

Assignment #2

Decision Tree

한양대학교 컴퓨터소프트웨어학부 2015004911 임성준
데이터사이언스(ITE4005, 12877) - 김상욱 교수님

1. 개요

Decision Tree는 의사 결정을 도와주는 구조의 일종으로, 의사 결정 규칙과 그 결과들을 tree구조로 구현한 모델이다. 훈련 데이터를 분석하여 label을 분류하는 작업을 하며 tree 모델을 구축한다. 그리고 실험 데이터를 통해 label을 분류하는 작업을 한다.

2. 목적

이 프로젝트의 목적은 주어진 training data를 가지고 데이터를 분석하여 트리 구조의 모델을 구축한다. 그리고 test data를 예측 모델을 통해 분류한다.

3. 개발환경

OS : macOS Big Sur 11.2.2
Language : Python 3.9.1
Source code editor : Visual Studio Code

4. 프로그램 구조 및 알고리즘

1. training data, test data, result data를 입력한다.
2. training data와 test data의 데이터를 저장한다.
3. training data를 분석하여 decision tree를 만든다.
4. Decision tree가 만들어지면 test data를 적용해 분류한다.
5. 적용된 결과값 df를 output file에 쓰는 것으로 프로그램을 종료한다.

5. 코드 설명

```
'''  
Node class  
Decision tree의 node 생성한다.  
'''  
class Node:  
    def __init__(self, attribute, is_leaf=False, label_name=None):  
        self.attribute = attribute  
        self.child = dict()  
        self.is_leaf = is_leaf  
        self.label_name = label_name
```

- `class Node` : attribute, child, is_leaf, label_name, 네 개의 변수를 가지고 있다.
- `__init__` : node를 초기화하여 생성하는 함수이다.

```
'''
Decision tree class
Decision tree의 구조와 이를 구현하는데 필요한 함수가 선언되어있다.
'''

class DecisionTree:
    def __init__(self, df, method):
        self.method = method
        self.label = df.columns[-1]
        self.attri_dict = {attribute: df[attribute].unique()
                           for attribute in df.columns if attribute!=self.label}
        self.root = self.create_decision_tree(df)
```

- `class DecisionTree` : df, method, label, attri_dict, root, 다섯 개의 변수를 가지고 있다.
- `__init__` : decision tree를 생성한다.

```
'''
Calculate entropy
주어진 data의 entropy를 구한다.
'''

def entropy(self, df):
    ## Info(D) = -Σ(p(i)*log2(p(i))) i:1~m (m은 class labels의 수)
    e = .0
    for _, value in df[self.label].value_counts().iteritems():
        p_value = value / len(df)
        ## log2(0)의 오류를 피하기 위해 1e-9를 더했다.
        e += -1.0 * (p_value * math.log(p_value + 1e-9, 2))
    return e

'''
Calculate information gain
주어진 data의 information gain을 구한다.
'''

def info_gain(self, df, attribute):
    ## InfoA(D) = Σ(|D(j)|/|D|*Info(D(j))) (|D|는 df의 크기)
    e = .0
    info_D = self.entropy(df)
    for a in self.attri_dict[attribute]:
        filtered_df = df[df[attribute] == a]
        info_A = self.entropy(filtered_df)
        e += info_A * (len(filtered_df) / len(df))

    return info_D - e
```

```

'''
Calculate split information
주어진 data의 split information을 구한다.
'''
def split_info(self, df, attribute):
    ## SplitInfoA(D) =  $-\sum(|D(j)|/|D| * \log_2(|D(j)|/|D|))$  (|D|는 df의 크기)
    e = .0
    for a in self.attri_dict[attribute]:
        filtered_df = df[df[attribute] == a]
        p_value = len(filtered_df) / len(df)
        value = math.log(p_value + 1e-9, 2)
        ## log2(0)의 오류를 피하기 위해 1e-9를 더했다.
        e += -1.0 * (p_value * value)

    return e

'''
Calculate gain ratio
주어진 data의 gain ratio를 구한다.
'''
def gain_ratio(self, df, attribute):
    ## GainRatio(A) = Gain(A)/SplitInfo(A)
    gain = self.info_gain(df, attribute)
    split_info = self.split_info(df, attribute)

    return gain / split_info

'''
Calculate gini
주어진 data의 gini를 구한다.
'''
def gini(self, df):
    ## gini(D) =  $1 - \sum p(j)^2$ 
    e = 1.0
    for _, value in df[self.label].value_counts().iteritems():
        p = value / len(df)
        e -= p ** 2

    return e

'''
Calculate gini index
주어진 data의 gini index를 구한다.
'''
# Calculate GiniA(D)
def gini_index(self, df, attribute, left, right):

```

```

    ## giniA(D) = |D1|gini(D1)+|D2|gini(D2) (여기서 D1은 left subset, D2는
right subset)
    ## Left subset
    left_df = df[df[attribute].isin(left)]
    left_size = len(left_df) / len(df)
    left_gini = self.gini(left_df)
    # Right subset
    right_df = df[df[attribute].isin(right)]
    right_size = len(right_df) / len(df)
    right_gini = self.gini(right_df)

    return (left_size * left_gini) + (right_size * right_gini)

