t-SNE可视化笔记

by Nuster@SDU

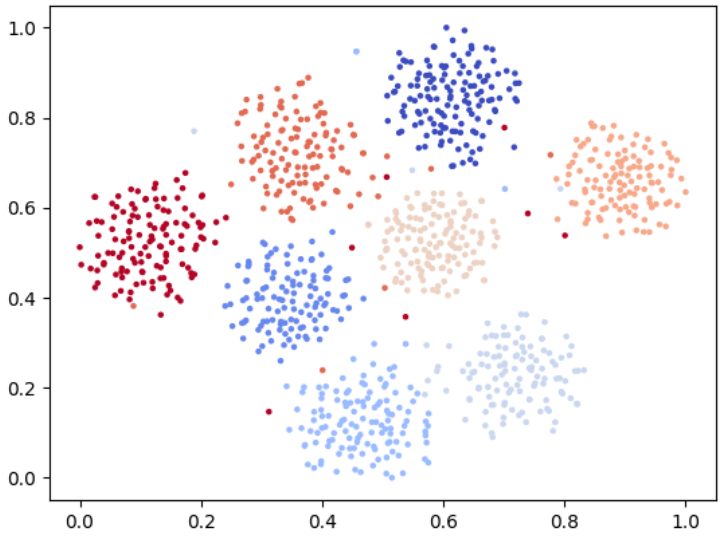
1 概述

t-SNE全称是t-distributed stochastic neighbor embedding，可以用于高维数据降维，在论文中常用来进行数据的可视化处理。

从本质上来说，t-SNE的降维属于无监督学习，原理好像类似使用概率进行各点的概率建模，然后优化概率分布的KL散度，具体原理可详见论文*Visualizing Data using t-SNE*，这里暂且只知概况，等日后有兴趣再研究一下。

使用t-SNE进行降维可视化，通常是将一组高维可分的数据点，向二维空间映射，且能能够依次体现它的高维可分性，一般会呈现出聚类特征。

以下图为例，该数据集是我用程序采样生成后进行t-SNE过程，然后绘制的点图，具体描述详见下文。这个图可以清楚地看出有8个聚类簇，而事实上这确实是由8个参数不同的生成器采样生成的1000个64维数据点，它们在高维上可分，映射到二维平面呈现聚类簇。



所以t-SNE能够一定程度上体现特征的质量，或者说可分性。

2 实验

这一部分主要有构建数据集、t-SNE数据降维和绘制散点图构成。

其中t-SNE数据降维调用sklearn包，调用的包在下面。

```
import random
import numpy as np
from sklearn import manifold
import matplotlib.pyplot as plt
```

构建数据集

数据集由C个高斯采样器生成，每个高斯采样器都能采样生成K维数据，在这K维数据中，每维都由一维高斯分布随机采样。

在实验中，使用N=1000，K=64，C=8。

对高斯采样器定义如下，其中dim=K，每次调用sample都会生成一个K维样本。

```
class Gauss():
    def __init__(self,tag,dim):
        self.tag=tag
        self.dim=dim
        self.mu=[np.random.random() for i in range(self.dim)]
        self.sigma=[np.random.random() for i in range(self.dim)]

    def sample(self):
        x=np.random.normal(self.mu,self.sigma)
        return x
```

我们构建一个数据集，代码如下，其中n_sample=N。

```
def sampleBatch(n_sample,dim):
    X,Y=[],[]
    gaussList=[Gauss(i,dim) for i in range(Category)]
    for i in range(n_sample):
        gauss=random.sample(gaussList,1)
        X.append(gauss[0].sample().tolist())
        Y.append(gauss[0].tag)

    return np.array(X),np.array(Y)
```

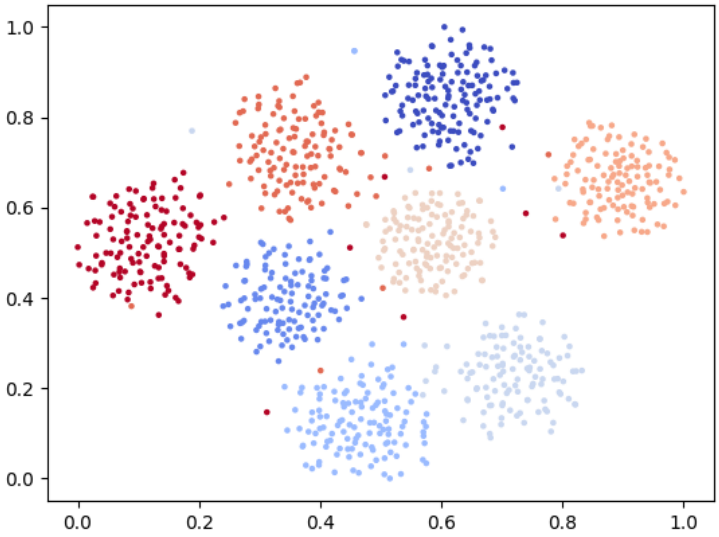
t-SNE降维和绘图

采样函数准备好了，接下来我们进行数据的t-SNE可视化实验。

```
if __name__ == '__main__':
    X,Y=sampleBatch(N,K)

    projector=manifold.TSNE(n_components=2)
    X_tsne=projector.fit_transform(X)

    X_tsne=normalize(X_tsne).T

    plt.scatter(X_tsne[0],X_tsne[1],s=5,c=Y,cmap='coolwarm')
    plt.show()
```

