Programming Assignment #2

컴퓨터소프트웨어학과 2017029470 김종원

1. Environment

- Mac OS (Monterey) M1 chip
- Python 3.10.3 (release March 16, 2022)

2. How to compile

Before compiling dt.py, python version 3 must be installed in your system.

```
[wonnx@wonnx project_2_decision_tree % python3 dt.py dt_train.txt dt_test.txt dt_result.txt
```

- Execution file name: dt.py
- Training file name: dt_train.txt, dt_train1.txt
- Test file name: dt_test.txt, dt.test1.txt
- Output file name: dt_result.txt, dt_result1.txt

3. Summary of algorithm

Decision Trees are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

- Create decision tree in a top-down recursive divided-and-conquer manner.
- At start, all the training examples are at the root.
- Examples are partitioned recursively based on the selected test attributes.
- The test attributes are selected based on a heuristic or statistic measure.
  (In this case in used information gain.)

4. Detailed description of codes

```python
if __name__ == "__main__":
    input_file = open(sys.argv[1], 'r')
    result = input_file.readline()
    attributes_name = result.strip().split('\t')
    class_name = attributes_name[-1]

    train_samples = list()
    attributes = defaultdict(set)
    class_label = set()

    while(True):
        dic = dict()
        line = input_file.readline().strip()
        if not line: break
        for idx, val in enumerate(line.split('\t')):
            dic[attributes_name[idx]] = val
            attributes[attributes_name[idx]].add(val)
        class_label.add(dic[class_name])
        train_samples.append(dic)
    input_file.close()
```

In order to train the decision tree model using the training data set given as the project specification, the samples were transferred to various types of data containers.

- **train_samples** is in the form of a list, which is a list containing training samples in the form of a dictionary.
- **attributes** are in the form of a dictionary, and the values that each attribute can has are collected in a set form. Since it is a collection of the attribute values of each sample of the training data, if the training data is not enough or have a lot of noise, the decision tree model may not be properly trained.
- **class_label** is the form of list, which contains the values that class label can have.

```python
# Algorithm for Decision Tree Induction
# use top-down recursive divide-and-conquer manner
def decision_tree(train_samples, attributes, class_label):

    # If all samples for given node belong to the same class
    # then stop the partitioning process
    class_name = list(attributes)[-1]
    labels = set([s[class_name] for s in train_samples])
    if len(labels) == 1: return list(labels)[0]

    # If there are no remaining attributes for further partitioning
    # majority voting is employed for classifying the leaf
    if len(attributes) == 1:
        class_count = dict.fromkeys(sorted(set(class_label)), 0)
        for s in train_samples: class_count[s[class_name]] += 1
        return Counter(class_count).most_common()[0][0]
```

When training the decision tree model, there are three conditions under which partitioning process should be stopped. In the code above, two out of three conditions are shown.

- All samples for given node belong to the same class.
  If all training samples have the same class label, we do not partition the decision tree anymore, and return the corresponding class label.
- There are no remaining attributes for further partitioning.
  A test attribute must be selected for partitioning, but if the attribute no longer exists, partitioning is not performed, and a class label is selected through majority voting.

```
    # Select the test attribute having the highest information gain
    dic = dict()
    total = len(train_samples)
    for A in list(attributes.keys())[:-1]:
        info_A = 0
        for val in attributes[A]:
            new_samples = [s for s in train_samples if s[A] == val]
            class_count = Counter([s[class_name] for s in new_samples])
            s_total = sum(class_count.values())
            entropy = 0
            for cnt in class_count.values():
                entropy += -(cnt/s_total) * math.log(cnt/s_total, 2)
            info_A += (s_total/total)*entropy
        dic[A] = info_A
    test_attribute = min(dic.keys(), key = lambda k : dic[k])
```

Among the attributes other than the class, the one with the highest information gain is selected as a new test attribute. In order to the information gain to be the largest, the sum of entropy of data samples divided by the corresponding attribute should be the smallest.

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- Pi is the probability that an arbitrary tuple in partitioned data samples which belongs to class.
- Obtain the entropy of each divided data set, add then all, and obtain the expected information when the samples are divided by the test attribute.

```
    # Create decision tree recursively
    subtree = dict()
    subtree[test_attribute] = dict()
    for val in attributes[test_attribute]:
        new_samples = [s for s in train_samples if s[test_attribute] == val]
        new_attributes = copy.deepcopy(attributes)
        del new_attributes[test_attribute]
        if len(new_samples) == 0:
            class_count = dict.fromkeys(sorted(set(class_label)), 0)
            for s in train_samples:
                class_count[s[class_name]] += 1
                subtree[test_attribute][val] = Counter(class_count).most_common()[0][0]
        else: subtree[test_attribute][val] = decision_tree(new_samples, new_attributes, class_label)

    return subtree
```

The decision tree is implemented by putting a dictionary in a dictionary. When the selection of the test attribute with the largest information gain is completed, the data samples are divided using the corresponding attribute and a subtree is created.

