实验二 聚类任务

姓名：张辰菁 班级：计算机2103 学号：2215015048

一、实验目的

1.学习使用sklearn的聚类算法；

2.学习使用sklearn的降维方法；

3.应用降维和聚类方法对cifar10、Mnist数据进行聚类。

二、实验过程

1.sklearn降维方法

sklearn库中有主成分分析（PCA）和t-分布随机邻域嵌入（t-SNE）两种降维方法。

1.1 PCA

PCA是一种线性降维技术，它通过将数据投影到主成分空间中，使得投影后的数据在新空间中的方差最大。主成分是原始数据特征的线性组合，且每个主成分相互正交。

优点：

- 简单高效，计算速度快。
- 保留了尽可能多的原始数据的方差信息。
- 结果可以解释为数据的主要方向。

缺点：

- 仅适用于线性关系的数据。
- 对噪声敏感。

1.2 T-SNE

t-SNE是一种非线性降维技术，特别适用于高维数据的可视化。t-SNE通过最小化高维空间中数据点间的相似性和低维空间中数据点间的相似性的差异来保持数据的局部结构。

优点：

- 非线性降维，能有效捕捉复杂的非线性结构。
- 非常适合高维数据的可视化（常用于将数据降到2维或3维）。

缺点：

- 计算复杂度高，适用于小数据集。
- 结果难以解释，主要用于可视化而非建模。

对比两种降维方法而言，对于大型数据集的降维和特征提取，PCA更为高效；对于数据可视化，特别是高维数据的探索性分析，t-SNE则更为强大。

2.sklearn聚类算法

常用的聚类算法有K-means，DBSCAN,OPTICS,Agglomerative Clustering，Spectral Clustering等算法。本次实验中使用了K-means、DBSCAN和Agglomerative Clustering算法。

2.1 K-means

K-Means 是一种基于划分的聚类方法，通过迭代优化将数据集分成K个簇，使得每个簇内的样本与簇中心的距离平方和最小。

```
from sklearn.cluster import AgglomerativeClustering
agg_clustering = AgglomerativeClustering(n_clusters=3)
labels = agg_clustering.fit_predict(X)
```

主要参数为n_clusters,为簇的数量。

2.2 DBSCAN

DBSCAN 是一种基于密度的聚类方法，通过识别密度相连的样本形成簇，能有效处理噪声和发现任意形状的簇。

```
from sklearn.cluster import DBSCAN
dbscan = DBSCAN(eps=0.5, min_samples=5)
labels = dbscan.fit_predict(X)
```

主要参数为eps(领域半径)，min_samples（最小领域密度）

2.3 Agglomerative Clustering

Agglomerative Clustering通过构建聚类树（树状图）来实现，从底部的每个样本开始，逐步将最近的两个簇合并，直到达到所需的簇数。

```
from sklearn.cluster import AgglomerativeClustering
agg_clustering = AgglomerativeClustering(n_clusters=3)
labels = agg_clustering.fit_predict(X)
```

主要参数为n_clusters，为簇的数量，

3.自编写K-means算法

本次主要使用pytorch中的numpy库来实现K-means算法。初始参数为簇心点个数和迭代次数，首先随机生成指定数量的簇心点，然后在每一次迭代中求出每个样本点距离那个中心点最近，对各个簇中样本点的坐标加权平均，计算新的中心点，以此来更新簇心的坐标。经过mat_iter次迭代后，簇心坐标确定，再根据簇心的坐标，来确定每个点分别属于哪个簇，确定每个点的标签。

