

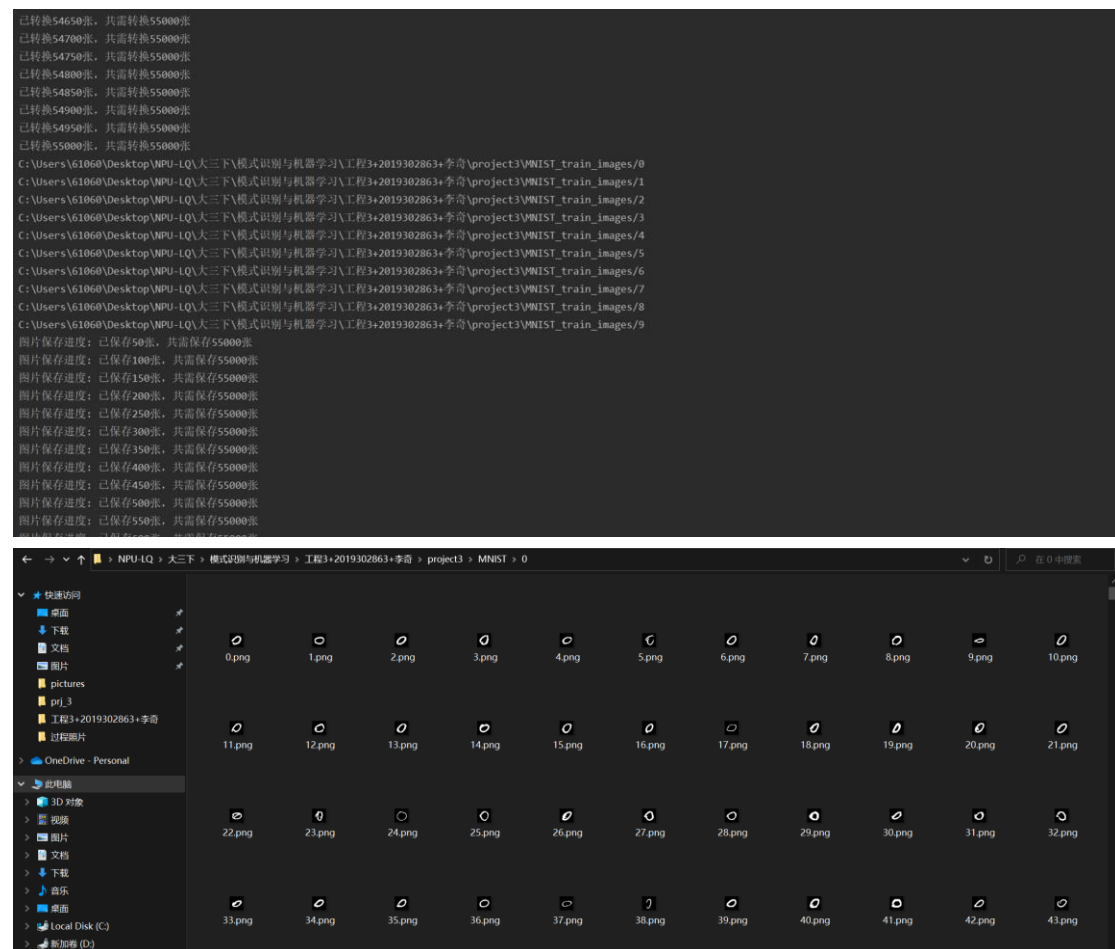
本次工程中我通过 sklearn 在 MNIST 数据集中对 DBSCAN、K-Means 等聚类方法进行了探索，并进行了可视化，展示了测试结果，在此之前，首先对 MNIST 数据集进行了预处理，工程总代码量一千行左右。

一、对实现的整体说明

1) MNIST_imageExtracting.py:

```
1  # -*- coding: utf-8 -*-
2  # @Time    : 2022/6/5 16:17
3  # @Author   : Li Qi
4  # @FileName: MNIST_imageExtracting.py
5  # @Function: Extract images from mnist and save as bmp type.
6
7  import ...
10
11  '''
12      mnist_dir    mnist数据集存储的路径
13      save_dir     提取结果存储的目录
14  '''
15  def extract_mnist(mnist_dir, save_dir):...
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78  if __name__ == '__main__':
79      mnist_dir = r"C:\Users\61060\Desktop\NPU-LQ\大三下\模式识别与机器学习\工程3+2019302863+李奇\project3"
80      save_dir = r"C:\Users\61060\Desktop\NPU-LQ\大三下\模式识别与机器学习\工程3+2019302863+李奇\project3\MNIST"
81      extract_mnist(mnist_dir, save_dir)
```

该.py 文件实现了根据原始数据集得到完整的 png 格式的 MNIST 数据集中并存储在指定文件夹下:














2) MNIST_imageFlatten.py:

```

1  # -*- coding: utf-8 -*-
2  # @Time    : 2022/6/5 18:54
3  # @Author   : Li Qi
4  # @FileName : MNIST_imageFlatten.py
5  # @Function : Save bmp image as txt with 1x784 array from 0 to 9
6  # -*- coding: utf-8 -*-
7  import ...
11
12  #此函数读取特定文件夹下的bmp格式图像
13  def get_imlist(path):...
16
17  def img_to_txt():...
38
39  img_to_txt()

```

该.py 文件实现了将提取出的照片转换为 1×784 的一维向量形式并存储在 txt 文件中，每一类存储在同一个 txt 文件，共计 10 个，还有一个 txt 文件存储标签：

 num_0.txt	2022/6/5 23:21	文本文档	185 KB
 num_1.txt	2022/6/5 23:21	文本文档	169 KB
 num_2.txt	2022/6/5 23:21	文本文档	181 KB
 num_3.txt	2022/6/5 23:21	文本文档	181 KB
 num_4.txt	2022/6/5 23:21	文本文档	177 KB
 num_5.txt	2022/6/5 23:21	文本文档	180 KB
 num_6.txt	2022/6/5 23:21	文本文档	180 KB
 num_7.txt	2022/6/5 23:21	文本文档	174 KB
 num_8.txt	2022/6/5 23:21	文本文档	183 KB
 num_9.txt	2022/6/5 23:21	文本文档	176 KB
 labels.txt	2022/6/5 23:21	文本文档	3 KB

