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

E10 Decision Tree

18340052

November 25, 2020

Contents

1 Datasets **2**

2 Decision Tree **2**

2.1 ID3 2

2.2 C4.5 and CART 3

3 Tasks **4**

4 Codes and Results **4**

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.

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 rules precondition if the accuracy of the rule improves without it.

C5.0 is Quinlans 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 `E10_YourNumber.pdf`, and send it to `ai_2020@foxmail.com`

4 Codes and Results

Code:

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import pickle as pk
import seaborn as sns

header = ['age', 'workclass', 'fnlwgt', 'education', 'education-num',
          'marital-status', 'occupation', 'relationship', 'race', 'sex',
          'capital-gain', 'capital-loss', 'hours-per-week', 'native-
              country', 'Salaries']
train_data_path = 'adult.data'
test_data_path = 'adult.test'
train_data = pd.read_csv(train_data_path, names=header)
test_data = pd.read_csv(test_data_path, names=header)
test_data.drop(0, inplace=True)
test_data.reset_index(drop=True, inplace=True)
train_data.replace(' ?', np.nan, inplace=True)
train_data.fillna(train_data.mode().iloc[0], inplace=True)
test_data.replace(' ?', np.nan, inplace=True)
test_data.fillna(test_data.mode().iloc[0], inplace=True)
continuous_cols = ['age', 'fnlwgt', 'education-num','capital-gain', '
    capital-loss', 'hours-per-week']
pos1 = int(len(train_data)/3)
pos2 = 2 * pos1
intervals = {}

for col in continuous_cols:
    i1 = sorted(train_data[col])[pos1]
    i2 = sorted(train_data[col])[pos2]
    intervals[col] = (range(0, i1+1), range(i1+1, i2+1),range(i2+1,
        sorted(train_data[col])[len(train_data)-1]+1))

rev_intervals = {}
for k, v in intervals.items():
```

```python
    tmp = {}
    for idx, r in enumerate(v):
        for i in r:
            tmp[i] = idx
    rev_intervals[k] = tmp

dsp_dict = {
    1: ['Private', 'Self-emp-not-inc', 'Self-emp-inc', 'Federal-gov', 'Local-gov', 'State-gov', 'Without-pay', 'Never-worked'],
    3: ['Bachelors', 'Some-college', '11th', 'HS-grad', 'Prof-school', 'Assoc-acdm', 'Assoc-voc', '9th', '7th-8th', '12th', 'Masters', '1st-4th', '10th', 'Doctorate', '5th-6th', 'Preschool'],
    5: ['Married-civ-spouse', 'Divorced', 'Never-married', 'Separated', 'Widowed', 'Married-spouse-absent', 'Married-AF-spouse'],
    6: ['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'],
    7: ['Wife', 'Own-child', 'Husband', 'Not-in-family', 'Other-relative', 'Unmarried'],
    8: ['White', 'Asian-Pac-Islander', 'Amer-Indian-Eskimo', 'Other', 'Black'],
    9: ['Female', 'Male'],
    13: ['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']
}

def dsp2numlist(idx):
    return list(range(len(dsp_dict[idx])))

AttrSet = [
    (0, [0, 1, 2], 'age'),
    (1, dsp2numlist(1), 'workclass'),
    (2, [0, 1, 2], 'fnlwgt'),
    (3, dsp2numlist(3), 'education'),
    (4, [0, 1, 2], 'education-num'),
    (5, dsp2numlist(5), 'marital-status'),
    (6, dsp2numlist(6), 'occupation'),
    (7, dsp2numlist(7), 'relationship'),
    (8, dsp2numlist(8), 'race'),
    (9, dsp2numlist(9), 'sex'),
    (10, [0, 1, 2], 'capital-gain'),
    (11, [0, 1, 2], 'capital-loss'),
```

```python
    (12, [0, 1, 2], 'hours-per-week'),
    (13, dsp2numlist(13), 'native-country')
]

train_label = [1 if val == ' >50K' else 0 for val in train_data['Salaries']]
train_input = []

for idx in range(len(train_data)):
    tmp = [dsp_dict[i].index(val.strip()) if int(i) in dsp_dict.keys()
            else rev_intervals[train_data.columns[i]].get(val, 2) for i, val in enumerate(train_data.iloc[idx][:-1])]
    train_input.append(tmp)
test_label = [1 if val == ' >50K.' else 0 for val in test_data['Salaries']]
test_input = []
for idx in range(len(test_data)):
    tmp = [dsp_dict[i].index(val.strip()) if int(i) in dsp_dict.keys()
            else rev_intervals[test_data.columns[i]].get(val, 2) for i, val in enumerate(test_data.iloc[idx][:-1])]
    test_input.append(tmp)

def Entropy(Data):
    labels = [sample[-1] for sample in Data]
    types = set(labels)
    types_counts = [labels.count(type) for type in types]
    probs = [prob/len(Data) for prob in types_counts]
    return -np.sum(probs*np.log2(probs))

def Gain(Data, attr):
    entropy = Entropy(Data)
    attr_num = attr[0]
    attr_vals = attr[1]
    entropys = [0 for val in attr_vals]
    weights = [0 for val in attr_vals]
    for idx, val in enumerate(attr_vals):
        sub_data = []
        for sample in Data:
            if sample[attr_num] == val:
                sub_data.append(sample)
                weights[idx] += 1
        entropys[idx] = Entropy(sub_data)
        weights[idx] /= len(Data)
    return entropy - np.sum(np.multiply(weights, entropys))

def Gini(Data):
    labels = [sample[-1] for sample in Data]
    types = set(labels)
    types_counts = [labels.count(type) for type in types]
```

