

IML 第二次作业

习题 3.2

令 $y = \frac{1}{1 + e^{-(\omega^\top x + b)}}$, $l(\beta) = \sum_{i=1}^m (-y_i \beta^\top \hat{x}_i + \ln(1 + e^{\beta^\top \hat{x}_i}))$, 这两个函数关于 w 和 $\beta = (w; b)$ 是二阶可微的, 分别计算二者的 Hessian 矩阵:

$$\begin{aligned} \frac{\partial y}{\partial \omega} &= \frac{e^{-(\omega^\top x + b)}}{\left[1 + e^{-(\omega^\top x + b)}\right]^2} x \\ \frac{\partial^2 y}{\partial \omega \partial \omega^\top} &= \frac{\partial}{\partial \omega^\top} \frac{\partial y}{\partial \omega} \\ &= \frac{\partial}{\partial \omega^\top} \frac{e^{-(\omega^\top x + b)}}{\left[1 + e^{-(\omega^\top x + b)}\right]^2} x \\ &= \frac{e^{-(\omega^\top x + b)} \left[1 - e^{-(\omega^\top x + b)}\right]}{\left[1 + e^{-(\omega^\top x + b)}\right]^3} x x^\top \\ &= y(1 - y)(1 - 2y) x x^\top \end{aligned}$$

矩阵 $x x^\top$ 半正定, 而 $y(1 - y)(1 - 2y) < 0$ (as $y \in (\frac{1}{2}, 1)$), 其 Hessian 矩阵不总非负, 即 y 是非凸的。

$$\begin{aligned} \frac{\partial l}{\partial \beta} &= \sum_{i=1}^m \left(-y_i \hat{x}_i + \frac{e^{\beta^\top \hat{x}_i}}{1 + e^{\beta^\top \hat{x}_i}} \hat{x}_i \right) \\ \frac{\partial^2 l}{\partial \beta \partial \beta^\top} &= \frac{\partial}{\partial \beta^\top} \frac{\partial l}{\partial \beta} \\ &= \frac{\partial}{\partial \beta^\top} \sum_{i=1}^m \left(-y_i \hat{x}_i + \frac{e^{\beta^\top \hat{x}_i}}{1 + e^{\beta^\top \hat{x}_i}} \hat{x}_i \right) \\ &= \sum_{i=1}^m \frac{e^{\beta^\top \hat{x}_i}}{(1 + e^{\beta^\top \hat{x}_i})^2} \hat{x}_i \hat{x}_i^\top \end{aligned}$$

矩阵 $\hat{x}_i \hat{x}_i^\top$ 半正定, 而 $\frac{e^{\beta^\top \hat{x}_i}}{(1 + e^{\beta^\top \hat{x}_i})^2} \hat{x}_i \hat{x}_i^\top > 0$, 所以其 Hessian 矩阵半正定, 即 $l(\beta)$ 是凸的。

习题 3.7

设类别 i 的 ECOC 码为 r_i , 其反码为 \tilde{r}_i , 定义 $d(r_i, r_j)$ 为其海明距离 (编码不同的位数)。对同等长度的编码, 理论上来说, 任意两个类别之间的编码距离越

远, 则越好。并且对于好的编码, 还要避免一个编码是另一个编码的反码的情况出现, 所以最大化的目标为

$$l = \prod_{1 \leq i < j \leq 4} d(r_i, r_j) d(r_i, \tilde{r}_j) + \sum_{1 \leq i < j \leq 4} d(r_i, r_j) d(r_i, \tilde{r}_j)$$

编写 C 代码程序 (程序代码附后), 搜索得出解为

$$C_1 = 0000000000 \quad C_2 = 101010100 \quad C_3 = 110011000 \quad C_4 = 111100000$$

事实上, T. G. Dietterich 等人 1995 年在 *Solving Multiclass Learning Problems via Error-Correcting Output Codes* 中指出得出在分类数为 4 时的计算方法, 并且最后两位可以任意取值, 对结论不造成影响。

作业 3.3

多分类情形下的 $S_b = \sum_{i=1}^N m_i (\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^\top$ 。

$$S_b = [(\boldsymbol{\mu}_1 - \boldsymbol{\mu}), (\boldsymbol{\mu}_2 - \boldsymbol{\mu}), \dots, (\boldsymbol{\mu}_N - \boldsymbol{\mu})] \begin{pmatrix} m_1 & 0 & 0 \\ 0 & \dots & 0 \\ 0 & 0 & m_N \end{pmatrix} \begin{pmatrix} (\boldsymbol{\mu}_1 - \boldsymbol{\mu})^\top \\ \dots \\ (\boldsymbol{\mu}_N - \boldsymbol{\mu})^\top \end{pmatrix}$$

记 $\mathbf{M} = \text{diag}(m_1, m_2, \dots, m_N)$, $\mathbf{A} = [(\boldsymbol{\mu}_1 - \boldsymbol{\mu}), (\boldsymbol{\mu}_2 - \boldsymbol{\mu}), \dots, (\boldsymbol{\mu}_N - \boldsymbol{\mu})]^\top$, 则

$$\begin{aligned} \text{rank } S_b &= \text{rank } \mathbf{A}^\top \mathbf{M} \mathbf{A} \\ &= \text{rank } \mathbf{A}^\top \mathbf{M}^{\frac{1}{2}} \mathbf{M}^{\frac{1}{2}} \mathbf{A} \\ &= \text{rank } \left(\mathbf{A}^\top \mathbf{M}^{\frac{1}{2}} \right) \left(\mathbf{A}^\top \mathbf{M}^{\frac{1}{2}} \right)^\top \\ &= \text{rank } \left(\mathbf{A}^\top \mathbf{M}^{\frac{1}{2}} \right) \\ &= \text{rank } \mathbf{A}^\top \end{aligned}$$

