

机器学习初步——决策树实验报告

07111904班 张驰 1120191600

1 实验目的

在本次实验中，要求实现一个决策树分类器，解决二分类问题。实验中所使用的训练集来源于乐学平台，根据其中的数据集建立一个决策树，对测试集中的数据进行相应的二分类预测。在本次实验中，数据集的所有属性都被离散为两个类别，即每个节点最多有两个子节点。通过构建二分类决策树，体验机器学习在实际生活中的运用，进一步理解机器学习这门课程。

本次实验使用python程序设计语言编写两个程序：inspection.py 和 decisionTree.py，分别用于检查数据和构建决策树。

2 Program1: 检测数据

本部分程序的代码位于inspection.py文件中，用于检测数据集中的数据，并输出决策树根的标签熵和使用多票数分类器进行分类的错误率。（熵的计算使用对数基数2进行运算）

2.1 程序分模块实现过程

inspection.py中主要包含两个函数，分别是load_tsv()函数和calc_Entropy_Error()函数，分别用于载入数据集和计算熵和错误率。本程序中的main函数较为简单，不再展示赘述。

2.1.1 load_tsv(inputfile)函数

load_tsv(inputfile)函数通过csv.reader()函数，读入分隔符为\t的tsv数据集，在其中读入数据集时，仅读入label（结果），函数代码如下所示，

```
def load_tsv(inputfile):
    data_file = open(input_file, "r")
    reader = csv.reader(data_file, delimiter='\t')
    headers = next(reader)
    label = []

    for row in reader:
        label.append(row[-1])

    data_file.close()
    return label
```

2.1.2 calc_Entropy_Error(label)函数

本函数用于计算熵和错误率，熵的计算以2为底数，错误率的计算采用多票数分类法。由于所有数据集集中的标签均被分为两个类别，所以在这个函数只需要统计label中的两个结果的数量，返回熵和错误率。

```
def calc_Entropy_Error(label):
    first_res = 0
    second_res = 0
    for i in label:
        if i == label[0]:
```

```

        first_res += 1
    else:
        second_res += 1

    if first_res < second_res:
        error = float(first_res)/(first_res + second_res)
    else:
        error = float(second_res)/(first_res + second_res)

    first_prob = float(first_res)/(first_res+second_res)
    second_prob = float(second_res)/(first_res+second_res)
    entropy = 0.0
    entropy -= first_prob * np.log2(first_prob)
    entropy -= second_prob * np.log2(second_prob)
    return entropy, error

```

2.2 程序运行说明

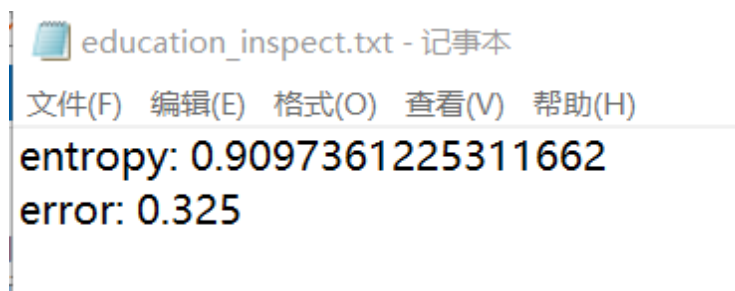
该程序运行使用命令行进行调用，inspection.py程序应该接受两个命令行参数，命令行参数如下所示：

命令行参数	参数的含义
input	要测试的数据文件的名称，例如：small_train.tsv
output	将结果输出到文件的名称，例如：small_inspect.txt

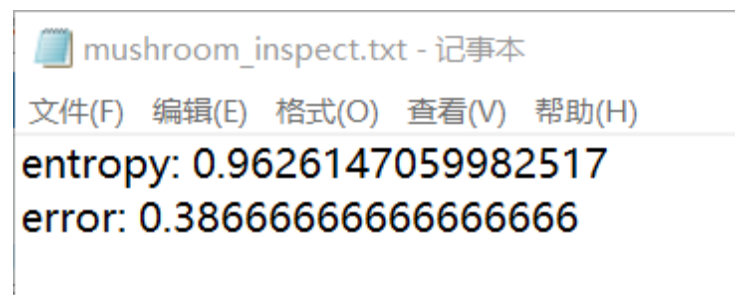
示例：`python inspection.py small_train.tsv small_inspect.txt`

