人工智能技术基础
实验指导书

人工智能课程组编写

北京交通大学电子信息工程学院

2019 年 2 月

实验四 决策树 ID3 算法

1 目的与要求

实验目的：

熟悉和掌握熵和信息增益的定义，掌握决策树 ID3 算法。

实验要求：

（1）编程语言 C++/python

（2）

2 实验原理：

决策树 ID3 算法

ID3(Examples, Target_attribute, Attributes)
创建树的 root 节点
如果 Examples 都为正,返回 label=+的单节点树 root
如果 Examples 都为反,返回 label=-的单节点树 root
如果 Attributes 为空，那么返回单节点 root，label=Examples 中最普遍的 Target_attribute 值
否则开始
A←Attributes 中分类 examples 能力最好的属性
root 的决策属性←A
对于 A 的每个可能值 vi
在 root 下加一个新的分支对应测试 A=vi
令 Examplesvi 为 Examples 中满足 A 属性值为 vi 的子集
如果 Examplesvi 为空
在这个新分支下加一个叶子节点，节点的 label=Examples 中最普遍的 Target_attribute 值
否则在新分支下加一个子树 ID3（ Examplesvi,Target_attribute,Attributes-{A}）
结束
返回 root

3 实验内容或题目

(1). 根据训练样例计算整体熵。
(2). 根据训练样例计算各属性的信息增益。
(3). 设计决策树 ID3 算法。

4 实验步骤与源程序

(1) 编写熵的计算程序。
(2) 编写信息增益的计算程序。
(3) 编写决策树 ID3 算法

5 实验报告要求

实验报告应至少包含如下内容：

- 实验题目
- 实验目的
- 实验环境
- 实验内容
- 实验结果（要求附上源程序）
- 实验中出现的问题
- 对问题的解决方案
- 实验总结

6 参考源代码

1、决策树的定义
决策树代表实例属性值约束的合取的析取式。从树根到树叶的每一条路径对应一组属性测试的合取，树本身对应这些合取的析取。
2、决策树的基本特征
1）通过把实例从根节点排列到某个叶子节点来分类实例
2）叶子节点即为实例所属的分类
3）树上每个节点说明了对实例的某个属性的测试
4）节点的每个后继分支对应于该属性的一个可能值
3、决策树的构建原则
1）树的生成,开始时所有的数据都在根节点,然后根据设定的标准选择测试属性,用不同的测试属性递归进行数据分割。
2）树的修剪,就是除去一些可能是噪音或异常的数据。
4、ID3算法
1）ID3 算法通过对一个训练例集进行学习生成一棵决策树,训练例集中的每一个例子都组织成属性—属性值对的形式。假设一个例子仅属于正例(符合被学习目标概念的例子) 或反例(不符合目标概念的例子) 两种分类之一,例子的所有属性都为离散属性。对于每个训练例集 Es ，如果正例的比例为P+ , 则反例比例就为P- = 1 - P+ , 熵的公式为:Ent ropy (Es) = - P+ log2 P+ - P- log2 P-(这里约定 log20 = 0)
2)若用属性 A 将训练例集 Es 分组, Entropy(Es)将会降低,新的期望信息量设为:New_Entropy (Esi , A) =Σi ∈Value (A)(| Esi | / | Es |) Ent ropy (Esi)
3）A 相对于 Es 的信息赢取 Gain (Es , A) , 即 Ent ropy (Es) 降低的数量,信息赢取越大的属性对训练例集越有利:Gain(Es ,A) = Entropy(Es) - New_Entropy(Esi , A)
5、代码如下：

```
// id3.cpp
#pragma warning (disable: 4996)
#include <stdio.h>
#include <stdlib.h>
#include <string>
#include <math.h>
#include <list>
using namespace std;
```

```
#define LN_2 0.693147180559945309417
#define ID3_ERROR 9999999
#define A_CHAR_MAX 16
#define A_VALUE_MAX 16
#define A_NUM_MAX     32
#define SAMPLES_MAX 256
#define ALL -1
#define NUL -2
#define YES         1
#define NO          0
#define NUKOWN -1
#define VALID '*'
#define INVALID '-'
// 属性
struct Attribute
{
char name[A_CHAR_MAX];                    // 属性名称
int num;                                  // 属性值个数
char att[A_VALUE_MAX][A_CHAR_MAX]; // 属性值
};
// 假设
struct Hypothesis
{
int num;                                  // 属性个数
Attribute an[A_NUM_MAX];                  // 属性集合
};
// 假设值
struct HypoValue
{
int value[A_NUM_MAX];
};
// 样本
struct Sample
{
HypoValue ev;                             // 假设
int result;                               // 正例/反例
};
Hypothesis g_Hypo;                        // 假设集合
Sample g_sa[SAMPLES_MAX];                 // 样本空间
int g_sn;                                 // 样本数
bool ReadHypothesis(const char* filename);
bool ReadSamples(const char* filename);
int CheckAllPositive();
int CheckAllNegative();
int CreateTree(char samplevalid[SAMPLES_MAX], char attvalid[A_NUM_MAX], FILE*, int);
int NotAllSame(char samplevalid[SAMPLES_MAX]);
```

```
int FindAtt(char attvalid[A_NUM_MAX], char samplevalid[SAMPLES_MAX]);
void Disaster(int);
int ID3(const char* filename);
// 从文件中读取假设集合
/*/ 文件格式
[集合个数 n]
[属性名称1] [属性值个数] [属性值1] [属性值2] [属性值3] ……
