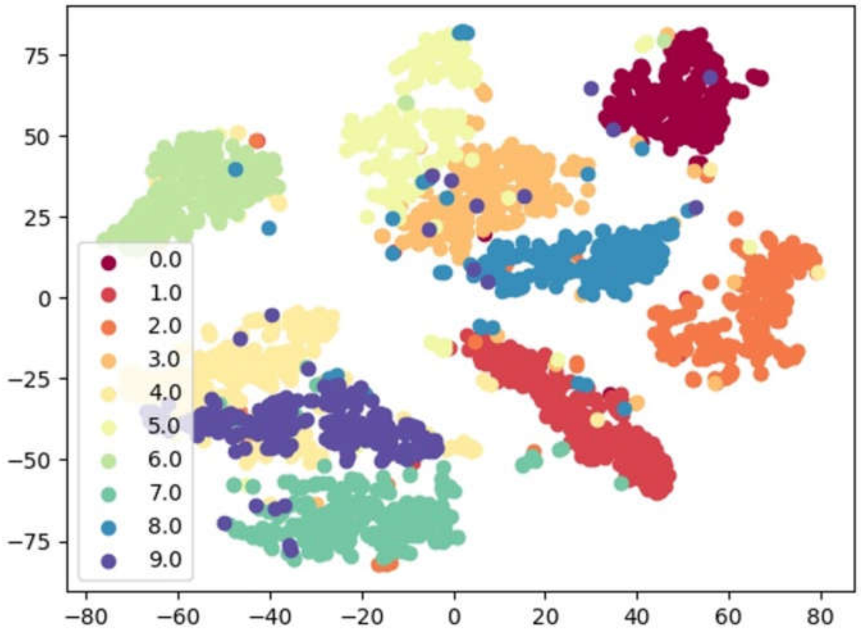
Python编程与人工智能实践

算法篇：数据降维-t-SNE

（t-Distributed Stochastic Neighbor Embedding）

tt分布随机近邻嵌入



于泓
鲁东大学
信息与电气工程学院

2021.9.30

t-SNE（t-Distributed Stochastic Neighbor Embedding）

- t-sne是一种非常常用的数据降维（数据可视化）

基本原理:

（1） 在高维空间构建一个概率分布拟合高维样本点间的相对位置关系

（2）在低维空间，也构建 一个概率分布，拟合低维样本点之间的位置关系

（3） 通过学习，调整低维数据点，令两个分布接近

参考文献: Van der Maaten L, Hinton G. Visualizing data using t-SNE[J]. Journal of Machine Learning Research, 2008, 9(2579-2605): 85.

SNE （Stochastic Neighbor Embedding）

相当于高斯分布

高维样本点间位置关系：

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k\neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)},$$

条件概率

方差固定
的高斯分布

低维样本点间位置关系：

$$q_{j|i} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k\neq i} \exp\left(-\|y_i - y_k\|^2\right)}.$$

利用KL散度（距离）
Kullback-Leibler divergences
衡量2个分布之间的差异

$$C = \sum_i KL(P_i||Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}},$$

通过梯度下降
调整**y** **使C**变小

实现降维

SNE 方法的主要缺点

（1）距离不对称（与实际不符）

改进：

$$p_{ij} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma^2\right)}{\sum_{k \neq l} \exp\left(-\|x_k - x_l\|^2 / 2\sigma^2\right)}, \qquad q_{ij} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k \neq l} \exp\left(-\|y_k - y_l\|^2\right)},$$

条件概率
变联合概率

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)}, \qquad q_{j|i} = \frac{\exp\left(-\|y_i - y_j\|^2\right)}{\sum_{k \neq i} \exp\left(-\|y_i - y_k\|^2\right)}.$$

具体实现：（1） $p_{ij} = p_{i|j} + p_{j|i}$ （2） $p_{ij} = \dfrac{p_{ij}}{\sum_i \sum_j p_{ij}}$