因为 $\sum_{i=1}^N m_i \boldsymbol{\mu}_i = \left(\sum_{i=1}^N m_i \right) \boldsymbol{\mu}$, 即 $\sum_{i=1}^N m_i (\boldsymbol{\mu}_i - \boldsymbol{\mu}) = \mathbf{0}$, 所以 $\text{rank } \mathbf{A}^\top \leq N - 1$ 。

作业 3.4

式 3.44 是 $\max_{\mathbf{W}} \frac{\text{tr}(\mathbf{W}^\top \mathbf{S}_b \mathbf{W})}{\text{tr}(\mathbf{W}^\top \mathbf{S}_w \mathbf{W})}$, 如果 \mathbf{W} 是一个解, 那么 $\alpha \mathbf{W}, \alpha \in \mathbb{R}$ 也是一个解, 于是可固定 $\text{tr}(\mathbf{W}^\top \mathbf{S}_w \mathbf{W}) = 1$, 求解 $-\text{tr}(\mathbf{W}^\top \mathbf{S}_b \mathbf{W})$ 的最小值。

由拉格朗日乘子法, 定义拉格朗日函数

$$L(\mathbf{W}, \lambda) = -\text{tr}(\mathbf{W}^\top \mathbf{S}_b \mathbf{W}) + \lambda (\text{tr}(\mathbf{W}^\top \mathbf{S}_w \mathbf{W}) - 1).$$

对上式关于 \mathbf{W} 求偏导得

$$\begin{aligned}\frac{\partial L(\mathbf{W}, \lambda)}{\partial \mathbf{W}} &= -\frac{\partial (\text{tr}(\mathbf{W}^T \mathbf{S}_b \mathbf{W}))}{\partial \mathbf{W}} + \lambda \frac{\partial (\text{tr}(\mathbf{W}^T \mathbf{S}_w \mathbf{W}) - 1)}{\partial \mathbf{W}} \\ &= -(\mathbf{S}_b + \mathbf{S}_b^T) \mathbf{W} + \lambda (\mathbf{S}_w + \mathbf{S}_w^T) \mathbf{W} \\ &= -2\mathbf{S}_b \mathbf{W} + 2\lambda \mathbf{S}_w \mathbf{W}\end{aligned}$$

令 $L(\mathbf{W}, \lambda) = 0$ 可得 $\mathbf{S}_b \mathbf{W} = \lambda \mathbf{S}_w \mathbf{W}$ 。

作业 3.5

对称性：

$$\begin{aligned}(\mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T)^T &= (\mathbf{X}^T)^T ((\mathbf{X}^T \mathbf{X})^{-1})^T \mathbf{X}^T \\ &= (\mathbf{X}^T)^T ((\mathbf{X}^T \mathbf{X})^T)^{-1} \mathbf{X}^T \\ &= \mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T\end{aligned}$$

幂等性：

$$\begin{aligned}(\mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T)^2 &= \mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \\ &= \mathbf{X} \mathbf{I} (\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T \\ &= \mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T\end{aligned}$$

所以矩阵 $\mathbf{X}(\mathbf{X}^T \mathbf{X})^{-1} \mathbf{X}^T$ 是投影矩阵。

将特征矩阵 \mathbf{X} 看作是一个由 d 个 n 维列向量组成的向量组。假设 $d < n$ 且所有列向量都线性无关，那 \mathbf{X} 张成的空间是 d 维度空间。真实值 \mathbf{y} 是一个 n 维空间中的 $n \times 1$ 向量。线性回归就是在 \mathbf{X} 张成的 d 维空间中，寻找 n 维空间中 \mathbf{y} 的投影，也就是一种降维的操作。

习题 4.1

用反证法。假设对于不含冲突数据的某个数据集，不存在与训练集一致的决策树，说明训练得到的任意一种决策树，都至少存在一个节点无法划分所有数据，否则决策树的构造过程保证其一定能够将当前节点所有数据划分出去，这与不含冲突数据矛盾。

习题 4.9

基于 4.4.2 节的定义 (式 4.9,4.10,4.11), 将基尼指数的计算推广为

$$\begin{aligned}\text{Gini_index}(D,a) &= \rho \times \text{Gini_index}(\tilde{D},a) \\ &= \rho \sum_{v=1}^{|V|} \tilde{r}_v \text{Gini_index}(\tilde{D}^v) \\ &= \rho \sum_{v=1}^{|V|} \tilde{r}_v \left(1 - \sum_{k=1}^{|y|} \tilde{p}_k^2 \right)\end{aligned}$$

作业 4.3

构造优化问题

$$\begin{aligned}\max H(\mathbf{p}) \\ \text{s.t. } \sum_k p_k = 1\end{aligned}$$

由拉格朗日乘数法, 其拉格朗日函数为

$$L(\lambda, \mathbf{p}) = H(\mathbf{p}) + \lambda(p_1 + \dots + p_K - 1)$$

对每个 p_i , 都令

$$\frac{\partial L}{\partial p_i} = -\log_2 e(\ln p_i + 1) + \lambda = 0$$

即 $\lambda = \log_2 e(\ln p_i + 1)$, 由于 $y = \ln x$ 是严格单调函数, 所以当最大值条件满足时 (即上式), 必有 $p_1 = \dots = p_K$, 即 X 服从均匀分布。

