

MachineLearningHomework3_DecisionTree

201250068 陈骏

环境配置

```
Python 3.7
Package:
    sklearn
    copy
    csv
    math
```

项目代码

代码结构

```
D: .
|  dataLoad.py # 读取csv文件
|  decisionTree.py # 决策树类
|  DTmath.py # 决策树相关数学类，提供信息增益及信息熵计算函数
|  log.txt
|  main.py
|  README.md
|
|_ .idea
|   | .gitignore
|   | DecisionTree.iml
|   | misc.xml
|   | modules.xml
|   | workspace.xml
|   |
|   |_ inspectionProfiles
|       profiles_settings.xml
|       Project_Default.xml
|
|_ dataResource
|   result.csv # 最终结果
|   testing.csv
|   training.csv
|   validation.csv
|
|_ __pycache__
|   dataLoad.cpython-37.pyc
|   decisionTree.cpython-37.pyc
|   DTmath.cpython-37.pyc
```

数据预处理

dataLoad.py 中，通过选取了 condition, date, usefulCount, sideEffects 四类属性作为决策树的分类属性。其中，condition 属性为枚举值，但存在大量的低频属性，即出现次数小于等于 10 次的样本，通过合并为其他类型进行预剪枝。date 类截取年份作为属性，usefulCount 为连续值，进行选取划分点进行区间化处理。sideEffects 不作处理。

DecisionTree实现

通过C4.5算法，计算每个节点选取不同的属性作为分类属性的信息增益，选取信息增益最大的属性作为当前节点的划分属性。当当前属性下所有的样本都为同一个分类，则当前节点返回叶节点。当当前属性为所有样本的最后一个分类属性，其叶节点返回当前数据集中出现次数最多的分类值。

代码内容

```
from DTmath import calculate_entropy
from DTmath import calculate_gain_sa
from sklearn.tree import DecisionTreeClassifier, export_graphviz
import copy

mat = "{:10}"

class Tree:

    def __init__(self):
        # 根节点，以及该节点的分类数目
        self.root = Node(0, 0)
        pass

    def train(self, train_data, result):
        self.root.data = len(train_data)
        self.root.classify_condition = "root"
        self.root.train(train_data, result)
        print("IIIIIIIIIIIIIIIIIIIIIIIIIIIIII\n\n\n\n\n\n\n\n")
        self.root.toString("")
        pass

    def valid(self, valid_data):
        result = []
        valid_len = len(valid_data)
        for i in range(valid_len):
            actual = self.root.input(valid_data[i])
            result.append(actual)
            print("Info : input data " + str(valid_data[i]))
            print("Info : tree decision " + str(actual))
        return result

    def input(self, data):

        self.root.input(data)
        pass


class Node:
    child = []
    depth = -1
    data = -1
```

```

def __init__(self, depth, data):
    self.child = []
    self.leaf = False
    self.label = ""
    self.classify_condition = ""
    self.depth = depth
    pass

def train(self, train_data, result):
    # print(mat.format("Info : train data " + str(train_data)))
    # print(mat.format("Info : target result " + str(result)))
    train_data_size = len(train_data)
    if train_data[0] is None:
        attribute_size = 0
    else:
        attribute_size = len(train_data[0])
    print("Info : current attribute num " + str(attribute_size))
    result_enum = []
    result_enum_num = []
    for i in range(len(result)):
        if not result_enum.__contains__(result[i]):
            result_enum_num.append(0)
            result_enum.append(result[i])
        result_enum_num[result_enum.index(result[i])] += 1
    result_enum.sort()
    # 目标值仅为 1 种
    if len(result_enum) == 1:
        self.child.append(Node(self.depth + 1, len(result)))
        self.child[0].leaf = True
        self.child[0].label = result_enum[0]
        # print("Info : only one class " + str(self.label))
        return
    # 还存在分类属性
    elif attribute_size >= 1:
        # print("Info : attribute classify")
        gain_max = -999
        attribute_index = -1
        gain_max_attribute_enum = None
        gain_max_attribute_enum_num = None
        s = calculate_entropy(result, result_enum)
        # print("Info : result entropy " + str(s))
        for i in range(attribute_size):
            attribute_enum_num = []
            attribute_enum = []
            data = []
            for j in range(train_data_size):
                data.append(train_data[j][i])
                if not attribute_enum.__contains__(train_data[j][i]):
                    attribute_enum.append(train_data[j][i])
                    attribute_enum_num.append(0)
                attribute_enum_num[attribute_enum.index(train_data[j][i])]
            += 1

            print("Info : attribute enum " + str(attribute_enum))
            print("Info : attribute enum count " + str(attribute_enum_num))
            # 计算信息增益率
            gain = s - calculate_gain_sa(data, attribute_enum, result,
result_enum)

