哈爾濱工程大學

课　程　设　计

课程名称：　机器学习第二次作业

任课教师：　钱 真

学　　号：　S322517454

姓　　名：　张云涛

完成时间：　2022/10/4

哈尔滨工程大学研究生院

1 作业要求

1. 样本属性可任意输入设定
2. 先验概率基于训练样本集合自动求得
3. 混淆矩阵维度可任意设定

2 实现方案

csv 文件中的训练样本数据支持增删属性，以满足要求 1 样本属性可任意输入设定。
先验概率在 BayesClassfier 类中实现基于训练样本集合自动求得，满足了要求 2。
混淆矩阵维度与训练样本数据种类一致，可支持多分类，满足要求 3。

BayesClassifier 实现贝叶斯分类的机制如下

1. 基于样本训练集计算先验概率
2. 根据输入的测试数据计算后验概率
3. 基于后验概率与输入的混淆矩阵计算条件风险
4. 取条件风险值最小对应的类型实现分类

3 程序源代码

```
1 # -*- encoding: utf-8 -*-
2 # @Author: 张云涛 S322517454
3 # @Date: 2022/10/04
4 # @File: bayes_classifier.py
5
6 import numpy as np
7 import pandas as pd
8 import collections
9
```

```
10 def load_data(data_path):
11     df = pd.read_csv(data_path)
12     # 删除属性
13     del df['编号']
14     del df['密度']
15     del df['含糖率']
16     data = df.values
17     return data
18 
19 class BayesClassifier:
20     """贝叶斯分类器"""
21     def __init__(self, data):
22         self.data = data
23         self.data_cnt = collections.Counter(self.data
               [:, -1])
24 
25     def cal_px(self):
26         """计算先验概率P(Xi)"""
27         data = self.data
28         p_x = dict()
29         all_data = np.array(data)
30         for k in range(all_data.shape[1]):
31             p_x[k] = dict()
32             tags = list(set(all_data[:, k]))
33             d = all_data[:, k]
34             tag_cnt = collections.Counter(d)
35             for tag in tags:
36                 # 平滑处理
37                 p_x[k][tag] = (tag_cnt[tag]+1)/float(
                       len(d)+len(tags))
38         # print("P(X)\t: ", p_x)
39         return p_x
40 
```

```
41      def cal_py(self):
42          """计算先验概率P(Y)"""
43          data = self.data
44          data_cnt = self.data_cnt
45          labels = data[:, -1]
46          p_y = dict.fromkeys(data_cnt.keys())
47          for tag in p_y.keys():
48              p_y[tag] = data_cnt[tag]/len(labels)
49          # print("P(Y)\t: ", p_y)
50          return p_y
51
52      def cal_pxy(self):
53          # 计算先验概率P(Xi|Y)
54          data = self.data
55          data_cnt = self.data_cnt
56          p_xy = dict.fromkeys(data_cnt.keys())
57          for tag in p_xy.keys():
58              p_xy[tag] = dict()
59          for label in data_cnt.keys():
60              sub_data = data[data[:, -1] == label]
61              sub_data = np.array(sub_data)
62              for k in range(sub_data.shape[1]):
63                  p_xy[label][k] = dict()
64                  d = sub_data[:, k]
65                  tags = list(set(data[:, k]))
66                  tag_cnt = collections.Counter(d)
67                  for tag in tags:
68                      # 平滑处理, 避免概率为0的情况
69                      p_xy[label][k][tag] = (tag_cnt[tag
                            ]+1)/float(len(d) + len(tags))
70          # print("P(Xi|Y)\t: ", p_xy)
71          return p_xy
72
```

```
73      def cal_pyx(self, test_data):
74          """计算P(Y|X)"""
75          tags = list(self.data_cnt.keys())
76          p_xy = self.cal_pxy()
77          p_y = self.cal_py()
78          p_x = self.cal_px()
79          print("先验概率:")
80          print("P(X):\t", p_x)
81          print("P(Y):\t", p_y)
82          print("P(X|Y):\t", p_xy)
83
84          p_yx = dict.fromkeys(self.data_cnt.keys())
85          for j, label in enumerate(tags):
86              p_yx[label] = np.math.log(p_y[label], 2)
87              for k, tag in enumerate(test_data):
88                  # 采用对数运算, 避免下溢
89                  p_yx[label] = p_yx[label] + np.math.
                        log(p_xy[label][k][tag], 2) - np.
                        math.log(p_x[k][tag], 2);
90          # print("P(Y|Xi)\t: ", p_yx)
91          print("后验概率:\nP(Y|X):\t", p_yx)
92          return p_yx
93
94      def cal_r(self, con_matrix, test_item):
95          '''计算条件风险'''
96          p_yx = self.cal_pyx(test_item)
97          r = dict.fromkeys(con_matrix.keys())
98          for p_tag in con_matrix.keys():   # 是否
99              r[p_tag] = 0
100             for tag in con_matrix[p_tag].keys():
101                 r[p_tag] += con_matrix[p_tag][tag] *
                        p_yx[tag]
102         # 对条件风险值进行排序, 返回最小值对应种类
```

```
103             res = sorted(r.items(), key=lambda kv:(kv[1],
                    kv[0]))
104             print("条件风险:\nR:\t", res)
105             return res[0][0]
106
107         def test_classifier(self, con_matrix, test_data):
108             """测试分类器"""
109             predict_vec = list()
110             for i, item in enumerate(test_data):
111                 print("\n", 20*'-', "␣第{0}次测试␣".format
                        (i+1), 20*'-')
112                 print("混淆矩阵:\t", con_matrix)
113                 print("测试数据:\t", item)
114                 res = self.cal_r(con_matrix, item)
115                 predict_vec.append(res)
116             print("\n测试结果:\t", predict_vec)
117             return predict_vec
118
119
120 if __name__ == '__main__':
121     data_path = './watermelon.csv'
122     train_data = load_data(data_path)
123     test_data = [['青绿', '蜷缩', '浊响', '清晰', '凹
            陷', '软粘'], ['浅白', '稍蜷', '浊响', '稍糊',
            '凹陷', '硬滑']]
124     classifier = BayesClassifier(train_data)
125     # 混淆矩阵
126     con_matrix ={'是':{'是':0, '否':1}, '否':{'是':1,
            '否':0}}
127     # classifier.cal_r(con_matrix, test_data)
128     res = classifier.test_classifier(con_matrix,
            test_data)
129     print("res:␣", res)
```

