School of Data and Computer Science, Sun Yat-sen University

E11 Decision Tree

17341189 姚森舰

2019 年 11 月 24 日

目录

1 Datasets

The UCI dataset (http://archive.ics.uci.edu/ml/index.php) is the most widely used dataset for machine learning. If you are interested in other datasets in other areas, you can refer to https://www.zhihu.com/question/63383992/answer/222718972.

Today's experiment is conducted with the **Adult Data Set** which can be found in http://archive.ics.uci.edu/ml/datasets/Adult.

Data Set Characteristics:	Multivariate	Number of Instances:	48842	Area:	Social
Attribute Characteristics:	Categorical, Integer	Number of Attributes:	14	Date Donated	1996-05-01
Associated Tasks:	Classification	Missing Values?	Yes	Number of Web Hits:	1305515

You can also find 3 related files in the current folder, `adult.name` is the description of **Adult Data Set**, `adult.data` is the training set, and `adult.test` is the testing set. There are 14 attributes in this dataset:

>50K, <=50K.

```
1. age: continuous.
2. workclass: Private, Self-emp-not-inc, Self-emp-inc, Federal-gov, Local-gov,
State-gov, Without-pay, Never-worked.
3. fnlwgt: continuous.
4. education: Bachelors, Some-college, 11th, HS-grad, Prof-school, Assoc-acdm,
Assoc-voc, 9th, 7th-8th, 12th, Masters, 5. 1st-4th, 10th, Doctorate, 5th-6th,
Preschool.
5. education-num: continuous.
6. marital-status: Married-civ-spouse, Divorced, Never-married, Separated,
Widowed, Married-spouse-absent, Married-AF-spouse.
7. occupation: Tech-support, Craft-repair, Other-service, Sales,
Exec-managerial, Prof-specialty, Handlers-cleaners, Machine-op-inspct,
Adm-clerical,Farming-fishing,Transport-moving,Priv-house-serv,Protective-serv,
Armed-Forces.
8. relationship: Wife,Own-child,Husband,Not-in-family,Other-relative,Unmarried.
9. race: White, Asian-Pac-Islander, Amer-Indian-Eskimo, Other, Black.
10. sex: Female, Male.
11. capital-gain: continuous.
12. capital-loss: continuous.
13. hours-per-week: continuous.
14. native-country: United-States, Cambodia,England,Puerto-Rico,Canada,Germany,
Outlying-US(Guam-USVI-etc),India,Japan,Greece, South,China,Cuba,Iran,Honduras,
```

Philippines, Italy, Poland, Jamaica, Vietnam, Mexico, Portugal, Ireland, France, Dominican−Republic, Laos, Ecuador, Taiwan, Haiti, Columbia, Hungary, Guatemala, Nicaragua, Scotland, Thailand, Yugoslavia, El−Salvador, Trinadad&Tobago, Peru, Hong, Holand−Netherlands.

Prediction task is to determine whether a person makes over 50K a year.

2 Decision Tree

2.1 ID3

ID3 (Iterative Dichotomiser 3) was developed in 1986 by Ross Quinlan. The algorithm creates a multiway tree, finding for each node (i.e. in a greedy manner) the categorical feature that will yield the largest information gain for categorical targets. Trees are grown to their maximum size and then a pruning step is usually applied to improve the ability of the tree to generalise to unseen data.

ID3 Algorithm:

1. Begins with the original set S as the root node.
2. Calculate the entropy of every attribute a of the data set S.
3. Partition the set S into subsets using the attribute for which the resulting entropy after splitting is minimized; or, equivalently, information gain is maximum.
4. Make a decision tree node containing that attribute.
5. Recur on subsets using remaining attributes.