模型定义

```
class KMeans:
    def __init__(self,n_clusters=5,max_iter=1000):
        self._n_clusters=n_clusters
        self._X=None
        self._y=None
        self._center = None
        self._max_iter=max_iter
    def fit(self,X):
        self._X=X
        n=X.shape[0]
```

```
        d=X.shape[1]
        #随机生成中心点
        self._center = np.array([[np.random.uniform(mi,mx) for mi,mx in
zip(X.min(axis=0),X.max(axis=0))] for _ in range(self._n_clusters)])
        step=0
        #迭代
        while step < self._max_iter:
            #求样本点与每个中心点的距离
            distances = np.array([np.sum((X-self._center[i,:])**2,axis=1) for i
in range(self._n_clusters)])
            #样本距离哪个最近中心点
            self._y = np.argmin(distances.T,axis=1)
            #对样本点加权平均计算新的中心点
            self._center = np.array([np.mean(X[self._y==i,:],axis=0) for i in
range(self._n_clusters)])
            step+=1
        y_pred = np.zeros(n)
        #确定每个点的标签
        for i in range(n):
            best_distance = np.inf
            best_y_pred = 0
            for j in range(self._n_clusters):
                distance = np.sum((X[i,:]-self._center[j,:])**2)
                if distance < best_distance:
                    best_distance = distance
                    best_y_pred = j
            y_pred[i]=best_y_pred
        return y_pred
```

训练过程

```
#使用自己实现的KMeans
n_clusters = 5
kmeans = KMeans(n_clusters)
y_pred = kmeans.fit(X_train)
```

三、实验内容及实验结果

1.读取数据集

本次实验读取了CIFAR10和Mnist数据集作为聚类的数据集。

```
def get_dataset_cifar(split):
    data_path = './datasets/cifar10/' + split
    dataset = ds.Cifar10Dataset(data_path)
    #归一化
    c_trans = []
    c_trans += [
        C.Normalize([0.4914, 0.4822, 0.4465], [0.2023, 0.1994, 0.2010]),
        C.HWC2CHW()
    ]
    dataset = dataset.map(operations=c_trans, input_columns="image",
num_parallel_workers=8)
```

```
    #获取迭代器
    dataset = dataset.batch(1)
    iterator_show = dataset.create_dict_iterator()
    class_a_num = 0
    class_b_num = 0
    class_c_num = 0
    x = []
    y = []

    #numpy 数据处理
    while class_a_num < 500:
        data = next(iterator_show)
        if data["label"].asnumpy() == 0:
            x.append(data["image"].reshape(-1).asnumpy())
            y.append(data["label"].asnumpy())
            class_a_num += 1
    while class_b_num < 500:
        data = next(iterator_show)
        if data["label"].asnumpy() == 5:
            x.append(data["image"].reshape(-1).asnumpy())
            y.append(data["label"].asnumpy())
            class_b_num += 1
    while class_c_num < 500:
        data = next(iterator_show)
        if data["label"].asnumpy() == 9:
            x.append(data["image"].reshape(-1).asnumpy())
            y.append(data["label"].asnumpy())
            class_c_num += 1
    x = np.array(x)
    y = np.array(y)
    print("x.shape:")
    print(x.shape)
    print("y.shape:")
    print(y.shape)
    return x, y
```

使用了mindspore自带的mindspore.dataset(as ds)中的Cifar10Dataset(data_path)来对CIFAR10数据集进行读取，对数据进行归一化，然后取出一批数据，使用dataset.create_dict_iterator()来创建一个迭代器来循环读取该数据集里的数据。这里选取了label=0，5，9的数据各500个读入x_train，y_train，最后需要将x_train和y_train转化为numpy数组。