作业 4.4

(a) 按各属性计算如下:

A 属性: $p_1 = p(A = T) = \frac{4}{10}$, $p_2 = p(A = F) = \frac{6}{10}$, $H = -p_1 \log_2 p_1 - p_2 \log_2 p_2 = 0.971$ 。

B 属性: $p_1 = p(B = T) = \frac{5}{10}$, $p_2 = p(B = F) = \frac{5}{10}$, $H = -p_1 \log_2 p_1 - p_2 \log_2 p_2 = 1$ 。

C 属性: $p_7 = p_5 = \frac{2}{10}$, $p_1 = p_2 = p_3 = p_4 = p_6 = p_8 = \frac{1}{10}$, $H = -\sum_{k=1}^8 p_k \log_2 p_k = 2.922$ 。

类别属性: $p_1 = p(+) = \frac{5}{10}$, $p_2 = p(-) = \frac{5}{10}$, $H = -p_1 \log_2 p_1 - p_2 \log_2 p_2 = 1.000$ 。

(b) 记整个数据集为 D , 由 (a) 得 $\text{Ent}(D) = 1$, 则 A 的信息增益为

$$\text{Gain}(D,A) = \text{Ent}(D) - \sum_{v=1}^2 \frac{|D^v|}{|D|} \text{Ent}(D^v)$$

在这个式子里

$$Ent(D^1) = -\frac{3}{4} \log_2 \frac{3}{4} - \frac{1}{4} \log_2 \frac{1}{4} = 0.811$$

$$Ent(D^2) = -\frac{2}{6} \log_2 \frac{2}{6} - \frac{4}{6} \log_2 \frac{4}{6} = 0.918$$

所以

$$Gain(D,A) = 1 - (\frac{4}{10} * 0.811 + \frac{6}{10} * 0.918) = 0.125$$

B 的信息增益为

$$Gain(D,B) = Ent(D) - \sum_{v=1}^2 \frac{|D^v|}{|D|} Ent(D^v)$$

在这个式子里

$$Ent(D^1) = -\frac{2}{5} \log_2 \frac{2}{5} - \frac{3}{5} \log_2 \frac{3}{5} = 0.971$$

$$Ent(D^2) = -\frac{3}{5} \log_2 \frac{3}{5} - \frac{2}{5} \log_2 \frac{2}{5} = 0.971$$

所以

$$Gain(D,B) = 1 - (\frac{5}{10} * 0.971 + \frac{5}{10} * 0.971) = 0.029$$

(c) C 属性是连续值，计算值可列表如下：

	a^1	a^2	a^3	a^4	a^5	a^6	a^7	a^8
	1.0	2.0	3.0	4.0	5.0	6.0	7.0	8.0
	t^1	t^2	t^3	t^4	t^5	t^6	t^7	
	1.5	2.5	3.5	4.5	5.5	6.5	7.5	
$Ent(D)$	1.0							
$Ent(D_t^-)$	0	0	0.918	0.811	1.0	0.985	0.991	
$Ent(D_t^+)$	0.991	0.954	0.985	0.918	1.0	0.918	0	
$ D_t^- $	1	2	3	4	6	7	9	
$ D_t^+ $	9	8	7	6	4	3	1	
$Gain(D, a, t)$	0.108	0.237	0.035	0.125	0	0.035	0.108	

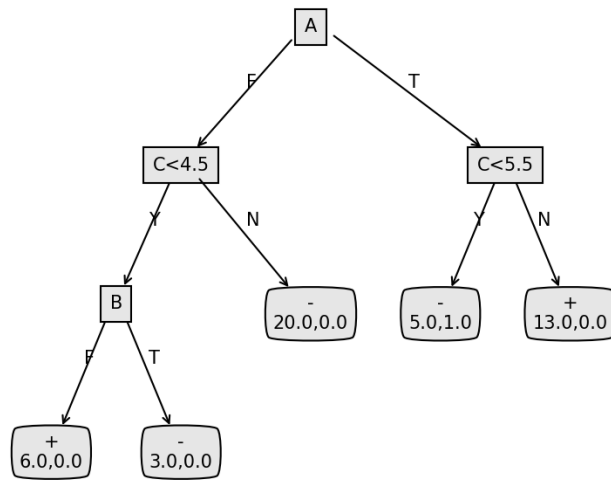
(d) 按书上公式计算如下：

$$Gini_index(D,A) = \frac{4}{10}(1 - \frac{3^2}{4^2} - \frac{1^2}{4^2}) + \frac{6}{10}(1 - \frac{2^2}{6^2} - \frac{4^2}{6^2}) = 0.417$$

$$Gini_index(D,B) = \frac{5}{10}(1 - \frac{2^2}{5^2} - \frac{3^2}{5^2}) + \frac{5}{10}(1 - \frac{2^2}{5^2} - \frac{3^2}{5^2}) = 0.48$$