[属性名称2] [属性值个数] [属性值1] [属性值2] [属性值3] ……
……
[属性名称 n] [属性值个数] [属性值1] [属性值2] [属性值3] ……
/*/
bool ReadHypothesis(const char* filename)
{
FILE* file;
if (fopen_s(&file, filename , "r"))
    return false;
int i,j,k;
fscanf(file, "%d\n", &g_Hypo.num);
for (i=0; i<g_Hypo.num ; i++)
{
    fscanf(file, "%s%d\n", g_Hypo.an[i].name, &k);
    g_Hypo.an[i].num = k;
    for (j=0; j<k; j++)
    {
      fscanf(file, "%s", g_Hypo.an[i].att[j]);
    }
    fscanf(file, "\n");
}
fclose(file);
return true;
}
// 从文件中读取样本
/*/ 文件格式
[样本个数 m]
[样本1属性1的值的序号] [样本1属性2的值的序号] …… [样本1属性 n 的值的序号] [1（正例）或者0（反例）]
[样本2属性1的值的序号] [样本2属性2的值的序号] …… [样本2属性 n 的值的序号] [1（正例）或者0（反例）]
……
[样本 m 属性1的值的序号] [样本 m 属性2的值的序号] …… [样本 m 属性 n 的值的序号] [1（正例）或者0（反例）]
/*/
bool ReadSamples(const char* filename)
{
FILE* file;
if (fopen_s(&file, filename , "r"))
```

```
    return false;
int i,j;
fscanf(file, "%d\n", &g_sn);
for (i=0; i<g_sn ; i++)
{
    for (j=0; j<g_Hypo.num; j++)
    {
      fscanf(file, "%d", &g_sa[i].ev.value[j]);
    }
    fscanf(file, "%d\n",&g_sa[i].result);
}
fclose(file);
return true;
}
int CheckAllPositive()
{
int i;
for(i=0;i<g_sn;i++)
{
    if (g_sa[i].result == NO)
    {
      return 0;
    }
}
return 1;
}
int CheckAllNegative()
{
int i;
for(i=0;i<g_sn;i++)
{
    if (g_sa[i].result == YES)
    {
      return 0;
    }
}
return 1;
}
int NotAllSame(char samplevalid[SAMPLES_MAX])
{
int i, y_tot = 0, n_tot = 0;
for (i=0; i<g_sn; i++)
{
    if (samplevalid[i] == VALID)
    {
      if (g_sa[i].result == YES)
```

```
        ++y_tot;
       if (g_sa[i].result == NO)
        ++n_tot;
    }
}
if (n_tot == 0)
    return 2; /* all yes */
else if (y_tot == 0)
    return 3; /* all no */
else
    return 1;
}
int FindAtt(char attvalid[A_NUM_MAX], char samplevalid[SAMPLES_MAX])
{
int i, j, l, y_tot = 0, n_tot = 0, y_tot_2, n_tot_2;
int tot_diff_atts;
int att_no = 0;
double max_inf_gain = -1.0;
double entropy, entropy_2, r_entropy_tot;
double att_entropy[A_NUM_MAX];
char valid_2[SAMPLES_MAX];

for (i=0; i<g_Hypo.num; i++)
{
    att_entropy[i] = -2.0;
}

for (i=0; i<g_sn; i++)
{
    if (samplevalid[i] == VALID)
    {
      if (g_sa[i].result == YES)
        ++y_tot;
      if (g_sa[i].result == NO)
        ++n_tot;
    }
}
if (y_tot == 0 || n_tot == 0)
    entropy = 0.0;
else
{
    entropy = 0.0 - ((y_tot/(double)(y_tot+n_tot))*log((y_tot/(double)(y_tot+n_tot))))
      - ((n_tot/(double)(y_tot+n_tot))*log((n_tot/(double)(y_tot+n_tot))));
}
for (i=0; i<g_Hypo.num; i++)
{
```

```
    if (attvalid[i] == VALID)
    {
     r_entropy_tot = 0.0;
     tot_diff_atts = g_Hypo.an[i].num;
     for (j=0; j<tot_diff_atts; j++)
     {
      memset (valid_2, INVALID, g_sn);
      for (l=0; l<g_sn; l++)
      {
       if ((g_sa[l].ev.value[i] == j+1) && (samplevalid[l] == VALID))
        valid_2[l] = VALID;
      }
      y_tot_2 = 0;
      n_tot_2 = 0;
      for (l=0; l<g_sn; l++)
      {
       if (valid_2[l] == VALID)
       {
        if (g_sa[l].result == YES)
         ++y_tot_2;
        if (g_sa[l].result == NO)
         ++n_tot_2;
       }
      }
      if (n_tot_2 == 0 || y_tot_2 == 0)
       entropy_2 = 0.0;
      else
      {
       entropy_2 = 0.0 -
