IML 第二次作业

习题 3.2

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

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

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

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

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

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

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

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

编写 C 代码程序（程序代码附后），搜索得出解为

$$C_1 = 000000000 \quad C_2 = 101010100 \quad C_3 = 110011000 \quad C_4 = 111100000$$

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 $\boldsymbol{S}_b = \sum_{i=1}^{N} m_i(\boldsymbol{\mu}_i - \boldsymbol{\mu})(\boldsymbol{\mu}_i - \boldsymbol{\mu})^\top$。

$$\boldsymbol{S}_b = [(\boldsymbol{\mu}_1 - \boldsymbol{\mu}), (\boldsymbol{\mu}_2 - \boldsymbol{\mu}), \ldots, (\boldsymbol{\mu}_N - \boldsymbol{\mu})] \begin{pmatrix} \boldsymbol{m}_1 & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \ldots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{m}_N \end{pmatrix} \begin{pmatrix} (\boldsymbol{\mu}_1 - \boldsymbol{\mu})^\top \\ \ldots \\ (\boldsymbol{\mu}_N - \boldsymbol{\mu})^\top \end{pmatrix}$$

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

$$\begin{aligned} \operatorname{rank} \boldsymbol{S}_b &= \operatorname{rank} \boldsymbol{A}^\top \boldsymbol{M} \boldsymbol{A} \\ &= \operatorname{rank} \boldsymbol{A}^\top \boldsymbol{M}^{\frac{1}{2}} \boldsymbol{M}^{\frac{1}{2}} \boldsymbol{A} \\ &= \operatorname{rank} \left(\boldsymbol{A}^\top \boldsymbol{M}^{\frac{1}{2}}\right) \left(\boldsymbol{A}^\top \boldsymbol{M}^{\frac{1}{2}}\right)^\top \\ &= \operatorname{rank} \left(\boldsymbol{A}^\top \boldsymbol{M}^{\frac{1}{2}}\right) \\ &= \operatorname{rank} \boldsymbol{A}^\top \end{aligned}$$

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 $\operatorname{rank} \boldsymbol{A}^\top \le N-1$。

作业 3.4

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

由拉格朗日乘子法, 定义拉格朗日函数

$$L(\boldsymbol{W}, \lambda) = -\operatorname{tr}\left(\boldsymbol{W}^{\mathrm{T}} \boldsymbol{S}_b \boldsymbol{W}\right) + \lambda \left(\operatorname{tr}\left(\boldsymbol{W}^{\mathrm{T}} \boldsymbol{S}_w \boldsymbol{W}\right) - 1\right).$$

对上式关于 $\boldsymbol{W}$ 求偏导得

$$\begin{aligned}\frac{\partial L(\boldsymbol{W},\lambda)}{\partial \boldsymbol{W}} &= -\frac{\partial\left(\operatorname{tr}\left(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{S}_b\boldsymbol{W}\right)\right)}{\partial \boldsymbol{W}} + \lambda\frac{\partial\left(\operatorname{tr}\left(\boldsymbol{W}^{\mathrm{T}}\boldsymbol{S}_w\boldsymbol{W}\right)-1\right)}{\partial \boldsymbol{W}}\\ &= -\left(\boldsymbol{S}_b+\boldsymbol{S}_b^{\mathrm{T}}\right)\boldsymbol{W} + \lambda\left(\boldsymbol{S}_w+\boldsymbol{S}_w^{\mathrm{T}}\right)\boldsymbol{W}\\ &= -2\boldsymbol{S}_b\boldsymbol{W} + 2\lambda\boldsymbol{S}_w\boldsymbol{W}\end{aligned}$$