保证对称

保证归一化

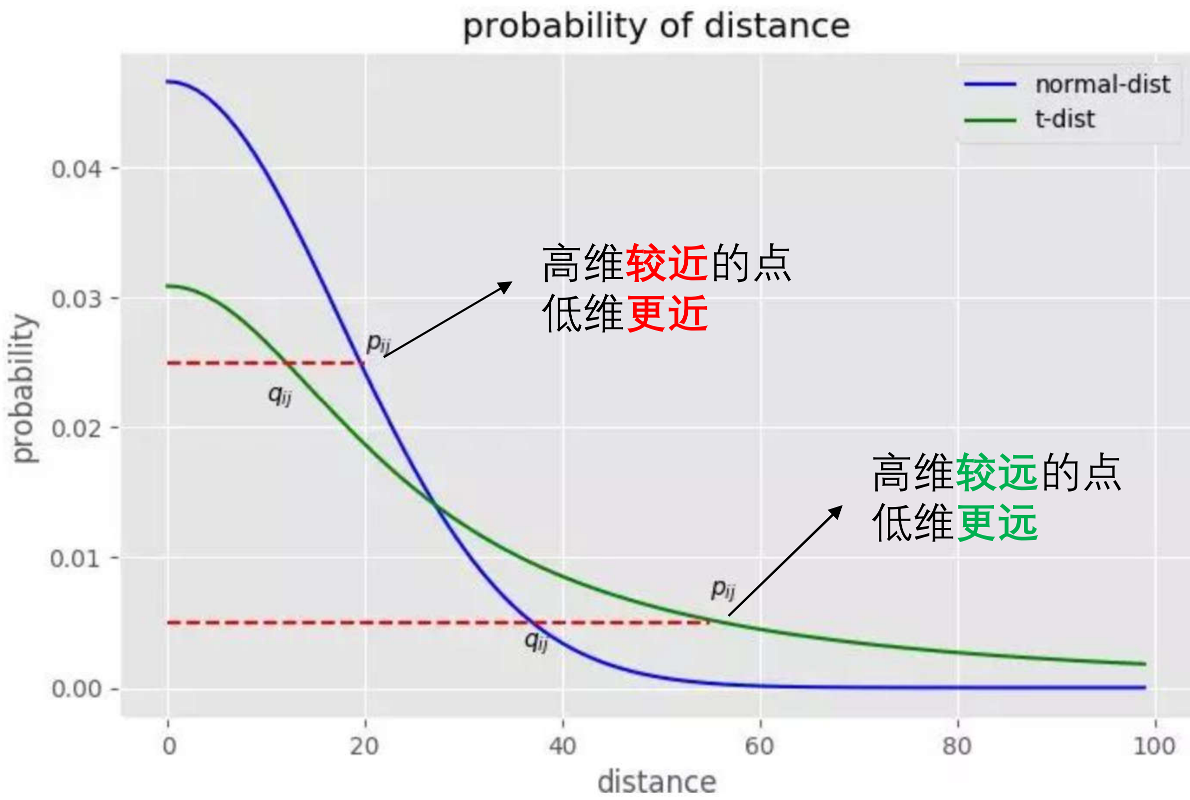
(2) 拥挤显现

从高维到低维进行转换的过程中，低维点的距离无法建模高维点之间的位置关系是的高维空间中距离较大的点对，在低维空间距离会变得较小

解决方法：利用拖尾较大的**student-t分布**来对低维点建模

变为：

$$q_{ij} = \frac{\left(1+\|y_i - y_j\|^2\right)^{-1}}{\sum_{k\neq l}\left(1+\|y_k - y_l\|^2\right)^{-1}}.$$



T-sne的一般步骤:

（1）计算 $p_{i|j}$ ，利用结果，计算 p_{ij}

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)},$$

$$\boxed{p_{ij} = p_{i|j} + p_{j|i} \qquad p_{ij} = \frac{p_{ij}}{\sum_i \sum_j p_{ij}}}$$

（2） 随机生成低维随机数并计算:

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}.$$

（3）利用梯度下降

令 C最小

$$C = KL(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

梯度公式:

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)\left(1 + \|y_i - y_j\|^2\right)^{-1}.$$

t-sne的一般步骤:

（1）计算 $p_{j|i}$ ，利用结果，计算 p_{ij}

?