3) DBSCAN&KMEANS final.py:

The screenshot shows the PyCharm IDE interface. The top menu bar includes File, Edit, View, Navigate, Code, Refactor, Run, Tools, VCS, Window, and Help. The top toolbar shows icons for saving, running, and other actions. The left sidebar displays the project structure for 'project3 [prj3]' located at 'C:\Users\61060\Desktop\NPU-1.0'. The main editor window shows the code for 'DBSCAN&KMEANS_final.py'. The code includes imports for 'random' and 'warnings', and defines functions for 'top1', 'MyAccuracyAndGeneralResults', 'GenerateMeasures', 'ViewDigit', and 'GenerateMeasures_2'. The bottom status bar indicates the current file is 'DBSCAN&KMEANS_final.py' and the interpreter is 'C:\Users\61060\Anaconda3\envs\pytorch\python.exe'.

该.py 文件对 DBSCAN、KMEANS、Gaussian Mixture 等聚类方法进行了探索，并对过程中的有效信息进行了可视化。包括以下几个阶段：数据读取、显示错误类别、降维手段测试、聚类方法的评价指标、聚类结果示例。

4) KMEANS_draft.py

该.py 文件按照 sklearn 中文手册中所写对 KMEANS 进行了初探。

二、具体实现描述

1) 数据集处理

在 Lecun 的网站上下载好 MNIST 数据集后，可以看到一共包含四个文件夹，每个文件夹下存储着同名数据文件，其格式如下所示：

■ t10k-images-idx3-ubyte	2022/6/7 14:00	文件夹
■ t10k-labels-idx1-ubyte	2022/6/7 14:00	文件夹
■ train-images-idx3-ubyte	2022/6/7 14:00	文件夹
■ train-labels-idx1-ubyte	2022/6/7 14:00	文件夹

```
TRAINING SET LABEL FILE (train-labels-idx1-ubyte):
[offset] [type]          [value]             [description]
0000     32 bit integer  0x00000801(2049)    magic number (MSB first)
0004     32 bit integer  60000               number of items
0008     unsigned byte  ??                  label
0009     unsigned byte  ??                  label
.....
xxxx     unsigned byte  ??                  label
The labels values are 0 to 9.
```

```
TRAINING SET IMAGE FILE (train-images-idx3-ubyte):
[offset] [type]          [value]             [description]
0000     32 bit integer  0x00000803(2051)    magic number
0004     32 bit integer  60000               number of images
0008     32 bit integer  28                  number of rows
0012     32 bit integer  28                  number of columns
0016     unsigned byte  ??                  pixel
0017     unsigned byte  ??                  pixel
.....
xxxx     unsigned byte  ??                  pixel
Pixels are organized row-wise. Pixel values are 0 to 255. 0 means
background (white), 255 means foreground (black).
```

```
TEST SET LABEL FILE (t10k-labels-idx1-ubyte):
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000801(2049) magic number (MSB first)
0004     32 bit integer  10000                number of items
0008     unsigned byte   ??                label
0009     unsigned byte   ??                label
.....
xxxx     unsigned byte   ??                label
The labels values are 0 to 9.
```

```
TEST SET IMAGE FILE (t10k-images-idx3-ubyte):
[offset] [type]          [value]          [description]
0000     32 bit integer  0x00000803(2051) magic number
0004     32 bit integer  10000                number of images
0008     32 bit integer  28                number of rows
0012     32 bit integer  28                number of columns
0016     unsigned byte   ??                pixel
0017     unsigned byte   ??                pixel
.....
xxxx     unsigned byte   ??                pixel
Pixels are organized row-wise. Pixel values are 0 to 255. 0 means
background (white), 255 means foreground (black).
```

之后我们通过 `extract_mnist()` 函数(MNIST_imageExtracting.py)对原始数据集进行处理, 首先加载并通过训练标签数量来得到训练图片数量:

```
19 # 加载mnist数据集
20 mnist = input_data.read_data_sets(mnist_dir)
21 # 获取训练图片数量
22 shape = mnist.train.images.shape
23 images_train_count = shape[0]
24 pixels_count_per_image = shape[1]
25 # 获取训练标签数量=训练图片数量
26 labels = mnist.train.labels
27 labels_train_count = labels.shape[0]
```

之后输出相应信息, 并创建数据的保存目录, 按标签保存:

```
29 if (images_train_count == labels_train_count):
30     print("训练集共包含%d张图片, %d个标签" % (images_train_count, labels_train_count))
31     print("每张图片包含%d个像素" % (pixels_count_per_image))
32     print("数据类型为", mnist.train.images.dtype)
33
34     # mnist图像数值的范围为[0,1], 需将其转换为[0,255]
35     for current_image_id in range(images_train_count):
36         for i in range(pixels_count_per_image):
37             if mnist.train.images[current_image_id][i] != 0:
38                 mnist.train.images[current_image_id][i] *= 255
39                 print(mnist.train.images[current_image_id][i])
40
41         if ((current_image_id + 1) % 50) == 0:
42             print("已转换%d张, 共需转换%d张" %
43                   (current_image_id + 1, images_train_count))
```

```
45 # 创建train images的保存目录,按标签保存
46 for i in range(10):
47     dir = "%s/%s" % (save_dir, i)
48     print(dir)
49     if not os.path.exists(dir):
50         os.mkdir(dir)
```

