 SVD 分解问题，求解小矩阵的 SVD 中依然没有避开对于协方差矩阵特征值的计算, 亦或者是求奇异值矩阵违逆的算法，在时间复杂度上也都有一定优化的空间。

2 添加高斯型噪声和稀疏型噪声下的降噪实验

首先对于 WNNM,LRTV 两种方法进行调参, 以 simu indian 为例, 在添加高斯噪音下, 其 WNNM 中参数 C 的最佳取值为 0.03,LRTV 中最佳的参数组合为 $\tau = 0.008, \lambda = 0.06, r = 9$. 其余结果不在此一一展示. 下面利用 CTV,RPCA,WNNM,LRTV 四种方法进行降噪, 以 simu indian 为例, 其低秩性和局部光滑性已经验证, 现在分别对其添加 $G = 0.1, 0.2, 0.3, 0.4$ 的高斯型噪声,$G = 0.1$ 下的降噪的图片结果如下：

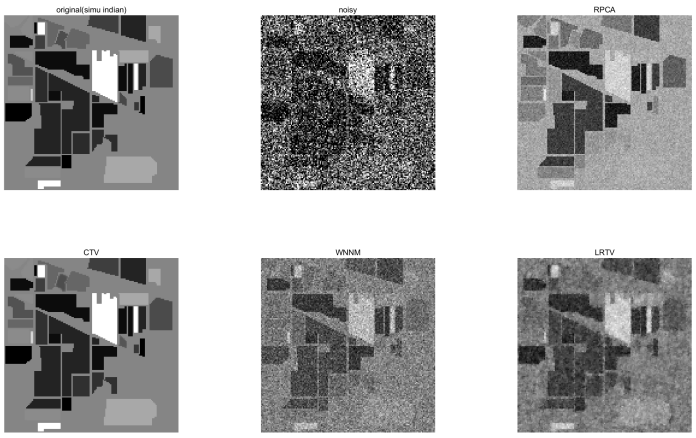

图 1: result of simu indian with Gaussian noise(G=0.1)

添加高斯型噪音后对比降噪图片的质量和时间, 结果如下表所示:

对于数据集 OriH 和 OriData 做类似的操作也可得到相应的结果, 篇幅原因在文末展示.

3 rSVD 的加速效果测试

在利用 rSVD 加速奇异值分解中, 生成的随机矩阵的列数 k 是是十分关键的参数, 直接影响到求解的效果和时间复杂度. 对于不同的数据集所选取的 k 得到的结果不尽相同, 下面首先以 OriH 为例, 测试 rSVD 的加速效果, 其中参数 $k_i = round(min(size(A))/5i)$, 结果如下

平均后对比, 利用随机 SVD 算法可以对单次 SVD 时间成本优化 39.2%,62.5%,70.5%, 75.2%, 而几种降噪算法的主要的耗时都是进行 SVD 分解, 因此选用随机 SVD 算法可以对程序有较大的加速效果.

根据 rsvd 的原理, 对程序进行改进和进一步加速, 得到三种加速的 rSVD 的算法:ieeesvdr, method1r,method2r. 现在分别考察其单次 SVD 对于时间的优化程度, 这里仍以 OriH 作为测试, 分别取随机生成矩阵的列数为 $r_1 = 8, r_2 = 16, r_3 = 24$, 其结果如下表所示:

4 嵌入三种加速随机 svd 算法后的降噪实验

需要注意的是, 几种随机 SVD 算法虽然能对降噪程序中单次 SVD 的实现效率带来极大的优化, 但同样由于忽略了一部分较小的奇异值, 对使得算法的收敛速度一定程度上降低. 为了

表 1: evaluation of simu indian(with Gaussian noise)

