

Python编程与人工智能实践

算法篇：朴素贝叶斯

于泓

鲁东大学

信息与电气工程学院

2021.3.20

朴素贝叶斯 (Naive Bayes)

- 一种基于统计概率的分类方法，通过计算样本发生的概率，来实现分类。

概率相关知识

x_1	x_2	x_3	x_4	Y
年龄	有工作	有自己的房子	信贷情况	是否放贷
0	0	0	0	No
0	0	0	1	No
0	1	0	1	Yes
0	1	1	0	Yes
0	0	0	0	No
1	0	0	0	No
1	0	0	1	No
1	1	1	1	Yes
1	0	1	2	Yes
1	0	1	2	Yes
2	0	1	2	Yes
2	0	1	1	Yes
2	1	0	1	Yes
2	1	0	2	Yes
2	0	0	0	No

关系:

$$P(XY)=P(X|Y)*P(Y)=P(Y|X)*P(X)$$

独立分布

$$P(Y)$$

$$P(Y=yes)=\frac{9}{15}=0.6$$

$$P(Y=no)=\frac{6}{15}=0.4$$

$$P(x_3)$$

$$P(x_3=0)=\frac{9}{15}=0.6$$

$$P(x_3=1)=\frac{6}{15}=0.4$$

联合分布

$$P(x_3, Y)$$

$$P(x_3=1, Y=yes)=\frac{6}{15}=0.4$$

$$P(x_3=0, Y=yes)=\frac{3}{15}=0.2$$

$$P(x_3=1, Y=no)=\frac{0}{15}=0$$

$$P(x_3=0, Y=no)=\frac{6}{15}=0.4$$

条件分布

$$P(x_3|Y)$$

$$P(x_3=1|Y=yes)=\frac{6}{9}=0.67$$

$$P(x_3=0|Y=yes)=\frac{3}{9}=0.33$$

$$P(Y|x_3)$$

$$P(Y=yes|x_3=0)=\frac{3}{9}=0.33$$

$$P(Y=no|x_3=0)=\frac{6}{9}=0.67$$

x_1	x_2	x_3	x_4	Y
年龄	有工作	有 自 己 的 房子	信贷情况	是否放贷
0	0	0	0	No
0	0	0	1	No
0	1	0	1	Yes
0	1	1	0	Yes
0	0	0	0	No
1	0	0	0	No
1	0	0	1	No
1	1	1	1	Yes
1	0	1	2	Yes
1	0	1	2	Yes
2	0	1	2	Yes
2	0	1	1	Yes
2	1	0	1	Yes
2	1	0	2	Yes
2	0	0	0	No

$$P(XY)=P(X|Y)*P(Y)=P(Y|X)*P(X)$$

$$P(Y|X)=\frac{P(X|Y)*P(Y)}{P(X)}$$

贝叶斯公式

已知X求Y

$$P(Y=\text{yes}|x_1,x_2,x_3,x_4)=\frac{P(x_1,x_2,x_3,x_4|Y=\text{yes})*P(Y=\text{yes})}{P(x_1,x_2,x_3,x_4)}$$

谁大 ?

$$P(Y=\text{no}|x_1,x_2,x_3,x_4)=\frac{P(x_1,x_2,x_3,x_4|Y=\text{no})*P(Y=\text{no})}{P(x_1,x_2,x_3,x_4)}$$

$$P(x_1,x_2,x_3,x_4|Y)=P(x_1|Y)P(x_2|Y)P(x_3|Y)P(x_4|Y)$$

似然函数

$$\text{LLK}(Y|X) = \log(P(Y)) + \sum_{i=1}^4 \log(P(x_i | Y))$$

谁大
 $\text{LLK}(Y=\text{yes}|X)$? $\text{LLK}(Y=\text{no}|X)$ 二
分类

谁最大?
 $\text{LLK}(Y=C_1|X)$ $\text{LLK}(Y=C_2|X)$ $\text{LLK}(Y=C_3|X)$ 多
分类

```
def createDataSet():
    dataSet = [[0, 0, 0, 0, 'no'], #数据集
               [0, 0, 0, 1, 'no'],
               [0, 1, 0, 1, 'yes'],
               [0, 1, 1, 0, 'yes'],
               [0, 0, 0, 0, 'no'],
               [1, 0, 0, 0, 'no'],
               [1, 0, 0, 1, 'no'],
               [1, 1, 1, 1, 'yes'],
               [1, 0, 1, 2, 'yes'],
               [1, 0, 1, 2, 'yes'],
               [2, 0, 1, 2, 'yes'],
               [2, 0, 1, 1, 'yes'],
               [2, 1, 0, 1, 'yes'],
               [2, 1, 0, 2, 'yes'],
               [2, 0, 0, 0, 'no']]
    labels = ['年龄', '有工作', '有自己的房子', '信贷情况']
    return dataSet, labels #返回数据
```