之后我们获得每个标签对应的图片数量,开始对照片进行存储,将他们存储在对应的文件夹下,存储过程如下所示:

```
52 # indices = [0, 0, 0, ..., 0]用来记录每个标签对应的图片数量
53 indices = [0 for x in range(0, 10)]
54 for i in range(images_train_count):
55     new_image = Image.new("L", (cols, rows))
56     # 遍历new_image 进行赋值
57     for r in range(rows):
58         for c in range(cols):
59             new_image.putpixel(
60                 (r, c), int(mnist.train.images[i][c + r * cols]))
61
62     # 获取第i张训练图片对应的标签
63     label = labels[i]
64     image_save_path = "%s/%s/%s.png" % (save_dir, label,
65                                         indices[label])
66     indices[label] += 1
67     new_image.save(image_save_path)
68
69     # 打印保存进度
70     if ((i + 1) % 50) == 0:
71         print("图片保存进度: 已保存%d张, 共需保存%d张" % (i + 1, images_train_count))
72 else:
73     print("图片数量与标签数量不一致!")
```

获取完整的 png 格式数据集后,我们按照实验要求,选取每类 100 个共 1000 个数据进行实验,为了提取出 1000 个数据,我们的思路是逐类别进行提取:在 img_to_txt()函数(MNIST_imageFlatten.py)中,每次提取时,先通过 get_imlist()函数(MNIST_imageFlatten.py)传输文件夹路径,之后建立(图像个数 \times (28 \times 28))的矩阵,打开图像后,将图像转化为数组,并将图像的矩阵形式转化为一维数组保存到先前创建的矩阵中,之后选取当前类别的 100 个图像,将他们转换成 1 \times 784 的格式存储到 txt 文件中。至于类标,由于此时的数据顺序没有被打乱,可以直接通过循环写入实现:

```
12 #此函数读取特定文件夹下的bmp格式图像
13 def get_imlist(path):
14
15     return [os.path.join(path,f) for f in os.listdir(path) if f.endswith('.png')]
16
17 def img_to_txt():
18     for n in range(10):
19         c = get_imlist(r"C:\Users\61060\Desktop\NPU-LQ\大三下\模式识别与机器学习\工程3+2019302863+李奇\project3\MNIST\%d"%n) #.r""是防止字符串转译
20         print(c) # 这里以list形式输出bmp格式的所有图像(带路径)
21         d = len(c) # 这可以以输出图像个数
22         data = numpy.empty((d, 28 * 28)) # 建立d*(28*28)的矩阵
23         while d > 0:
24             img = Image.open(c[d - 1]) # 打开图像
25             img_ndarray = numpy.asarray(img, dtype='float64') # 将图像转化为数组
26             data[d - 1] = numpy.ndarray.flatten(img_ndarray) # 将图像的矩阵形式转化为一维数组保存到data中
27             d = d - 1
28             print(data)
29
30         with open('num_%d.txt'%n, 'ab') as f:
31             for i in range(100):
32                 A = numpy.array(data[i]).reshape(1, 784)
33                 savetxt(f, A, fmt="%0.0f") # 将数据保存到txt文件中
34         with open("labels.txt", "w") as f:
35             for i in range(10):
36                 for j in range(100):
37                     f.write('%d\n' % i)
```

至此，对于数据集的处理已经完成。

2) 实验过程

我们首先将先前处理好的 txt 见保存的数据读取进来，并将数据打乱、拆分、归一化，得到可以直接使用的特征和标签集合：

```

84     print("##### 数据读取阶段 #####")
85     X = []
86     y = []
87     for i in range(10):
88         a = np.loadtxt('num_%d.txt'%i)
89         X = np.append(X,a)
90     X = np.reshape(X,(1000,784))
91     y = np.loadtxt('labels.txt')
92     X = np.float64(X)
93     y = np.float64(y)
94     random_state = check_random_state(0) # 随机数种子
95     permutation = random_state.permutation(X.shape[0]) # 随机排列数组，打乱样本
96     X = X[permutation]
97     y = y[permutation]
98     X = X.reshape((X.shape[0]-1))
99     X = pd.DataFrame(data=X)
100    y = pd.Series(data=y)
101    X_train,X_test,y_train,y_test = train_test_split(X,y,test_size=0.1) # 拆分验证集
102
103    scaler = StandardScaler() # 数据归一化
104    x_train = scaler.fit_transform(X_train)
105    x_test = scaler.transform(X_test)
106    print(f'Dimensions of Training Set: {X_train.shape}')
107    print(f'Dimensions of Labels for the Training Set: {y_train.shape}')
108    print(f'Dimensions of Testing Set: {X_test.shape}')
109    print(f'Dimensions of Labels for the Testing Set: {y_test.shape}')

```

```

##### 数据读取阶段 #####
Dimensions of Training Set: (900, 784)
Dimensions of Labels for the Training Set: (900,)
Dimensions of Testing Set: (100, 784)
Dimensions of Labels for the Testing Set: (100,)

```

实验要求跟真实类别对比，展示至少 1 个不正确的结果，猜测原因。所以我们设置了显示错误类别模块，在这里我们重新读取了数据，操作流程同上，区别在于这里没有打乱顺序，之后用 t-SNE 将数据降维，用降维后的数据对 kmeans 进行训练，得到聚类标签：

```

127    pca = PCA(n_components=784, random_state=23003)
128    XX_PCA = pca.fit_transform(XX)
129    XX_PCA = pd.DataFrame(data=XX_PCA)
130
131    n_components = 2
132    learning_rate = 300
133    perplexity = 30
134    early_exaggeration = 12
135    init = 'random'
136
137    tSNE = TSNE(n_components=n_components, learning_rate=learning_rate,
138                perplexity=perplexity, early_exaggeration=early_exaggeration,
139                init=init, random_state=random_state)
140    XX_train_tSNE = tSNE.fit_transform(XX_PCA.loc[:,99])
141    XX_train_tSNE = pd.DataFrame(data=XX_train_tSNE)
142
143    kmeans = KMeans(n_clusters=10, n_init=10,
144                   max_iter=300, tol=0.0001, random_state=random_state,
145                   n_jobs=2)
146    kmeans = kmeans.fit(XX_train_tSNE)
147    yy_pre = kmeans.predict(XX_train_tSNE)