4 程序训练数据

训练样本数据保存在一个 csv 文件中，由程序进行加载。

编号	色泽	根蒂	敲声	纹理	脐部	触感	密度	含糖率	好瓜
1	青绿	蜷缩	浊响	清晰	凹陷	硬滑	0.697	0.46	是
2	乌黑	蜷缩	沉闷	清晰	凹陷	硬滑	0.774	0.376	是
3	乌黑	蜷缩	浊响	清晰	凹陷	硬滑	0.634	0.264	是
4	青绿	蜷缩	沉闷	清晰	凹陷	硬滑	0.608	0.318	是
5	浅白	蜷缩	浊响	清晰	凹陷	硬滑	0.556	0.215	是
6	青绿	稍蜷	浊响	清晰	稍凹	软粘	0.403	0.237	是
7	乌黑	稍蜷	浊响	稍糊	稍凹	软粘	0.481	0.149	是
8	乌黑	稍蜷	浊响	清晰	稍凹	硬滑	0.437	0.211	是
9	乌黑	稍蜷	沉闷	稍糊	稍凹	硬滑	0.666	0.091	否
10	青绿	硬挺	清脆	清晰	平坦	软粘	0.243	0.267	否
11	浅白	硬挺	清脆	模糊	平坦	硬滑	0.245	0.057	否
12	浅白	蜷缩	浊响	模糊	平坦	软粘	0.343	0.099	否
13	青绿	稍蜷	浊响	稍糊	凹陷	硬滑	0.639	0.161	否
14	浅白	稍蜷	沉闷	稍糊	凹陷	硬滑	0.657	0.198	否
15	乌黑	稍蜷	浊响	清晰	稍凹	软粘	0.36	0.37	否
16	浅白	蜷缩	浊响	模糊	平坦	硬滑	0.593	0.042	否
17	青绿	蜷缩	沉闷	稍糊	稍凹	硬滑	0.719	0.103	否

5 程序运行结果