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)},$$

$$\boxed{p_{ij} = p_{i|j} + p_{j|i} \qquad p_{ij} = \frac{p_{ij}}{\sum_i \sum_j p_{ij}}}$$

（2） 随机生成低维随机数并计算:

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l} \left(1 + \|y_k - y_l\|^2\right)^{-1}}.$$

（3）利用梯度下降

令 C最小

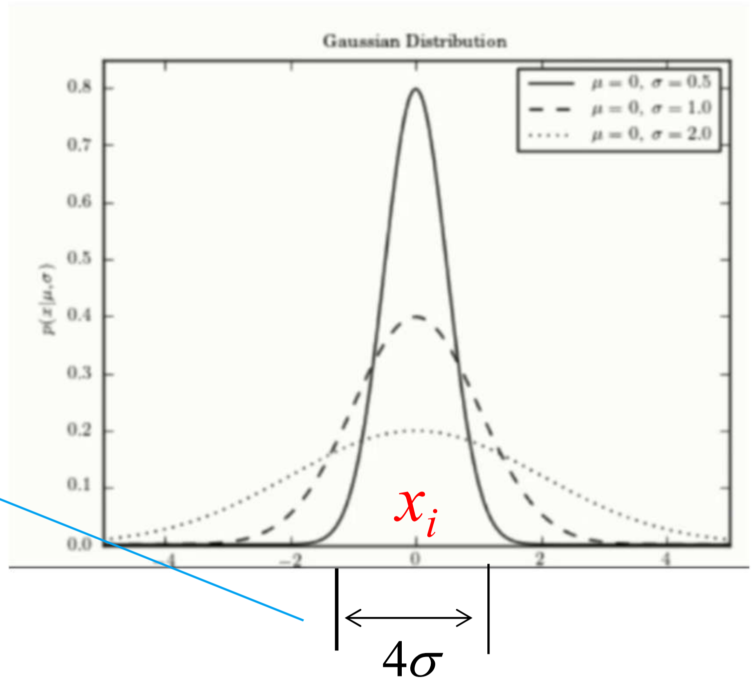
$$C = KL(P\|Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}}.$$

梯度公式:

$$\frac{\delta C}{\delta y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)\left(1 + \|y_i - y_j\|^2\right)^{-1}.$$

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2/2\sigma_i^2\right)}{\sum_{k\neq i}\exp\left(-\|x_i - x_k\|^2/2\sigma_i^2\right)},$$

物理意义:



如何确定 σ ？

设置一个固定的参数:
Perplexity 表示分布的熵

设法调节每个 σ_i

令 $\log(\text{Per}) = -\sum_j p_{j|i}\log(p_{j|i})$

熵和 σ_i 成正比 可以通过二分查找法 确定 σ_i

在实际代码中设置 $\text{beta}_i = \dfrac{1}{2\sigma_i^2}$ 寻找最优***beta***

距离x_i 小于 2σ
的点对**P**的计算起主要作用

当x_i临近点较多时：减少 σ
当x_i临近点较少时：增大 σ

Stochastic Neighbor

代码实现

```
import numpy as np
import matplotlib.pyplot as plt
from test_PCA import pca
# 计算任意两点之前距离 ||x_i-x_j||^2
# X 维度 [N,D]
def cal_pairwise_dist(X):
    sum_X = np.sum(np.square(X), 1)
    D = np.add(np.add(-2 * np.dot(X, X.T), sum_X).T, sum_X)
    #返回任意两个点之间距离
    return D
```

去掉ii点

$$p_{j|i} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_k\|^2 / 2\sigma_i^2\right)},$$

```
# 计算P_ij 以及 log松弛度
def calc_P_and_entropy(D,beta=1.0):
    P = np.exp(-D.copy() * beta)
    sumP = np.sum(P)
    # 计算熵
    log_entropy = np.log(sumP) + beta * np.sum(D * P) / sumP

    P = P/sumP
    return P,log_entropy
```