```

由于聚类标签是随机给的，并不是按照原始的手写数字的 0~9 排列，比如原来标签是 1 的数据，在聚类后得到的标签可能是 4，因此我们需要将两者对应，由于数据顺序未被打乱，因此我们可以每一百个数据处理一次，通过 top1()函数(DBSCAN&KMEANS_final.py)得到这一百个数据中出现次数最多的标签并存储下来，最终会得到 10 个标签，将他们存储在一个 list 中，他们在 list 中的索引就代表原数据的标签：

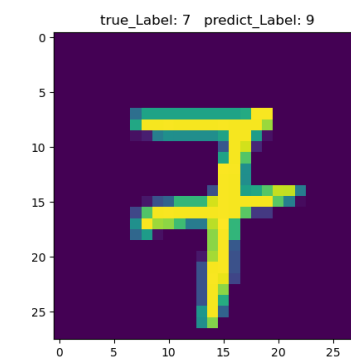
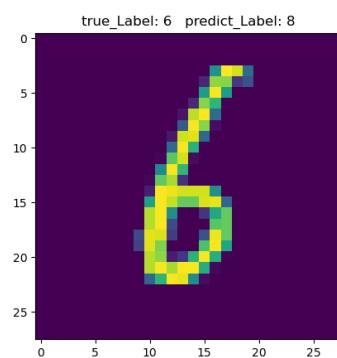
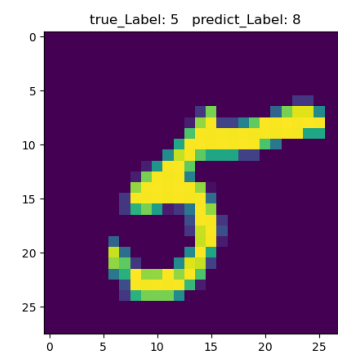
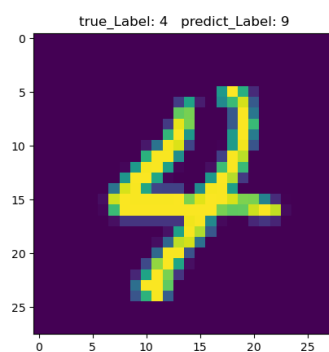
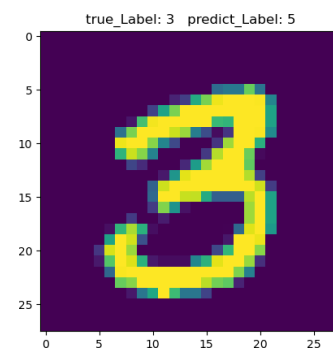
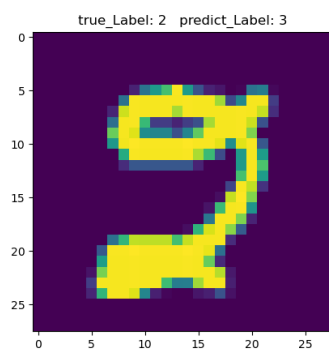
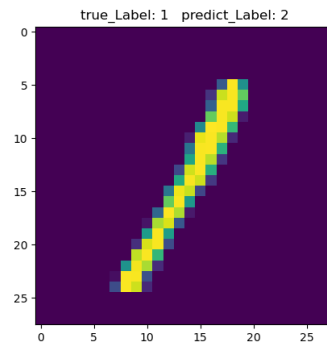
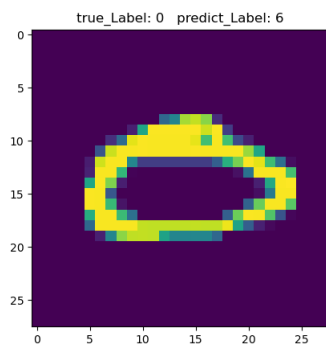
```
150     print("yy_pre: ")
151     print(yy_pre)
152     maxlabel = []
153     mid = (yy_pre[0:100]).tolist()
154     maxlabel.append(top1(mid))
155
156     mid = (yy_pre[100:200]).tolist()
157     maxlabel.append(top1(mid))
158
159     mid = (yy_pre[200:300]).tolist()
160     maxlabel.append(top1(mid))
161
162     mid = (yy_pre[300:400]).tolist()
163     maxlabel.append(top1(mid))
164
165     mid = (yy_pre[400:500]).tolist()
166     maxlabel.append(top1(mid))
167
168     mid = (yy_pre[500:600]).tolist()
169     maxlabel.append(top1(mid))
```

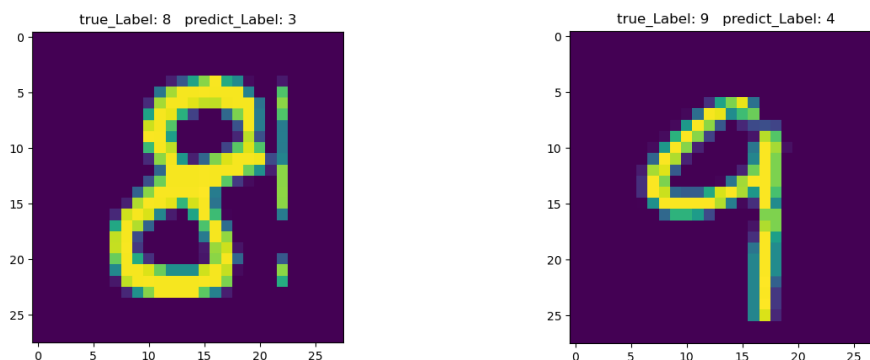
```
maxlabel:
[4, 7, 3, 6, 8, 9, 0, 5, 2, 1]
```

