二分kmeans实验报告

姓名：季马泽宇

学号：1950089

算法介绍

kmeans

- 算法介绍

  k均值聚类是最著名的划分聚类算法，由于简洁和效率使得他成为所有聚类算法中最广泛使用的。给定一个数据点集合和需要的聚类数目k，k由用户指定，k均值算法根据某个距离函数反复把数据分入k个聚类中。

- 伪代码

  ```
  选择k个点作为初始质心。
  repeat 将每个点指派到最近的质心，形成k个簇 重新计算每个簇的质心 until 质心不发生变化
  ```

- 代码

  ```python
  # 计算欧氏距离
  def euclid(point1,point2,dimension):
      dis=0
      for i in range(dimension):
          dis+=(point1[i]-point2[i])**2
      return dis**0.5
  # 更新centers
  def update_centers(dataset,labels,clusters):
      length,dimension = dataset.shape
      centers=np.zeros([clusters,dimension])
      # 计算每个cluster的个数
      numbers=np.zeros(clusters)
      for label in labels:
          numbers[label]+=1
      for i,point in enumerate(dataset):
          for di in range(dimension):
  ```

```
            centers[labels[i]][di]+=point[di]
    for i in range(clusters):
        # print(centers[i])
        # print(numbers[i])
        if(numbers[i]!=0):
            centers[i]/=numbers[i]
    return centers
# 参数为数据集、簇个数、迭代次数（这里直接指定迭代次数，偷一点点懒）
def k_means(dataset,clusters,iters,visualization=False,reduced_data=[]):
    plt.ion()
    # 样本个数
    length,dimension = dataset.shape
    # 随机生成初始簇心
    centers = dataset[random.sample(range(0,length),clusters)]
    #print(centers.shape) # clusters*dimension
    # 数据标签,代表每类所属于的簇
    labels = []
    for _ in range(iters):
        for center in centers:
            tmp_list = [euclid(dataset[i],center,dimension) for i in
range(length)]
            labels.append(tmp_list)
        labels = np.argmin(labels,axis=0)
        centers = update_centers(dataset,labels,clusters)
        # 分布可视化
        if visualization:
            pplot(reduced_data,labels)
        plt.pause(1)
        if(_!=iters-1):
            labels = []
    plt.ioff()
    return labels
```

二分kmeans

- 算法介绍

  为克服K-Means算法收敛于局部最小值问题，提出了二分K-Means算法。

  二分K-Means算法首先将所有点作为一个簇，然后将该簇一分为二。之后选择其中一个簇继续进行划分，选择哪一个簇进行划分取决于对其划分是否可以最大程度降低SSE的值。上述基于SSE的划分过程不断重复，直到得到用户指定的簇数目为止。

- 伪代码：

```
初始化簇表，使之包含由所有的点组成的簇。
repeat
   {对选定的簇进行多次二分试验}
   for i=1 to 试验次数 do
       使用基本k均值，二分选定的簇。
   endfor
   从二分试验中选择具有最小误差的两个簇。
   将这两个簇添加到簇表中。
until 簇表中包含k个簇
```

- 代码：

```
# 计算sse（簇内误差平方和）
```

```
def sse(dataset,labels,clusters):
    length,dimension = dataset.shape
    data = []
    if(clusters>1):
        for i in range(clusters):
            data.append([])
        for i,dt in enumerate(dataset):
            data[labels[i]].append(dt)
    else:
        data.append([])
        data[0] = dataset
    res=0
    for k in range(clusters):
        for point1 in data[k]:
            for point2 in data[k]:
                res+=euclid(point1,point2,dimension)**2
    return res
# 二分kmeans,iters代表每次尝试不同点进行二分的次数
def bi_kmeans(dataset,iters,clusters):
    length = len(dataset)
    # 初始化标签
    labels=[]
    for i in range(length):
        labels.append(0)
    # 分割簇数-1次
    for it in range(clusters-1):
        # 计算每类sse
        data = []
        for i in range(it+1):
            data.append([])
        for i,dt in enumerate(dataset):
            data[labels[i]].append(dt)
        max_sse=0
        tag=0
        for i in range(it+1):
            dt = np.array(data[i])
            tmp_sse = sse(dt,[],1)
            if tmp_sse>max_sse:
                max_sse=tmp_sse
                tag=i
        idx=[]
        for i in range(length):
            if labels[i]==tag:
                idx.append(i)
        dataset_to_divide = data[tag]
        dataset_to_divide = np.array(dataset_to_divide)
        f_labels =
k_means(dataset_to_divide,2,1,visualization=False,reduced_data=dataset)
        f_sse = sse(dataset_to_divide,f_labels,2)
        for i in range(iters-1):
            tmp_labels
=k_means(dataset_to_divide,2,1,visualization=False,reduced_data=dataset_to_d
ivide)
            tmp_sse = sse(dataset_to_divide,tmp_labels,2)
            if tmp_sse<f_sse:
                f_sse=tmp_sse
                f_labels=tmp_labels
        for i,idxx in enumerate(idx):
```

```
            if f_labels[i]==0:
                labels[idxx]=it+1
        pplot(dataset,labels)
    return labels
```