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

作业 3.5

对称性：

$$\begin{aligned}(\boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top)^\top &= (\boldsymbol{X}^\top)^\top((\boldsymbol{X}^\top\boldsymbol{X})^{-1})^\top\boldsymbol{X}^\top\\ &= (\boldsymbol{X}^\top)^\top((\boldsymbol{X}^\top\boldsymbol{X})^\top)^{-1}\boldsymbol{X}^\top\\ &= \boldsymbol{X}(\boldsymbol{X}^\top\boldsymbol{X})^{-1}\boldsymbol{X}^\top\end{aligned}$$

幂等性：

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

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

将特征矩阵 $\boldsymbol{X}$ 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 $\boldsymbol{X}$ 张成的空间是 d 维度空间。真实值 $\boldsymbol{y}$ 是一个 n 维空间中的 $n\times 1$ 向量。线性回归就是在 $\boldsymbol{X}$ 张成的 d 维空间中，寻找 n 维空间中 $\boldsymbol{y}$ 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

$$\max H(\boldsymbol{p})$$
$$\text{s.t. } \sum_k p_k = 1$$

由拉格朗日乘数法，其拉格朗日函数为

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

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

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 $Ent(D) = 1$，则 A 的信息增益为

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

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	

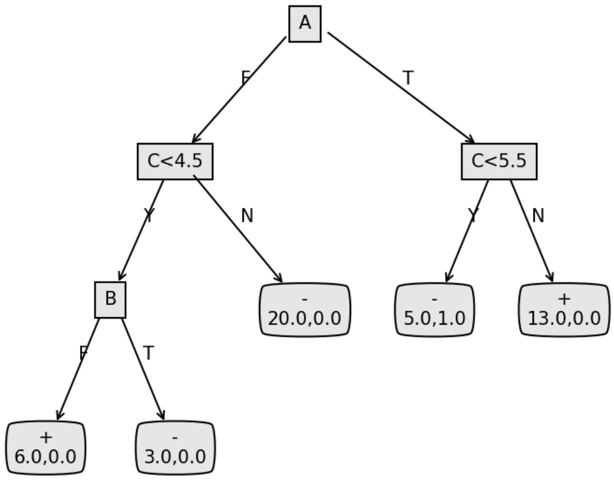
（d）按书上公式计算如下：

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

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

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

```cpp
#include <iostream>
#include <vector>
#include <string>

int dist(int a, int b) {
  int dist_return = 0;
  for (int i = 0; i <= 8; i++) {
    dist_return += ((a >> i) & 1) ^ ((b >> i) & 1);
  }
  return dist_return;
}

int L(int a, int b, int c, int d) {
  int d1 = dist(a, b);
  int d2 = dist(a, c);
  int d3 = dist(a, d);
  int d4 = dist(b, c);
  int d5 = dist(b, d);
  int d6 = dist(c, d);
  int d1_ = dist(a, ~b);
  int d2_ = dist(a, ~c);
  int d3_ = dist(a, ~d);
  int d4_ = dist(b, ~c);
  int d5_ = dist(b, ~d);
  int d6_ = dist(c, ~d);
  return d1 * d2 * d3 * d4 * d5 * d6 * d1_ * d2_ * d3_ * d4_ * d5_ * d6_ +
         d1 * d1_ + d2 * d2_ + d3 * d3_ + d4 * d4_ + d5 * d5_ + d6 * d6_;
}

int main() {
  int hi = 0;
  int hj = 0;
  int hk = 0;
  int hg = 0;
  int hs = 0;

  const int min = 0b000000000;
  const int max = 0b111111111;
  for (int i = min; i <= max - 3; i++) {
    for (int j = i + 1; j <= max - 2; j++) {
```

```
        for (int k = j + 1; k <= max - 1; k++) {
          for (int g = k + 1; g <= max; g++) {
            int ns = L(i, j, k, g);
            if (ns > hs) {
              hs = ns;
              hi = i;
              hj = j;
              hk = k;
              hg = g;
            }
          }
        }
      }
      printf("process: i= %4x\n", i);
    }

    printf("%4x, %4x, %4x, %4x, socre: %d\n", hi, hj, hk, hg, hs);
}
```