在得到标签的对应关系后，我们每一类选取一个错误图像进行可视化，可视化由 ViewDigit_W()函数(DBSCAN&KMEANS_final.py)来实现：

```
67     def ViewDigit_W(example_i, predict_label, X, y):
68         label = y.loc[example_i]
69         image = X.loc[example_i:].values.reshape([28,28])
70         image = np.fliplr(image)
71         image = np.rot90(image, 1)
72         plt.title(f'true_Label: {int(label)}    predict_Label: {predict_label}')
73         plt.imshow(image)
74         plt.show()
```

```
186     wrong0 = []
187     wrong0_label = []
188     mid = yy_pre[0:100]
189     for i in range(100):
190         if mid[i] != maxlabel[0]:
191             wrong0.append(i)
192             wrong0_label.append(mid[i])
193     print("wrong0: ")
194     print(wrong0)
195     print("wrong0_label: ")
196     print(wrong0_label)
197     print("第一个错误示例显示中...")
198     ViewDigit_W(wrong0[1], maxlabel.index(wrong0_label[1]), X, yy)
```



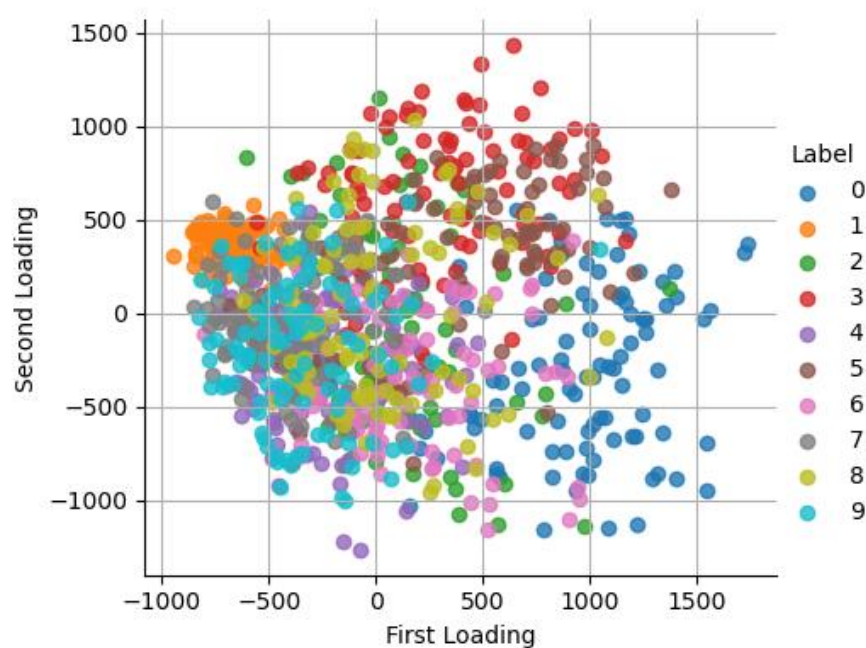
可以看到 10 个被错误分类的例子，选取几个分析：0 被预测成 6，可能是由于它在书写的时候收笔时多出来一块尾巴，使得上面比较的厚，同时尾巴与 0 之间还有缝隙，看起来像 6；2 预测成 3，可能是起笔时带了一个弯曲部分，看起来像 3；4 预测成 9，可能是书写时那两笔在桑放趋于闭合；9 预测成 4 则相反，由于书写的 9 没有完全闭合，导致看起来像 4。

接下来是对降维的探索，我们使用了 PCA、SVD、t_SNE 等三个降维方法，他们在 sklearn 中均有相应实现，结果如下所示：

PCA:

简单地说，PCA 的数学定义是：一个正交化线性变换，把数据变换到一个新的坐标系中，使得这一数据的任何投影的第一大方差在第一个坐标（称为第一主成分）上，第二大方差在第二个坐标（第二主成分）上，依次类推。

```
##### 降维手段：主成分分析(PCA) #####
Variance explained by all 784 principal components: 1.0000000000000007
Variance explained by first 10 principal components: [0.48279898]
Variance explained by first 20 principal components: [0.6473502]
Variance explained by first 100 principal components: [0.9255185]
Variance explained by first 150 principal components: [0.95928952]
Variance explained by first 200 principal components: [0.97626926]
Variance explained by first 250 principal components: [0.98631086]
```



SVD:

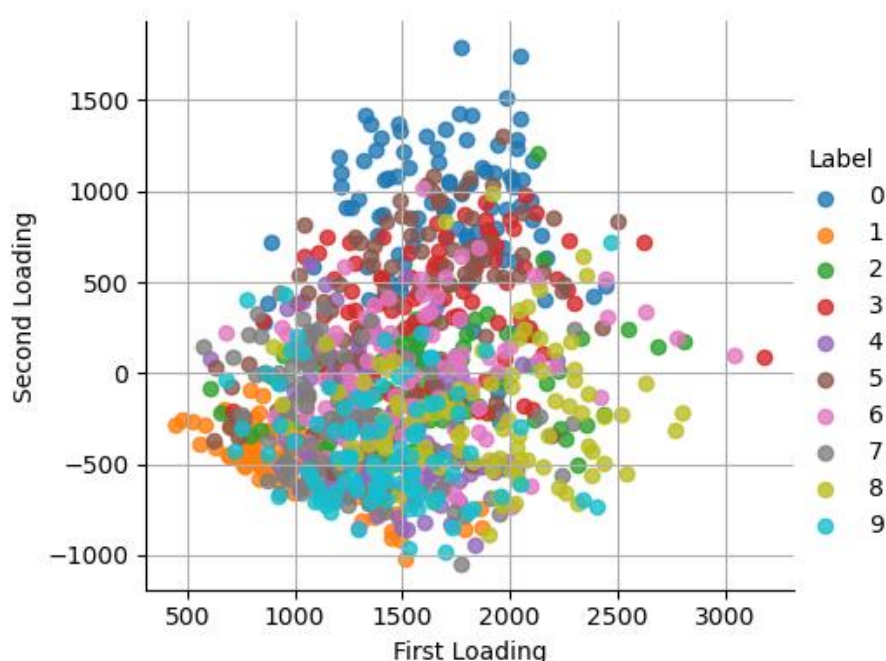
奇异值分解（SVD）是一种允许显式分析阿达玛条件的工具，也是数据分析中降维的一种好方法。矩阵论告诉我们，任何矩阵 $X \in R^{k \times n}$ 都可以重写为：