		RPCA	CTV	WNNM	LRTV
G=0.1	PSNR	23.7734	50.8164	23.5351	29.2611
	SSIM	0.6718	0.9994	0.6631	0.9265
	ergas	9.6092	0.4399	9.8605	5.1585
	times	4.1686	47.7877	4.6269	91.7575
G=0.2	PSNR	21.6823	50.8164	22.3984	26.9107
	SSIM	0.5786	0.9994	0.5620	0.8867
	ergas	12.2029	0.4399	11.2193	6.7417
	times	3.7615	43.2092	4.4203	94.0694
G=0.3	PSNR	20.5133	50.8164	21.5236	25.2754
	SSIM	0.5217	0.9994	0.5012	0.8512
	ergas	13.9619	0.4399	12.3948	8.1376
	times	3.7630	43.7176	4.4095	94.6543
G=0.4	PSNR	19.5999	50.8164	20.7769	24.0622
	SSIM	0.4799	0.9994	0.4565	0.8172
	ergas	15.4780	0.4399	13.4946	9.3660
	times	3.9450	45.6125	4.4466	95.7970

使时间尽可能缩短, 同时尽量维持降噪后的图像质量, 需要对不同随机矩阵的列数 R 进行试验. 由于只是为了验证其加速效果, 这里只对数据集 simu indian 做嵌入加速 SVD 算法后的降噪实验. 由于只是为了验证其对于程序加速的效果, 在这里只对耗时较长的 CTV 和 LRTV 两种算法进行实验。对于 ieee 算法, 其结果如下所示：

对于结果进行分析

表 2: time cost of rsvd with different k

svd	0.1513	0.1482	0.1460	0.1518	0.1512	0.1547	0.1565	0.1556	0.1466	0.1504
rsvd(k_1)	0.0918	0.0964	0.0929	0.0934	0.0953	0.1034	0.1029	0.0989	0.0951	0.0947
rsvd(k_2)	0.0549	0.0538	0.0554	0.0543	0.0544	0.0526	0.0525	0.0534	0.0538	0.0566
rsvd(k_3)	0.0444	0.0443	0.0449	0.0438	0.0440	0.0473	0.0468	0.0448	0.044	0.0438
rsvd(k_4)	0.0404	0.0400	0.0425	0.0408	0.0381	0.0387	0.0390	0.0377	0.0373	0.0412

表 3: time cost of different svd methods

svd	0.1513	0.1482	0.1460	0.1518	0.1512	0.1547	0.1565	0.1556	0.1466	0.1504
$ieee_{r1}$	0.0156	0.0153	0.0158	0.0162	0.0159	0.0159	0.0157	0.0157	0.0157	0.0159
$method1_{r1}$	0.0097	0.0103	0.0101	0.0103	0.0104	0.0096	0.0102	0.0093	0.0101	0.0106
$method2_{r1}$	0.0093	0.0101	0.0102	0.0097	0.0106	0.0101	0.0100	0.0093	0.0098	0.0096
$ieee_{r2}$	0.0200	0.0192	0.0194	0.0202	0.0195	0.0190	0.0197	0.0191	0.0194	0.0210
$method1_{r2}$	0.0136	0.0134	0.0134	0.0140	0.0141	0.0140	0.0155	0.0149	0.0131	0.0141
$method2_{r2}$	0.0140	0.0141	0.0150	0.0154	0.0138	0.0135	0.0153	0.0140	0.0141	0.0139
$ieee_{r3}$	0.0296	0.0298	0.0334	0.0311	0.0302	0.0314	0.0318	0.0319	0.0308	0.0318
$method1_{r3}$	0.0197	0.0193	0.0242	0.0207	0.0206	0.0219	0.0201	0.0206	0.0198	0.0211
$method2_{r3}$	0.0191	0.0201	0.0245	0.0198	0.0202	0.0196	0.0201	0.0200	0.0198	0.0195

表 4: ieee result of simu Indian(with salt and pepper noise)

	SVD		$ieee_{11}$		$ieee_{22}$		$ieee_{33}$		$ieee_{44}$	
	CTV	LRTV	CTV	LRTV	CTV	LRTV	CTV	LRTV	CTV	LRTV
PSNR	49.77	41.41	37.01	33.98	47.34	41.22	48.61	41.39	48.99	41.33
SSIM	0.99	0.99	0.98	0.96	0.99	0.99	0.99	0.99	0.99	0.99
ergas	0.50	1.28	2.15	2.96	0.66	1.31	0.57	1.29	0.55	1.30
times	62.52	65.69	54.59	60.23	51.60	60.12	48.06	60.87	49.68	61.47