二分kmeans的改进（结合层次聚类）

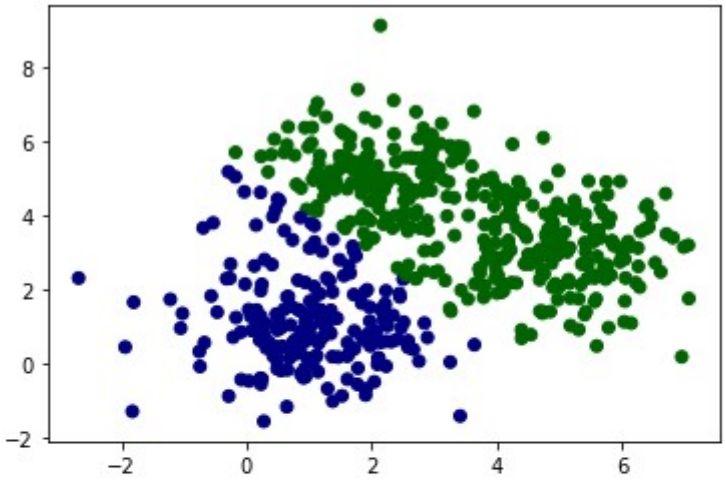
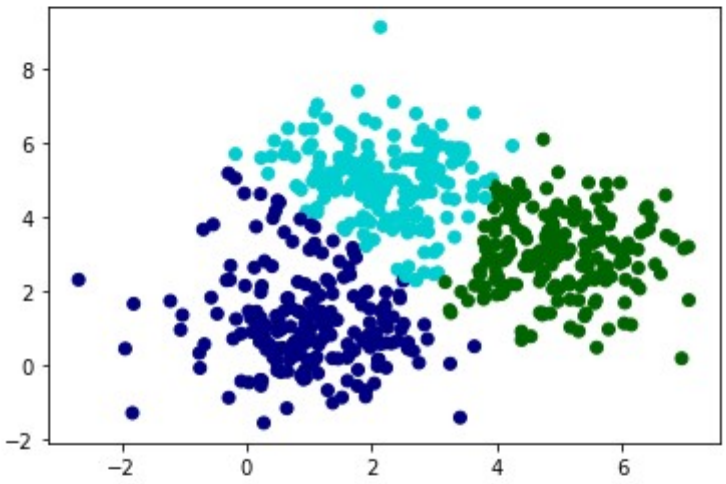
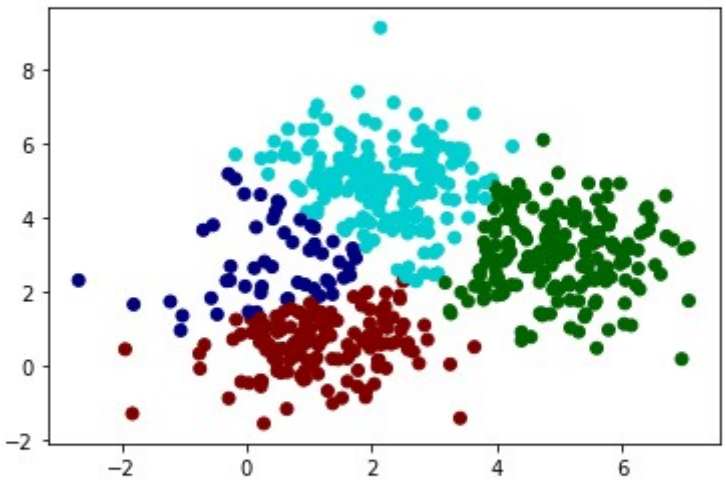
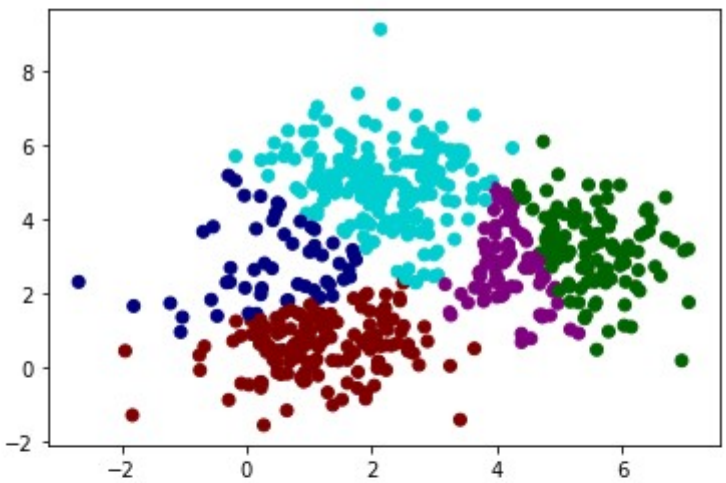
- **K-means聚类**，最大的特点就是可以**快速处理大量数据**，但其仅能处理定量数据而不能处理分类数据，并且K-means聚类需要自主设定聚类类别的数量，不能自动寻找最优聚类类别数量，可能导致结果质量不稳定。
- **层次聚类**，又叫系统聚类，基本思路是将多个样本各作为一类，计算样本两两之间的距离，合并距离最近的两类成新的一类，然后再计算距离，再合并，直到只有一类为止。层次聚类可以处理分类数据和定量数据，但处理速度相对较慢，通常情况下需要结合相关结果进行主观判断聚类类别数量。
- 结合层次聚类与kmeans聚类，先使用二分kmeans将数据划分成很多小的聚类，再通过层次聚类将一些相似度高的节点进行融合，最终达到目标的类别数目
- 代码:

