学习向量量化

> 今天我将介绍一种和k-Means很相似的算法，叫做学习向量量化（Learning Vector Quantization）算法，这种算法也是希望通过寻找一组原型向量来刻画聚类的结构。这种“聚类”算法很奇特，它不再是无监督学习，而是一种需要预设标签的学习算法，学习过程通过这些监督信息来辅助聚类。

算法介绍

```
输入：样本集、原型向量的个数、各个原型向量预设类别标记、学习率（0，1）
过程：
 1. 初始化一组原型向量
 2. repeat
 3.     从样本集中随机选取样本
 4.     计算样本与各个原型向量之间的距离
 5.     找出与样本最近的原型向量
 6.     if 样本标签与最近的原型向量一致
 7.         原型向量更新：p = p + eta * (x - p)
 8.     else
 9.         原型向量更新：p = p - eta * (x - p)
10. until 满足停止条件
```

算法可以说是很清晰了，大白话来说，就是先选几个向量作为原型向量，然后在样本集中随机选取样本，计算和原型向量的距离，找出最邻近的原型向量，看看他们的标签是不是一样，一样的话原型向量向样本按照学习率靠拢，否则远离。就上面的过程，不断的迭代，直到满足迭代的停止条件。需要声明一点，这个迭代条件的设置有不同的方法，可以设置最大迭代轮次、也可以设置原型向量的更新阈值（如更新值很小就停止），需要根据需求设置。
整个算法最重要的部分集中在7、9两行，这两行的更新公式其实很好理解，eta是学习率，p是原型向量。若样本和原型向量标签一致，说明真实的原型向量是靠近这个样本的，那么就要向这个样本的方向靠拢，也就是+(x - p)这部分；若不一致，说明不在一个簇里，那么就该远离，远离的方向是-(x - p)。
LVD算法其实也是一种基于竞争的学习，这点和无监督的SOM算法挺像的。LVD算法可以被视为一种网络，由输入层、竞争层、输出层组成。输入层很容易理解，就是接受样本的输入；竞争层可以被视为神经元之间的竞争，也就是原型向量之间的竞争，离得最近的神经元（原型向量）获胜，赢者通吃（winner-take-all）；输出层负责输出分类结果。不论是如何理解这个算法，其实本质都是一样的，也就是同类靠拢、异类远离。

算法复现

```
# -*- coding: utf-8 -*-
"""
Created on Tue Jan 29 20:22:18 2019

@author: zmddzf
"""
import numpy as np
import random
from tqdm import tqdm
import matplotlib.pyplot as plt


class LVQ:
    """
    学习向量化算法实现
    attributes:
        train:LVQ
        predict: 预测一个样本所属的簇
    """
    def __init__(self, D, T, lr, maxEpoch):
        """
        初始化LVQ，构造器
        :param D: 训练集，格式为[[array, label],...]
        :param T: 原型向量类别标记
        :param lr: 学习率，0-1之间
        :param maxEpoch: 最大迭代次数
        """
        self.D = D
        self.T = T
        self.lr = lr
```

```
        self.maxEpoch = maxEpoch
        self.P = []
        #初始化原型向量，随机选取
        for t in T:
            while True:
                p = random.choice(self.D)
                if p[1] != t:
                    pass
                else:
                    self.P.append(p)
                    break

    def __dist(self, p1, p2):
        """
        私有属性，计算距离
        :param p1: 向量1
        :param p2: 向量2
        :return dist: 距离
        """
        dist = np.linalg.norm(p1 - p2)
        return dist

    def train(self):
        """
        训练LVQ
        :return self.P: 训练后的原型向量
        """
        for epoch in tqdm(range(self.maxEpoch)):
            x = random.choice(self.D) #从训练集随机选取样本
            dist = []
            for p in self.P:
                dist.append(self.__dist(p[0], x[0])) #计算距离列表

            t = self.P[dist.index(min(dist))][1] #确定对应最小距离原型向量的类别
            if t == x[1]:
                #若类别一致，则靠拢
                self.P[dist.index(min(dist))][0] = self.P[dist.index(min(dist))][0] + self.lr*(x[0] - self.P[dist.index(min(dist))][0])
            else:
                #若类别不同，则远离
                self.P[dist.index(min(dist))][0] = self.P[dist.index(min(dist))][0] - self.lr*(x[0] - self.P[dist.index(min(dist))][0])
        return self.P

    def predict(self, x):
        """
        预测样本所属的簇
        :param x: 样本向量
        :return label: 样本的分类结果
        """
        dist = []
        for p in self.P:
            dist.append(self.__dist(p[0], x))
        label = self.P[dist.index(min(dist))][1]
        return label



#生成实验数据集，数据集是两个正态分布二维点集
mu1 = 2; sigma1 = 1
mu2 = 4; sigma2 = 1
#生成第一个正态分布
samples1 = np.array([np.random.normal(mu1, sigma1, 50), np.random.normal(mu1, sigma1, 50)])
samples1 = samples1.T.tolist()
label1 = [1 for i in range(50)]
#生成第二个正态分布
samples2 = np.array([np.random.normal(mu2, sigma2, 50), np.random.normal(mu2, sigma2, 50)])
samples2 = samples2.T.tolist()
label2 = [0 for i in range(50)]
#合并生成数据集
samples = samples1 + samples2
labels = label1 + label2

#修改数据格式
data = []
for s, l in zip(samples, labels):
    data.append([np.array(s), l])

#开始训练
lvq = LVQ(data, [0, 1], 0.1, 5000)
vector = lvq.train()

#使用lvq分类
prediction = []
for i in data:
```

```
    prediction.append(lvq.predict(i[0]))

#计算accuracy
accuracy = 0
for pred, label in zip(prediction, labels):
    if pred == label:
        accuracy += 1
accuracy = accuracy / len(data)
print("accuracy of LVQ:", accuracy)

#画图展示原型向量和散点
plt.figure(figsize=(15,10))
plt.scatter(np.array(samples).T[0], np.array(samples).T[1], c = labels)
plt.scatter(vector[0][0][0], vector[0][0][1], marker = '*', s = 300)
plt.scatter(vector[1][0][0], vector[1][0][1], marker = '*', s = 300)

plt.show()
```