PCA 和流形学习算法

学号：PB21000224　　　　姓名：陈鸿绪　　　　日期：12.23

一. 背景理论：

PCA（主成分分析）：

1. 协方差矩阵：PCA 的目标是最大化数据中的方差。通过计算协方差矩阵来衡量数据之间的相关性，协方差矩阵中的每个元素表示两个特征之间的相关性。PCA 通过对协方差矩阵进行特征值分解，得到特征值和特征向量，其中特征值较大的特征向量对应的方向就是数据的主成分。
2. 算法原理：PCA 通过对原始特征矩阵进行特征值分解，得到特征值和特征向量。其中，特征值较大的特征向量对应的方向就是数据的主成分。因此，主成分的数目可以通过保留前 k 个最大特征值对应的特征向量得到，其中 k 是预定的主成分数目。
3. PCA 的主要应用：PCA 是一种广泛使用的降维技术，其应用场景包括图像压缩、人脸识别、文本挖掘、推荐系统等。通过降维技术可以将高维度的数据转化为低维度的数据，从而更好地理解和分析数据的内在规律和结构。

MDS（多维缩放）：

1. 基本思想: 是将高维空间中的数据点投影到低维空间中，同时保持数据点之间的距离关系尽可能地接近原始数据点之间的距离。MDS 的目标是使得降维后的数据点在低维空间中的位置能够反映它们在高维空间中的相似性。
2. 算法原理：MDS 的理论基础包括距离度量和相似性度量。在 MDS 中，数据点之间的距离通常使用欧氏距离或余弦相似度等度量方式来计算。为了在降维后的空间中保持原始数据点之间的距离关系，MDS 采用优化算法来最小化所有点对之间的距离误差的平方和。
3. MDS 的主要应用：现在广泛应用于社会科学、生物学、信息科学等多个领域。例如，在生物学中，MDS 算法可以用于分析基因表达数据的相似性和差异性；在信息科学中，MDS 算法可以用于数据挖掘和知识发现。

二． 实验步骤：

a. 数据处理：

1. 读取 mnist 数据集，查看数据集的部分数据。
2. 将数据集分割成标签和属性两部分。

b. PCA 的算法实现步骤：

1. 标准化数据(即采用中心化让样本重心在原点)。
2. 计算数据集协方差矩阵。
3. 对协方差矩阵进行特征值分解、选择主成分并降维。在选择主成分时，通常保留前 k 个最大的特征值对应的特征向量，构建出新的矩阵来近似原始数据集主成分。
4. 将原始数据投影到这个新的矩阵上，从而实现降维。
5. 最后，将 Sklearn 的 PCA 降维的结果和以上实现的 PCA 降维结果比较。（实验中分别选取了 2 个与 3 个主成分）

c. MDS 的算法实现步骤：

1. 计算高维空间中样本之间的距离，得到距离矩阵（可以是欧式距离，也可以采用其他距离函数）。
2. 使用特征值分解，得到对角阵和正交向量组。
3. 选取对角阵中最大的若干个特征值和对应的向量，将 $X'\Lambda'X'^T$ 分解成 ZZ^T ,即有 $Z = X'\Lambda'^{\frac{1}{2}}$ 。得到高维坐标对应的低维坐标。
4. 选取不同主成分实验中分别选取了 2 个与 3 个主成分），结合标签，进行低维向量绘图。

三． 关键代码：

1. PCA 算法的实现：

```
def PCA(X,dim=2) :

    mean_df = X.mean()
```

```
    #为了样本中心化计算样本重心位置
    centred_df = X.sub(mean_df,axis=1)
    #进行样本中心化
    cov_centred_df = np.dot(centred_df.T,centred_df)
    #计算协方差矩阵
    PCA_eigenvalues,PCA_eigenvector = np.linalg.eigh(cov_centred_df)
    #得到向量组是正交矩阵的特征值分解
    PCA_eigenvector_T = PCA_eigenvector.T
    #向量组需要转置
    PCA_eigenvector_T = np.flip(PCA_eigenvector_T,axis=0)
    PCA_eigenvector_T_cut = PCA_eigenvector_T[:dim]
    #选取前 dim 个特征值最大对应的特征向量
    PCA_Result = pd.DataFrame(np.dot(centred_df,PCA_eigenvector_T_cut.T))
    #将矩阵作用在原来的数据集上
    return PCA_Result
    #返回低维坐标结果
```