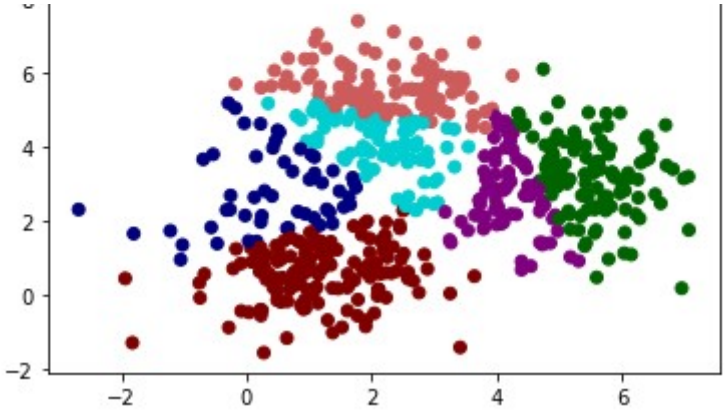
```
# 计算两个簇之间的平均距离
def average_distance(data1,data2):
    #print(data1.shape)
    _,dimension = data1.shape
    len1=len(data1)
    len2=len(data2)
    ssum=0;
    for i in range(len1):
        for j in range(len2):
            ssum+=euclid(data1[i],data2[j],dimension)
    #print(ssum,len1,len2)
    return ssum/(len1*len2)

# 层次聚类,每次将距离（使用平均距离）最小的两个簇合二为一
def hac(dataset,labels,clusters,target_clusters):
    if target_clusters>=clusters:
        return labels
    for it in range(clusters-target_clusters):
        # 找距离最短的两个簇
        data = []
        for i in range(clusters-it):
            data.append([])
        for i,dt in enumerate(dataset):
            data[labels[i]].append(dt)
        min_dis=1e10 #inf
        label1=0
        label2=0
        for i in range(clusters-it):
            for j in range(clusters-it):
                tmp_dis=average_distance(np.array(data[i]),np.array(data[j]))
                if(tmp_dis<min_dis and i!=j):
                    min_dis=tmp_dis
                    label1=i
                    label2=j
        label1,label2=max(label1,label2),min(label1,label2)
        #print(label1,label2)
        for i in range(len(labels)):
            if labels[i]==label1:
```

```
                labels[i]=label2
        for i in range(len(labels)):
            if labels[i]>label1:
                labels[i]-=1
        pplot(dataset,labels)
        plt.show()
    return labels
```