作业 4.4 的代码

仅提供 C4.5 算法部分。

```
from math import log
import operator
import os

import re
from numpy import inf
import copy


# 计算信息熵
def calcShannonEnt(dataSet, labelIndex):
    # type: (list) -> float
    numEntries = 0  # 样本数(按权重计算)
    labelCounts = {}
    for featVec in dataSet:  # 遍历每个样本
        if featVec[labelIndex] != 'N':
            weight = float(featVec[-2])
            numEntries += weight
            currentLabel = featVec[-1]  # 当前样本的类别
            if currentLabel not in labelCounts.keys():  # 生成类别字典
```

```
                labelCounts[currentLabel] = 0
            labelCounts[currentLabel] += weight  # 数据集的倒数第二个值用来标记样本权重
    shannonEnt = 0.0
    for key in labelCounts:  # 计算信息熵
        prob = float(labelCounts[key]) / numEntries
        shannonEnt = shannonEnt - prob * log(prob, 2)
    return shannonEnt


def splitDataSet(dataSet, axis, value, LorR='N'):
    """
    type: (list, int, string or float, string) -> list
    划分数据集
    axis:按第几个特征划分
    value:划分特征的值
    LorR: N 离散属性; L 小于等于value值; R 大于value值
    """
    retDataSet = []
    featVec = []
    if LorR == 'N':  # 离散属性
        for featVec in dataSet:
            if featVec[axis] == value:
                reducedFeatVec = featVec[:axis]
                reducedFeatVec.extend(featVec[axis + 1:])
                retDataSet.append(reducedFeatVec)
    elif LorR == 'L':
        for featVec in dataSet:
            if featVec[axis] != 'N':
                if float(featVec[axis]) < value:
                    retDataSet.append(featVec)
    elif LorR == 'R':
        for featVec in dataSet:
            if featVec[axis] != 'N':
                if float(featVec[axis]) > value:
                    retDataSet.append(featVec)
    return retDataSet


def splitDataSetWithNull(dataSet, axis, value, LorR='N'):
    """
    type: (list, int, string or float, string) -> list
    划分数据集
```

```python
    axis:按第几个特征划分
    value:划分特征的值
    LorR: N 离散属性; L 小于等于value值; R 大于value值
    """
    retDataSet = []
    nullDataSet = []
    featVec = []
    totalWeightV = calcTotalWeight(dataSet, axis, False)  # 非空样本权重
    totalWeightSub = 0.0
    if LorR == 'N':  # 离散属性
        for featVec in dataSet:
            if featVec[axis] == value:
                reducedFeatVec = featVec[:axis]
                reducedFeatVec.extend(featVec[axis + 1:])
                retDataSet.append(reducedFeatVec)
            elif featVec[axis] == 'N':
                reducedNullVec = featVec[:axis]
                reducedNullVec.extend(featVec[axis + 1:])
                nullDataSet.append(reducedNullVec)
    elif LorR == 'L':
        for featVec in dataSet:
            if featVec[axis] != 'N':
                if float(featVec[axis]) < value:
                    retDataSet.append(featVec)
            elif featVec[axis] == 'N':
                nullDataSet.append(featVec)
    elif LorR == 'R':
        for featVec in dataSet:
            if featVec[axis] != 'N':
                if float(featVec[axis]) > value:
                    retDataSet.append(featVec)
            elif featVec[axis] == 'N':
                nullDataSet.append(featVec)

    totalWeightSub = calcTotalWeight(retDataSet, -1, True)  # 计算此分支中非空样本的总权重
    for nullVec in nullDataSet:  # 把缺失值样本按权值比例划分到分支中
        nullVec[-2] = float(nullVec[-2]) * totalWeightSub / totalWeightV
        retDataSet.append(nullVec)

    return retDataSet
```

```python


def calcTotalWeight(dataSet, labelIndex, isContainNull):
    """
    type: (list, int, bool) -> float
    计算样本集对某个特征值的总样本树（按权重计算）
    :param dataSet: 数据集
    :param labelIndex: 特征值索引
    :param isContainNull: 是否包含空值的样本
    :return: 返回样本集的总权重值
    """
    totalWeight = 0.0