对于Mnist数据集的读取同理。

```
def get_dataset_MNIST(split):
    data_path = './datasets/MNIST_Data/' + split
    dataset = ds.MnistDataset(data_path)
    class_a_num = 0
    class_b_num = 0
    class_c_num = 0
    class_d_num = 0
    class_e_num = 0
    x = []
    y = []
    #归一化
    c_trans = [
        C.Rescale(1.0 / 255.0, 0),
        C.Normalize(mean=(0.1307,), std=(0.3081,)),
```

```
        C.HWC2CHW()
    ]
    dataset = dataset.map(operations=c_trans, input_columns="image",
num_parallel_workers=8)
    #获取迭代器
    dataset = dataset.batch(1)
    iterator_show = dataset.create_dict_iterator()
    #numpy 数据处理
    while class_a_num < 500:
        data = next(iterator_show)
        if data["label"].asnumpy() == 0:
            x.append(data["image"].reshape(-1).asnumpy())
            y.append(data["label"].asnumpy())
            class_a_num += 1
    while class_b_num < 500:
        data = next(iterator_show)
        if data["label"].asnumpy() == 5:
            x.append(data["image"].reshape(-1).asnumpy())
            y.append(data["label"].asnumpy())
            class_b_num += 1
    while class_c_num < 500:
        data = next(iterator_show)
        if data["label"].asnumpy() == 9:
            x.append(data["image"].reshape(-1).asnumpy())
            y.append(data["label"].asnumpy())
            class_c_num += 1
    while class_d_num < 500:
        data = next(iterator_show)
        if data["label"].asnumpy() == 1:
            x.append(data["image"].reshape(-1).asnumpy())
            y.append(data["label"].asnumpy())
            class_d_num += 1
    while class_e_num < 500:
        data = next(iterator_show)
        if data["label"].asnumpy() == 7:
            x.append(data["image"].reshape(-1).asnumpy())
            y.append(data["label"].asnumpy())
            class_e_num += 1
    x = np.array(x)
    y = np.array(y)
    print("x.shape:")
    print(x.shape)
    print("y.shape:")
    print(y.shape)
    return x, y
```

读取结果

```
x1,y1=get_dataset_cifar("train")
x2,y2=get_dataset_MNIST("train")
```

```
x.shape:
(1500, 3072)
y.shape:
(1500, 1)
x.shape:
(2500, 784)
y.shape:
(2500, 1)
```

```
print(x1[1:10])
print(x2[1:10])
```

```
[[ 813.1913   813.1913   813.1913  ...  793.7985   793.7985   793.7985 ]
 [1253.132   1238.3025  1238.3025  ... 1246.5348  1246.5348  1261.4601 ]
 [1144.3826  1119.6669  1119.6669  ... 1171.9078  1171.9078  1206.7338 ]
 ...
 [ 674.783    689.6124   704.4419  ...  445.53976  475.39053  460.46515]
 [ 640.1809   605.57886  566.0336  ...  525.1418   520.1667   564.9428 ]
 [ 650.0672   645.1241   645.1241  ... 1246.5348  1246.5348  1231.6094 ]]
[[-0.42421296 -0.42421296 -0.42421296 ... -0.42421296 -0.42421296
  -0.42421296]
 [-0.42421296 -0.42421296 -0.42421296 ... -0.42421296 -0.42421296
  -0.42421296]
 [-0.42421296 -0.42421296 -0.42421296 ... -0.42421296 -0.42421296
  -0.42421296]
 ...
 [-0.42421296 -0.42421296 -0.42421296 ... -0.42421296 -0.42421296
  -0.42421296]
 [-0.42421296 -0.42421296 -0.42421296 ... -0.42421296 -0.42421296
  -0.42421296]
 [-0.42421296 -0.42421296 -0.42421296 ... -0.42421296 -0.42421296
  -0.42421296]]
```

2.降维

由于CIFAR10和Mnist为图像数据集，均为高维数据，在对其进行可视化前，需要对数据进行降维；另外，使用降维后的数据进行训练，可以提升训练速度和效率，因为有许多可能有相互联系的特征被合并了，剩余的特征之间都是相互独立的，更加具有训练价值。然而，由于降维后的数据会损失原有的部分信息，可能会导致训练效果变差。

```
#pca 降维
pca = PCA(n_components=2)
x2_lower_pca = pca.fit_transform(x2)
#tsne 降维
t_sne = TSNE(n_components=2)
x1_lower_tsne = t_sne.fit_transform(x1)
```

降维后的shape

```
76 ms
print('x2_lower.pca.shape')
print(x2_lower_pca.shape)
print('x1_lower.tsne.shape')
print(x1_lower_tsne.shape)

x2_lower.pca.shape
(2500, 2)
x1_lower.tsne.shape
(1500, 2)
```