计算 $P(X|Y)$

```
# 获取概率模型, 输入feat np.array格式 大小[N,D]
def trainPbmodel_X(feats):
    N,D = np.shape(feats)

    model = {}
    # 对每一维度的特征进行概率统计
    for d in range(D):
        data = feats[:,d].tolist()
        keys = set(data)
        N = len(data)
        model[d] = {}
        for key in keys:
            model[d][key] = float(data.count(key)/N)
    return model
```

```
# datas: list格式 每个元素表示1个特征序列
# labs: list格式 每个元素表示一个标签
def trainPbmodel(datas,labs):
```

```
    # 定义模型
```

```
    model = {}
```

```
    # 获取分类的类别
```

```
    keys = set(labs)
```

```
    for key in keys:
```

```
        # 获得P(Y)
```

```
        Pbmodel_Y = labs.count(key)/len(labs)
```

```
    # 收集标签为Y的数据
```

```
    index = np.where(np.array(labs)==key)[0].tolist()
```

```
    feats = np.array(datas)[index]
```

```
    # 获得 P(X|Y)
```

```
    Pbmodel_X = trainPbmodel_X(feats)
```

```
    # 模型保存
```

```
    model[key]={}
```

```
    model[key]["PY"] = Pbmodel_Y
```

```
    model[key]["PX"] = Pbmodel_X
```

```
    return model
```

```
{'yes': {'PY': 0.6, 'PX': {0: {0: 0.2222222222222222, 1: 0.3333333333333333, 2: 0.4444444444444444}, 1: {0: 0.4444444444444444, 1: 0.5555555555555556}, 2: {0: 0.3333333333333333, 1: 0.6666666666666666}, 3: {0: 0.1111111111111111, 1: 0.4444444444444444, 2: 0.4444444444444444}}}, 'no': {'PY': 0.4, 'PX': {0: {0: 0.5, 1: 0.3333333333333333, 2: 0.1666666666666666}, 1: {0: 1.0}, 2: {0: 1.0}, 3: {0: 0.6666666666666666, 1: 0.3333333333333333}}}}
```

```

# feat : list格式 一条输入特征
# model: 训练的概率模型
# keys : 考察标签的种类
def getPbfromModel(feats,model,keys):
    results={}
    eps = 0.00001
    for key in keys:
        # 获取P(Y)
        PY = model.get(key,eps).get("PY")

        # 分别获取 P(X|Y)
        model_X = model.get(key,eps).get("PX")
        list_px=[]
        for d in range(len(feats)):
            pb = model_X.get(d,eps).get(feats[d],eps)
            list_px.append(pb)

        result = np.log(PY) + np.sum(np.log(list_px))
        results[key]= result
    return results

```

```

if __name__ == '__main__':
    dataSet, labels = createDataSet()

    datas = [i[:-1] for i in dataSet]
    labs = [i[-1] for i in dataSet]

    model = trainPbmodel(datas,labs)
    print(model)
    feat = [0,1,0,1]
    result = getPbfromModel(feats,model,keys)

    print(result)

    # 遍历结果找到最大值
    for key,value in result.items():
        if(value == max(result.values())):
            print(key)

```

```

{'yes': -4.512232190328822, 'no': -14.220975666072437}
yes

```


隐形眼镜数据集

['年龄','近视/远视','是否散光','是否眼干']

训练

	train-lenses.txt	DecisionTree.py	train-len
1	青年	近视	否 干涩 不配镜
2	青年	近视	否 正常 软镜片
3	青年	近视	是 干涩 不配镜
4	青年	远视	否 干涩 不配镜
5	青年	远视	是 干涩 不配镜
6	中年	近视	否 干涩 不配镜
7	中年	近视	否 正常 软镜片
8	中年	近视	是 干涩 不配镜
9	中年	近视	是 正常 硬镜片
10	中年	远视	否 干涩 不配镜
11	中年	远视	否 正常 软镜片
12	中年	远视	是 干涩 不配镜
13	中年	远视	是 正常 不配镜
14	老年	近视	否 正常 不配镜
15	老年	近视	是 干涩 不配镜
16	老年	近视	是 正常 硬镜片
17	老年	远视	否 干涩 不配镜
18	老年	远视	否 正常 软镜片
19	老年	远视	是 干涩 不配镜
20	老年	远视	是 正常 不配镜
21	青年	远视	是 正常 硬镜片