    for featVec in dataSet:  # 遍历每个样本
        weight = float(featVec[-2])
        if isContainNull is False and featVec[labelIndex] != 'N':
            totalWeight += weight  # 非空样本树，按权重计算
        if isContainNull is True:
            totalWeight += weight  # 总样本数，按权重计算
    return totalWeight


def calcGain(dataSet, labelIndex, labelPropertyi):
    """
    type: (list, int, int) -> float, int
    计算信息增益,返回信息增益值和连续属性的划分点
    dataSet: 数据集
    labelIndex: 特征值索引
    labelPropertyi: 特征值类型，0为离散，1为连续
    """
    baseEntropy = calcShannonEnt(dataSet, labelIndex)  # 计算根节点的信息熵
    featList = [example[labelIndex] for example in dataSet]  # 特征值列表
    uniqueVals = set(featList)  # 该特征包含的所有值
    newEntropy = 0.0
    totalWeight = 0.0
    totalWeightV = 0.0
    totalWeight = calcTotalWeight(dataSet, labelIndex, True)  # 总样本权重
    totalWeightV = calcTotalWeight(dataSet, labelIndex, False)  # 非空样本权重
    if labelPropertyi == 0:  # 对离散的特征
        for value in uniqueVals:  # 对每个特征值，划分数据集，计算各子
```

```
集的信息熵
        if value != 'N':
            subDataSet = splitDataSet(dataSet, labelIndex, value)
            totalWeightSub = 0.0
            totalWeightSub = calcTotalWeight(subDataSet, labelIndex, True)
            prob = totalWeightSub / totalWeightV
            newEntropy += prob * calcShannonEnt(subDataSet, labelIndex)
else:  # 对连续的特征
    uniqueValsList = list(uniqueVals)
    if 'N' in uniqueValsList:
        uniqueValsList.remove('N')
    sortedUniqueVals = sorted(uniqueValsList)  # 对特征值排序
    listPartition = []
    minEntropy = inf
    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
        totalWeightLeft = calcTotalWeight(dataSet, labelIndex, True)
        probLeft = totalWeightLeft / totalWeightV
        minEntropy = probLeft * calcShannonEnt(dataSet, labelIndex)
    else:
        for j in range(len(sortedUniqueVals) - 1):  # 计算划分点
            partValue = (float(sortedUniqueVals[j]) + float(
                sortedUniqueVals[j + 1])) / 2
            # 对每个划分点，计算信息熵
            dataSetLeft = splitDataSet(dataSet, labelIndex, partValue, 'L')
            dataSetRight = splitDataSet(dataSet, labelIndex, partValue, 'R')
            totalWeightLeft = 0.0
            totalWeightLeft = calcTotalWeight(dataSetLeft, labelIndex, True)
            totalWeightRight = 0.0
            totalWeightRight = calcTotalWeight(dataSetRight, labelIndex, True)
            probLeft = totalWeightLeft / totalWeightV
            probRight = totalWeightRight / totalWeightV
            Entropy = probLeft * calcShannonEnt(dataSetLeft, labelIndex) + \
                      probRight * calcShannonEnt(dataSetRight,
```

```
labelIndex)
                if Entropy < minEntropy:  # 取最小的信息熵
                    minEntropy = Entropy
        newEntropy = minEntropy
    gain = totalWeightV / totalWeight * (baseEntropy - newEntropy)
    return gain


def calcGainRatio(dataSet, labelIndex, labelPropertyi):
    """
    type: (list, int, int) -> float, int
    计算信息增益率,返回信息增益率和连续属性的划分点
    dataSet: 数据集
    labelIndex: 特征值索引
    labelPropertyi: 特征值类型, 0为离散, 1为连续
    """
    baseEntropy = calcShannonEnt(dataSet, labelIndex)  # 计算根节点的信息熵
    featList = [example[labelIndex] for example in dataSet]  # 特征值列表
    uniqueVals = set(featList)  # 该特征包含的所有值
    newEntropy = 0.0
    bestPartValuei = None
    IV = 0.0
    totalWeight = 0.0
    totalWeightV = 0.0
    totalWeight = calcTotalWeight(dataSet, labelIndex, True)  # 总样本权重
    totalWeightV = calcTotalWeight(dataSet, labelIndex, False)  # 非空样本权重
    if labelPropertyi == 0:  # 对离散的特征
        for value in uniqueVals:  # 对每个特征值, 划分数据集, 计算各子集的信息熵
            subDataSet = splitDataSet(dataSet, labelIndex, value)