$$X = UDV^T$$

上述方程的右侧称为矩阵 X 的奇异值分解， $U \in R^{k \times k}$ 是正交矩阵。 $D \in R^{k \times n}$ 是一个对角矩阵，其中每个值都是 X 的奇异值。

通过这种方式，我们将特征的原始矩阵数据的秩降低到较小的秩，以便可以使用较小秩矩阵中的一些向量的线性组合来重新创建原始矩阵。较小的秩矩阵捕获原始特征空间中最重要的元素。

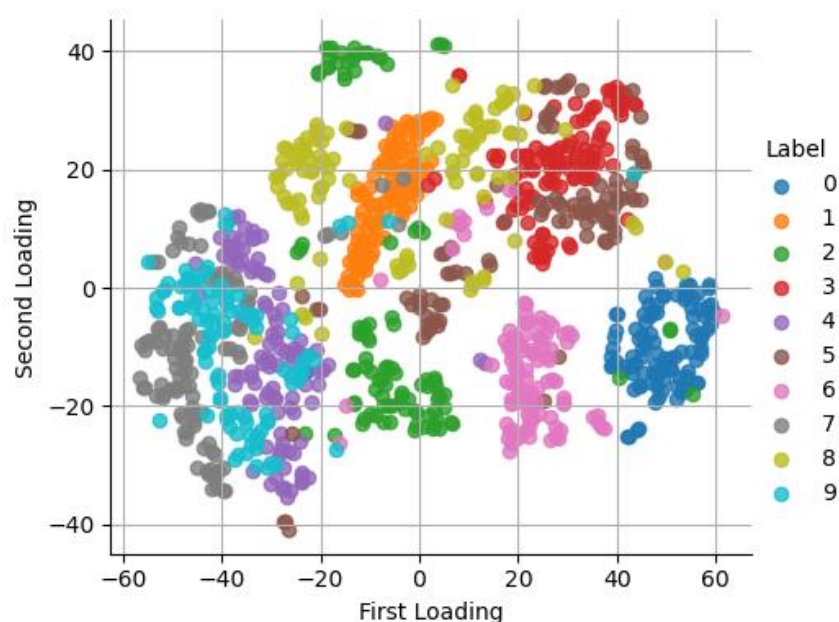
这与 PCA 非常相似。在 PCA 中，协方差矩阵的特征分解用于降维。在奇异值分解中，我们对数据矩阵进行奇异值分解。在具体实现中，我们使用了截断奇异值分解，即我们通过丢弃最小特征值来考虑低秩稳定矩阵。

**t-SNE:**

t-Distributed Stochastic Neighbor Embedding (t-SNE) 是一种降维技术，用于在二维或三维的低维空间中表示高维数据集，从而使其可视化。与其他降维算法(如 PCA)相比，t-SNE 创建了一个缩小的特征空间，相似的样本由附近的点建模，不相似的样本由高概率的远点建模。

在高水平上，t-SNE 为高维样本构建了一个概率分布，相似的样本被选中的可能性很高，而不同的点被选中的可能性极小。然后，t-SNE 为低维嵌入中的点定义了相似的分佈。最后，t-SNE 最小化了两个分布之间关于嵌入点位置的 Kullback-Leibler (KL) 散度。

t-SNE 是一种集降维与可视化于一体的技术，它是基于 SNE 可视化的改进，解决了 SNE 在可视化后样本分布拥挤、边界不明显的特点，是目前较好的降维可视化手段。



接下来是部分聚类指标在模型中的值，实验要求计算带矫正的互信息（AMI）和剪影值（silhouette），其中互信息是用于评价相同数据的两个标签之间的相似性度量，Silhouette 值用来描述一个目标对于目标所在簇与其他簇之间的相似性。其范围是从-1~+1，这个值越大表明目标与自己所在簇之间的匹配关系度越高，与其他簇的匹配关系度越低。如果这个值越高，那么聚类结果越好，如果是很小或是负值，那么可能是分簇太多或是太少造成的。二者在 sklearn 包中均已实现，我们调用相关函数即可，结果如下：

```
##### Silhouette-Score & AMI for K-MEANS #####
n_clusters: 2    silhouette_score: 0.46788379549980164    AMI: 0.31198970629706146
n_clusters: 3    silhouette_score: 0.4724043011665344    AMI: 0.4583800483560295
n_clusters: 4    silhouette_score: 0.46773561835289    AMI: 0.49676620448990183
n_clusters: 5    silhouette_score: 0.4488189220428467    AMI: 0.5457176529410347
n_clusters: 6    silhouette_score: 0.46926426887512207    AMI: 0.5738273389089655
n_clusters: 7    silhouette_score: 0.47255364060401917    AMI: 0.6421030407006398
n_clusters: 8    silhouette_score: 0.45184701681137085    AMI: 0.6638652747054428
n_clusters: 9    silhouette_score: 0.4559392035007477    AMI: 0.6656654586123963
n_clusters: 10   silhouette_score: 0.4494220018386841    AMI: 0.6491726776560429
n_clusters: 11   silhouette_score: 0.4247298836708069    AMI: 0.6372007288336002
n_clusters: 12   silhouette_score: 0.42532867193222046    AMI: 0.6474500586353596

##### Silhouette-Score & AMI for DBSCAN #####
n_eps: 3    silhouette_score: -0.21921955049037933    AMI: 0.04840807638870555
n_eps: 4    silhouette_score: -0.0018632421270012856    AMI: 0.5163972137570899
n_eps: 5    silhouette_score: 0.354154109954834    AMI: 0.6726404899897631
n_eps: 6    silhouette_score: 0.41156646609306335    AMI: 0.6202125161265987
n_eps: 7    silhouette_score: 0.30296647548675537    AMI: 0.45109420018303026
```