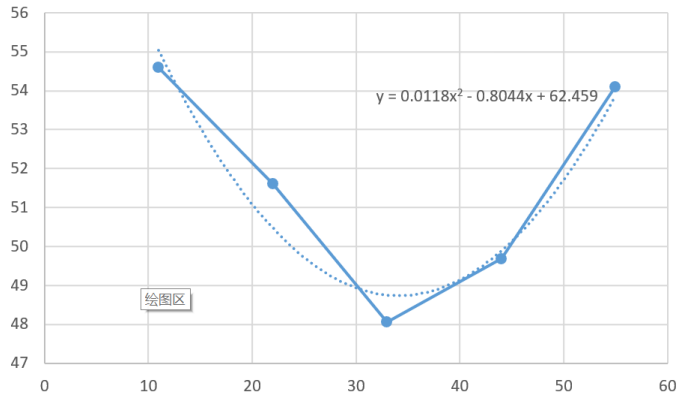

图 2: relationship bewteen R and times

可以看出降噪的时间随着 R 的增大先减少后增加, 利用二次函数拟合后, 其最小值应在 $R = 34$ 时取得, 约为 $0.15 * min(size(A))$, 对于其他数据集也有类的结论.

对于其余两种加速算法, 结果如下

表 5: method1 result of simu Indian(with salt and pepper noise)

	$method1_{11}$		$method1_{22}$		$method1_{33}$		$method1_{44}$	
	CTV	LRTV	CTV	LRTV	CTV	LRTV	CTV	LRTV
PSNR	40.20	34.46	47.53	41.13	48.66	41.09	48.99	41.21
SSIM	0.99	0.98	0.99	0.99	0.99	0.99	0.99	0.99
ergas	1.49	2.72	0.64	1.33	0.56	1.33	0.54	1.31
times	46.93	56.79	46.86	61.22	47.23	61.38	48.85	61.17

表 6: method2 result of simu Indian(with salt and pepper noise)

	$method2_{11}$		$method2_{22}$		$method2_{33}$		$method2_{44}$	
	CTV	LRTV	CTV	LRTV	CTV	LRTV	CTV	LRTV
PSNR	39.84	35.97	47.57	41.13	48.58	41.17	49.02	41.18
SSIM	0.99	0.98	0.99	0.99	0.99	0.99	0.99	0.99
ergas	1.55	2.33	0.65	1.33	0.57	1.32	0.55	1.33
times	46.31	57.82	46.74	61.07	46.37	61.49	48.74	60.90

结合实验结果可以看出, 当随机矩阵列数较小的时候, 虽然单次 SVD 的时间有较大的优化, 但是结果的收敛次数也相应的增加, 从而导致时间上出现不仅没有缩短反而增长的情况. 为了进一步提升效率, 一个可行的方法是根据程序进行的不同阶段对于该阶段内的随机 SVD 设置不同的 R 值. 通过实验发现, 在开始的几次迭代中, R 值得差别并不会导致迭代结果有较大的区别, 而在后期精度较高时, 较小的 R 值收敛的更慢, 因此合适的方法是对于目前程序达到的精度进行分段, 当精度较低时使用较小的 R 值加速收敛, 精度高时改用较大的 R 值。分别以 $C1 = 10^{-4}, C2 = 10^{-5}, C3 = 10^{-6}$ 为分界线, 对于数据集 simu indian 利用加速的 CTV 进行降噪, 得到的结果如下:

表 7: speed-change result of simu Indian(with salt and pepper noise)

	ieee C1	C2	C3	method1 C1	C2	C3	method2 C1	C2	C3
PSNR	40.78	39.17	38.57	42.41	40.83	40.77	42.40	41.36	40.29
SSIM	0.993	0.990	0.988	0.996	0.993	0.993	0.996	0.994	0.992
ergas	1.42	1.69	1.82	1.19	1.41	1.40	1.17	1.32	1.49
times	48.12	44.50	43.02	51.51	48.72	42.87	45.26	43.97	41.41