Recursion on a subset may stop in one of these cases:

- every element in the subset belongs to the same class; in which case the node is turned into a leaf node and labelled with the class of the examples.
- there are no more attributes to be selected, but the examples still do not belong to the same class. In this case, the node is made a leaf node and labelled with the most common class of the examples in the subset.
- there are no examples in the subset, which happens when no example in the parent set was found to match a specific value of the selected attribute.

ID3 shortcomings:

- ID3 does not guarantee an optimal solution.
- ID3 can overfit the training data.
- ID3 is harder to use on continuous data.

Entropy:

Entropy $H(S)$ is a measure of the amount of uncertainty in the set S.

$$H(S) = \sum_{x \in X} -p(x) \log_2 p(x)$$

where

- S is the current dataset for which entropy is being calculated
- X is the set of classes in S
- $p(x)$ is the proportion of the number of elements in class x to the number of elements in set S.

Information gain:

Information gain $IG(A)$ is the measure of the difference in entropy from before to after the set S is split on an attribute A. In other words, how much uncertainty in S was reduced after splitting set S on attribute A.

$$IG(S, A) = H(S) - \sum_{t \in T} p(t)H(t) = H(S) - H(S \mid A)$$

where

- $H(S)$ is the entropy of set S
- T is the subsets created from splitting set S by attribute A such that $S = \cup_{t \in T} t$
- $p(t)$ is the proportion of the number of elements in t to the number of elements in set S
- $H(t)$ is the entropy of subset t.

2.2 C4.5 and CART

C4.5 is the successor to ID3 and removed the restriction that features must be categorical by dynamically defining a discrete attribute (based on numerical variables) that partitions the continuous attribute value into a discrete set of intervals. C4.5 converts the trained trees (i.e. the output of the ID3 algorithm) into sets of if-then rules. These accuracy of each rule is then evaluated to determine the order in which they should be applied. Pruning is done by removing a rule’s precondition if the accuracy of the rule improves without it.

C5.0 is Quinlan’s latest version release under a proprietary license. It uses less memory and builds smaller rulesets than C4.5 while being more accurate.

CART (Classification and Regression Trees) is very similar to C4.5, but it differs in that it supports numerical target variables (regression) and does not compute rule sets. CART constructs binary trees using the feature and threshold that yield the largest information gain at each node.

3 Tasks

- Given the training dataset `adult.data` and the testing dataset `adult.test`, please accomplish the prediction task to determine whether a person makes over 50K a year in `adult.test` by using ID3 (or C4.5, CART) algorithm (C++ or Python), and compute the accuracy.
    1. You can process the continuous data with **bi-partition** method.
    2. You can use prepruning or postpruning to avoid the overfitting problem.
    3. You can assign probability weights to solve the missing attributes (data) problem.
- Please finish the experimental report named `E11_YourNumber.pdf`, and send it to `ai_201901@foxmail.com`