2.3 程序运行结果展示

该程序在education_train.tsv数据集上的运行结果如下图所示：



该程序在mushroom_train.tsv数据集上的运行结果如下图所示：



该程序在politicians_train.tsv数据集上的运行结果如下图所示：

```
politicians_inspect.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
entropy: 0.9905894286537543
error: 0.4429530201342282
```

该程序在small_train.tsv数据集上的运行结果如下图所示：

```
small_inspect.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
entropy: 0.996316519558962
error: 0.4642857142857143
```

3 Program2：构建决策树 Decision Tree Learner

3.1 构建决策树的方法

决策树是基于树状结构来进行决策的，一般地，一棵决策树包含一个根节点、若干个内部节点和若干个叶节点。其中，每个内部节点代表一个属性上的判断，每个树支代表一个判断的条件，每个叶节点代表一种分类结果，根节点包含样本全集。

在本次实验中，构建决策树采用的ID3方法，ID3算法的核心是根据信息增益来进行划分的特征，然后递归地构建决策树。

3.1.1 特征选择

特征选择从当前数据特征中选择一个特征作为划分依据，选择为当前特征中信息增益最大的属性节点。

信息增益的计算方式如下：

$$Entropy(熵) = - \sum_{i=1}^n p(x_i) \log_2 p(x_i)$$

其中 $p(x_i)$ 表示的是 x_i 出现的概率，n是分类的数目，在本次二分类实验中，n=2

条件熵计算方式：

$$Entropy(Y|X) = \sum_{i=1}^n p(x_i) H(Y|X = x_i)$$

信息增益的计算方式如下：

$$Gain(A) = Entropy(Dataset) - Entropy(Dataset|A)$$

根据信息增益的大小，选择相应的特征，对训练数据进行划分

3.1.2 递归建树的步骤

1. 从根节点开始，计算所有可能的特征的信息增益，选择信息增益最大的特征作为节点的划分特征；
2. 由该特征的不同取值建立子节点；
3. 再对子节点递归1-2步，构建决策树；
4. 满足递归终止判定条件时，停止递归，对每个结点根据多数票分类法，得到最终的决策树。

3.2 程序实现过程

decisionTree.py程序代码过长，仅对核心过程（计算信息增益的过程、递归构建决策树的过程以及预测的过程）进行介绍。

3.2.1 计算信息增益的过程

其中，`entropy()` 函数用于计算数据集总体的熵，`condition_entropy()` 用于计算条件熵，`gain()` 函数用于计算某一个特征的信息增益。

```
def entropy(label):  
    """  
    计算结果列表的熵  
    :param label: 标签列表  
    :return: 熵  
    """  
    first_res = 0  
    second_res = 0  
    for i in label:  
        if i == label[0]:  
            first_res += 1  
        else:  
            second_res += 1  
  
    first_prob = float(first_res)/(first_res+second_res)  
    second_prob = float(second_res)/(first_res+second_res)  
    entropy = 0.0  
    if first_prob != 0:  
        entropy -= first_prob * np.log2(first_prob)  
    if second_prob != 0:  
        entropy -= second_prob * np.log2(second_prob)  
    return entropy  
  
def condition_entropy(attr_value_list, result_list):  
    """  
    计算条件熵  
    :param attr_value_list: 属性的特征向量  
    :param result_list: 结果列表  
    :return: 条件熵  
    """  
    entropy_dict = collections.defaultdict(list)  
    for attr_value, value in zip(attr_value_list, result_list):  
        entropy_dict[attr_value].append(value)
```

```

con_ent = 0.0
attr_len = len(attr_value_list)
for value in entropy_dict.values():
    p = len(value)/attr_len * entropy(value)
    con_ent += p
return con_ent

def gain(attr_value_list, result_list):
    """
    获取某一特征的信息增益
    :param attr_value_list: 属性的特征向量
    :param result_list: 结果列表
    :return: 信息增益
    """
    ent = entropy(result_list)
    con_ent = condition_entropy(attr_value_list, result_list)
    return ent - con_ent

```

3.2.2 定义决策树节点类:

为了方便打印输出，并做出相关的判断定义决策树的节点，决策树的节点包含以下信息：

1. 已经被选择过的特征属性的索引
2. 待检验的判断条件，对应的属性列的索引值
3. 当前节点中等待判断的数据
4. 当前节点中匹配的label值
5. 当前分支的判断结果（如果有判断结果则为叶子节点，如果没有判断结果则为内部节点）
6. 当前节点信息增益最高的特征为True的子树（1下的子树）
7. 当前节点信息增益最高的特征为False时的子树（0下的子树）
8. 当前节点处于决策树的深度
9. 整棵树要求的最大深度
10. 判断属性内容列表
11. 标签值的集合
12. 左子树和右子树对应的属性的判断表