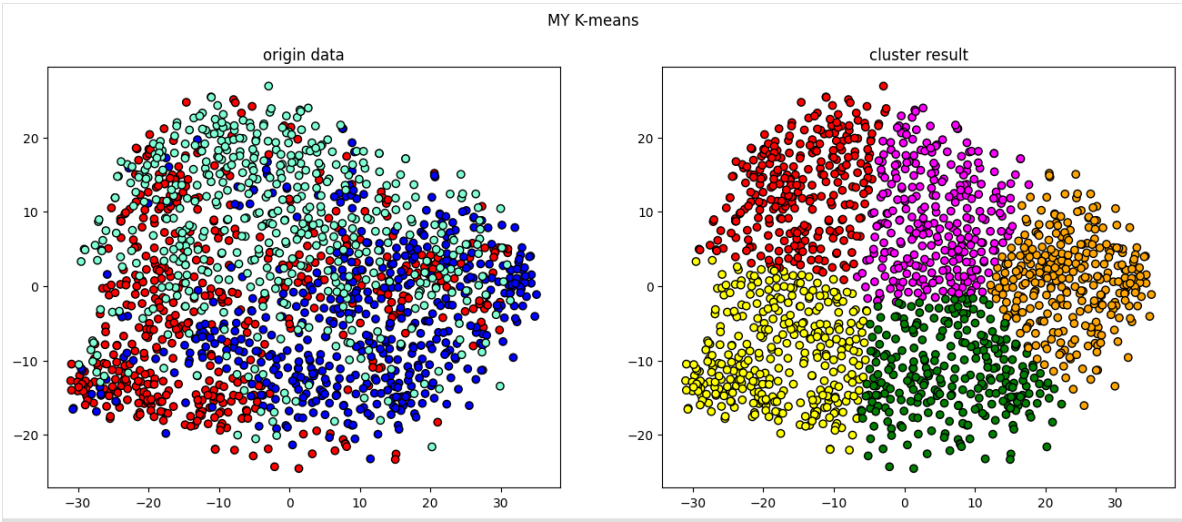
3. 应用聚类算法

3.1 K-means

使用自编K-means算法进行聚类。

```
#自定义K-means
K_means=KMeans(n_clusters=5)
y_pred = K_means.fit(x1_lower_tsne)
visualize(x1_lower_tsne,y1,y_pred,title='K-means')
```

聚类结果



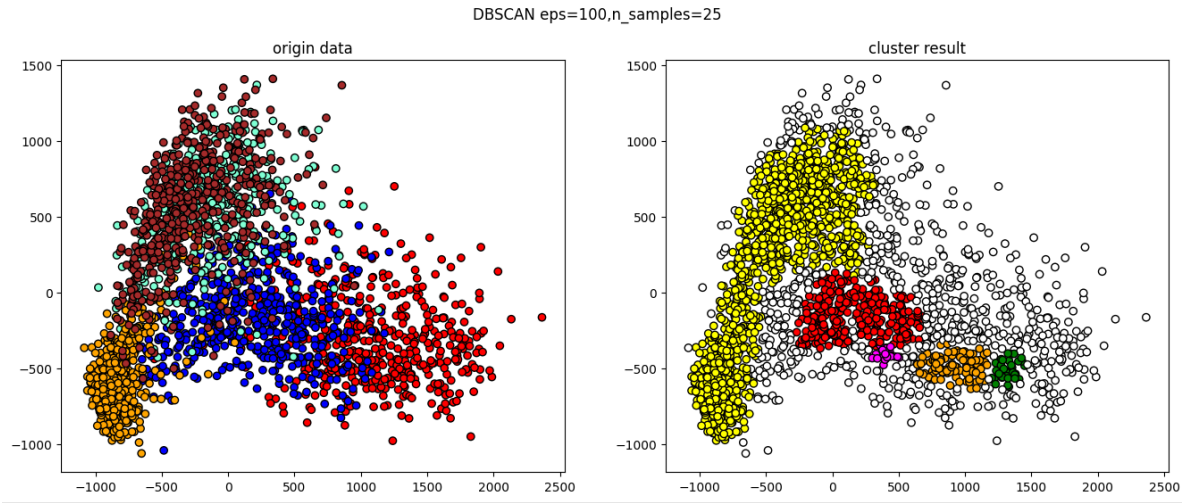
可以看到K-means得到的簇更偏向于球形，由于K-means算法是基于层次划分的聚类算法，如果是非凸集合的数据，则不能够很好地识别相应的簇。