4 Codes and Results

4.1 Codes

```python
import math
from time import *


class Node(object):
    """ 决策树的节点 """
    def __init__(self, label=None, attribute=None, branches=None):
        self.attribute = attribute       # 当前节点划分的属性标签
        self.label = label               # 保存的是针对当前分支的类别划分结果，叶节点的该属性才重要
        self.branches = branches         # 分支的字典


def load_data(filename):
    """
    加载训练和测试数据集，并做一些初步的处理，删除掉无关的属性
    :param filename: str
    :return: 数据集列表
    """
    with open(filename, 'r') as f:
        dataset = []
        for line in f.readlines()[:-1]:
            record = line.strip().split(', ')
            record[0] = int(record[0])
            record[4] = int(record[4])
            record[10] = int(record[10])
            record[11] = int(record[11])
            record[12] = int(record[12])
            # 去除最后的点
            if record[-1][-1] == '.':
                record[-1] = record[-1][:-1]
            # 删除部分属性，删除后会前移
            del(record[2])
            del(record[2])
            dataset.append(record)
        return dataset


def get_miss_attributes(dataset):
    """
    返回数据集中有缺失值的属性，以及对应缺失次数
    :param dataset:
```

```python
    :return: dict 键值是缺失属性的序号
    """
    miss = {}
    for record in dataset:
        for i in range(len(record)):
            if record[i] == '?':
                miss[i] = miss.get(i, 0) + 1
    # for data in train_data:
    #     print(data)
    # print(miss)
    return miss


def get_attributes_domain(dataset):
    """
    返回每种属性的取值范围
    :param dataset: 数据集
    :return: dict 每种属性的取值范围
    """
    num = len(dataset[0]) - 1
    attribute_domain = {}
    # 遍历属性, 遍历数据集, 记录所有取值情况
    for i in range(num):
        domain = set()
        for record in dataset:
            domain.add(record[i])
        attribute_domain[i] = domain
    return attribute_domain


def get_attributes_max_branch(dataset):
    """
    每个属性取值出现次数最多的取值, 用于填充
    :param dataset:
    :return: dict 有缺失的属性的出现最频繁的值
    """
    # 首先找到哪些属性有缺失
    miss_attributes = get_miss_attributes(dataset)
    miss_attributes_max_branches = {}
    # 遍历有缺失值的属性, 统计找出有缺失值的属性取值最频繁的值, 用于后面填充
    for i in miss_attributes.keys():
        count = {}
        for record in dataset:
            count[record[i]] = count.get(record[i], 0) + 1
        # 出现频率最高
        max_branch = max(count.items(), key=lambda x: x[1])
        miss_attributes_max_branches[i] = max_branch[0]
```

```python
    return miss_attributes_max_branches

# miss_attributes_max_branches = get_attributes_max_branch(train_data)
# print(miss_attributes_max_branches)


def precondition(dataset):
    """
    预处理，划分连续属性，以及将缺失值填充为该属性出现次数最多的取值
    :param dataset: 数据集，列表
    :return: 处理后的数据集 list， 有用属性序号列表
    """
    # 得到有缺失的属性，以及该属性出现次数最多的取值
    miss_attributes_max_branches = get_attributes_max_branch(dataset)
    # print(miss_attributes_max_branches)
    for record in dataset:
        for i in miss_attributes_max_branches.keys():
            if record[i] == '?':
                record[i] = miss_attributes_max_branches[i]

    for record in dataset:
        # 年龄划分成7类
        if record[0] <= 20:
            record[0] = 1
        if 20 < record[0] <= 24:
            record[0] = 2
        if 24 < record[0] <= 34:
            record[0] = 3
        if 34 < record[0] <= 44:
            record[0] = 4
        if 44 < record[0] <= 54:
            record[0] = 5
        if 54 < record[0] <= 64:
            record[0] = 6
        if record[0] > 64:
            record[0] = 7

        # 关于资本的一些属性，分为2类
        if record[8] != 0:
            record[8] = 1

        if record[9] != 0:
            record[9] = 1

        # 一周工作时长，分3类
        if record[10] <= 36:
            record[10] = 1
```

```python
        if 36 < record[10] <= 72:
            record[10] = 2
        if record[10] > 72:
            record[10] = 3

        if record[-1] == '<=50K':
            record[-1] = 0
        else:
            record[-1] = 1
    # 顺便返回有用属性的顺序列表, int
    attrs = []
    for i in range(len(dataset[0])-1):
        attrs.append(i)
    # for data in dataset:
    #     print(data)
    return dataset, attrs


def cal_InforEntropy(dataset):
    """
    计算当前子集的信息熵值
    :param dataset: 最后一列为标签值, 其他为属性值
    :return: 返回信息熵的结果
    """
    total = len(dataset)
    label_counts = {}           # 统计各个label的数量, 考虑可以用于多分类
    for record in dataset:
        label = record[-1]
        label_counts[label] = label_counts.get(label, 0) + 1

    InforEntropy = 0.0