2. MDS 算法的实现：

```
def MDS(df, dim=2) :
    B = np.array([np.zeros(len(df))]*len(df))
    #初始化低维空间的内积矩阵
    dist_X = np.array([np.zeros(len(df))]*len(df))
    #初始化距离矩阵
    centred_df_np = np.array(centred_df)
    for i in range(len(df)) :
```

```
        for j in range(len(df)) :
            #以下距离矩阵中都取了平方是为了后续不用计算过程中再取平方
            dist_X[i][j] = np.sum(abs(centred_df_np[i]-centred_df_np[j])**6)**(1/3)
            #上面采用欧式距离衡量高维空间的样本距离
            #dist_X[i][j] = np.max(centred_df_np[i]-centred_df_np[j])**2
            #dist_X[i][j] = np.sum((centred_df_np[i]-centred_df_np[j])**2)
            #dist_X[i][j] = np.sum(np.sum(centred_df_np[i]-centred_df_np[j]))**2
            #以上采取其他的距离度量
    dist_X_sum = dist_X.sum(axis = 0)/len(df)
    dist_X_sum_all = dist_X.sum()/(len(df)**2)
    for i in range(len(df)) :
        for j in range(len(df)) :
            B[i][j] = -0.5*(dist_X[i][j]-dist_X_sum[j]-dist_X_sum[i]+dist_X_sum_all)
    #根据公式计算内积矩阵 B
    MDS_eigenvalues,MDS_eigenvector = np.linalg.eigh(B)
    #得到向量组是正交矩阵的特征值分解
    MDS_Result = np.dot(np.diag(np.sqrt(np.flip(MDS_eigenvalues)[0:dim])),np.flip(MDS_eigenvector.T,axis=0)[0:dim]).T
    #通过 B 的分解得到 Z 的矩阵，即获得高维坐标对应的低维坐标
    return MDS_Result
    #返回低维坐标结果
```

四． 结果分析和结论：

1. PCA 降维实验结果：

	0	1
0	-673.858840	-29.990507
1	-254.873896	-936.709765
2	-358.126501	781.144783
3	-867.130962	-358.526281
4	-582.996280	-934.002072
...	...	...
403	-440.586048	-712.932881
404	255.758636	55.149979
405	294.652460	-494.401949
406	-721.124659	1058.315331
407	688.723540	45.993653

本人 PCA 算法实验结果（二维）

	0	1
0	-673.859423	29.990567
1	-254.874155	936.707884
2	-358.127160	-781.146024
3	-867.132434	358.525890
4	-582.995521	934.001496
...	...	...
403	-440.586076	712.932230
404	255.758177	-55.150727
405	294.653182	494.400511
406	-721.125334	-1058.316185
407	688.723080	-45.994028

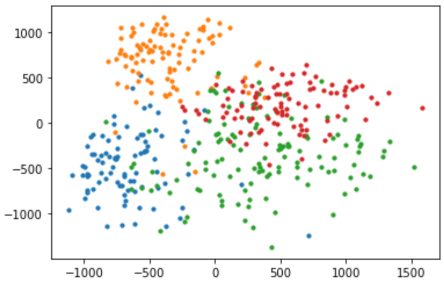
Sklearn 库中 PCA 算法实验结果（二维）

	0	1	2
0	-673.858840	-29.990507	314.968169
1	-254.873896	-936.709765	20.991886
2	-358.126501	781.144783	607.774357
3	-867.130962	-358.526281	-268.695038
4	-582.996280	-934.002072	185.565198
...	...	...	...
403	-440.586048	-712.932881	183.953253
404	255.758636	55.149979	28.615792
405	294.652460	-494.401949	190.933909
406	-721.124659	1058.315331	89.133345
407	688.723540	45.993653	-445.437476

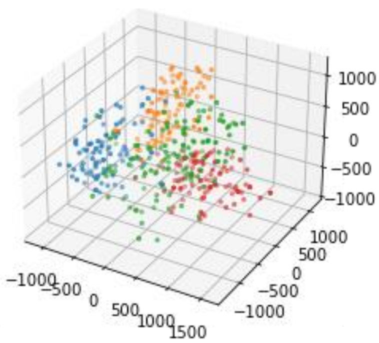
本人 PCA 算法实验结果（三维）

	0	1	2
0	-673.858793	29.990473	314.944120
1	-254.873741	936.710645	21.085634
2	-358.125779	-781.145779	607.768561
3	-867.130168	358.528369	-268.321378
4	-582.996082	934.000799	185.414423
...	...	...	...
403	-440.585998	712.931686	183.847050
404	255.758940	-55.151568	28.480088
405	294.652318	494.402184	190.909205
406	-721.124947	-1058.314486	89.177537
407	688.723971	-45.993367	-445.339769

Sklearn 库中 PCA 算法实验结果（三维）



降维到二维的可视化



降维到三维的可视化

分析与结论：对比两者结果，可以发现两者的误差大部分在 1e-3 精度以内，两者降维得到的坐标绝对值基本一致，误差主要是特征值分解时 Sklearn 和本人实现算法中采用的不同迭代算法或终止条件的差异导致。正负号的差异则是由于特征值分解的特征向量可以有不同的值。在最后可视化结果中，无论是二维还是三维，虽然可以明显看到四个都各自形成了簇，但发现标签的四种类别重叠的部分很多，说明 PCA 降维提取主成分之后并没有有效将差别拉开。最后降维结果并不是很理想。