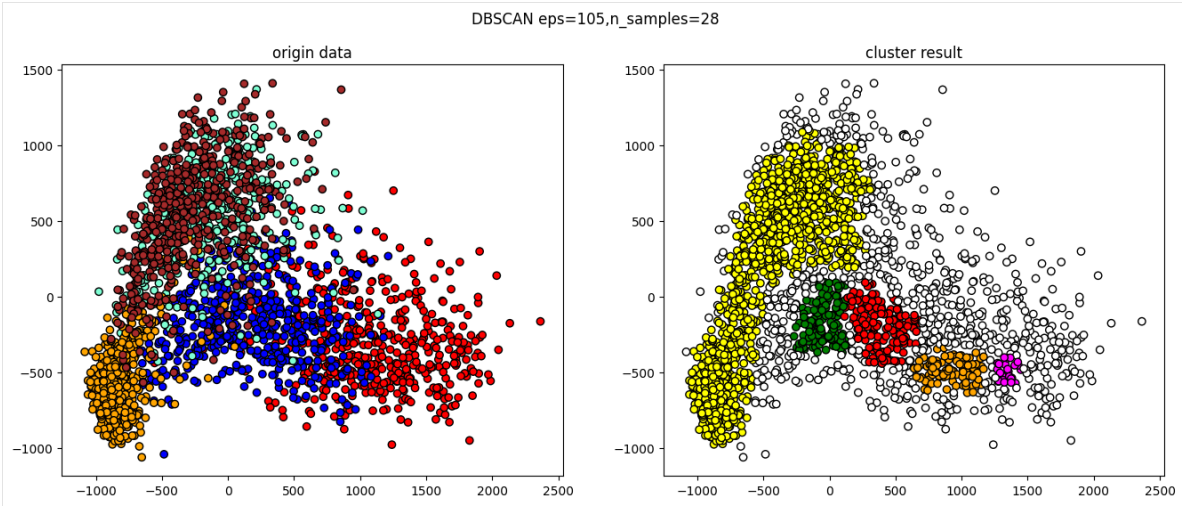
3.2 DBSCAN

使用sk-learn库中的DBSCAN函数进行聚类。

```
eps = 105
n_samples = 28
DBSCAN_preds=DBSCAN(eps=eps, min_samples=n_samples).fit_predict(x2_lower_pca)
visualize(x2_lower_pca,y2,DBSCAN_preds,title=f'DBSCAN eps={eps},n_samples=
{n_samples}')
```