```python
    probs = [prob/len(Data) for prob in types_counts]
    return 1 - np.sum(np.power(probs, 2))

def Gini_index(Data, attr):
    gini = Gini(Data)
    attr_num = attr[0]
    attr_vals = attr[1]
    ginis = [0 for val in attr_vals]
    weights = [0 for val in attr_vals]
    for idx, val in enumerate(attr_vals):
        sub_data = []
        for sample in Data:
            if sample[attr_num] == val:
                sub_data.append(sample)
                weights[idx] += 1
        ginis[idx] = Gini(sub_data)
        weights[idx] /= len(Data)
    return np.sum(np.multiply(weights, ginis))

def chooseBestAttr(Data, Attrset, method='ID3'):
    best_attr = Attrset[0]
    best_gain = -1
    best_gini = np.Inf
    for attr_tuple in Attrset:
        gain = Gain(Data, attr_tuple)
        best_attr = attr_tuple if gain > best_gain else best_attr
        best_gain = gain if gain > best_gain else best_gain
    return best_attr

def splitData(Data, attr_num, attr_val):
    sub_data = []
    for sample in Data:
        if sample[attr_num] == attr_val:
            sub_data.append(sample[:attr_num] + sample[attr_num+1:])
    return sub_data

def getMajority(Data):
    labels = [sample[-1] for sample in Data]
    types = list(set(labels))
    types_counts = [labels.count(type) for type in types]
    major = 0
    max_count = 0
    for idx, type_count in enumerate(types_counts):
        major = types[idx] if max_count < type_count else major
        max_count = type_count if max_count < type_count else max_count
    return str(major)

def GenerateTree(Data, Attrset, method='ID3'):
    labels = [sample[-1] for sample in Data]
```

```
    if len(set(labels)) == 1:
        return str(labels[0])
    if len(Attrset) == 0:
        return getMajority(Data)
    flag = False
    for attr_tuple in Attrset:
        if len(set([sample[attr_tuple[0]] for sample in Data])) != 1:
            flag = True
            break
    if not flag:
        return getMajority(Data)
    best_attr = chooseBestAttr(Data, Attrset, method)
    attr_num = best_attr[0]
    attr_vals = best_attr[1]
    attr_name = best_attr[2]
    for idx, attr in enumerate(Attrset):
        if attr[0] > attr_num:
            Attrset[idx] = (attr[0]-1, attr[1], attr[2])
    del(Attrset[Attrset.index(best_attr)])
    Node = {attr_name: {}}
    for val in attr_vals:
        sub_data = splitData(Data, attr_num, val)
        if len(sub_data) == 0:
            return getMajority(Data)
        else:
            Node[attr_name][val] = GenerateTree(sub_data, Attrset[:], method)
    return Node

def Classifier(DecisionTree, AttrSet, SampleData):
    root = list(DecisionTree.keys())[0]
    for attr_tuple in AttrSet:
        if root == attr_tuple[2]:
            key = SampleData[attr_tuple[0]]
    succ = DecisionTree[root][key]
    if isinstance(succ, dict):
        return Classifier(succ, AttrSet, SampleData)
    else:
        return succ

def show_accuracy(DecisionTree, AttrSet, testing_data, log=False):
    labels = [sample[-1] for sample in testing_data]
    res = []
    for sample in testing_data:
        res.append(Classifier(DecisionTree, AttrSet, sample[:-1]))
    check = [labels[idx] + int(res[idx]) for idx in range(len(testing_data))]
    if log:
        print("Total Accuracy: %.5f" % (1-check.count(1)/len(
```

```python
            testing_data)))
    else:
        return 1 - check.count(1)/len(testing_data)

testing_attrset = [(0, [0, 1, 2, 3], '2nd'), (1, [0, 1, 2, 3],'3rd'),
    (2, [0, 1, 2, 3, 4, 5, 6], '4th')]
X_train = [data + [train_label[idx]] for idx, data in enumerate(
    train_input)]
X_test = [data + [test_label[idx]] for idx, data in enumerate(test_input
    )]
SalaryPredict_DT_ID3 = GenerateTree(X_train, AttrSet[:])
print("="*10+' Testing ID3 '+"="*10)
show_accuracy(SalaryPredict_DT_ID3, AttrSet, X_test, True)
```

Result: