Decision Tree

Environment

OS: Mac OS, Windows 10

Language: Python 3.9

Run with the command

```
// In the Decision Tree folder.

python dt.py [trainset_file_path] [testset_file_path] [result_file_path]

Ex) python dt.py dt_train.txt dt_test.txt dt_result.txt
```

Train set format

[attribute_name_1]\t[attribute_name_1]\t ... [attribute_name_n]\n

[attribute_1]\t[attribute_1]\t ... [attribute_n]\n

[attribute_1]\t[attribute_1]\t ... [attribute_n]\n

[attribute_1]\t[attribute_1]\t ... [attribute_n]\n

...

- Except for the first row, each row is a tuple.
- All the attributes are categorical.
- A n-th attribute is a class label that the corresponding tuple belongs to.

Ex)

```
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
...
```

Test set format

[attribute_name_1]\t[attribute_name_1]\t ... [attribute_name_n-1]\n

[attribute_1]\t[attribute_1]\t ... [attribute_n-1]\n

[attribute_1]\t[attribute_1]\t ... [attribute_n-1]\n

[attribute_1]\t[attribute_1]\t ... [attribute_n-1]\n

...

- A test set does not have a class label.

Ex)

```
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
```

Result format

[attribute_name_1]\t[attribute_name_1]\t ... [attribute_name_n]\n

[attribute_1]\t[attribute_1]\t ... [attribute_n]\n

[attribute_1]\t[attribute_1]\t ... [attribute_n]\n

[attribute_1]\t[attribute_1]\t ... [attribute_n]\n

...

- A n-th attribute is predicted class label value by the decision tree model.

Ex)

```
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
...
```

How it works

Class DecisionTreeNode

```
class DecisionTreeNode:
    def __init__(self, test_attribute_name=None, label=None):
        self.children = {}
        self.test_attribute_name = test_attribute_name
        self.label = label  #type: str
```

Making a decision tree model

1. Given the data frame, calculate entropy w.r.t. the class label.
    a. If the data frame size is 0, set the root label to None.
        i. The root label value will be changed later with the majority of class label values in the parent node.
    b. If the calculated entropy is 0, which means that every tuple has the same class label, set the current node's label to the corresponding class label value.
2. With the entropy of the class label, select a test attribute by calculating a gain ratio.
    a. Chose the gain ratio as an attribute selection measure.
        i. Tried an information gain for attribute selection measure.
        ii. It seemed that the gain ratio was better.
3. If the decision tree cannot find an appropriate test attribute, set the class label for that node with the majority of class label values in the local data frame.
4. Repeat {1} - {3} recursively for the local data frames projected with the selected test attribute values.
5. If there's no tuple for some attribute values, make the node for that value and set the label of the node with the majority of class label values in the local data frame.

```
def decision_tree(df, root):
    global column_name_value_pair
    if df.size == 0:
        root.label = None
        return root

    df_entropy = calculate_entropy(df, column_idx=-1)

    if df_entropy == 0:  # All tuples have same class label.
        root.label = df.iloc[-1, -1]
        return root

    root.test_attribute_name = select_test_attribute(df, df_entropy)
    majority = df.iloc[:, -1].mode().iat[0]

    if root.test_attribute_name is None:  # No more attributes for test attribute.
        root.label = majority
        return root

    for key, local_df in df.groupby(root.test_attribute_name):
        root.children[key] = decision_tree(local_df.drop(root.test_attribute_name, axis=1), DecisionTreeNode())
```

```
    for value in column_name_value_pair[root.test_attribute_name]:
        if root.children.get(value) is None:
            root.children[value] = DecisionTreeNode(test_attribute_name=root.test_attribute_name, label=majority)

    return root
```

Selecting a test attribute

1. Initialize a maximum to -1.
2. Calculate split info and entropy for each partition projected with the attribute value.
3. Get a delta entropy by subtracting the calculated entropy from df_entropy.
4. Divide the delta entropy with the split info to get a gain ratio.
5. If the gain ratio is greater than the current maximum, substitute it with the gain ratio.
6. Repeat {2} - {4} for every column except class label column.

```
def select_test_attribute(df, df_entropy):
    test_attribute = (None, -1)

    for column_idx in range(0, df.columns.size - 1):
        split_info = 0
        entropy = 0
        column_name = df.columns[column_idx]
        for key, local_df in df.groupby(column_name):
            probability = local_df.__len__() / df.__len__()
            split_info += -probability * np.log2(probability)
            entropy += probability * calculate_entropy(local_df, -1)
        delta_entropy = df_entropy - entropy
        gain_ratio = delta_entropy / split_info
        if test_attribute[1] < gain_ratio:
            test_attribute = (column_name, gain_ratio)

    return test_attribute[0]
```

Calculating an entropy

1. Project the data frame with the given column idx.
2. For each projected data frame, get the size of it and divide it by length of the original data frame to get a probability.
3. Calculate an entropy.

```
def calculate_entropy(df, column_idx):
    probabilities = df.groupby(df.columns[column_idx]).size().div(len(df))
    entropy = probabilities.apply(lambda x: -x * np.log2(x)).sum()
    return entropy
```

Predict a class label

- Apply traverse_tree for each tuple in test data frame.

```
def predict_class_label(test_df, root):
    label_vector = test_df.apply(lambda x: traverse_tree(root, x), axis=1)
    return label_vector

def traverse_tree(root, row):
    while root.label is None:
        root = root.children[row[root.test_attribute_name]]
    return root.label
```