A 属性划分后的基尼指数最小，所以是最优划分。

(e) 使用 Python 实现 C4.5 决策树算法，生成的决策树如下：



使用的代码附后。

习题 3.7 的代码

```
1 #include <iostream>
2 #include <vector>
3 #include <string>
4
5 int dist(int a, int b) {
6     int dist_return = 0;
7     for (int i = 0; i <= 8; i++) {
8         dist_return += ((a >> i) & 1) ^ ((b >> i) & 1);
9     }
10    return dist_return;
11 }
12
13 int L(int a, int b, int c, int d) {
14     int d1 = dist(a, b);
15     int d2 = dist(a, c);
16     int d3 = dist(a, d);
17     int d4 = dist(b, c);
18     int d5 = dist(b, d);
19     int d6 = dist(c, d);
20     int d1_ = dist(a, ~b);
21     int d2_ = dist(a, ~c);
22     int d3_ = dist(a, ~d);
23     int d4_ = dist(b, ~c);
24     int d5_ = dist(b, ~d);
25     int d6_ = dist(c, ~d);
26     return d1 * d2 * d3 * d4 * d5 * d6 * d1_ * d2_ * d3_ * d4_ * d5_ *
27         d6_ +
28         d1 * d1_ + d2 * d2_ + d3 * d3_ + d4 * d4_ + d5 * d5_ + d6 *
29         d6_;
30 }
31
32 int main() {
33     int hi = 0;
34     int hj = 0;
35     int hk = 0;
36     int hg = 0;
37     int hs = 0;
38
39     const int min = 0b0000000000;
40     const int max = 0b1111111111;
41     for (int i = min; i <= max - 3; i++) {
42         for (int j = i + 1; j <= max - 2; j++) {
```

```
41     for (int k = j + 1; k <= max - 1; k++) {
42         for (int g = k + 1; g <= max; g++) {
43             int ns = L(i, j, k, g);
44             if (ns > hs) {
45                 hs = ns;
46                 hi = i;
47                 hj = j;
48                 hk = k;
49                 hg = g;
50             }
51         }
52     }
53 }
54 printf("process: i= %4x\n", i);
55 }
56
57 printf("%4x, %4x, %4x, %4x, socre: %d\n", hi, hj, hk, hg, hs);
58 }
```

作业 4.4 的代码

仅提供 C4.5 算法部分。

```
1 from math import log
2 import operator
3 import os
4
5 import re
6 from numpy import inf
7 import copy
8
9
10 # 计算信息熵
11 def calcShannonEnt(dataSet, labelIndex):
12     # type: (list) -> float
13     numEntries = 0 # 样本数(按权重计算)
14     labelCounts = {}
15     for featVec in dataSet: # 遍历每个样本
16         if featVec[labelIndex] != 'N':
17             weight = float(featVec[-2])
18             numEntries += weight
19             currentLabel = featVec[-1] # 当前样本的类别
20             if currentLabel not in labelCounts.keys(): # 生成类别字典
```



```

21         labelCounts[currentLabel] = 0
22         labelCounts[currentLabel] += weight # 数据集的倒数第二个
值用来标记样本权重
23     shannonEnt = 0.0
24     for key in labelCounts: # 计算信息熵
25         prob = float(labelCounts[key]) / numEntries
26         shannonEnt = shannonEnt - prob * log(prob, 2)
27     return shannonEnt
28
29
30 def splitDataSet(dataSet, axis, value, LorR='N'):
31     """
32     type: (list, int, string or float, string) -> list
33     划分数据集
34     axis: 按第几个特征划分
35     value: 划分特征的值
36     LorR: N 离散属性; L 小于等于value值; R 大于value值
37     """
38     retDataSet = []
39     featVec = []
40     if LorR == 'N': # 离散属性
41         for featVec in dataSet:
42             if featVec[axis] == value:
43                 reducedFeatVec = featVec[:axis]
44                 reducedFeatVec.extend(featVec[axis + 1:])
45                 retDataSet.append(reducedFeatVec)
46     elif LorR == 'L':
47         for featVec in dataSet:
48             if featVec[axis] != 'N':
49                 if float(featVec[axis]) < value:
50                     retDataSet.append(featVec)
51     elif LorR == 'R':
52         for featVec in dataSet:
53             if featVec[axis] != 'N':
54                 if float(featVec[axis]) > value:
55                     retDataSet.append(featVec)
56     return retDataSet
57
58
59 def splitDataSetWithNull(dataSet, axis, value, LorR='N'):
60     """
61     type: (list, int, string or float, string) -> list
62     划分数据集