            totalWeightSub = 0.0
            totalWeightSub = calcTotalWeight(subDataSet, labelIndex, True)
            if value != 'N':
                prob = totalWeightSub / totalWeightV
                newEntropy += prob * calcShannonEnt(subDataSet, labelIndex)
            prob1 = totalWeightSub / totalWeight
            IV -= prob1 * log(prob1, 2)
```

```python
    else:  # 对连续的特征
        uniqueValsList = list(uniqueVals)
        if 'N' in uniqueValsList:
            uniqueValsList.remove('N')
            # 计算空值样本的总权重，用于计算IV
            totalWeightN = 0.0
            dataSetNull = splitDataSet(dataSet, labelIndex, 'N')
            totalWeightN = calcTotalWeight(dataSetNull, labelIndex, True)
            probNull = totalWeightN / totalWeight
            if probNull > 0.0:
                IV += -1 * probNull * log(probNull, 2)

        sortedUniqueVals = sorted(uniqueValsList)  # 对特征值排序
        listPartition = []
        minEntropy = inf

        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
            totalWeightLeft = calcTotalWeight(dataSet, labelIndex, True)
            probLeft = totalWeightLeft / totalWeightV
            minEntropy = probLeft * calcShannonEnt(dataSet, labelIndex)
            IV = -1 * probLeft * log(probLeft, 2)
        else:
            for j in range(len(sortedUniqueVals) - 1):  # 计算划分点
                partValue = (float(sortedUniqueVals[j]) + float(
                    sortedUniqueVals[j + 1])) / 2
                # 对每个划分点，计算信息熵
                dataSetLeft = splitDataSet(dataSet, labelIndex, partValue, 'L')
                dataSetRight = splitDataSet(dataSet, labelIndex, partValue, 'R')
                totalWeightLeft = 0.0
                totalWeightLeft = calcTotalWeight(dataSetLeft, labelIndex, True)
                totalWeightRight = 0.0
                totalWeightRight = calcTotalWeight(dataSetRight, labelIndex, True)
                probLeft = totalWeightLeft / totalWeightV
                probRight = totalWeightRight / totalWeightV
                Entropy = probLeft * calcShannonEnt(
```

```
                    dataSetLeft, labelIndex) + probRight * calcShannonEnt(dataSetRight, labelIndex)
                if Entropy < minEntropy:  # 取最小的信息熵
                    minEntropy = Entropy
                    bestPartValuei = partValue
                    probLeft1 = totalWeightLeft / totalWeight
                    probRight1 = totalWeightRight / totalWeight
                    IV += -1 * (probLeft1 * log(probLeft1, 2) + probRight1 * log(probRight1, 2))

        newEntropy = minEntropy
    gain = totalWeightV / totalWeight * (baseEntropy - newEntropy)
    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
        IV = 0.0000000001
    gainRatio = gain / IV
    return gainRatio, bestPartValuei


# 选择最好的数据集划分方式
def chooseBestFeatureToSplit(dataSet, labelProperty):
    """
    type: (list, int) -> int, float
    :param dataSet: 样本集
    :param labelProperty: 特征值类型，1 连续， 0 离散
    :return: 最佳划分属性的索引和连续属性的划分值
    """
    numFeatures = len(labelProperty)  # 特征数
    bestInfoGainRatio = 0.0
    bestFeature = -1
    bestPartValue = None  # 连续的特征值，最佳划分值
    gainSum = 0.0
    gainAvg = 0.0
    for i in range(numFeatures):  # 对每个特征循环
        infoGain = calcGain(dataSet, i, labelProperty[i])
        gainSum += infoGain
    gainAvg = gainSum / numFeatures
    for i in range(numFeatures):  # 对每个特征循环
        infoGainRatio, bestPartValuei = calcGainRatio(dataSet, i, labelProperty[i])
        infoGain = calcGain(dataSet, i, labelProperty[i])
        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
```

```
                bestInfoGainRatio = infoGainRatio