    for item in label_counts.items():
        prob = float(item[1]) / total
        InforEntropy -= prob * math.log(prob, 2)
    # print(InforEntropy)
    return InforEntropy


def split_dataset(dataset, attribute, value):
    """
    根据选定的属性划分数据集
    :param dataSet:
    :param attribute: 选定属性的序号
    :param value: 该属性的取值
    :return: 在该属性上取该值的子集
    """
    # 遍历数据集, 只要在该属性上取该值的数据, 就取出来
```

```python
    branch = []
    for record in dataset:
        if record[attribute] == value:
            branch.append(record)
    return branch


def select_best_attribute(dataset, attributes_domain, remaining_attributes):
    """
    计算信息增益，选出信息增益最大的属性，返回用于划分的属性序号
    :param dataset:
    :param attributes_domain: 每种属性的取值范围
    :param remaining_attributes: 还未用于划分的属性集
    :return: 返回用于划分的属性序号
    """
    total = len(dataset)                            # 总的数据条数
    rootEntropy = cal_InforEntropy(dataset)         # 根节点信息熵
    InforGain_list = []
    for attribute in remaining_attributes:
        InforEntropy = 0.0                          # 计算按该属性划分的信息熵
        for value in attributes_domain[attribute]:
            branch = split_dataset(dataset, attribute, value)
            # 该属性取值的数据集占总数的比例
            prob = len(branch) / total
            InforEntropy += prob * cal_InforEntropy(branch)
        InforGain_list.append((rootEntropy - InforEntropy, attribute))
    max_IG = max(InforGain_list, key=lambda IG: IG[0])
    return max_IG[1]


def build_tree(dataset, parent_label, remaining_attributes, attributes_domain):
    """
    递归建立决策树
    :param dataset:
    :param parent_label: 父节点label
    :param remaining_attributes: 还未用于划分的属性集
    :param attributes_domain: 每种属性的取值范围
    :return: 决策树节点
    """
    labels = [record[-1] for record in dataset]
    # 该分支无数据，为叶节点
    if len(dataset) == 0:
        return Node(label=parent_label, attribute=None, branches=None)
    # 样本全属于同一类
    if labels.count(labels[0]) == len(labels):
        return Node(label=labels[0], attribute=None, branches=None)
    # 全部属性都分完了，叶节点，其分类为其中样本数最多的类
```

```python
    if len(remaining_attributes) == 0:
        # 出现频率最高, list.count 函数对象
        return Node(label=max(labels, key=labels.count), attribute=None, branches=None)
    # D中样本在剩余的属性集A上取值相同
    diff = False
    for attr in remaining_attributes:
        for record in dataset:
            if record[attr] != dataset[0][attr]:
                diff = True
                break
        if diff:
            break
    if not diff:
        return Node(label=max(labels, key=labels.count), attribute=None, branches=None)

    # 找到信息增益最大的属性, 并准备用它来划分数据集, 同时将该属性标记未已使用过
    best_attribute = select_best_attribute(dataset, attributes_domain,
        remaining_attributes)
    remaining_attributes.remove(best_attribute)
    branches = {}
    parent_label = max(labels, key=labels.count)
    # 把该属性每一个取值对应的数据集子集拿出来, 递归建立子树, 并记录到该节点的branches中
    for value in attributes_domain[best_attribute]:
        branch = split_dataset(dataset, best_attribute, value)
        branches[value] = build_tree(branch, parent_label, remaining_attributes[:],
            attributes_domain)

    return Node(attribute=best_attribute, label=parent_label, branches=branches)


def test(test_dataset, root):
    """
    在生成的决策树上测试, 返回正确率
    :param test_dataset:
    :param root: 之前生成的决策树根节点
    :return: 正确率
    """
    right = 0
    for record in test_dataset:
        cur = root
        # 只要有分支就不是叶节点
        while cur.branches:
            # 流向 该条数据 在 该节点的属性 的取值 对应的分支
            cur = cur.branches[record[cur.attribute]]
        if cur.label == record[-1]:
            right += 1
    return right/len(test_dataset)
```

```python

if __name__ == '__main__':
    train_data = load_data('adult.data')
    test_data = load_data('adult.test')      # 含标签

    train_data, attributes = precondition(train_data)
    test_data, a = precondition(test_data)
    attributes_domain = get_attributes_domain(train_data)
    # print(attributes)
    t1 = time()
    # 初始标签其实无所谓，此外，属性集上一开始所有属性都未用过
    root = build_tree(train_data, -1, attributes, attributes_domain)
    t2 = time()
    accuracy_train = test(train_data, root)
    accuracy_test = test(test_data, root)
    t3 = time()
    print('Accuracy on training data set: {:.4}%'.format(accuracy_train*100))
    print('Accuracy on testing data set: {:.4}%'.format(accuracy_test*100))
    print('Building decision tree time cost: {:.4}s'.format(t2 - t1))
    print('Testing time cost: {:.4}s'.format(t3 - t2))
```