```

```
63     axis:按第几个特征划分
64     value:划分特征的值
65     LorR: N 离散属性; L 小于等于value值; R 大于value值
66     """
67     retDataSet = []
68     nullDataSet = []
69     featVec = []
70     totalWeightV = calcTotalWeight(dataSet, axis, False) # 非空样本权重
71     totalWeightSub = 0.0
72     if LorR == 'N': # 离散属性
73         for featVec in dataSet:
74             if featVec[axis] == value:
75                 reducedFeatVec = featVec[:axis]
76                 reducedFeatVec.extend(featVec[axis + 1:])
77                 retDataSet.append(reducedFeatVec)
78             elif featVec[axis] == 'N':
79                 reducedNullVec = featVec[:axis]
80                 reducedNullVec.extend(featVec[axis + 1:])
81                 nullDataSet.append(reducedNullVec)
82     elif LorR == 'L':
83         for featVec in dataSet:
84             if featVec[axis] != 'N':
85                 if float(featVec[axis]) < value:
86                     retDataSet.append(featVec)
87             elif featVec[axis] == 'N':
88                 nullDataSet.append(featVec)
89     elif LorR == 'R':
90         for featVec in dataSet:
91             if featVec[axis] != 'N':
92                 if float(featVec[axis]) > value:
93                     retDataSet.append(featVec)
94             elif featVec[axis] == 'N':
95                 nullDataSet.append(featVec)
96
97     totalWeightSub = calcTotalWeight(retDataSet, -1, True) # 计算此分支中非空样本的总权重
98     for nullVec in nullDataSet: # 把缺失值样本按权值比例划分到分支中
99         nullVec[-2] = float(nullVec[-2]) * totalWeightSub /
totalWeightV
100         retDataSet.append(nullVec)
101
102     return retDataSet
```

```

103
104
105 def calcTotalWeight(dataSet, labelIndex, isContainNull):
106     """
107     type: (list, int, bool) -> float
108     计算样本集对某个特征值的总样本数（按权重计算）
109     :param dataSet: 数据集
110     :param labelIndex: 特征值索引
111     :param isContainNull: 是否包含空值的样本
112     :return: 返回样本集的总权重值
113     """
114     totalWeight = 0.0
115     for featVec in dataSet: # 遍历每个样本
116         weight = float(featVec[-2])
117         if isContainNull is False and featVec[labelIndex] != 'N':
118             totalWeight += weight # 非空样本数，按权重计算
119         if isContainNull is True:
120             totalWeight += weight # 总样本数，按权重计算
121     return totalWeight
122
123
124 def calcGain(dataSet, labelIndex, labelPropertyi):
125     """
126     type: (list, int, int) -> float, int
127     计算信息增益，返回信息增益值和连续属性的划分点
128     dataSet: 数据集
129     labelIndex: 特征值索引
130     labelPropertyi: 特征值类型，0为离散，1为连续
131     """
132     baseEntropy = calcShannonEnt(dataSet, labelIndex) # 计算根节点的信息熵
133     featList = [example[labelIndex] for example in dataSet] # 特征值列表
134     uniqueVals = set(featList) # 该特征包含的所有值
135     newEntropy = 0.0
136     totalWeight = 0.0
137     totalWeightV = 0.0
138     totalWeight = calcTotalWeight(dataSet, labelIndex, True) # 总样本权重
139     totalWeightV = calcTotalWeight(dataSet, labelIndex, False) # 非空样本权重
140     if labelPropertyi == 0: # 对离散的特征
141         for value in uniqueVals: # 对每个特征值，划分数据集，计算各子

```

```

集的信息熵
142         if value != 'N':
143             subDataSet = splitDataSet(dataSet, labelIndex, value)
144             totalWeightSub = 0.0
145             totalWeightSub = calcTotalWeight(subDataSet,
labelIndex, True)
146             prob = totalWeightSub / totalWeightV
147             newEntropy += prob * calcShannonEnt(subDataSet,
labelIndex)
148         else: # 对连续的特征
149             uniqueValsList = list(uniqueVals)
150             if 'N' in uniqueValsList:
151                 uniqueValsList.remove('N')
152             sortedUniqueVals = sorted(uniqueValsList) # 对特征值排序
153             listPartition = []
154             minEntropy = inf
155             if len(sortedUniqueVals) == 1: # 如果只有一个值, 可以看作只有
左子集, 没有右子集
156                 totalWeightLeft = calcTotalWeight(dataSet, labelIndex,
True)
157                 probLeft = totalWeightLeft / totalWeightV
158                 minEntropy = probLeft * calcShannonEnt(dataSet, labelIndex
)
159             else:
160                 for j in range(len(sortedUniqueVals) - 1): # 计算划分点
161                     partValue = (float(sortedUniqueVals[j]) + float(
sortedUniqueVals[j + 1])) / 2
162                     # 对每个划分点, 计算信息熵
163                     dataSetLeft = splitDataSet(dataSet, labelIndex,
partValue, 'L')
164                     dataSetRight = splitDataSet(dataSet, labelIndex,
partValue, 'R')
165                     totalWeightLeft = 0.0
166                     totalWeightLeft = calcTotalWeight(dataSetLeft,
labelIndex, True)
167                     totalWeightRight = 0.0
168                     totalWeightRight = calcTotalWeight(dataSetRight,
labelIndex, True)
169                     probLeft = totalWeightLeft / totalWeightV
170                     probRight = totalWeightRight / totalWeightV
171                     Entropy = probLeft * calcShannonEnt(dataSetLeft,
labelIndex) + \
172                                     probRight * calcShannonEnt(dataSetRight,

```

```

labelIndex)
174         if Entropy < minEntropy: # 取最小的信息熵
175             minEntropy = Entropy
176             newEntropy = minEntropy
177             gain = totalWeightV / totalWeight * (baseEntropy - newEntropy)
178             return gain
179
180
181 def calcGainRatio(dataSet, labelIndex, labelPropertyi):
182     """
183     type: (list, int, int) -> float, int
184     计算信息增益率, 返回信息增益率和连续属性的划分点
185     dataSet: 数据集
186     labelIndex: 特征值索引
187     labelPropertyi: 特征值类型, 0为离散, 1为连续
188     """
189     baseEntropy = calcShannonEnt(dataSet, labelIndex) # 计算根节点的信息熵
190     featList = [example[labelIndex] for example in dataSet] # 特征值列表
191     uniqueVals = set(featList) # 该特征包含的所有值
192     newEntropy = 0.0
193     bestPartValuei = None
194     IV = 0.0
195     totalWeight = 0.0
196     totalWeightV = 0.0
197     totalWeight = calcTotalWeight(dataSet, labelIndex, True) # 总样本权重
198     totalWeightV = calcTotalWeight(dataSet, labelIndex, False) # 非空样本权重
199     if labelPropertyi == 0: # 对离散的特征
200         for value in uniqueVals: # 对每个特征值, 划分数据集, 计算各子集的信息熵
201             subDataSet = splitDataSet(dataSet, labelIndex, value)
202             totalWeightSub = 0.0
203             totalWeightSub = calcTotalWeight(subDataSet, labelIndex,
True)
204             if value != 'N':
205                 prob = totalWeightSub / totalWeightV
206                 newEntropy += prob * calcShannonEnt(subDataSet,
labelIndex)
207             prob1 = totalWeightSub / totalWeight
208             IV -= prob1 * log(prob1, 2)