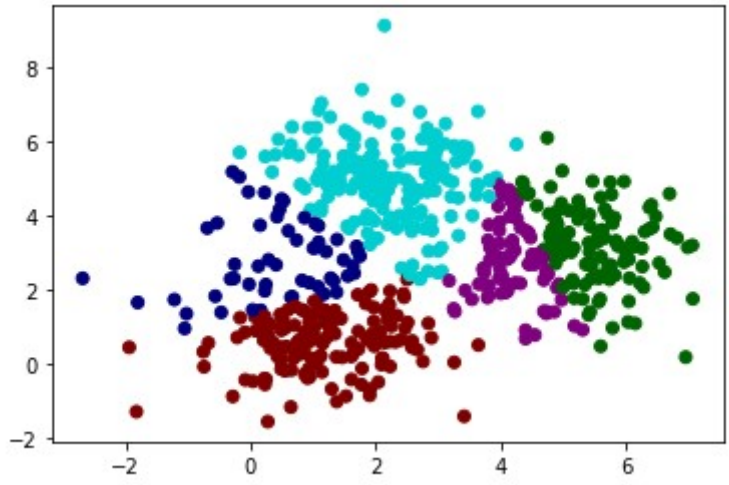
下面是随便生产的数据用于代码测试

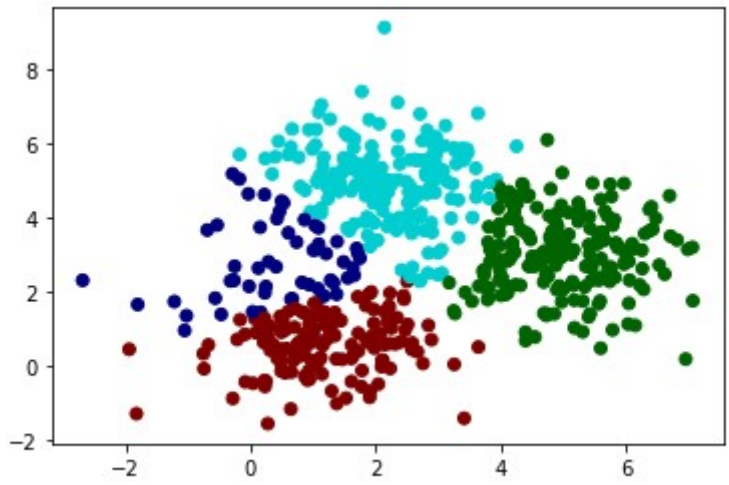
- 二分kmeans

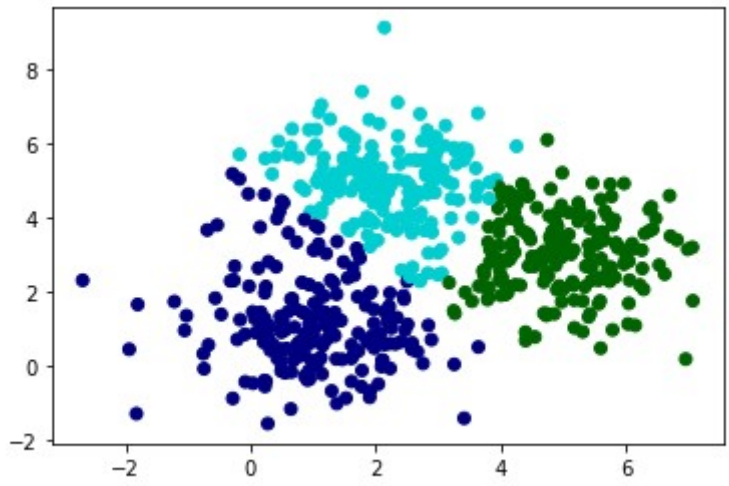





- 层次聚类







样例测试与可视化

样例

使用经典的MNIST手写数字进行测试

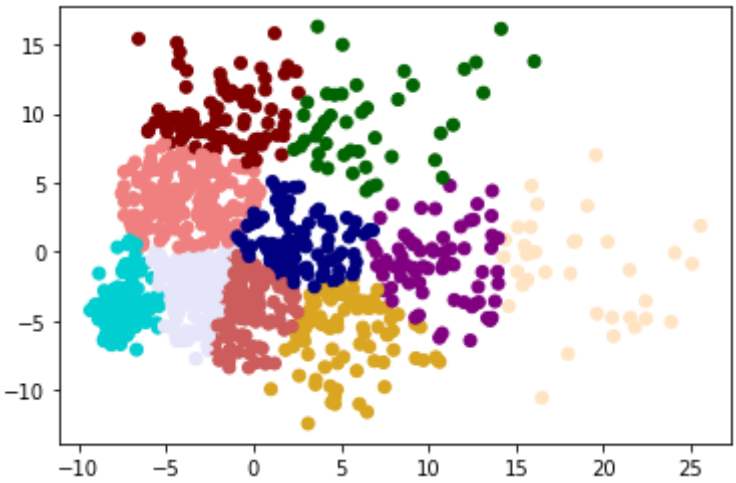
```
import torch
import torch.nn as nn
import torch.utils.data as Data
import torchvision
import matplotlib.pyplot as plt
import argparse
import numpy as np

torch.manual_seed(1)    # 随机种子

# Hyper Parameters
parser = argparse.ArgumentParser(description='Training a model')
parser.add_argument('--batch_size', type=int, default=50, help='')
parser.add_argument('--epoch', type=int, default=2, help='')
parser.add_argument('--lr', type=float, default=0.001, help='')
opt =parser.parse_args(args=[])

DOWNLOAD_MNIST=False
# 下载训练集
train_data = torchvision.datasets.MNIST(