$$\log(\mathrm{Per}) = -\sum_j p_{j|i} \log(p_{j|i})$$

```python
# 二值搜索寻找最优的 sigma
def binary_search(D, init_beta,logU, tol=1e-5, max_iter=50):

    beta_max = np.inf
    beta_min = -np.inf
    beta = init_beta

    P,log_entropy=calc_P_and_entropy(D,beta)
    diff_log_entropy = log_entropy - logU

    m_iter = 0
    while np.abs(diff_log_entropy)> tol and m_iter<max_iter:
        # 交叉熵比期望值大，增大beta
        if diff_log_entropy>0:
            beta_min = beta
            if beta_max == np.inf or beta_max == -np.inf:
                beta = beta*2
            else:
                beta = (beta+beta_max)/2.
        # 交叉熵比期望值小， 减少beta
        else:
            beta_max = beta
            if beta_min == -np.inf or beta_min == -np.inf:
                beta = beta/2
            else:
                beta = (beta + beta_min)/2.

        # 重新计算
        P,log_entropy=calc_P_and_entropy(D,beta)

        diff_log_entropy = log_entropy - logU

        m_iter = m_iter+1
```

```python
        # 重新计算
        P,log_entropy=calc_P_and_entropy(D,beta)

        diff_log_entropy = log_entropy - logU

        m_iter = m_iter+1

    # 返回最优的 beta 以及所对应的 P
    return P, beta
```

```python
# 给定一组数据 datas : [N,D]
# 计算联合概率 P_ij : [N,N]
def p_joint(datas, target_perplexity):

    N,D = np.shape(datas)
    # 计算两两之间的距离
    distances = cal_pairwise_dist(datas)

    beta = np.ones([N,1])  # beta = 1/(2*sigma^2)
    logU = np.log(target_perplexity)
    p_conditional = np.zeros([N,N])
    # 对每个样本点搜索最优的sigma(beta) 并计算对应的P
    for i in range(N):
        if i %500 ==0:
            print("Compute joint P for %d points"%(i))
        # 删除 i -i 点
        Di = np.delete(distances[i,:],i)
        # 进行二值搜索, 寻找 beta
        # 使 log_entropy 最接近 logU
        P, beta[i] = binary_search(Di, beta[i],logU)

        # 在ii的位置插0
        p_conditional[i] = np.insert(P,i,0)

    # 计算联合概率
    P_join = p_conditional + p_conditional.T
    P_join = P_join/np.sum(P_join)

    print("Mean value of sigma: %f" % np.mean(np.sqrt(1 / beta)))
    return P_join
```

```python
def estimate_tsen(datas,labs,dim,target_perplexity,plot=False):

    N,D = np.shape(datas)

    # 随机初始化低维数据Y
    Y = np.random.randn(N, dim)

    # 计算高维数据的联合概率
    print("Compute P_joint")
    P = p_joint(datas, target_perplexity

    # 开始若干轮对 P 进行放大
    P = P*4.
    P = np.maximum(P, 1e-12)

    # 开始进行迭代训练
    # 训练相关参数
    max_iter = 1500
    initial_momentum = 0.5
    final_momentum = 0.8
    eta = 500    # 学习率
    min_gain = 0.01
    dY = np.zeros([N, dim]) # 梯度
    iY = np.zeros([N, dim]) # Y的变化
    gains = np.ones([N, dim])
```

取值5-50

通常较大

```python
for m_iter in range(max_iter):

    # 计算 Q
    Q,num= q_tsne(Y)

    # 计算梯度
    PQ = P - Q
    for i in range(N):
        dY[i, :] = np.sum(np.tile(PQ[:, i] * num[:, i], (dim, 1)).T * (Y[i, :] - Y), 0)
```

$$\frac{\delta C}{\delta y_i} = 4\sum_j (p_{ij} - q_{ij})(y_i - y_j)\left(1 + \|y_i - y_j\|^2\right)^{-1}$$

```python
def q_tsne(Y):
    N = np.shape(Y)[0]
    sum_Y = np.sum(np.square(Y), 1)
    num = -2. * np.dot(Y, Y.T)
    num = 1. / (1. + np.add(np.add(num, sum_Y).T, sum_Y))
    num[range(N), range(N)] = 0.
    Q = num / np.sum(num)
    Q = np.maximum(Q, 1e-12)
    return Q,num
```