                bestFeature = i
                bestPartValue = bestPartValuei
    return bestFeature, bestPartValue


# 通过排序返回出现次数最多的类别
def majorityCnt(classList, weightList):
    classCount = {}
    for i in range(len(classList)):
        if classList[i] not in classCount.keys():
            classCount[classList[i]] = 0.0
        classCount[classList[i]] += round(float(weightList[i]),1)

    # python 2.7
    # sortedClassCount = sorted(classCount.iteritems(),
    #                           key=operator.itemgetter(1), reverse=True)
    sortedClassCount = sorted(classCount.items(),
                              key=operator.itemgetter(1), reverse=True)
    if len(sortedClassCount) == 1:
        return (sortedClassCount[0][0],sortedClassCount[0][1],0.0)
    return (sortedClassCount[0][0], sortedClassCount[0][1], sortedClassCount[1][1])


# 创建树, 样本集 特征 特征属性(0 离散, 1 连续)
def createTree(dataSet, labels, labelProperty):
    classList = [example[-1] for example in dataSet]  # 类别向量
    weightList = [example[-2] for example in dataSet]  # 权重向量
    if classList.count(classList[0]) == len(classList):  # 如果只有一个类别, 返回
        totalWeiht = calcTotalWeight(dataSet,0,True)
        return (classList[0], round(totalWeiht,1),0.0)
    #totalWeight = calcTotalWeight(dataSet, 0, True)
    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
        return majorityCnt(classList)
    bestFeat, bestPartValue = chooseBestFeatureToSplit(dataSet,
                                                       labelProperty)  # 最优分类特征的索引
    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
        return majorityCnt(classList, weightList)
    if labelProperty[bestFeat] == 0:  # 对离散的特征
        bestFeatLabel = labels[bestFeat]
        myTree = {bestFeatLabel: {}}
        labelsNew = copy.copy(labels)
        labelPropertyNew = copy.copy(labelProperty)
        del (labelsNew[bestFeat])  # 已经选择的特征不再参与分类
        del (labelPropertyNew[bestFeat])
        featValues = [example[bestFeat] for example in dataSet]
        uniqueValue = set(featValues)  # 该特征包含的所有值
        uniqueValue.discard('N')
        for value in uniqueValue:  # 对每个特征值，递归构建树
            subLabels = labelsNew[:]
            subLabelProperty = labelPropertyNew[:]
            myTree[bestFeatLabel][value] = createTree(
                splitDataSetWithNull(dataSet, bestFeat, value), subLabels,
                subLabelProperty)
    else:  # 对连续的特征，不删除该特征，分别构建左子树和右子树
        bestFeatLabel = labels[bestFeat] + '<' + str(bestPartValue)
        myTree = {bestFeatLabel: {}}
        subLabels = labels[:]
        subLabelProperty = labelProperty[:]
        # 构建左子树
        valueLeft = 'Y'
        myTree[bestFeatLabel][valueLeft] = createTree(
            splitDataSetWithNull(dataSet, bestFeat, bestPartValue, 'L'), subLabels,
            subLabelProperty)
        # 构建右子树
        valueRight = 'N'
        myTree[bestFeatLabel][valueRight] = createTree(
            splitDataSetWithNull(dataSet, bestFeat, bestPartValue, 'R'), subLabels,
            subLabelProperty)
    return myTree


# 测试算法
def classify(inputTree, classList, featLabels, featLabelProperties, testVec):
    firstStr = list(inputTree.keys())[0]  # 根节点
```

```
    firstLabel = firstStr
    lessIndex = str(firstStr).find('<')
    if lessIndex > -1:  # 如果是连续型的特征
        firstLabel = str(firstStr)[:lessIndex]
    secondDict = inputTree[firstStr]
    featIndex = featLabels.index(firstLabel)  # 跟节点对应的特征
    classLabel = {}
    for classI in classList:
        classLabel[classI] = 0.0
    for key in secondDict.keys():  # 对每个分支循环
        if featLabelProperties[featIndex] == 0:  # 离散的特征