在上述代码中，首先先构建数据集，然后调用sklearn中manifold包的TSNE函数，其中参数n_components为降维后的维度，要绘制平面散点图所以为2。然后normalize是归一化函数(自己随手写一个)，归一化到[0,1]范围内才好绘图。最后用pyplot绘制散点图即可。

最后结果就是之前的那个图。



还是把normalize函数放一下吧hhhh。

```
def normalize(X):
    maxcol=X.max(axis=0)
    mincol=X.min(axis=0)

    tmp=np.zeros((X.shape[0],X.shape[1]))
    for i in range(X.shape[1]):
        tmp[:,i]=(X[:,i]-mincol[i])/(maxcol[i]-mincol[i])
    return tmp
```

一些延伸

我把参数调了一下，得到了一些经验上的关系，其实这本身也很intuitive了。

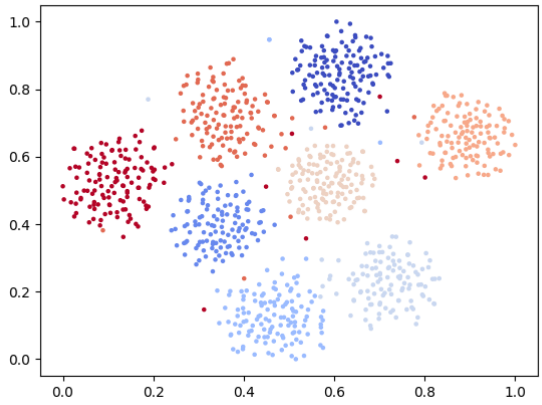
随着K增大，簇的分类会更加明晰(图AB对照)。

随着N增大，簇的分类会更加明晰(图AC对照)。

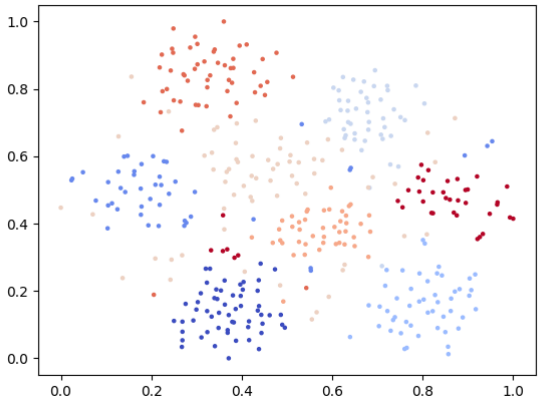
随着C减小，簇的分类会更加明晰(图CD对照)。

对比一下下面四张图

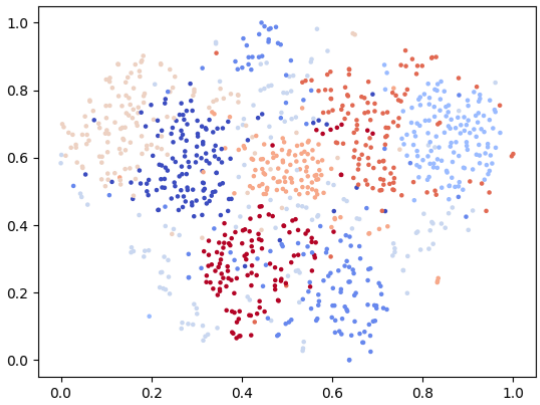
图A：N=1000，K=64，C=8。



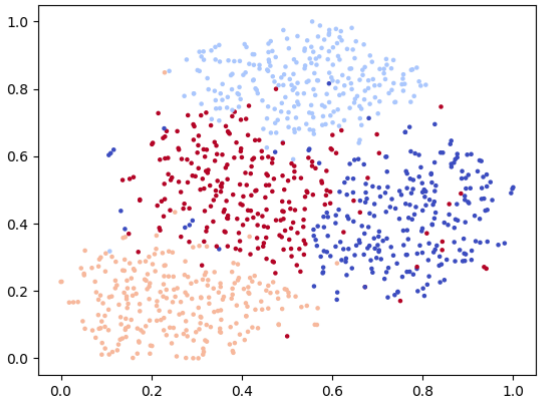
图B：N=400，K=64，C=8。



图C：N=1000，K=20，C=8。



图D：N=1000，K=20，C=4。



感觉思路就是cluster嘛。

3 总结

这个算法好像2008年才提出来，相对而言不是很早，用的话直接sklearn一句代码就出来了。最后还是把传到GitHub上吧，需要的话可以参考一下。