```

- `entropy` : 주어진 df의 entropy를 계산한다.
- `info_gain` : 주어진 df의 information gain을 계산한다.
- `split_info` : 주어진 df의 split information을 계산한다.
- `gain_ratio` : 주어진 df의 gain ratio를 계산한다.
- `gini` : 주어진 df의 gini를 계산한다.
- `gini_index` : 주어진 df의 gini index를 계산한다.

```

...
Create decision tree
주어진 data로 decision tree를 생성한다.
...

def create_decision_tree(self, df):
    ## Data frame의 classification이 잘 되었을 때, leaf node를 return한다.
    if df[self.label].nunique() == 1:
        label_name = df[self.label].unique()[0]
        return Node(None, True, label_name)

    ## Data frame의 classification이 다 되지 않았을 때, majority voting을 진행한다.
    elif len(df.columns) == 1:
        ## majority voting
        majority_list =
df_train[self.label].value_counts().sort_values(ascending=False)
        majority = majority_list.index[0]
        return Node(None, True, majority)

    ## method에 따라 attribute를 결정한 후, node를 생성한다.
    ## method == 0 : Information gain
    if self.method == 0:
        info_dict = {attribute: self.info_gain(df, attribute)
                     for attribute in df.columns if attribute!=self.label}
        target_attri = sorted(info_dict.items(), key=lambda x: x[1],
reverse=True)[0][0]

```

```

    ## method == 1 : Gain ratio
    elif self.method == 1:
        ratio_dict = {attribute: self.gain_ratio(df, attribute)
                       for attribute in df.columns if attribute != self.label}
        target_attri = sorted(ratio_dict.items(), key=lambda x: x[1],
                              reverse=True)[0][0]

        node = Node(target_attri)

        ## data의 수가 가장 많은 label을 가져온다.
        majority_list =
df_train[self.label].value_counts().sort_values(ascending=False)
        node.label_name = majority_list.index[0]

        ## gini index를 이용하여 적절한 branch를 결정한다.
        a = self.attri_dict[node.attribute]
        gini_dict = dict()
        for i in range(1, len(a)):
            left = tuple(a[:i])
            right = tuple(a[i:])
            gini_dict[(left, right)] = self.gini_index(df, node.attribute, left,
right)
        branch = sorted(gini_dict.items(), key=lambda x: x[1], reverse=False)[0]
[0]

        for b in branch:
            filtered_df = df[df[node.attribute].isin(b)]
            if len(filtered_df) > 0:
                node.child[tuple(b)] = self.create_decision_tree(filtered_df)
            else:
                majority_list =
df_train[self.label].value_counts().sort_values(ascending=False)
                majority = majority_list.index[0]
                node.child[tuple(b)] = Node(None, True, majority)
        return node

```

- `create_decision_tree` : 주어진 df로 decision tree를 새로 생성한다.
majority voting를 통해 가장 많은 수의 label을 선택하고, pruning해주는 작업을 반복한다.
pruning한 뒤, 남은 df를 다시 classify 한다.
df가 완전히 classify 될 때까지 재귀적으로 실행된다.

```

'''
Find leaf
주어진 data의 leaf node를 찾는다.
'''
def find_leaf(self, data):
    node = self.root

```

```

a = data[node.attribute]
while not node.is_leaf:
    for child, next_node in node.child.items():
        if a in child:
            break
    node = next_node
    if node.attribute:
        a = data[node.attribute]

return node.label_name

```

- `find_leaf` : data에서 leaf node를 찾아 label_name을 반환한다.

```

'''
Classify
주어진 data를 classify 한다.
'''

def classify(self, df):
    label = [self.find_leaf(df.loc[i]) for i in range(len(df))]
    df[self.label] = label
    return df

```

- `classify` : df를 구현한 tree에 적용하여 분석한다.

```

'''
Main 함수
'''

if __name__ == "__main__":
    ## read argv
    train_file = sys.argv[1]
    test_file = sys.argv[2]
    output_file = sys.argv[3]

    ## read file
    df_train = pd.read_csv(train_file, sep="\t")
    df_test = pd.read_csv(test_file, sep="\t")

    ## Build decision tree
    dt = DecisionTree(df_train, 0)

    ## Classify data
    df = dt.classify(df_test)
    df.to_csv(output_file, index=False, sep="\t")

```

- `__main__` : 인자값을 받고, training data를 읽고, test data를 적용한 뒤, result data를 출력한다.

6. 실행 방법 및 실행 결과

```
data — -zsh — 80x24
limsungjun@Sungjuns-MacBookPro-16 data % python3 dt.py dt_train.txt dt_test.txt dt_result.txt
limsungjun@Sungjuns-MacBookPro-16 data %
```

- `python3 dt.py dt_train.txt dt_test.txt dt_result.txt`

python으로 구현했기에 terminal 상에서는 python을 실행하는 명령어 `python3`, `training file`, `test file`, `output file` 순서로 입력해서 실행한다.

dt_result.txt					
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	

제공된 dt_train.txt와 dt_test.txt로 출력된 dt_result.txt 결과값이다.