```

```

209     else: # 对连续的特征
210         uniqueValsList = list(uniqueVals)
211         if 'N' in uniqueValsList:
212             uniqueValsList.remove('N')
213             # 计算空值样本的总权重, 用于计算IV
214             totalWeightN = 0.0
215             dataSetNull = splitDataSet(dataSet, labelIndex, 'N')
216             totalWeightN = calcTotalWeight(dataSetNull, labelIndex,
True)
217             probNull = totalWeightN / totalWeight
218             if probNull > 0.0:
219                 IV += -1 * probNull * log(probNull, 2)
220
221             sortedUniqueVals = sorted(uniqueValsList) # 对特征值排序
222             listPartition = []
223             minEntropy = inf
224
225             if len(sortedUniqueVals) == 1: # 如果只有一个值, 可以看作只有
左子集, 没有右子集
226                 totalWeightLeft = calcTotalWeight(dataSet, labelIndex,
True)
227                 probLeft = totalWeightLeft / totalWeightV
228                 minEntropy = probLeft * calcShannonEnt(dataSet, labelIndex
)
229                 IV = -1 * probLeft * log(probLeft, 2)
230             else:
231                 for j in range(len(sortedUniqueVals) - 1): # 计算划分点
232                     partValue = (float(sortedUniqueVals[j]) + float(
sortedUniqueVals[j + 1])) / 2
233                     # 对每个划分点, 计算信息熵
234                     dataSetLeft = splitDataSet(dataSet, labelIndex,
partValue, 'L')
235                     dataSetRight = splitDataSet(dataSet, labelIndex,
partValue, 'R')
236                     totalWeightLeft = 0.0
237                     totalWeightLeft = calcTotalWeight(dataSetLeft,
labelIndex, True)
238                     totalWeightRight = 0.0
239                     totalWeightRight = calcTotalWeight(dataSetRight,
labelIndex, True)
240                     probLeft = totalWeightLeft / totalWeightV
241                     probRight = totalWeightRight / totalWeightV
242                     Entropy = probLeft * calcShannonEnt(

```

```

244         dataSetLeft, labelIndex) + probRight *
    calcShannonEnt(dataSetRight, labelIndex)
245         if Entropy < minEntropy: # 取最小的信息熵
246             minEntropy = Entropy
247             bestPartValuei = partValue
248             probLeft1 = totalWeightLeft / totalWeight
249             probRight1 = totalWeightRight / totalWeight
250             IV += -1 * (probLeft1 * log(probLeft1, 2) +
    probRight1 * log(probRight1, 2))
251
252     newEntropy = minEntropy
253     gain = totalWeightV / totalWeight * (baseEntropy - newEntropy)
254     if IV == 0.0: # 如果属性只有一个值, IV为0, 为避免除数为0, 给个很
    小的值
255         IV = 0.0000000001
256     gainRatio = gain / IV
257     return gainRatio, bestPartValuei
258
259
260 # 选择最好的数据集划分方式
261 def chooseBestFeatureToSplit(dataSet, labelProperty):
262     """
263     type: (list, int) -> int, float
264     :param dataSet: 样本集
265     :param labelProperty: 特征值类型, 1 连续, 0 离散
266     :return: 最佳划分属性的索引和连续属性的划分值
267     """
268     numFeatures = len(labelProperty) # 特征数
269     bestInfoGainRatio = 0.0
270     bestFeature = -1
271     bestPartValue = None # 连续的特征值, 最佳划分值
272     gainSum = 0.0
273     gainAvg = 0.0
274     for i in range(numFeatures): # 对每个特征循环
275         infoGain = calcGain(dataSet, i, labelProperty[i])
276         gainSum += infoGain
277     gainAvg = gainSum / numFeatures
278     for i in range(numFeatures): # 对每个特征循环
279         infoGainRatio, bestPartValuei = calcGainRatio(dataSet, i,
    labelProperty[i])
280         infoGain = calcGain(dataSet, i, labelProperty[i])
281         if infoGainRatio > bestInfoGainRatio and infoGain > gainAvg:
    # 取信息增益高于平均增益且信息增益率最大的特征