节点类的代码如下所示：

```

class DecisionNode(object):
    def __init__(
        self, col=-1, data_set=None, labels=None,
        results=None, tb=None, fb=None, depth=None, max_depth=None,
        table=None, attr_lst=None, labelset=None):
        self.has_calc_index = [] # 已经计算过的特征索引
        self.col = col # col 是待检验的判断条件，对应列索引值
        self.data_set = data_set # 节点的 待检测数据
        self.labels = labels # 对应当前列必须匹配的值
        self.results = results # 保存的是针对当前分支的结果，有值则表示该点是叶子

```

节点

```

self.tb = tb    # 当信息增益最高的特征为True时的子树
self.fb = fb    # 当信息增益最高的特征为False时的子树
self.depth = depth    # 节点处于决策树的深度
self.max_depth = max_depth    # 节点的最大深度
self.table = table #0, 1对应的属性的判断条件表
self.attr_lst = attr_lst #判断属性内容列表
self.label_set = labelset #标签值的集合

```

3.2.3 定义决策树类:

决策树类仅包含两个属性特征：决策树的根节点，以及决策树特征数目。

决策树类包含的方法

1. `majorlabel()` 方法：采用采用多数投票的方式建立叶子节点，选择叶子节点的的判断属性
2. `count_label()` 方法：用于输出，计算标签的个数并输出
3. `build_tree()` 方法，递归使用该方法建立决策树结构
4. `train()` 方法，进行训练
5. `_predict()` 方法以及 `predict()` 方法，对整个数据集进行预测

决策树类的代码如下所示：

```

class DecisionTree():
    def __init__(self):
        self.feature_num = 0
        self.root = None

    def majorlabel(self, labels):
        """
        采用majorvote的方式建立叶子节点
        :param labels:
        :return: major结果
        """

    def count_label(self, labels, labelset):
    def build_tree(self, node: DecisionNode):
        """
        递归建立决策树结构
        :param node: 树的节点
        :return:
        """

    def _predict(self, data_test, node):
        """
        对单个的数据进行预测
        :param data_test: 单个数据
        :param node: 节点
        :return: 预测单个的结果
        """

    def predict(self, data_test_set):
        """
        对整个数据集进行预测

```

```

:param data_test_set: 所有数据
:return: 预测结果

```

3.2.4 递归构建决策树的过程

首先要设定递归停止的条件，递归停止的条件如下所示，满足下列条件之一即成立

1. 结点的深度超过了设定的最大深度，则停止递归。（在此情况下，使用多数表决的方法）
2. 决策树的结点的所有样本属于同一类
3. 没有剩余属性可以用来进一步划分样本

递归停止的判断函数如下所示：

```

def end_condition(result_list, max_depth, current_depth):
    """
    递归判定的结束条件
    :param result_list: 结果列表
    :param max_depth: 最大深度
    :param current_depth: 当前深度
    :return: true表示结束，false表示结束
    """
    if current_depth > max_depth:
        return True
    else:
        result = collections.Counter()
        result.update(result_list)
        return len(result) == 1

```

递归建立决策树的过程位于 `build_tree()` 函数，根据输出的要求，采用DFS的方式递归建立决策树。

```

def build_tree(self, node: DecisionNode):
    """
    递归建立决策树结构
    :param node: 树的节点
    :return:
    """
    if end_condition(node.labels, node.max_depth, node.depth):
        node.results = self.majorlabel(node.labels)
        self.count_label(node.labels, node.label_set)
        # print("为叶子节点，结果为： {}".format(node.results))
        return
    best_index = choose_best_feature(node.data_set, node.labels,
node.has_calc_index)
    node.col = best_index
    # print("best_index =", best_index)

    self.count_label(node.labels, node.label_set)

    #根据信息增益的最大进行划分 采用DFS的方法