聚类结果





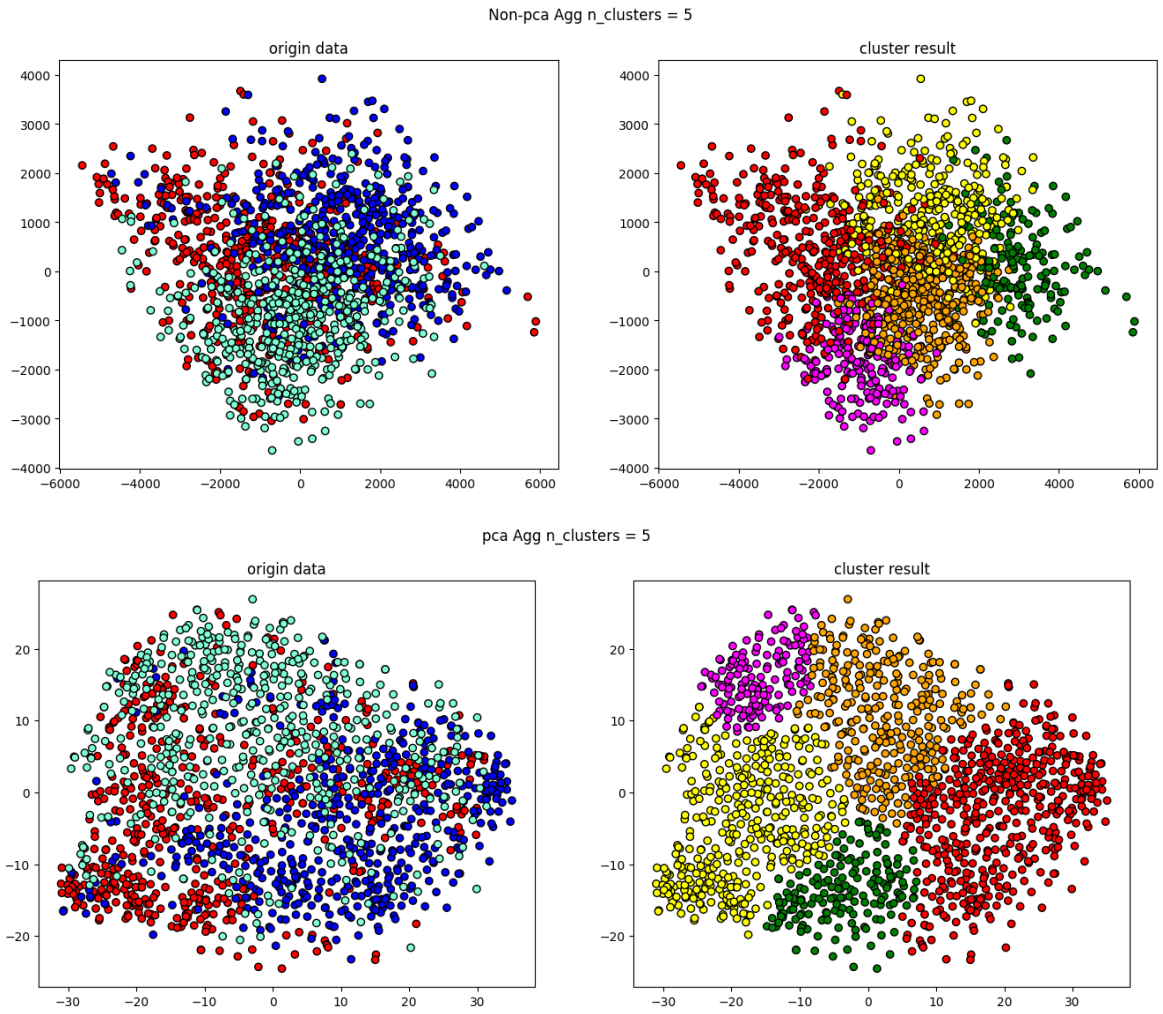
可以看到聚类效果比较差，这是因为DBSCAN算法对于参数非常敏感，需要花费大量时间调整参数才能达到理想的效果，而参数的设置往往依据经验。而对于聚簇不太明显的数据，想要得到一个理想的聚类结果，就需要一点一点调试两个参数。

3.3 Agglomerative Clustering

使用sk-learn库函数进行聚类。

```
n_clusters = 5
ypreds_Agglomerative_pca_lower =
AgglomerativeClustering(n_clusters=n_clusters).fit_predict(x1_lower_tsne)
Agglomerative_preds =
AgglomerativeClustering(n_clusters=n_clusters).fit_predict(x1)
visualize(x1,y1,Agglomerative_preds,title=f'Non-pca Agg n_clusters =
{n_clusters}')
visualize(x1_lower_tsne,y1,ypreds_Agglomerative_pca_lower,title=f'pca Agg
n_clusters = {n_clusters}')
```

聚类结果



可以看到，降维之后再进行聚类，聚类结果可视化(降维后可视化)更加直观清晰。

四、总结

1.本次实验通过使用sk-learn库中的聚类算法和自编K-means算法对CIFAR10和Mnist数据集中的数据进行了聚类。

2.Mindspore中对于数据的归一化与pytorch略有不同。Mindspore自带的数据加载函数(如ds.Cifar10Dataset(data_path)) 中归一化的函数不作为参数，而需要在接下来先定义归一化transform，再使用dataset = dataset.map(operations=c_trans, input_columns="image", num_parallel_workers=8)来对数据进行归一化；而pytorch中自带的数据加载函数(如torchvision.datasets.CIFAR10())中,transform在定义之后，是作为数据加载函数的transform参数传入的。