```

```

        # print("Info : Gain(S, A) " + str(gain))
        # 选取最大信息增益率的属性作为分类
        if gain > gain_max:
            gain_max = gain
            attribute_index = i
            gain_max_attribute_enum = attribute_enum
            gain_max_attribute_enum_num = attribute_enum_num
        # 选取最大信息增益的属性作为分类属性之后，传入节点的子节点
        print("Info : select attribute " + str(gain_max_attribute_enum))
        for i in range(len(gain_max_attribute_enum)):
            self.child.append(Node(depth=self.depth + 1,
data=gain_max_attribute_enum_num[i]))
            self.child[i].classify_condition = gain_max_attribute_enum[i]
            # 加载子节点建立决策数需要的数据的result
            child_data = []
            child_result = []
            for j in range(len(train_data)):
                if train_data[j][attribute_index] ==
gain_max_attribute_enum[i]:
                    temp = copy.deepcopy(train_data[j])
                    temp.remove(train_data[j][attribute_index])
                    child_result.append(result[j])
                    child_data.append(temp)
            self.child[i].train(child_data, child_result)
        # 仅剩当前属性，为根节点，确定分类值
        else:
            self.child.append(Node(depth=self.depth + 1, data=len(result)))
            self.child[0].leaf = True
            avg = sum(list(result)) / len(result)
            self.child[0].label = [int(avg), int(avg) + 1]
            result_index = -1
            result_count = -1
            for i in range(len(result_enum)):
                count = list(result).count(result_enum[i])
                if count > result_count:
                    result_index = i
            self.child[0].label = result_enum[result_index]

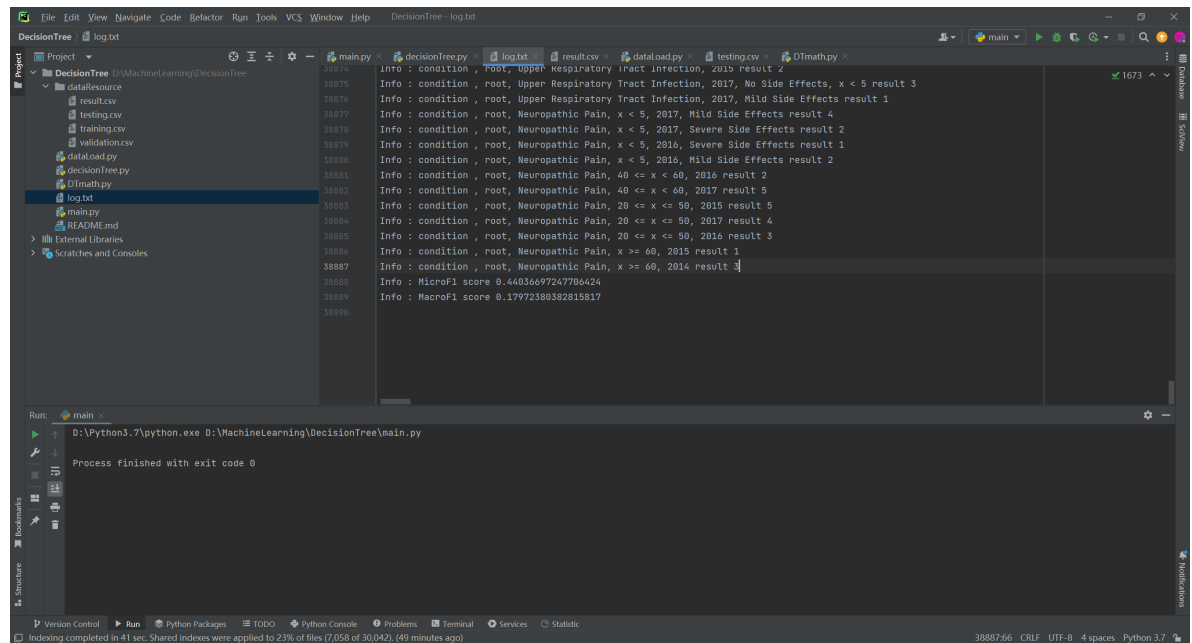
def input(self, data):
    for i in range(len(self.child)):
        if self.child[i].leaf:
            return self.child[i].label
        if list(data).__contains__(self.child[i].classify_condition):
            return self.child[i].input(data)
    return self.child[0].input(data)

def toString(self, condition):
    if self.leaf:
        print("Info : condition " + condition + " result " +
str(self.label))
    else:
        for i in self.child:
            i.toString(condition + ", " + self.classify_condition)