```

```

#生成左子树:
for i in range(node.depth):
    print("\t", end="")
    print("{} = {}: ".format(node.attr_lst[node.col], node.table[node.col]
[1]), end="")

    tb_index = [i for i, value in enumerate(node.data_set) if value[best_index]]
    tb_data_set = [node.data_set[x] for x in tb_index]
    tb_data_labels = [node.labels[x] for x in tb_index]
    tb_table = node.table
    tb_node = DecisionNode(data_set=tb_data_set, labels=tb_data_labels,
table=tb_table, attr_lst=node.attr_lst, labelset=node.label_set)
    tb_node.has_calc_index = list(node.has_calc_index)
    tb_node.has_calc_index.append(best_index)
    tb_node.depth = node.depth + 1
    tb_node.max_depth = node.max_depth
    node.tb = tb_node

#生成右子树:
# print("右子树: ")
fb_index = [i for i, value in enumerate(node.data_set) if not
value[best_index]]
fb_data_set = [node.data_set[x] for x in fb_index]
fb_data_labels = [node.labels[x] for x in fb_index]
fb_table = node.table
fb_node = DecisionNode(data_set=fb_data_set, labels=fb_data_labels,
table=fb_table, attr_lst=node.attr_lst, labelset=node.label_set)
fb_node.has_calc_index = list(node.has_calc_index)
fb_node.has_calc_index.append(best_index)
fb_node.depth = node.depth + 1
fb_node.max_depth = node.max_depth
node.fb = fb_node

if tb_index:
    # print("-----建立左子树-----")
    self.build_tree(node.tb)
else:
    self.count_label(node.tb.labels, node.tb.label_set)

for i in range(node.depth):
    print("\t", end="")
    print("{} = {}: ".format(node.attr_lst[node.col], node.table[node.col]
[0]), end="")

    if fb_index:
        # print("-----建立右子树-----")
        self.build_tree(node.fb)
    else:

```



```
self.count_label(node.fb.labels, node.fb.label_set)
```

左右子树的节点的深度均为父亲节点的深度加1，左右节点的待检测数据和已经计算过的特征索引序列均进行了相应的筛选。左右子树节点中某些数据集中共同的信息直接继承自父节点的信息。

3.2.5 使用决策树预测测试集的结果

使用决策树预测测试集中的信息，使用已经构建好的决策树 `predict()` 函数，对测试集中的数据进行相应的预测。预测结果即从决策树的根节点开始，根据判断的信息，逐步走向叶子节点，并输出叶子节点的预测结果。判断该节点是否是叶子节点的方式时判断该节点的 `result` 是否为空，如果为空说明该节点为内部节点，如果不为空，说明该节点为叶子节点。

```
def _predict(self, data_test, node):  
    """  
    对单个的数据进行预测  
    :param data_test: 单个数据  
    :param node: 节点  
    :return: 预测单个的结果  
    """  
    if node.results:  
        return node.results  
    col = node.col  
  
    if data_test[col]:  
        return self._predict(data_test, node.tb)  
    else:  
        return self._predict(data_test, node.fb)  
  
def predict(self, data_test_set):  
    """  
    对整个数据集进行预测  
    :param data_test_set: 所有数据  
    :return: 预测结果  
    """  
    test_ans = []  
    for data_test in data_test_set:  
        test_ans.append(self._predict(data_test, self.root))  
    return test_ans
```

3.3 程序运行说明

该程序运行使用命令行进行调用，`decisionTree.py` 程序应该接受命令行参数，命令行参数如下所示：

命令行参数	参数的含义
train_input	输入相应训练数据的路径，例如：small_train.tsv
test_input	输入相应测试数据的路径，例如：small_test.tsv
max_depth	生成决策树的最大深度，例如：2
train_out	将决策树对训练数据的预测写入标签文件的路径，例如：small_2_train.labels
test_out	将决策树对测试数据的预测写入标签文件的路径，例如：small_2_test.labels
metrics_out	将训练集和测试集分类的错误指标写入文件的路径，例如：small_2_metrics.txt

示例：`python decisionTree.py politicians_train.tsv politicians_test.tsv 2`

`pol_2_train.labels pol_2_test.labels pol_2_metrics.txt`

3.4 程序运行结果展示

3.4.1 对数据集politicians进行训练和预测

输出结果如下所示：

```
D:\张驰\学习\机器学习初步\大作业\HW\handout>python decisionTree.py politicians_train.t
sv politicians_test.tsv 3 pol_3_train.labels pol_3_test.labels pol_3_metrics.txt
[83 democrat/66 republican]
    Superfund_right_to_sue = y: [28 democrat/64 republican]
        Aid_to_nicaraguan_contras = n: [13 democrat/58 republican]
            Export_south_africa = y: [13 democrat/38 republican]
            Export_south_africa = n: [0 democrat/20 republican]
        Aid_to_nicaraguan_contras = y: [15 democrat/6 republican]
            Mx_missile = n: [12 democrat/0 republican]
            Mx_missile = y: [3 democrat/6 republican]
    Superfund_right_to_sue = n: [55 democrat/2 republican]
        Export_south_africa = y: [55 democrat/1 republican]
            Immigration = y: [9 democrat/1 republican]
            Immigration = n: [46 democrat/0 republican]
        Export_south_africa = n: [0 democrat/1 republican]
```