```

```

282         bestInfoGainRatio = infoGainRatio
283         bestFeature = i
284         bestPartValue = bestPartValue[i]
285     return bestFeature, bestPartValue
286
287
288 # 通过排序返回出现次数最多的类别
289 def majorityCnt(classList, weightList):
290     classCount = {}
291     for i in range(len(classList)):
292         if classList[i] not in classCount.keys():
293             classCount[classList[i]] = 0.0
294         classCount[classList[i]] += round(float(weightList[i]), 1)
295
296     # python 2.7
297     # sortedClassCount = sorted(classCount.iteritems(),
298     #                             key=operator.itemgetter(1), reverse=True
299     # )
300     sortedClassCount = sorted(classCount.items(),
301                               key=operator.itemgetter(1), reverse=True
302     )
303     if len(sortedClassCount) == 1:
304         return (sortedClassCount[0][0], sortedClassCount[0][1], 0.0)
305     return (sortedClassCount[0][0], sortedClassCount[0][1],
306             sortedClassCount[1][1])
307
308
309 # 创建树，样本集 特征 特征属性 (0 离散, 1 连续)
310 def createTree(dataSet, labels, labelProperty):
311     classList = [example[-1] for example in dataSet] # 类别向量
312     weightList = [example[-2] for example in dataSet] # 权重向量
313     if classList.count(classList[0]) == len(classList): # 如果只有一个类别，返回
314         totalWeight = calcTotalWeight(dataSet, 0, True)
315         return (classList[0], round(totalWeight, 1), 0.0)
316     # totalWeight = calcTotalWeight(dataSet, 0, True)
317     if len(dataSet[0]) == 1: # 如果所有特征都被遍历完了，返回出现次数最多的类别
318         return majorityCnt(classList)
319     bestFeat, bestPartValue = chooseBestFeatureToSplit(dataSet,
320                                                         labelProperty)
321
322     # 最优分类特征的索引
323     if bestFeat == -1: # 如果无法选出最优分类特征，返回出现次数最多的

```


类别

```

319         return majorityCnt(classList, weightList)
320     if labelProperty[bestFeat] == 0: # 对离散的特征
321         bestFeatLabel = labels[bestFeat]
322         myTree = {bestFeatLabel: {}}
323         labelsNew = copy.copy(labels)
324         labelPropertyNew = copy.copy(labelProperty)
325         del (labelsNew[bestFeat]) # 已经选择的特征不再参与分类
326         del (labelPropertyNew[bestFeat])
327         featValues = [example[bestFeat] for example in dataSet]
328         uniqueValue = set(featValues) # 该特征包含的所有值
329         uniqueValue.discard('N')
330         for value in uniqueValue: # 对每个特征值，递归构建树
331             subLabels = labelsNew[:]
332             subLabelProperty = labelPropertyNew[:]
333             myTree[bestFeatLabel][value] = createTree(
334                 splitDataSetWithNull(dataSet, bestFeat, value),
subLabels,
335                 subLabelProperty)
336     else: # 对连续的特征，不删除该特征，分别构建左子树和右子树
337         bestFeatLabel = labels[bestFeat] + '<' + str(bestPartValue)
338         myTree = {bestFeatLabel: {}}
339         subLabels = labels[:]
340         subLabelProperty = labelProperty[:]
341         # 构建左子树
342         valueLeft = 'Y'
343         myTree[bestFeatLabel][valueLeft] = createTree(
344             splitDataSetWithNull(dataSet, bestFeat, bestPartValue, 'L'
), subLabels,
345             subLabelProperty)
346         # 构建右子树
347         valueRight = 'N'
348         myTree[bestFeatLabel][valueRight] = createTree(
349             splitDataSetWithNull(dataSet, bestFeat, bestPartValue, 'R'
), subLabels,
350             subLabelProperty)
351     return myTree
352
353 # 测试算法
354 def classify(inputTree, classList, featLabels, featLabelProperties,
testVec):
355     firstStr = list(inputTree.keys())[0] # 根节点

```

```

357     firstLabel = firstStr
358     lessIndex = str(firstStr).find('<')
359     if lessIndex > -1: # 如果是连续型的特征
360         firstLabel = str(firstStr)[:lessIndex]
361     secondDict = inputTree[firstStr]
362     featIndex = featLabels.index(firstLabel) # 跟节点对应的特征
363     classLabel = {}
364     for classI in classList:
365         classLabel[classI] = 0.0
366     for key in secondDict.keys(): # 对每个分支循环
367         if featLabelProperties[featIndex] == 0: # 离散的特征
368             if testVec[featIndex] == key: # 测试样本进入某个分支
369                 if type(secondDict[key]).__name__ == 'dict': # 该分支
不是叶子节点，递归
370                     classLabelSub = classify(secondDict[key],
classList, featLabels,
371                                             featLabelProperties, testVec
)
372                     for classKey in classLabel.keys():
373                         classLabel[classKey] += classLabelSub[classKey]
]
374         else: # 如果是叶子，返回结果
375             for classKey in classLabel.keys():
376                 if classKey == secondDict[key][0]:
377                     classLabel[classKey] += secondDict[key][1]
378                 else:
379                     classLabel[classKey] += secondDict[key][2]
380         elif testVec[featIndex] == 'N': # 如果测试样本的属性值缺
失，则进入每个分支
381             if type(secondDict[key]).__name__ == 'dict': # 该分支
不是叶子节点，递归
382                 classLabelSub = classify(secondDict[key],
classList, featLabels,
383                                             featLabelProperties, testVec
)
384             for classKey in classLabel.keys():
385                 classLabel[classKey] += classLabelSub[key]
386         else: # 如果是叶子，返回结果
387             for classKey in classLabel.keys():
388                 if classKey == secondDict[key][0]:
389                     classLabel[classKey] += secondDict[key][1]
390                 else:
391                     classLabel[classKey] += secondDict[key][2]