            if testVec[featIndex] == key:  # 测试样本进入某个分支
                if type(secondDict[key]).__name__ == 'dict':  # 该分支不是叶子节点，递归
                    classLabelSub = classify(secondDict[key], classList, featLabels,
                                             featLabelProperties, testVec)
                    for classKey in classLabel.keys():
                        classLabel[classKey] += classLabelSub[classKey]
                else:  # 如果是叶子， 返回结果
                    for classKey in classLabel.keys():
                        if classKey == secondDict[key][0]:
                            classLabel[classKey] += secondDict[key][1]
                        else:
                            classLabel[classKey] += secondDict[key][2]
            elif testVec[featIndex] == 'N':  # 如果测试样本的属性值缺失，则进入每个分支
                if type(secondDict[key]).__name__ == 'dict':  # 该分支不是叶子节点，递归
                    classLabelSub = classify(secondDict[key], classList, featLabels,
                                             featLabelProperties, testVec)
                    for classKey in classLabel.keys():
                        classLabel[classKey] += classLabelSub[key]
                else:  # 如果是叶子， 返回结果
                    for classKey in classLabel.keys():
                        if classKey == secondDict[key][0]:
                            classLabel[classKey] += secondDict[key][1]
                        else:
                            classLabel[classKey] += secondDict[key][2]
```

```
        else:
            partValue = float(str(firstStr)[lessIndex + 1:])
            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
                # 进入左子树
                if type(secondDict[key]).__name__ == 'dict':  # 该分支不是叶子节点，递归
                    classLabelSub = classify(secondDict[key], classList, featLabels,
                                             featLabelProperties, testVec)
                    for classKey in classLabel.keys():
                        classLabel[classKey] += classLabelSub[classKey]
                else:  # 如果是叶子， 返回结果
                    for classKey in classLabel.keys():
                        if classKey == secondDict[key][0]:
                            classLabel[classKey] += secondDict[key][1]
                        else:
                            classLabel[classKey] += secondDict[key][2]
            elif float(testVec[featIndex]) <= partValue and key == 'Y':  # 进入左子树
                if type(secondDict['Y']).__name__ == 'dict':  # 该分支不是叶子节点，递归
                    classLabelSub = classify(secondDict['Y'], classList, featLabels,
                                                featLabelProperties, testVec)
                    for classKey in classLabel.keys():
                        classLabel[classKey] += classLabelSub[classKey]
                else:  # 如果是叶子， 返回结果
                    for classKey in classLabel.keys():
                        if classKey == secondDict[key][0]:
                            classLabel[classKey] += secondDict['Y'][1]
                        else:
                            classLabel[classKey] += secondDict['Y'][2]
            elif float(testVec[featIndex]) > partValue and key == 'N':
                if type(secondDict['N']).__name__ == 'dict':  # 该分支不是叶子节点，递归
                    classLabelSub = classify(secondDict['N'], classList, featLabels,
                                                featLabelProperties,
```

```
testVec)
                    for classKey in classLabel.keys():
                        classLabel[classKey] += classLabelSub[classKey]
                else:  # 如果是叶子， 返回结果
                    for classKey in classLabel.keys():