5 降噪实验结果

该部分展示了添加高斯噪声和椒盐噪声下三个数据集的降噪实验结果

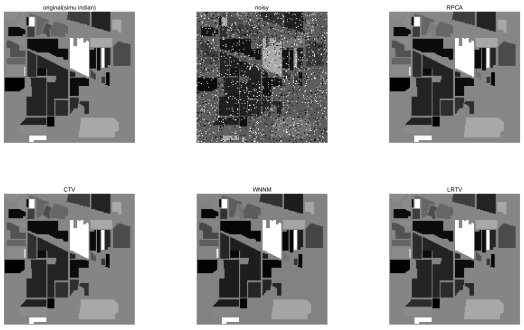

图 3: result of simu indian with Salt and Pepper noise(S=0.1)

表 8: evaluation of simu indian(with Salt and Pepper noise)

		RPCA	CTV	WNNM	LRTV
S=0.1	PSNR	43.1355	50.8164	38.9460	41.4424
	SSIM	0.9975	0.9994	0.9924	0.9949
	ergas	1.0243	0.4399	1.6783	1.2816
	times	4.0816	49.1001	5.2465	84.8005
S=0.2	PSNR	40.6649	50.8164	38.6259	41.3590
	SSIM	0.9964	0.9994	0.9918	0.9949
	ergas	1.3153	0.4399	1.7398	1.2733
	times	3.5699	43.6897	4.7321	87.9256
S=0.3	PSNR	37.5030	50.8164	36.5020	39.7694
	SSIM	0.9932	0.9994	0.9873	0.9937
	ergas	1.8696	0.4399	2.2095	1.5091
	times	10.0579	80.5749	4.5074	68.0389
S=0.4	PSNR	34.2944	50.8164	36.4194	35.7440
	SSIM	0.9857	0.9994	0.9864	0.9789
	ergas	2.7800	0.4399	2.2304	2.3611
	times	3.9074	46.0250	4.7211	90.6928

图 4: result of Ori H with Gaussian noise(G=0.1)

表 9: evaluation of OriH(with Gaussian noise)

		RPCA	CTV	WNNM	LRTV
G=0.1	PSNR	24.3402	27.5266	23.5210	26.3040
	SSIM	0.8406	0.8509	0.7098	0.7700
	ergas	12.5675	9.0085	15.0662	10.9359
	times	8.7499	99.4597	15.6153	87.6846
G=0.2	PSNR	21.9934	25.6089	21.4390	22.9227
	SSIM	0.7641	0.7821	0.6197	0.6116
	ergas	16.2932	11.2215	19.1472	16.1406
	times	7.8356	87.5609	17.9862	88.7730
G=0.3	PSNR	20.6760	24.5321	20.3728	22.8161
	SSIM	0.7094	0.7333	0.5552	0.6137
	ergas	18.8081	12.6412	21.6478	16.3399
	times	7.8755	94.4844	16.9421	86.0183
G=0.4	PSNR	19.7787	23.7677	20.2314	21.9550
	SSIM	0.6640	0.6944	0.5443	0.5632
	ergas	20.7796	13.7723	22.0031	18.0429
	times	7.9382	92.7861	17.8691	86.6167

图 5: result of Ori H with Salt and Pepper noise(S=0.1)

表 10: evaluation of OriH(with Salt and Pepper noise)

		RPCA	CTV	WNNM	LRTV
S=0.1	PSNR	43.8722	48.2578	41.7783	42.6209
	SSIM	0.9987	0.9992	0.9965	0.9943
	ergas	1.7119	0.9509	1.8393	1.6711
	times	9.2886	89.4334	12.0584	92.3794
S=0.2	PSNR	42.1348	47.0079	41.5237	21.3358
	SSIM	0.7641	0.7821	0.2112	0.5617
	ergas	0.9982	0.9990	0.9963	0.8656
	times	8.9039	88.7880	12.1055	68.6084
S=0.3	PSNR	40.4269	45.7588	41.2409	19.0970
	SSIM	0.9973	0.9986	0.9962	0.8403
	ergas	2.3833	1.2504	1.9928	17.7482
	times	8.9240	88.4497	12.0287	68.9703
S=0.4	PSNR	37.7006	43.9060	37.8073	17.4980
	SSIM	0.9945	0.9980	0.9903	0.7691
	ergas	3.0836	1.5203	3.0526	21.8102
	times	8.6999	89.0533	9.6192	69.2051

