学习向量量化 (LVQ)

作者：Lonery
原文链接：https://ld246.com/article/1614744595834
来源网站：链滴

学习向量量化

“学习向量量化”(Learning Vector Quantization，简称LVQ)是一种利用监督信息辅助的聚类算法像K-Meas算法一样也是通过调整一组类似于质心的点来进行聚类。

算法描述
</br>

<p1 style="text-indent:2em">

假设给定样本D={(X1,Y1),(X2,Y2),...,(Xm,Ym)},X表示样本的属性，Y表示样本的类别标记。LVQ目的学习一组n维原型向量{P1,P2,...Pq}，当样本点距离这组原型向量的某个分向量距离最近时那么它的类标记应当与这个原型分向量一致。原型向量的训练过程和K-Means算法大同小异，初始时通过随机一原型向量(可以从样本点中随机提取)，当与原型向量最近的样本点类别标记与此原型向量的类别不同，那么要调整原型向量，使得原型向量离当前的样本点远一些，相反则需要调整它离当前样本点近一，当算法满足停止条件时(达到最大迭代次数，或原型向量更新变化很小甚至不变时)则可以返回这一原型向量。

</p1>

输入：样本集 $D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\ldots,(\boldsymbol{x}_m,y_m)\}$;
原型向量个数 q, 各原型向量预设的类别标记 $\{t_1,t_2,\ldots,t_q\}$;
学习率 $\eta\in(0,1)$.

过程：
1: 初始化一组原型向量 $\{\boldsymbol{p}_1,\boldsymbol{p}_2,\ldots,\boldsymbol{p}_q\}$
2: **repeat**
3: 从样本集 D 随机选取样本 $(\boldsymbol{x}_j,y_j)$;
4: 计算样本 $\boldsymbol{x}_j$ 与 $\boldsymbol{p}_i$ $(1\leqslant i\leqslant q)$ 的距离: $d_{ji}=\|\boldsymbol{x}_j-\boldsymbol{p}_i\|_2$;
5: 找出与 $\boldsymbol{x}_j$ 距离最近的原型向量 $\boldsymbol{p}_{i^*}$, $i^*=\arg\min_{i\in\{1,2,\ldots,q\}} d_{ji}$;
6: **if** $y_j=t_{i^*}$ **then**
7: $\boldsymbol{p}'=\boldsymbol{p}_{i^*}+\eta\cdot(\boldsymbol{x}_j-\boldsymbol{p}_{i^*})$
8: **else**
9: $\boldsymbol{p}'=\boldsymbol{p}_{i^*}-\eta\cdot(\boldsymbol{x}_j-\boldsymbol{p}_{i^*})$
10: **end if**
11: 将原型向量 $\boldsymbol{p}_{i^*}$ 更新为 $\boldsymbol{p}'$
12: **until** 满足停止条件

输出：原型向量 $\{\boldsymbol{p}_1,\boldsymbol{p}_2,\ldots,\boldsymbol{p}_q\}$

图 9.4 学习向量量化算法

原文链接：学习向量量化 (LVQ)

代码实现：

生成数据集：

```
X=datasets.make_blobs(n_samples=1000,centers=3) #1000个样本点分为3类
```

初始化原型向量：

```
P=np.zeros((q,col)) #原型向量
    for i in range(q):   #初始化原型向量
        index=np.where(sample[1]==Label[i])[0]
        choose=np.random.randint(0,len(index),1)
        P[i,:]=sample[0][index[choose],:]
```

训练主体：

```
for i in range(1000):   #训练
    choose=np.random.randint(0,row,1) #随机选取一个样本
    dis=np.linalg.norm(sample[0][choose,:]-P,axis=1) #计算与原型向量的距离
    y=dis.tolist().index(min(dis))  #获取距离最近的原型向量下标
    if Label[y]==sample[1][choose]: #更新原型向量
        P[y,:]=P[y,:]+eta*(sample[0][choose,:]-P[y,:])
    else:
        P[y,:]=P[y,:]-eta*(sample[0][choose,:]-P[y,:])
```

完整代码：

```
from sklearn import datasets
import matplotlib.pyplot as plt
import numpy as np
X=datasets.make_blobs(n_samples=1000,centers=3) #1000个样本点分为3类

def lvq(sample,q,Label,eta):
    if q!=len(Label):
        return 0
    row,col=np.shape(sample[0]) #获取样本集的规格
    P=np.zeros((q,col)) #原型向量
    for i in range(q):   #初始化原型向量
        index=np.where(sample[1]==Label[i])[0]
        choose=np.random.randint(0,len(index),1)
        P[i,:]=sample[0][index[choose],:]
    for i in range(1000):   #训练
        choose=np.random.randint(0,row,1) #随机选取一个样本
        dis=np.linalg.norm(sample[0][choose,:]-P,axis=1) #计算与原型向量的距离
        y=dis.tolist().index(min(dis))  #获取距离最近的原型向量下标
        if Label[y]==sample[1][choose]: #更新原型向量
            P[y,:]=P[y,:]+eta*(sample[0][choose,:]-P[y,:])
        else:
            P[y,:]=P[y,:]-eta*(sample[0][choose,:]-P[y,:])
    IDX=[]  #分类标记
    for i in sample[0]:  #以距离最近的标记为样本的类别
        D=np.linalg.norm(i-P,axis=1)
```

```
        y=D.tolist().index(min(D))
        IDX.append(Label[y])
    plot(IDX,sample[0],max(Label)+1,P)
    return P
def plot(a,X,k,p):  #绘画板块
    m=k
    for j in range(m):
        index=[i for i,v in enumerate(a) if v==j]
        x=[]
        y=[]
        for k in index:
            x.append(X[k][0])
            y.append(X[k][1])
        plt.scatter(x,y)
    plt.scatter(p[:,0],p[:,1],marker='x')
    plt.show()
```

测试代码:

```
lvq(X,5,[0,1,0,1,2],0.3)
```

```
array([[  7.02402226,   3.74801884],
       [-10.38672182,   3.03633487],
       [  8.20578187,   4.94912081],
       [ -8.30953565,   3.69883677],
       [  7.69178827,  -7.90175793]])
```