                        if classKey == secondDict[key][0]:
                            classLabel[classKey] += secondDict['N'][1]
                        else:
                            classLabel[classKey] += secondDict['N'][2]

    return classLabel


# 存储决策树
def storeTree(inputTree, filename):
    import pickle
    fw = open(filename, 'w')
    pickle.dump(inputTree, fw)
    fw.close()


# 读取决策树，文件不存在返回None
def grabTree(filename):
    import pickle
    if os.path.isfile(filename):
        fr = open(filename)
        return pickle.load(fr)
    else:
        return None


# 测试决策树正确率
def testing(myTree, classList, data_test, labels, labelProperties):
    error = 0.0
    for i in range(len(data_test)):
        classLabelSet = classify(myTree, classList, labels, labelProperties, data_test[i])
        maxWeight = 0.0
        classLabel = ''
        for item in classLabelSet.items():
            if item[1] > maxWeight:
                classLabel = item[0]
```

```
        if classLabel != data_test[i][-1]:
            error += 1
    return float(error)


# 测试投票节点正确率
def testingMajor(major, data_test):
    error = 0.0
    for i in range(len(data_test)):
        if major[0] != data_test[i][-1]:
            error += 1
    # print 'major %d' %error
    return float(error)


# 后剪枝
def postPruningTree(inputTree, classSet, dataSet, data_test, labels, labelProperties):
    firstStr = list(inputTree.keys())[0]
    secondDict = inputTree[firstStr]
    classList = [example[-1] for example in dataSet]
    weightList = [example[-2] for example in dataSet]
    featkey = copy.deepcopy(firstStr)
    if '<' in firstStr:  # 对连续的特征值，使用正则表达式获得特征标签和value
        featkey = re.compile("(.+<)").search(firstStr).group()[:-1]
        featvalue = float(re.compile("(<.+)").search(firstStr).group()[1:])
    labelIndex = labels.index(featkey)
    temp_labels = copy.deepcopy(labels)
    temp_labelProperties = copy.deepcopy(labelProperties)
    if labelProperties[labelIndex] == 0:  # 离散特征
        del (labels[labelIndex])
        del (labelProperties[labelIndex])
    for key in secondDict.keys():  # 对每个分支
        if type(secondDict[key]).__name__ == 'dict':  # 如果不是叶子节点
            if temp_labelProperties[labelIndex] == 0:  # 离散的
                subDataSet = splitDataSet(dataSet, labelIndex, key)
                subDataTest = splitDataSet(data_test, labelIndex, key)
            else:
                if key == 'Y':
                    subDataSet = splitDataSet(dataSet, labelIndex,
```

```
featvalue,
                                                'L')
                    subDataTest = splitDataSet(data_test, labelIndex,
                                                featvalue, 'L')
                else:
                    subDataSet = splitDataSet(dataSet, labelIndex,
featvalue,
                                                'R')
                    subDataTest = splitDataSet(data_test, labelIndex,
                                                featvalue, 'R')
            if len(subDataTest) > 0:
                inputTree[firstStr][key] = postPruningTree(secondDict[
key], classSet,
                                                            subDataSet,
subDataTest,
                                                            copy.deepcopy(
labels),
                                                            copy.deepcopy(

labelProperties))
    if testing(inputTree, classSet, data_test, temp_labels,
               temp_labelProperties) <= testingMajor(majorityCnt(
classList, weightList),
                                                        data_test):
        return inputTree
    return majorityCnt(classList,weightList)
```