$$q_{ij} = \frac{\left(1 + \|y_i - y_j\|^2\right)^{-1}}{\sum_{k \neq l}\left(1 + \|y_k - y_l\|^2\right)^{-1}}.$$

```
    # Perform the update
    if m_iter < 20:
        momentum = initial_momentum
    else:
        momentum = final_momentum
    gains = (gains + 0.2) * ((dY > 0.) != (iY > 0.)) + \
            (gains * 0.8) * ((dY > 0.) == (iY > 0.))
    gains[gains < min_gain] = min_gain
    iY = momentum * iY - eta * (gains * dY)
    Y = Y + iY

    # Y 取中心化
    Y = Y - np.tile(np.mean(Y, 0), (N, 1))

    # Compute current value of cost function
    if (m_iter + 1) % 10 == 0:
        C = np.sum(P * np.log(P / Q))
        print("Iteration %d: loss is %f" % (m_iter + 1, C))

    # 停止放大P
    if m_iter == 100:
        P = P / 4.

    if plot and m_iter % 100 == 0:
        print("Draw Map")
        draw_pic(Y,labs,name = "%d.jpg"%(m_iter))

return Y
```

点的偏移对点点之间的距离无影响

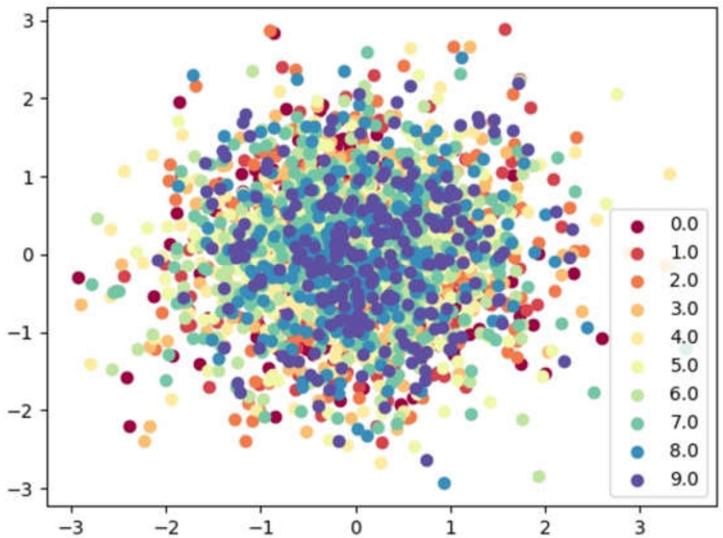
```python
def draw_pic(datas,labs,name = '1.jpg'):
    plt.cla()
    unque_labs = np.unique(labs)
    colors = [plt.cm.Spectral(each)
      for each in np.linspace(0, 1,len(unque_labs))]
    p=[]
    legends = []
    for i in range(len(unque_labs)):
        index = np.where(labs==unque_labs[i])
        pi = plt.scatter(datas[index, 0], datas[index, 1], c =[colors[i]] )
        p.append(pi)
        legends.append(unque_labs[i])

    plt.legend(p, legends)
    plt.savefig(name)
```

2500 * 784 （27x27）

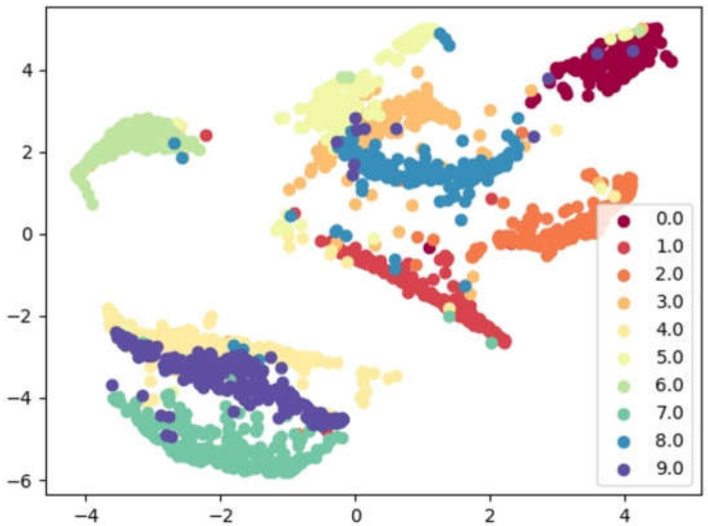
```python
if __name__ == "__main__":

    mnist_datas = np.loadtxt("mnist2500_X.txt")
    mnist_labs = np.loadtxt("mnist2500_labels.txt")

    print("first reduce by PCA")
    datas, _= pca(mnist_datas, 30)
    X = datas.real

    Y = estimate_tsen(X,mnist_labs,2,30, plot=True)

    draw_pic(Y,mnist_labs,name = "final.jpg")
```