```

决策树结果

详见 `log.txt` 以condition开头的打印信息及result.csv文件。



The screenshot shows an IDE interface for a project named 'DecisionTree'. The file explorer on the left lists files including 'dataResource', 'result.csv', 'testing.csv', 'training.csv', 'validation.csv', 'dataLoad.py', 'decisionTree.py', 'DTmath.py', 'log.txt', 'main.py', and 'README.md'. The main editor window displays the content of 'log.txt', which contains a series of log messages starting with 'Info : condition' and ending with 'Info : MacroF1 score 0.17972388382815817'. The Run console at the bottom shows the command 'D:\Python3.7\python.exe D:\MachineLearning\DecisionTree\main.py' and the message 'Process finished with exit code 0'.

```
38876 Info : condition , root, Upper Respiratory Tract Infection, 2015 result 2
38876 Info : condition , root, Upper Respiratory Tract Infection, 2017, No Side Effects, x < 5 result 3
38876 Info : condition , root, Upper Respiratory Tract Infection, 2017, Mild Side Effects result 1
38877 Info : condition , root, Neuropathic Pain, x < 5, 2017, Mild Side Effects result 4
38878 Info : condition , root, Neuropathic Pain, x < 5, 2017, Severe Side Effects result 2
38879 Info : condition , root, Neuropathic Pain, x < 5, 2016, Severe Side Effects result 1
38880 Info : condition , root, Neuropathic Pain, x < 5, 2016, Mild Side Effects result 2
38881 Info : condition , root, Neuropathic Pain, 40 <= x < 60, 2016 result 2
38882 Info : condition , root, Neuropathic Pain, 40 <= x < 60, 2017 result 5
38883 Info : condition , root, Neuropathic Pain, 20 <= x <= 50, 2015 result 5
38884 Info : condition , root, Neuropathic Pain, 20 <= x <= 50, 2017 result 4
38885 Info : condition , root, Neuropathic Pain, 20 <= x <= 50, 2016 result 3
38886 Info : condition , root, Neuropathic Pain, x >= 60, 2015 result 1
38887 Info : condition , root, Neuropathic Pain, x >= 60, 2014 result 3
38888 Info : MacroF1 score 0.44036697247784424
38889 Info : MacroF1 score 0.17972388382815817
38890
```

Run console output:

```
D:\Python3.7\python.exe D:\MachineLearning\DecisionTree\main.py
Process finished with exit code 0
```