4.2 Results

只实现了 ID3 的方法，最终生成的决策树在训练集上的准确率为 90.92%，在测试集上的准确率为 81.64%，预剪枝后可达 83.3% 运行结果如下：

```
D:\Anaconda\python.exe F:/桌面/人工智能/E11_20191120_DT/E11.PY
Accuracy on training data set: 90.92%
Accuracy on testing data set: 81.64%
Building decision tree time cost: 15.89s
Testing time cost: 0.1s
```

从上面也可以看到建树的时间相对来说比较长，所以可以将决策树写进文件，但是好像 16s 也不是特别长，所以就没实现了。

对于缺失值，用该属性上取值出现次数最多的值来填补，就像助教所说的，我们有理由相信缺失的值有很大概率是那个出现最频繁的取值。

对于连续值，我是自己分类的，比如年龄，我是找了一个合理的年龄段划分，因为年龄段和工资的关系是比较明显的，一周工作时长也类似。对于后面有两个属性 capital-gain 和 capital-loss，是与资本相关的，同样，这两个数据不为 0 的人很大概率年薪也比较高，所以手动划分为了两类，一种是取值为 0，另一种是不为 0.

此外删除了 2 个属性，一是 education，我认为它和 education-num 是两个非常相关的属性，取

其中之一即可。还有一个是 fnlwgt，我查到这个属性有两种说法，有的说它是普查人员的员工号，有人说是被调查人员的背景的数值化，如果是前者，可以直接删掉，如果是后者，肉眼观察没法找出它与工资的大概关系，如果不删除，使用西瓜书上的做法，也就是 bi-partition 的方法，但是把这个背景（教育，家庭?）属性仅仅分为两类，是不太合理的，于是想用该方法求出划分后信息增益最大的几个分界点来划分，但是又不知道取多少个分界点合适。用二分的方法的话，准确率只能提高 2% 左右，但是运行时间就长很多，所以干脆去掉了这个属性。

对于剪枝处理，我试了下预剪枝，一是限制节点包含的最少样本数，二是限制树的深度，这两种方法还挺有效的，都能够使在测试集上的结果增加 2% 的准确率，但想再增加就很难了。如果限制节点最少包含 5 个样本，也就是将 build_tree() 函数中第一个判断改为 if len(dataset) < 5，可以得到下面的结果：

```
D:\Anaconda\python.exe F:/桌面/人工智能/E11_20191120_DT/E11.PY
Accuracy on training data set: 85.5%
Accuracy on testing data set: 83.37%
Building decision tree time cost: 16.22s
Testing time cost: 0.08677s
```

可以看到在训练集上准确率下降了，一定程度上减小过拟合，也就使得在测试集上表现更好了。限制决策树的深度只需在该函数添加 1 个 num 参数，每次递归调用时加 1，到了限制深度就返回，结果和上面类似。