测试

	test-lenses.txt	train-lenses.txt	Decision
1	青年	远视	否 正常 软镜片
2	老年	近视	否 干涩 不配镜
3	青年	近视	是 正常 硬镜片

```
if __name__ == '__main__':
    with open("train-lenses.txt", 'r') as f:
        lines = f.read().splitlines()

    dataSet = [line.split('\t') for line in lines]
```

```
datas = [i[:-1] for i in dataSet]
labs = [i[-1] for i in dataSet]
```

```
keys = set(labs)
model = trainPbmodel(datas, labs)
print(model)
```

测试

```
with open("test-lenses.txt", 'r') as f:
    lines = f.read().splitlines()

for line in lines:
    data = line.split('\t')
    lab_true = data[-1]
    feat = data[:-1]
    result = getPbfromModel(feat, model, keys)

    key_out = ""
    for key, value in result.items():
        if (value == max(result.values())):
            key_out = key
    print("输入特征: ")
    print(data)
    print(result)
    print("预测结果 %s 医生推荐 %s"%(key_out, lab_true))
```

```
{'不配镜': {'PY': 0.6666666666666666, 'PX': {0: {'老年': 0.35714285714285715, '中年': 0.35714285714285715, '青年': 0.2857142857142857}, 1: {'近视': 0.42857142857142855, '远视': 0.5714285714285714}, 2: {'是': 0.5714285714285714, '否': 0.42857142857142855}, 3: {'干涩': 0.7857142857142857, '正常': 0.21428571428571427}}}, '硬镜片': {'PY': 0.14285714285714285, 'PX': {0: {'老年': 0.3333333333333333, '中年': 0.3333333333333333, '青年': 0.3333333333333333}, 1: {'近视': 0.6666666666666666, '远视': 0.3333333333333333}, 2: {'是': 1.0}, 3: {'正常': 1.0}}}, '软镜片': {'PY': 0.19047619047619047, 'PX': {0: {'老年': 0.25, '中年': 0.5, '青年': 0.25}, 1: {'近视': 0.5, '远视': 0.5}, 2: {'否': 1.0}, 3: {'正常': 1.0}}}}
```

```
输入特征:
['青年', '远视', '否', '正常', '软镜片']
{'不配镜': -4.605586765873308, '硬镜片': -15.656060191361762, '软镜片': -3.737669618283368}
预测结果 软镜片 医生推荐 软镜片
输入特征:
['老年', '近视', '否', '干涩', '不配镜']
{'不配镜': -3.370842302880618, '硬镜片': -26.475838475772044, '软镜片': -15.250595083253597}
预测结果 不配镜 医生推荐 不配镜
输入特征:
['青年', '近视', '是', '正常', '硬镜片']
{'不配镜': -4.605586765873308, '硬镜片': -3.4499875458315876, '软镜片': -15.250595083253597}
预测结果 硬镜片 医生推荐 硬镜片
```

$$f_{yes}(X; \lambda)$$

$$P(Y=yes|x_1, x_2, x_3, x_4) = \frac{P(x_1, x_2, x_3, x_4 | Y=yes) * P(Y=yes)}{P(x_1, x_2, x_3, x_4)}$$

谁大？

一般都相等

$$P(Y=no|x_1, x_2, x_3, x_4) = \frac{P(x_1, x_2, x_3, x_4 | Y=no) * P(Y=no)}{P(x_1, x_2, x_3, x_4)}$$

$$f_{no}(X; \lambda)$$

```
{'yes': {'PY': 0.6, 'PX': {0: {0: 0.2222222222222222, 1: 0.3333333333333333, 2: 0.4444444444444444}, 1: {0: 0.4444444444444444, 1: 0.5555555555555556}, 2: {0: 0.3333333333333333, 1: 0.6666666666666666}, 3: {0: 0.1111111111111111, 1: 0.4444444444444444, 2: 0.4444444444444444}}}, 'no': {'PY': 0.4, 'PX': {0: {0: 0.5, 1: 0.3333333333333333, 2: 0.1666666666666666}, 1: {0: 1.0}, 2: {0: 1.0}, 3: {0: 0.6666666666666666, 1: 0.3333333333333333}}}}
```

(1) 由属于 Y_{yes} 类的数据训练而成; λ 由 Y_{yes} 类数据统计得到

$$f_{yes}(X; \lambda)$$

(2) 当输入属于 Y_{yes} 类的数据时, 输出较大值, 当输入不属于 Y_{yes} 数据时输出较小值

(3) 设定 f_{yes} 的函数形式为一个分支函数

其他函数形式?