之后向 DBSCAN、KMEANS、GM 等模型输入不同降维方法处理过的数据并输出过程中展示的模型信息，结果如下所示：

```
##### 模型设置: DBSCAN on X_PCA #####
```

```
Estimated number of clusters: 0
Estimated number of noise points: 1000
Homogeneity: 0.0000
Completeness: 1.0000
V-measure: 0.0000
Adjusted Rand Index: 0.0000
Adjusted Mutual Information: 0.0000
The overall Accuracy with dbscan on X_PCA is: 0.1
The clusters found are:
  cluster clusterCount
0        -1          1000
```

```
##### 模型设置: DBSCAN on X_svd #####
```

```
Estimated number of clusters: 0
Estimated number of noise points: 1000
Homogeneity: 0.0000
Completeness: 1.0000
V-measure: 0.0000
Adjusted Rand Index: 0.0000
Adjusted Mutual Information: 0.0000
The overall Accuracy with dbscan on X_svd is: 0.1
The clusters found are:
  cluster clusterCount
0        -1          1000
```

```
##### 模型设置: DBSCAN on X_tSNE #####
```

```
Estimated number of clusters: 9
Estimated number of noise points: 97
Homogeneity: 0.6453
Completeness: 0.7165
V-measure: 0.6790
Adjusted Rand Index: 0.5079
Adjusted Mutual Information: 0.6726
The overall Accuracy with dbscan on X_tSNE is: 0.636
The clusters found are:
  cluster clusterCount
0         3           280
1         1           137
2         0           118
3         4           101
4        -1            97
5         2            86
6         5            81
7         6            64
8         8            19
9         7            17
```

我们使用了三个指标:

同质性: 如果所有聚类只包含属于单个类的数据点, 则聚类结果满足同质性。此度量与标签的绝对值无关: 类或簇标签值的排列不会以任何方式更改分数值。

完整性: 如果给定类的所有数据点都是同一个集群的元素, 则集群结果满足完整性。此

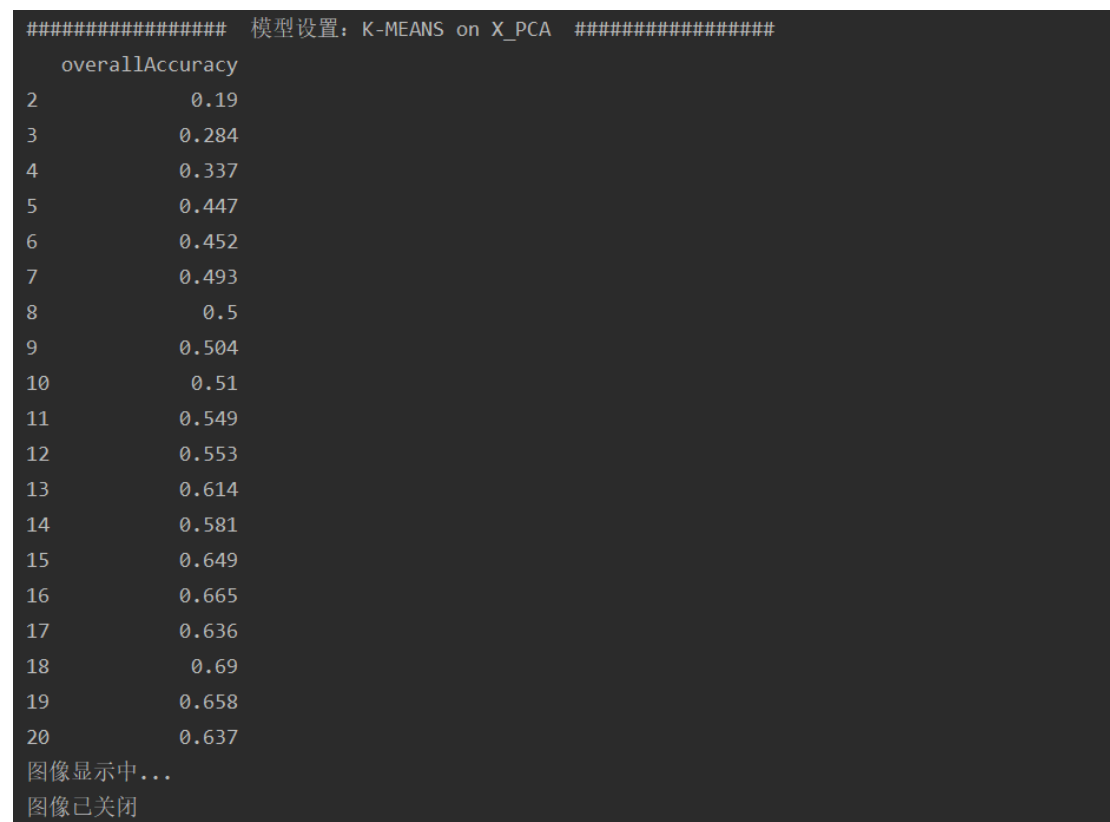
度量与标签的绝对值无关：类或簇标签值的排列不会以任何方式更改分数值。

V-测度：V-测度是同质性和完整性之间的调和平均值。

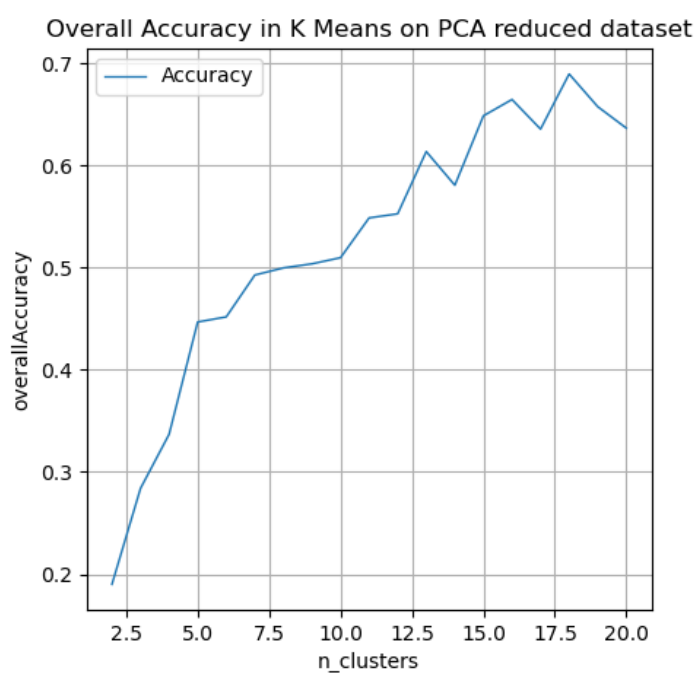
随机调整的兰德指数：兰德指数通过考虑预测和真实聚类中分配在相同或不同聚类中的所有样本对和计数对，计算两个聚类之间的相似性度量。然后将原始 RI 分数“根据机会调整”为 ARI 分数。因此，调整后的兰德指数对于独立于簇和样本数量的随机标记而言，确保其值接近 0.0，而对于簇相同的情况，则确保其值精确为 1.0。

调整后的互信息：调整后的互信息 (AMI) 是对互信息 (MI) 分数的调整，以考虑机会。它解释了这样一个事实，即对于两个集群数量较大的集群，MI 通常较高，而不管实际上是否共享了更多的信息。

可以看到当使用 t-SNE 时在 K_MEANS 中同质性和完整性均有较好水平，分别是 0.6543、0.7165，AMI 值也有较高水平，说明数据质量较高，最终形成了 10 个聚簇，其中聚簇 3 中数据最多，有 280 个。



可以看到使用 X_PCA 作为输入时，总体准确率随聚类数量的变化情况，总体随着聚类数量的增多而上升，在聚类数量为 18 时达到最高值。



```
##### 模型设置: K-MEANS on X_tSNE #####
```

```
Homogeneity: 0.4657
```

```
Completeness: 0.4870
```

```
V-measure: 0.4761
```

```
Adjusted Rand Index: 0.3120
```

```
Adjusted Mutual Information: 0.4662
```

```
overallAccuracy
```

```
2          0.199
```

```
3          0.296
```

```
4          0.375
```

```
5          0.432
```

```
6          0.512
```

```
7          0.597
```

```
8          0.677
```

```
9          0.706
```

```
10         0.723
```

```
11         0.702
```

```
12         0.728
```

```
13         0.719
```

```
14         0.729
```

```
15         0.752
```

```
16         0.756
```

```
17         0.734
```

```
18         0.733
```

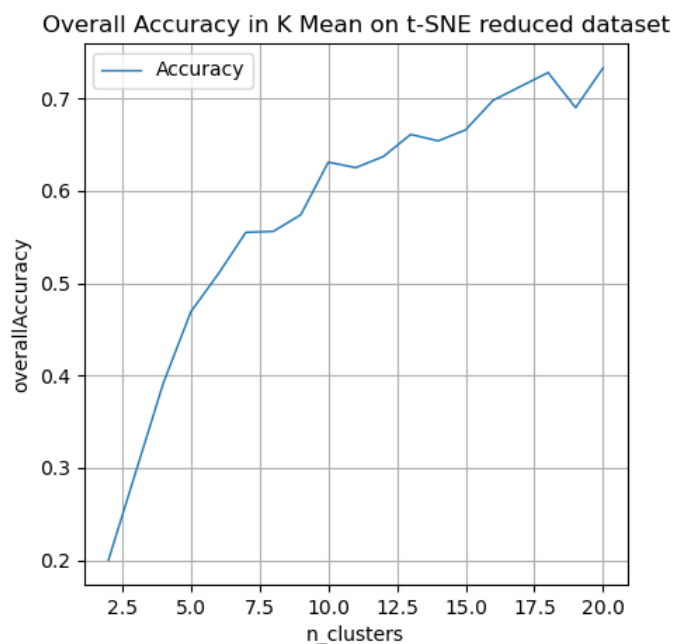
```
19         0.765
```

```
20         0.771
```

```
图像显示中...
```

```
图像已关闭
```

可以看到使用 X_tSNE 作为输入时, 总体准确率随聚类数量的变化情况, 总体随着聚类数量的增多而上升, 在聚类数量为 18 时达到最高值。



```
##### 模型设置: GM on X_PCA #####
0.377
Homogeneity: 0.2861
Completeness: 0.2910
V-measure: 0.2885
Adjusted Rand Index: 0.1650
Adjusted Mutual Information: 0.2753
  cluster  clusterCount
0         3           148
1         1           147
2         5           106
3         9           104
4         0           103
5         6            98
6         8            86
7         4            83
8         7            63
9         2            62
```

可以看到当使用 X_PCA 时在 GM 中同质性和完整性均一般，分别是 0.2861、0.2910，AMI 值也较小，最终形成了 10 个聚簇，其中聚簇 3 中数据最多，有 148 个。

最后将每个聚类方法的效果可视化，如下图所示，可以看到 K_MEANS 和 DBSCAN 均产生了较好的聚类效果：