    root='./mnist/',
    train=True,
    transform=torchvision.transforms.ToTensor(),
    download=DOWNLOAD_MNIST,
)
test_data = torchvision.datasets.MNIST(root='./mnist/', train=False)

# bacth_size=50
train_loader = Data.DataLoader(dataset=train_data, batch_size=opt.batch_size,
shuffle=True)

# 取前2000个作为测试集
test_x = torch.unsqueeze(test_data.test_data, dim=1).type(torch.FloatTensor)
[:1000]/255.   # shape from (2000, 28, 28) to (2000, 1, 28, 28), value in
range(0,1)
test_y = test_data.test_labels[:1000]

# 展平数据
data=test_x.detach().numpy()
data=np.squeeze(data,1)
data=data.reshape(1000,28*28)
# 标准化
data = scale(data)
```

降维+坐标

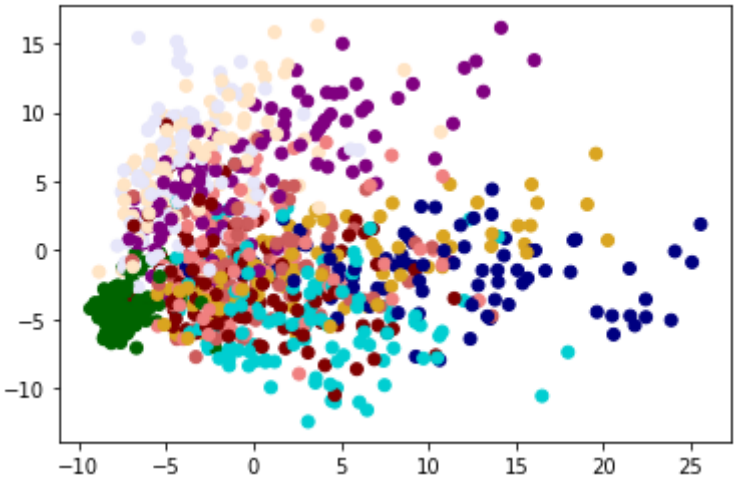
PCA降维

```
# 使用主成分分析降维
reduced_data = PCA(n_components=2).fit_transform(data)
labels=k_means(reduced_data,10,3,True,reduced_data)
```