对于高维数据先用PCA降维

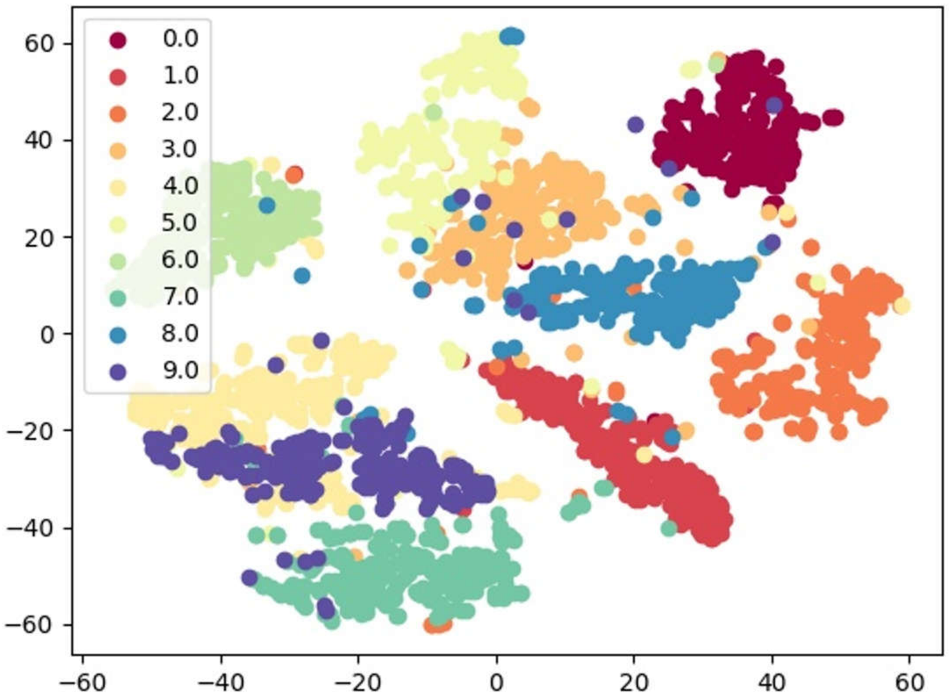
结果:



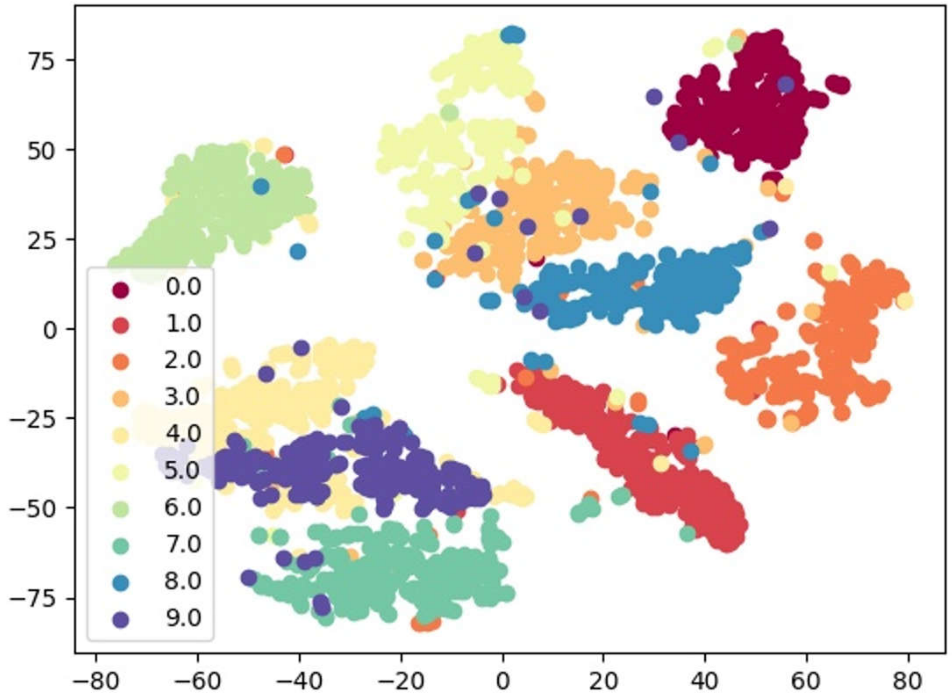
0



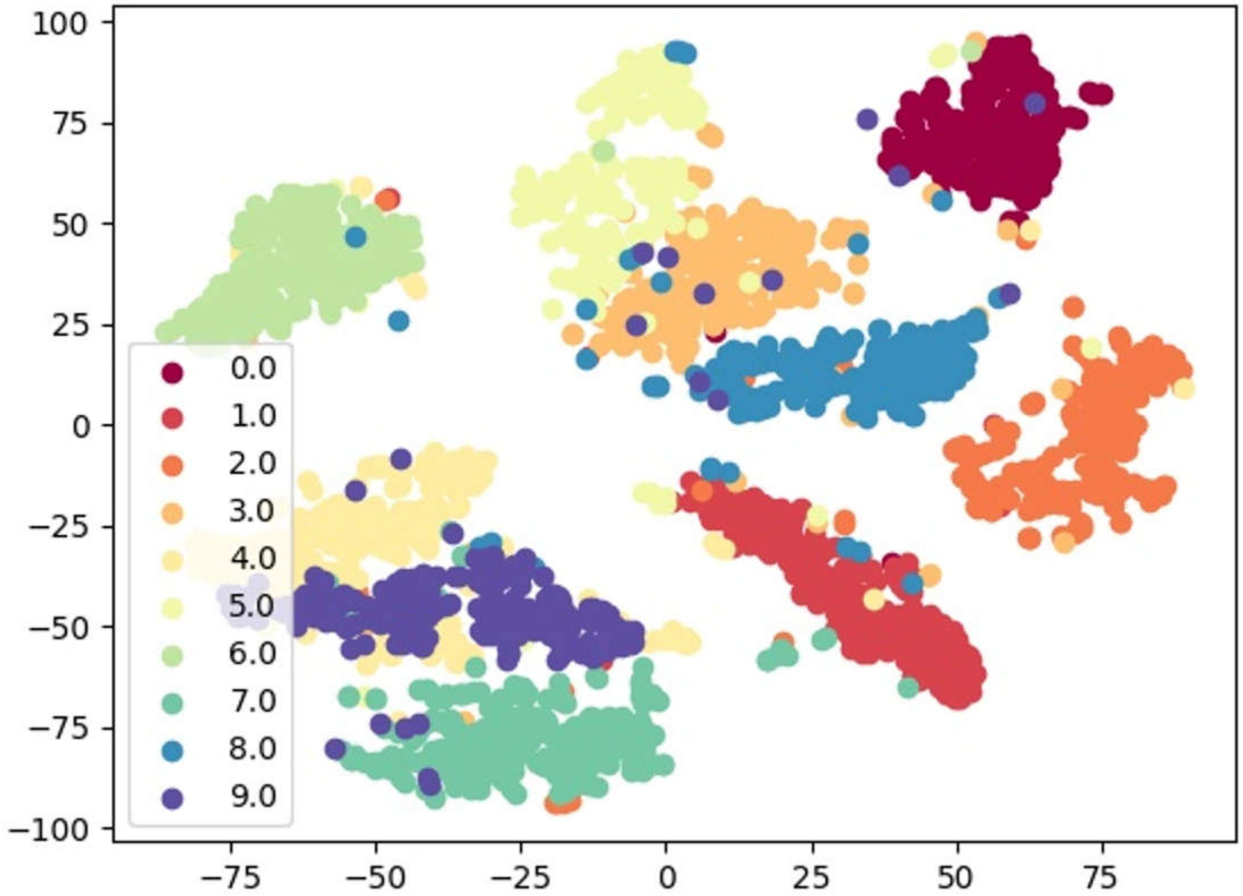
100



500



1000


100
75
50
25
0
−25
−50
−75
−100
−75
−50
−25
0
25
50
75
0.0
1.0
2.0
3.0
4.0
5.0
6.0
7.0
8.0
9.0

final