```

```

392         else:
393             partValue = float(str(firstStr)[lessIndex + 1:])
394             if testVec[featIndex] == 'N': # 如果测试样本的属性值缺
失，则对每个分支的结果加和
395                 # 进入左子树
396                 if type(secondDict[key]).__name__ == 'dict': # 该分支
不是叶子节点，递归
397                     classLabelSub = classify(secondDict[key],
classList, featLabels,
398                                             featLabelProperties, testVec
)
399                 for classKey in classLabel.keys():
400                     classLabel[classKey] += classLabelSub[classKey]
]
401             else: # 如果是叶子， 返回结果
402                 for classKey in classLabel.keys():
403                     if classKey == secondDict[key][0]:
404                         classLabel[classKey] += secondDict[key][1]
405                     else:
406                         classLabel[classKey] += secondDict[key][2]
407             elif float(testVec[featIndex]) <= partValue and key == 'Y'
: # 进入左子树
408                 if type(secondDict['Y']).__name__ == 'dict': # 该分支
不是叶子节点，递归
409                     classLabelSub = classify(secondDict['Y'],
classList, featLabels,
410                                             featLabelProperties,
testVec)
411                 for classKey in classLabel.keys():
412                     classLabel[classKey] += classLabelSub[classKey]
]
413             else: # 如果是叶子， 返回结果
414                 for classKey in classLabel.keys():
415                     if classKey == secondDict[key][0]:
416                         classLabel[classKey] += secondDict['Y'][1]
417                     else:
418                         classLabel[classKey] += secondDict['Y'][2]
419             elif float(testVec[featIndex]) > partValue and key == 'N':
420                 if type(secondDict['N']).__name__ == 'dict': # 该分支
不是叶子节点，递归
421                     classLabelSub = classify(secondDict['N'],
classList, featLabels,
422                                             featLabelProperties,

```

```
testVec)
423         for classKey in classLabel.keys():
424             classLabel[classKey] += classLabelSub[classKey]
425     else: # 如果是叶子, 返回结果
426         for classKey in classLabel.keys():
427             if classKey == secondDict[key][0]:
428                 classLabel[classKey] += secondDict['N'][1]
429             else:
430                 classLabel[classKey] += secondDict['N'][2]
431
432     return classLabel
433
434
435 # 存储决策树
436 def storeTree(inputTree, filename):
437     import pickle
438     fw = open(filename, 'w')
439     pickle.dump(inputTree, fw)
440     fw.close()
441
442
443 # 读取决策树, 文件不存在返回None
444 def grabTree(filename):
445     import pickle
446     if os.path.isfile(filename):
447         fr = open(filename)
448         return pickle.load(fr)
449     else:
450         return None
451
452
453 # 测试决策树正确率
454 def testing(myTree, classList, data_test, labels, labelProperties):
455     error = 0.0
456     for i in range(len(data_test)):
457         classLabelSet = classify(myTree, classList, labels,
458                                 labelProperties, data_test[i])
459         maxWeight = 0.0
460         classLabel = ''
461         for item in classLabelSet.items():
462             if item[1] > maxWeight:
```

```

463         if classLabel != data_test[i][-1]:
464             error += 1
465     return float(error)
466
467
468 # 测试投票节点正确率
469 def testingMajor(major, data_test):
470     error = 0.0
471     for i in range(len(data_test)):
472         if major[0] != data_test[i][-1]:
473             error += 1
474     # print 'major %d' %error
475     return float(error)
476
477
478 # 后剪枝
479 def postPruningTree(inputTree, classSet, dataSet, data_test, labels,
480                     labelProperties):
481     firstStr = list(inputTree.keys())[0]
482     secondDict = inputTree[firstStr]
483     classList = [example[-1] for example in dataSet]
484     weightList = [example[-2] for example in dataSet]
485     featkey = copy.deepcopy(firstStr)
486     if '<' in firstStr: # 对连续的特征值, 使用正则表达式获得特征标签
487         # 和 value
488         featkey = re.compile("(.+<)").search(firstStr).group()[:-1]
489         featvalue = float(re.compile("<(.+)").search(firstStr).group()
490                           [1:])
491     labelIndex = labels.index(featkey)
492     temp_labels = copy.deepcopy(labels)
493     temp_labelProperties = copy.deepcopy(labelProperties)
494     if labelProperties[labelIndex] == 0: # 离散特征
495         del (labels[labelIndex])
496         del (labelProperties[labelIndex])
497     for key in secondDict.keys(): # 对每个分支
498         if type(secondDict[key]).__name__ == 'dict': # 如果不是叶子节点
499             if temp_labelProperties[labelIndex] == 0: # 离散的
500                 subDataSet = splitDataSet(dataSet, labelIndex, key)
501                 subDataTest = splitDataSet(data_test, labelIndex, key)
502             else:
503                 if key == 'Y':
504                     subDataSet = splitDataSet(dataSet, labelIndex,

```

```
    featvalue ,
502                                     'L')
503         subDataSet = splitDataSet(data_test , labelIndex ,
504                                   featvalue , 'L')
505     else :
506         subDataSet = splitDataSet(dataSet , labelIndex ,
    featvalue ,
507                                     'R')
508         subDataSet = splitDataSet(data_test , labelIndex ,
509                                   featvalue , 'R')
510         if len(subDataSet) > 0:
511             inputTree[firstStr][key] = postPruningTree(secondDict[
    key] , classSet ,
512                                                         subDataSet ,
    subDataSet ,
513                                                         copy.deepcopy(
    labels) ,
514                                                         copy.deepcopy(
515                                                         labelProperties))
516         if testing(inputTree , classSet , data_test , temp_labels ,
517                   temp_labelProperties) <= testingMajor(majorityCnt(
    classList , weightList) ,
518                                                         data_test) :
519             return inputTree
520         return majorityCnt(classList , weightList)
```