- If the size of the divided data sample is not 0, the subtree is recursively created.
- However, if there are no samples with each value of the attribute, the corresponding node is filled through majority voting, rather than recursively create a subtree.

```
    test_file = open(sys.argv[2], 'r')
    test_attributes_name = test_file.readline().strip().split('\t')
    test_samples = list()

    while(True):
        dic = dict()
        line = test_file.readline().strip()
        if not line: break
        for idx, val in enumerate(line.split('\t')):
            dic[test_attributes_name[idx]] = val
        test_samples.append(dic)
    test_file.close()

    for sample in test_samples:
        result += '\t'.join(list(sample.values())) + '\t'
        result += classify(decision_tree, sample, list(attributes.keys())[:-1]) + '\n'

    result_file = open(sys.argv[3], 'w')
    result_file.write(result)
    result_file.close()
```

Classify the class label of the test data using the decision tree learned using the training data. In the same way as when training the decision tree, the test data were divided into several data containers.

- **test_samples** is type of list, which contains test samples in the form of a dictionary.

```
# Find the leaf node recursively for the test data set
def classify(dt, sample, attributes):
    node = list(dt.keys())[0]
    next_node = sample[node]
    if isinstance(dt[node][next_node], str): return dt[node][next_node]
    else: return classify(dt[node][next_node], sample, attributes)
```

For each test sample, traverses the decision tree and classifies the class label value. The decision tree is a form that contains a dictionary in the dictionary. If the value in the dictionary is a string, it indicates a class label, and if it is a dictionary, it means that the data set is partitioned by the test attribute.

5. Testing result

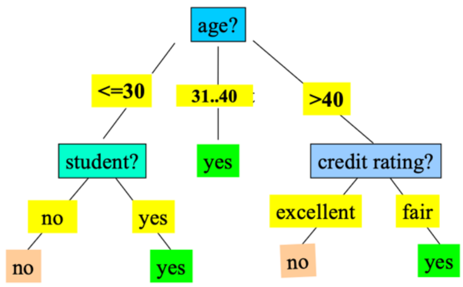
```
age income  student credit_rating   Class:buys_computer
<=30     high     no  fair     no
<=30     high     no  excellent    no
31...40 high     no  fair     yes
>40 medium  no  fair     yes
>40 low yes fair     yes
>40 low yes excellent    no
31...40 low yes excellent    yes
<=30     medium  no  fair     no
<=30     low yes fair     yes
>40 medium  yes fair     yes
<=30     medium  yes excellent    yes
31...40 medium  no  excellent    yes
31...40 high     yes fair     yes
>40 medium  no  excellent    no
```

dt_train.txt

The pircture above is the picture of training data samples. The first line shows attributes and class, and the attribute and class values of each training sample are shown from the next line. The form of the decision tree trained using this training data is as follows.

```
{'age': {'<=30': {'student': {'yes': 'yes', 'no': 'no'}}, '31...40': 'yes', '>40': {'credit_rating': {'fair': 'yes', 'excellent': 'no'}}}}
```

The decision tree is the form of dicionary in the dictionary, and if it is no longer partitioned, a class label is included. The above results are graphically shown as follows.



Using the above learned decision tree, classify the class label of the test data.

```
age income  student credit_rating
<=30     low no  fair
<=30     medium  yes fair
31...40 low no  fair
>40 high     no  fair
>40 low yes excellent
```

```
age income  student credit_rating   Class:buys_computer
<=30     low no  fair     no
<=30     medium  yes fair     yes
31...40 low no  fair     yes
>40 high     no  fair     yes
>40 low yes excellent    no
```

The test data is a sample that has values of attributes other than class as shown in the picture on the left. The result of classifying the test data using the decision tree is shown in the picture on the right.

```
C:₩Users₩labor₩data>dt_test.exe dt_result.txt dt_answer.txt
5 / 5

C:₩Users₩labor₩data>dt_test.exe dt_result1.txt dt_answer1.txt
320 / 346
```

The picture above is the result of scoring using the testing program. I did not get the best accuracy by setting the max depth of the tree, but I was able to get a good accuracy of about 92.4%.