降维后的聚类效果：



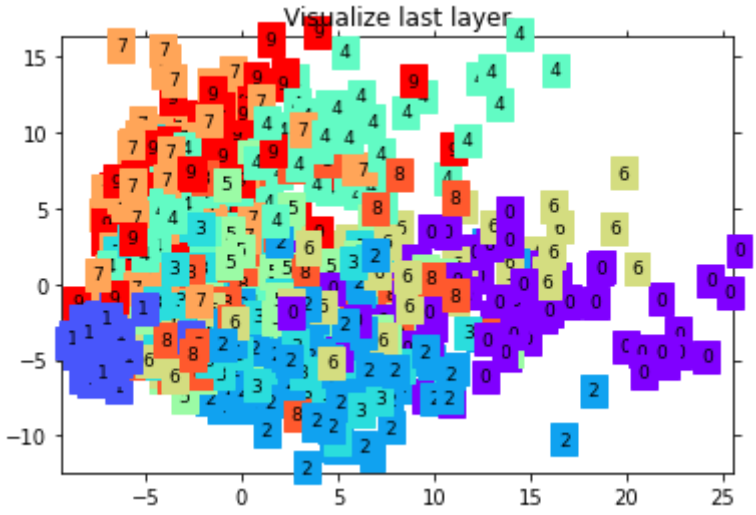
使用数字标签：



让我们看的更清楚一点：

可以看出使用pca降维后，可以看到聚类算法还是有效果的。虽然效果看上去不是那么好，但这其实不是聚类的错，而是pca难以提取出图片的主要特征，考虑使用其他降维算法（TSNE或者autoencoder）

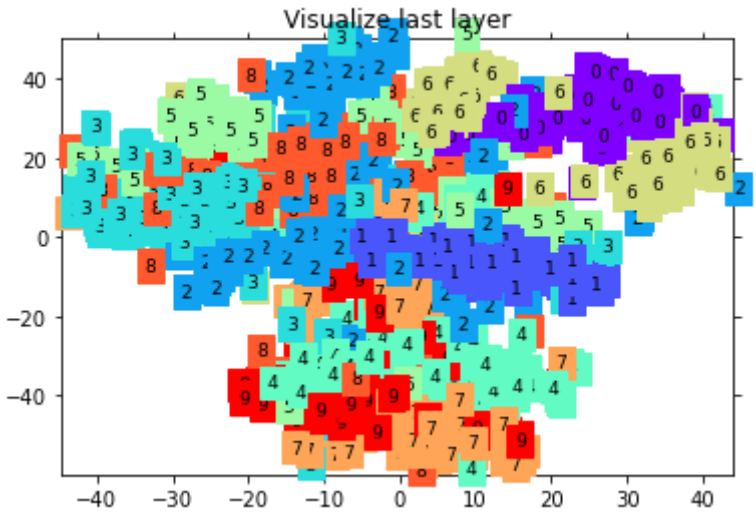
```
from matplotlib import cm
# 数字绘制
def plot_with_labels(lowDWeights, labels):
    plt.cla()
    X, Y = lowDWeights[:, 0], lowDWeights[:, 1]
    for x, y, s in zip(X, Y, labels):
        c = cm.rainbow(int(255 * s / 9)); plt.text(x, y, s, backgroundcolor=c,
fontsize=9)
    plt.xlim(X.min(), X.max()); plt.ylim(Y.min(), Y.max()); plt.title('Visualize
last layer'); plt.show(); plt.pause(0.01)
plot_with_labels(reduced_data,test_y.detach().numpy())
```



TSNE降维

使用TSNE降维后，可以非常清楚的看到聚类的效果，聚类算法倾向于将同一种数字当作一类。

```
from sklearn.manifold import TSNE
# 使用tsne降维度
tsne = TSNE(perplexity=30, n_components=2, init='pca', n_iter=5000)
reduced_data = tsne.fit_transform(data)
labels=k_means(reduced_data,10,3,True,reduced_data)
# 用真实数字标签看看效果咋样
pplot(reduced_data,test_y)
plot_with_labels(reduced_data,test_y.detach().numpy())
```



相似度矩阵

最后，我们将聚类好的数据根据数字标签进行排序，并计算相似度矩阵，可以看出很明显每一段数字都和自己一类自相关程度很高，对角线颜色非常深。

```
# 根据标签排序
idx = np.argsort(labels)
data_ = data[idx]
data_ = data_.reshape(1000,1,-1)
reduced_data_ = reduced_data[idx]
# 计算相似度矩阵
from sklearn.metrics.pairwise import cosine_similarity
similarity_matrix = np.zeros((1000,1000))
```

```
for i in range(1000):
    for j in range(1000):
        similarity_matrix[i][j]=euclid(reduced_data_[i],reduced_data_[j],2)
pic=similarity_matrix
for i in range(1000):
    for j in range(1000):
        pic[i][j]=-pic[i][j]
for i in range(1000):
        pic[i][i]=0
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
pic=scaler.fit_transform(pic)
plt.imshow(pic,cmap='Reds')
plt.colorbar()
```