五、完整源代码

```
from sklearn.cluster import KMeans, DBSCAN, AgglomerativeClustering
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import mindspore.dataset as ds
import matplotlib.pyplot as plt
import numpy as np
import mindspore.dataset.vision.c_transforms as C
import mindspore.dataset.transforms.c_transforms as C2
from mindspore import dtype as mstype
```

```
def visualize(X, y, y_pred, title=None, type='pca'):
    # 可视化,pca降维或者tsne
    if type == 'pca':
        pca = PCA(n_components=2)
        X_dim_reduction = pca.fit_transform(X)

    elif type == 'tsne':
        t_sne = TSNE(n_components=2)
        X_dim_reduction = t_sne.fit_transform(X)
    else:
        X_dim_reduction = X
    # print(X_dim_reduction.shape)
    plt.figure(figsize=(16, 6))

    def get_color(y):
        color_dict = {-1: 'white', 0: 'red', 1: 'orange', 2: 'yellow', 3:
'green', 4: 'magenta',
                      5: 'blue', 6: 'cyan', 7: 'brown', 8: 'purple', 9:
'aquamarine'}
        color = []
        for i in range(y.shape[0]):
            color.append(color_dict.get(y[i].item(), 'olive'))
        return color
    plt.suptitle(title, fontweight=16)
    plt.subplot(121)
    plt.title('origin data')
    #使用降维后的两个维度进行绘图
    plt.scatter(X_dim_reduction[:, 0], X_dim_reduction[:, 1], c=get_color(y),
edgecolors='black')
    plt.subplot(122)
    plt.title('cluster result')
    plt.scatter(X_dim_reduction[:, 0], X_dim_reduction[:, 1],
c=get_color(y_pred), edgecolors='black')
    plt.show()
```

```
def get_dataset_cifar(split):
    data_path = './datasets/cifar10/' + split
    dataset = ds.Cifar10Dataset(data_path)
    #归一化
    c_trans = []
    c_trans += [
        C.Normalize([0.4914, 0.4822, 0.4465], [0.2023, 0.1994, 0.2010]),
        C.HWC2CHW()
    ]
    dataset = dataset.map(operations=c_trans, input_columns="image",
num_parallel_workers=8)

    #获取迭代器
    dataset = dataset.batch(1)
    iterator_show = dataset.create_dict_iterator()
    class_a_num = 0
    class_b_num = 0
    class_c_num = 0
    x = []
    y = []

    #numpy 数据处理
```

```
    while class_a_num < 500:
        data = next(iterator_show)
        if data["label"].asnumpy() == 0:
            x.append(data["image"].reshape(-1).asnumpy())
            y.append(data["label"].asnumpy())
            class_a_num += 1
    while class_b_num < 500:
        data = next(iterator_show)
        if data["label"].asnumpy() == 5:
            x.append(data["image"].reshape(-1).asnumpy())
            y.append(data["label"].asnumpy())
            class_b_num += 1
    while class_c_num < 500:
        data = next(iterator_show)
        if data["label"].asnumpy() == 9:
            x.append(data["image"].reshape(-1).asnumpy())
            y.append(data["label"].asnumpy())
            class_c_num += 1
    x = np.array(x)
    y = np.array(y)
    print("x.shape:")
    print(x.shape)
    print("y.shape:")
    print(y.shape)
    return x, y
```

```
def get_dataset_MNIST(split):
    data_path = './datasets/MNIST_Data/' + split
    dataset = ds.MnistDataset(data_path)
    class_a_num = 0
    class_b_num = 0
    class_c_num = 0
    class_d_num = 0
    class_e_num = 0
    x = []
    y = []
    #归一化
    c_trans = [
        C.Rescale(1.0 / 255.0, 0),
        C.Normalize(mean=(0.1307,), std=(0.3081,)),
        C.HWC2CHW()
    ]
    dataset = dataset.map(operations=c_trans, input_columns="image",
num_parallel_workers=8)
    #获取迭代器
    dataset = dataset.batch(1)
    iterator_show = dataset.create_dict_iterator()
    #numpy 数据处理
    while class_a_num < 500:
        data = next(iterator_show)
        if data["label"].asnumpy() == 0:
            x.append(data["image"].reshape(-1).asnumpy())
            y.append(data["label"].asnumpy())
            class_a_num += 1
    while class_b_num < 500:
        data = next(iterator_show)
        if data["label"].asnumpy() == 5:
```

```
            x.append(data["image"].reshape(-1).asnumpy())
            y.append(data["label"].asnumpy())
            class_b_num += 1
    while class_c_num < 500:
        data = next(iterator_show)
        if data["label"].asnumpy() == 9:
            x.append(data["image"].reshape(-1).asnumpy())
            y.append(data["label"].asnumpy())
            class_c_num += 1
    while class_d_num < 500:
        data = next(iterator_show)
        if data["label"].asnumpy() == 1:
            x.append(data["image"].reshape(-1).asnumpy())
            y.append(data["label"].asnumpy())
            class_d_num += 1
    while class_e_num < 500:
        data = next(iterator_show)
        if data["label"].asnumpy() == 7:
            x.append(data["image"].reshape(-1).asnumpy())
            y.append(data["label"].asnumpy())
            class_e_num += 1
    x = np.array(x)
    y = np.array(y)
    print("x.shape:")
    print(x.shape)
    print("y.shape:")
    print(y.shape)
    return x, y
```

```
x1,y1=get_dataset_cifar("train")
x2,y2=get_dataset_MNIST("train")
```

```
class MYKMeans:
    def __init__(self,n_clusters=5,max_iter=1000):
        self._n_clusters=n_clusters
        self._X=None
        self._y=None
        self._center = None
        self._max_iter=max_iter

    def fit(self,X):
        self._X=X
        n=X.shape[0]
        d=X.shape[1]
        #随机生成中心点
        self._center = np.array([[np.random.uniform(mi,mx) for mi,mx in
zip(X.min(axis=0),X.max(axis=0))] for _ in range(self._n_clusters)])
        step=0
        #迭代
        while step < self._max_iter:
            #求样本点与每个中心点的距离
            distances = np.array([np.sum((X-self._center[i,:])**2,axis=1) for i
in range(self._n_clusters)])
            #样本距离哪个最近中心点
            self._y = np.argmin(distances.T,axis=1)
            #对样本点加权平均计算新的中心点
```

```
            self._center = np.array([np.mean(X[self._y==i,:],axis=0) for i in
range(self._n_clusters)])
            step+=1
        y_pred = np.zeros(n)
        #确定每个点的标签
        for i in range(n):
            best_distance = np.inf
            best_y_pred = 0
            for j in range(self._n_clusters):
                distance = np.sum((X[i,:]-self._center[j,:])**2)
                if distance < best_distance:
                    best_distance = distance
                    best_y_pred = j
            y_pred[i]=best_y_pred
        return y_pred
```

```
#pca 降维
pca = PCA(n_components=2)
x2_lower_pca = pca.fit_transform(x2)
#tsne 降维
t_sne = TSNE(n_components=2)
x1_lower_tsne = t_sne.fit_transform(x1)
```

```
print('x2_lower.pca.shape')
print(x2_lower_pca.shape)
print('x1_lower.tsne.shape')
print(x1_lower_tsne.shape)
```

```
K_means=MYKMeans(n_clusters=5)
y_pred = K_means.fit(x1_lower_tsne)
visualize(x1_lower_tsne,y1,y_pred,title='MY K-means')
```

```
eps = 105
n_samples = 23
DBSCAN_preds=DBSCAN(eps=eps, min_samples=n_samples).fit_predict(x2_lower_pca)
visualize(x2_lower_pca,y2,DBSCAN_preds,title=f'DBSCAN eps={eps},n_samples=
{n_samples}')
```

```
n_clusters = 5
ypreds_Agglomerative_pca_lower =
AgglomerativeClustering(n_clusters=n_clusters).fit_predict(x1_lower_tsne)
Agglomerative_preds =
AgglomerativeClustering(n_clusters=n_clusters).fit_predict(x1)
visualize(x1,y1,Agglomerative_preds,title=f'Non-pca Agg n_clusters =
{n_clusters}')
visualize(x1_lower_tsne,y1,ypreds_Agglomerative_pca_lower,title=f'pca Agg
n_clusters = {n_clusters}')
```