2. MDS 降维实验结果

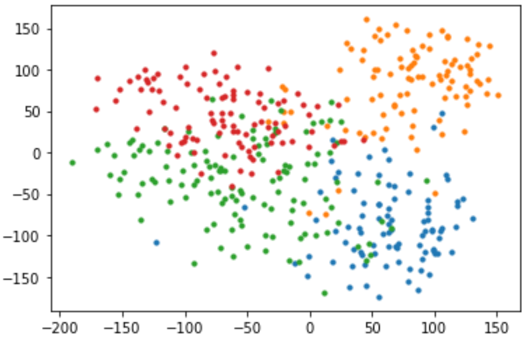
以下实验中，高维空间的距离度量采用的是向量的 6 范数：

	0	1
0	79.653134	-25.984306
1	-1.269283	-125.901484
2	78.880410	101.381580
3	65.122259	-81.119057
4	34.454844	-136.278325
...	...	...
403	34.760867	-115.946111
404	-34.186077	9.565877
405	-45.074975	-54.032979
406	133.719213	118.083273
407	-97.498400	31.176169

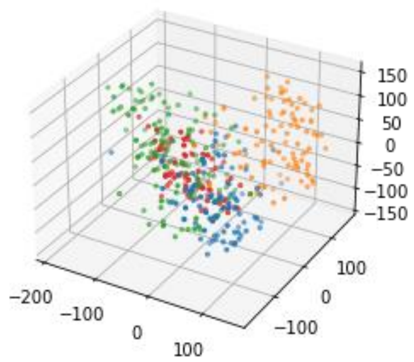
本人 MDS 算法实验结果（二维）

	0	1	2
0	79.653134	-25.984306	31.851668
1	-1.269283	-125.901484	-5.659310
2	78.880410	101.381580	81.057249
3	65.122259	-81.119057	2.742145
4	34.454844	-136.278325	41.163370
...	...	...	...
403	34.760867	-115.946111	5.712740
404	-34.186077	9.565877	4.113060
405	-45.074975	-54.032979	24.598715
406	133.719213	118.083273	39.620505
407	-97.498400	31.176169	-87.697376

本人 MDS 算法实验结果（三维）



降维到二维的可视化



降维到三维的可视化

分析与结论：实际上在构造距离矩阵的时候，如果高维空间采取欧式距离度量，则情况就会退化到 PCA 降维，和之前的实验结果一致。所以这里实验采用的是 6 范数的距离度量（其他 p 范数也可以）。可视化结果可以发现无论是二维还是三维，虽然可以明显看到四

个都各自形成了簇，但发现标签的四种类别重叠部分相对较多。所以对于该数据集，MDS 也不是很好的降维方式。在换了其他的距离度量情况下，本人也分别进行了实验分析，但是结果不理想，发现对该数据集，只有在 p 范数距离度量下，实验结果才相对比较理想。

3. t-SNE 降维实验结果

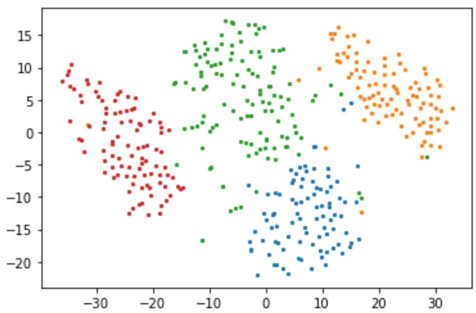
t-SNE 算法本人没有手动实现，而是使用 Sklearn 库中现有函数，得到以下实验结果。

	0	1
0	-1.508494	-22.028236
1	0.277970	-9.516426
2	29.076769	-0.030416
3	14.920293	-17.671289
4	3.829377	-16.127720
...	...	...
403	1.214772	-12.650958
404	-9.841883	-5.521832
405	-1.140674	6.568618
406	22.503975	7.189423
407	-24.148630	-9.120502

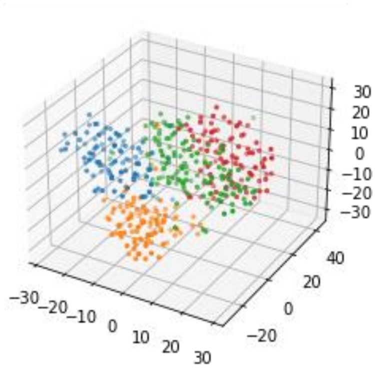
Skearn t-SNE 结果（二维）

	0	1	2
0	-1.135612	-15.578084	-0.808352
1	-1.458159	-9.917426	16.609097
2	6.067926	-23.972282	-10.119480
3	-28.185875	-16.074564	18.112934
4	-8.623219	-20.887428	15.802938
...	...	...	...
403	-3.606115	-16.048019	14.117353
404	11.627268	5.200423	-12.811044
405	6.241803	-1.720730	5.645647
406	-15.141407	-13.969590	-18.219048
407	8.732494	37.091667	4.003466

Skearn t-SNE 结果（三维）



降维到二维的可视化



降维到三维的可视化

分析与结论：无论是二维还是三维，可以发现 t-SNE 的实验结果非常理想，不同类别簇之间的距离被拉得相对较远，基本没有重叠的部分，所以基本可以认定 t-SNE 在此数据集上的效果是三者中最好的。且由此可以合理猜测：对于 PCA 降维不能有效拉开不同簇之间距离的情况，t-SNE 算法可以将不同簇之间距离有效控制从而避免过多重叠。