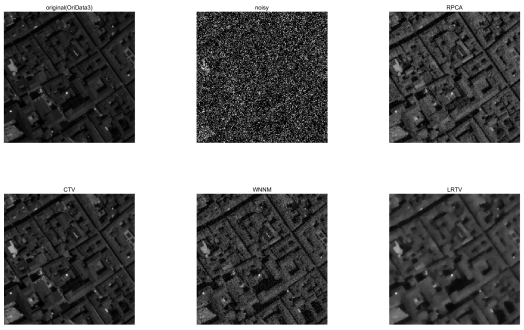

图 6: result of OriData3 with Gaussian noise(G=0.1)

表 11: evaluation of OriData3(with Gaussian noise)

		RPCA	CTV	WNNM	LRTV
G=0.1	PSNR	23.4205	47.3306	25.0086	23.9387
	SSIM	0.7278	0.9981	0.7239	0.5344
	ergas	15.0790	1.0527	13.3354	18.4379
	times	2.3309	32.4339	2.7046	46.7776
G=0.2	PSNR	21.0794	47.3696	23.2616	22.8655
	SSIM	0.6094	0.9981	0.6217	0.4527
	ergas	19.6560	1.0502	15.3276	16.2875
	times	2.0031	28.5206	2.6317	52.3192
G=0.3	PSNR	19.7970	47.3692	22.0327	22.0545
	SSIM	0.5363	0.9981	0.5447	0.3908
	ergas	22.7100	1.0494	17.5780	17.8880
	times	2.7402	32.7798	2.7742	53.5267
G=0.4	PSNR	18.9516	47.3895	21.1837	22.0407
	SSIM	0.4773	0.9981	0.4931	0.3970
	ergas	25.0006	1.0477	19.3766	17.6883
	times	2.0333	30.0052	2.8566	52.6576

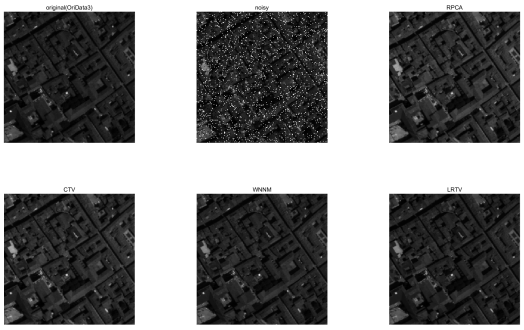

图 7: result of OriData3 with Salt and Pepper noise(S=0.1)

表 12: evaluation of OriData3(with Salt and Pepper noise)

		RPCA	CTV	WNNM	LRTV
S=0.1	PSNR	39.0419	47.3878	44.0895	38.6802
	SSIM	0.9961	0.9981	0.9959	0.9849
	ergas	2.3917	1.0477	1.3745	2.7051
	times	2.2512	28.9894	2.8537	45.7612
S=0.2	PSNR	36.6363	47.3914	43.9010	36.1037
	SSIM	0.9935	0.9981	0.9958	0.9741
	ergas	3.1349	1.0474	1.4145	3.5746
	times	2.2039	29.0117	2.8254	46.2757
S=0.3	PSNR	34.6662	47.3652	43.6393	33.8382
	SSIM	0.9889	0.9981	0.9955	0.9561
	ergas	3.9272	1.0500	1.4650	4.6173
	times	2.2381	29.9864	2.8736	47.1214
S=0.4	PSNR	32.3781	47.3662	42.2230	31.5482
	SSIM	0.9794	0.9981	0.9942	0.9289
	ergas	5.1258	1.0498	1.7412	5.9269
	times	2.1566	29.7949	2.7946	47.2150