((y_tot_2/(double)(y_tot_2+n_tot_2))*log((y_tot_2/(double)(y_tot_2+n_tot_2))))
                   -
((n_tot_2/(double)(y_tot_2+n_tot_2))*log((n_tot_2/(double)(y_tot_2+n_tot_2))));
      }
      r_entropy_tot = r_entropy_tot + (entropy_2 * ((n_tot_2+y_tot_2)/(double)(n_tot+y_tot)));
     }
     att_entropy[i] = entropy - r_entropy_tot;
    }
}
for (l=0; l<g_Hypo.num; l++)
{
    if (att_entropy[l] >= max_inf_gain)
    {
     max_inf_gain = att_entropy[l];
     att_no = l;
    }
}
```

```
if (max_inf_gain == 0.0)
{
    return ID3_ERROR;
}
return att_no;
}
void Disaster(int num)
{
switch(num)
{
case 1: printf("** ID3 failure **\n");
    break;
case 2:
    break;
default:
    break;
}
}
int CreateTree(char samplevalid[SAMPLES_MAX], char attvalid[A_NUM_MAX], FILE *opf, int
tab_cnt)
{
char samplevalid2[SAMPLES_MAX];
char attvalid2[A_NUM_MAX];
int j, l, i, ret, tot_diff_atts, att_no;
for (i=0; i<tab_cnt+tab_cnt; i++)
{
    fprintf(opf,"\t");
}
tab_cnt++;
if ((att_no = FindAtt(attvalid, samplevalid)) == ID3_ERROR)
{
    return ID3_ERROR;
}
strncpy(attvalid2, attvalid, g_Hypo.num);
attvalid2[att_no] = INVALID;
fprintf(opf, "[%s]\n", g_Hypo.an[att_no].name);
tot_diff_atts = g_Hypo.an[att_no].num;
for (j=0; j<tot_diff_atts; j++)
{
    //valid[M1-1] = '\0';
    strncpy(samplevalid2, samplevalid, g_sn);
    for (l=0; l<g_sn; l++)
    {
      if (g_sa[l].ev.value[att_no] != j+1)
      {
        samplevalid2[l] = INVALID;
```

```
      }
    }
    if ((ret = NotAllSame(samplevalid2)) == 1)
    {
      for (i=0; i<tab_cnt+tab_cnt-1; i++)
      {
        fprintf(opf, "\t");
      }
      fprintf(opf, " %s\n", g_Hypo.an[att_no].att[j]);
      if (CreateTree(samplevalid2, attvalid2, opf, tab_cnt) == ID3_ERROR)
      {
        return ID3_ERROR;
      }
    }
    else
    {
      for (i=0; i<tab_cnt+tab_cnt-1; i++)
      {
        fprintf(opf, "\t");
      }
      if (ret == 2)
      {
        fprintf(opf, " %s\t - YES\n", g_Hypo.an[att_no].att[j]);
      }
      else
      {
        fprintf(opf, " %s\t - NO\n", g_Hypo.an[att_no].att[j]);
      }
    }
}
return 1;
}

// ID3算法，结果保存在 filename 文件中
int ID3(const char* filename)
{
int tab_cnt = 0;
char samplevalid[SAMPLES_MAX];
char attvalid[A_NUM_MAX];
FILE *opf;
if ((opf = fopen(filename, "w")) == NULL)
{
    printf("File error : Cannot create output file TREE\n");
    return 0;
}
fprintf(opf, "\n");
```

```
if (CheckAllPositive())
{
    fprintf(opf,"HALT:all_positive\n");
    fclose(opf);
    return 1;
}
if (CheckAllNegative())
{
    fprintf(opf,"HALT:all_negative\n");
    fclose(opf);
    return 1;
}
memset (samplevalid, VALID, g_sn);
memset (attvalid, VALID, g_Hypo.num);
if (CreateTree(samplevalid, attvalid, opf, tab_cnt) == ID3_ERROR)
{
    return 0;
}
fclose(opf);
return 1;
}
int main(int arc, char** argv)
{
// 读取假设和样本
if (!ReadHypothesis(argv[1]))
{
    printf("read hypothesis file error");
    return 0;
}
if (!ReadSamples(argv[2]))
{
    printf("read samples file error");
    return 0;
}
if (ID3(argv[3]))
    printf ("Decision Tree has been created with ID3 and stored in %s\n", argv[3]);
else
    Disaster(1);
getchar();
return 0;
}
```

编译后运行：id3 hypothesis2.txt samples2.txt tree.txt