对于训练集和测试集的预测标签文件：pol_3_train.labels，pol_3_test.labels不再进行展示，pol_3_metrics.txt结果如下所示：

 pol_3_metrics.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

error(train): 0.11409395973154363

error(test): 0.1686746987951807

3.4.2 对数据集small进行训练和预测

输出结果如下所示：

```
D:\张驰\学习\机器学习初步\大作业\HW\handout>python decisionTree.py small_train.tsv sma
ll_test.tsv 2 small_2_train.labels small_2_test.labels small_2_metrics.txt
[15 democrat/13 republican]
    Anti_satellite_test_ban = n: [2 democrat/12 republican]
        Export_south_africa = y: [2 democrat/7 republican]
        Export_south_africa = n: [0 democrat/5 republican]
    Anti_satellite_test_ban = y: [13 democrat/1 republican]
        Export_south_africa = y: [13 democrat/0 republican]
        Export_south_africa = n: [0 democrat/1 republican]
```

对于训练集和测试集的预测标签文件：small_2_train.labels，small_2_test.labels不再进行展示，small_2_metrics.txt结果如下所示：

```
small_2_metrics.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
error(train): 0.07142857142857142
error(test): 0.14285714285714285
```

3.4.3 对数据集education进行训练和预测

```
D:\张驰\学习\机器学习初步\大作业\HW\handout>python decisionTree.py education_train.tsv
education_test.tsv 3 edu_3_train.labels edu_3_test.labels edu_3_metrics.txt
[135 A/65 notA]
  F = A: [119 A/23 notA]
    M4 = notA: [63 A/21 notA]
      M2 = notA: [26 A/18 notA]
        M2 = A: [37 A/3 notA]
      M4 = A: [56 A/2 notA]
        P1 = A: [41 A/0 notA]
        P1 = notA: [15 A/2 notA]
    F = notA: [16 A/42 notA]
      M2 = notA: [3 A/27 notA]
        M4 = notA: [0 A/22 notA]
        M4 = A: [3 A/5 notA]
      M2 = A: [13 A/15 notA]
        M4 = notA: [7 A/14 notA]
        M4 = A: [6 A/1 notA]
```


对于训练集和测试集的预测标签文件不再进行展示，edu_3_metrics.txt结果如下所示：

```
edu_3_metrics.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)
error(train): 0.17
error(test): 0.205
```

3.4.4 对数据集mushroom进行训练和预测

```
D:\张驰\学习\机器学习初步\大作业\HW\handout>python decisionTree.py mushroom_train.tsv
mushroom_test.tsv 4 mushroom_4_train.labels mushroom_4_test.labels mushroom_4_metrics.
txt
[3680 0/2320 1]
  odor_foul = 0: [3680 0/688 1]
    gill-size_broad = 0: [288 0/616 1]
      odor_none = 0: [96 0/576 1]
        gill-spacing_close = 1: [0 0/480 1]
        gill-spacing_close = 0: [96 0/96 1]
      odor_none = 1: [192 0/40 1]
        stalk-surface-above-ring_silky = 0: [192 0/8 1]
        stalk-surface-above-ring_silky = 1: [0 0/32 1]
    gill-size_broad = 1: [3392 0/72 1]
      spore-print-color_green = 0: [3392 0/0 1]
      spore-print-color_green = 1: [0 0/72 1]
  odor_foul = 1: [0 0/1632 1]
```

对于训练集和测试集的预测标签文件不再进行展示，mushroom_4_metrics.txt结果如下所示：

 mushroom_4_metrics.txt - 记事本

文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

error(train): 0.017333333333333333

error(test): 0.02071563088512241

4 实验心得体会

在本次实验中，我手动通过python实现了二分类决策树的构建，并在老师给定的数据集上完美运行并输出了相应的决策树、训练集预测、测试集预测以及预测结果的错误率。在本次决策树实验中，通过建立决策树模型，对新给定的数据集进行分析训练，从而对测试集的相关数据进行预测。

机器学习是对大量数据进行分析，寻找统计规律，建模，并使用模型对新数据进行预测和分析的方法。通过本次手动构建决策树的过程，我进一步深入理解了决策树的相关知识，对机器学习有了进一步深入的理解。