Programming Assignment #2 Report

2016024766

김서현

1. 구현한 Decision Tree 알고리즘에 대한 설명

Decision Tree는 Tree 형태로 data set을 분류하여 새로운 data set의 class label을 분류 및 예측하는 모델입니다. 구현한 코드에서는 Training data set을 이용해 decision tree를 구성하고, 만들어진 decision tree를 이용해서 test dataset의 class label을 예측합니다.

1) Decision Tree 구성.

   A. 우선 전체 dataset를 root node에 두고 root node에 대해서 아래의 과정을 진행합니다.

   B. tree의 node에서는 매번 information gain을 이용해 test attribute를 선택합니다.

   C. Decision tree를 구성하는 과정 중 information gain이 0.1 이하이거나, 노드 내의 데이터 수가 2개 이하인 경우 더 이상 branch를 내리지 않고 majority vote를 진행해서 class label을 정해줍니다. (pre-pruning)

   D. Decision tree의 branch를 내릴 수 있는 경우에는 test attribute를 기준으로 branch를 내리고 하위 노드에서 B), C)의 과정을 재귀적으로 반복합니다.

   E. Branch를 내릴 수 없는 경우(남은 attribute가 없는 경우, 해당 노드의 class label이 모두 동일한 경우)에는 class label을 지정합니다. (필요에 따라 majority vote 진행)

2) Decision tree를 이용해 Test data set 분류

   A. test data set의 데이터를 하나씩 읽으며 decision tree를 사용할 수 있도록 dictionary 형태로 변환시킨 뒤 decision tree를 사용해 class label을 분류합니다.

   B. 만일 test data가 tree에 없는 attribute나 tree에 없는 attribute value를 갖고 있는 경우에는 training data set에서의 전체 데이터에 대한 majority class label로 분류합니다.

3) 위의 과정에서 분류된 class label을 test dataset에 추가하고 output file을 출력합니다.

2. 코드에 대한 설명

1) Main

```python
if __name__ == '__main__':
    if len(sys.argv) < 4:
        print(
            "USAGE: python dt.py <training filename> <test filename> <output filename>")
        sys.exit()
    train_file = sys.argv[1]
    test_file = sys.argv[2]
    output_file = sys.argv[3]
    df = pd.read_csv(f'./data/{train_file}', sep='\t')
    dt = pd.read_csv(f'./data/{test_file}', sep='\t')
    # class label
    label = df.columns[-1]
    # class label values
    label_val = df[label].unique()
    total = df[label].count()
    # 전체 dataset에서 각 class label을 갖는 데이터 수 저장
    majority_table = df[label].value_counts(sort=True)
    major_dict = {}
    for i in range(len(majority_table)):
        major_dict[majority_table.index[i]] = majority_table[i]
    major_result = majority_vote(df, label, major_dict)
    attributes = df.columns[:-1]
```

dt.py의 main함수입니다. 우선 argument로 입력 받은 training file에서 class label과 attribute들을 읽어오고, 전체에 대한 majority class label을 계산합니다. 만일 argument가 누락되었다면 USAGE를 띄우고 프로그램을 종료합니다.

```python
# build decision tree
tree = decision_tree(df, attributes, 1, major_dict)
test_total = len(dt)
res = []
# test dataset을 한줄씩 decision tree에 넣어서 classify함
for i in range(test_total):
    test = dt.iloc[i]
    # test data를 dictionary 형태로 재구성
    test_data = {}
    for k in range(len(test)):
        test_data[test.index[k]] = test[k]
    # decision tree에 적용
    ans = classify(tree, test_data, major_result)
    res.append(ans)
# test dataset에 class label을 추가
dt.loc[:, label] = res
print(dt)
# result 출력
dt.to_csv(f'./test/{output_file}', sep="\t")
```

Training dataset을 이용해 decision tree를 구성합니다. 그 뒤에 test dataset의 data를 하나씩 읽으면서 decision tree에 적용 가능하도록 python의 dictation 형태로 변환합

니다. Test data에 대한 class label 분류가 종료되면 그 값을 배열에 저장함으로써 모든 test dataset의 class label을 저장합니다. 그 결과 배열을 test dataset에 추가하고, 이를 output file에 출력합니다.

2) Select attribute

```
# select test attribute
def select_attribute(df, attributes, label_val, info_d, label):
    selected_attr = ""
    selected_gain = 0
    for index in attributes:
        total = df[index].count()
        attributes_idx = df[index].unique()
        info = 0
        # 값이 없으면 0으로 채움
        table = df.groupby([index, label]).size().unstack(fill_value=0).stack()
        # attribute마다 information gain을 구하고 가장 큰 것을 test attribute로 설정
        for attr in attributes_idx:
            total_small = table[attr].sum()
            info_ = entropy(table, label_val, total_small, attr)
            info += (total_small/total)*info_
        information_gain = info_d - info

        if selected_gain < information_gain:
            selected_attr = index
            selected_gain = information_gain
    return selected_attr, selected_gain
```

Decision tree의 노드에서 test attribute를 선택하는 함수입니다. Attribute selection measure로 information gain을 사용하였습니다. 각 attribute마다 class label을 갖고 있는 데이터 개수를 세기 위해 pandas의 groupby()를 사용하였습니다. attribute value와 class label로 묶었는데 데이터가 없는 경우에는 개수를 0으로 설정하여 table을 채웠습니다. attribute마다 information gain을 계산하였고, 그 중 information gain이 가장 큰 attribute와 그 information gain 값을 리턴합니다.

3) Entropy

```
# calculate entropy
def entropy(table, label_val, total_small, attr):
    info = 0
    for j in label_val:
        cnt = table[attr][j]
        pi = cnt/total_small
        if pi:
            info -= pi*np.log2(pi)
    return info
```

입력 받은 attribute에 대한 Entropy를 계산하는 함수입니다.

4) Majority vote

```
# majority voting
def majority_vote(table, label, major_dict):
    majority = table[label].value_counts(sort=True)
    major_num = majority[0]
    final = majority.index[0]
    cnt = major_dict[final]
    for i in range(len(majority)):
        # majority 개수가 같은 것들이 있을 때에는 전체 데이터에서 가장 수가 적은 label 선택
        if majority[i] == major_num:
            if cnt > major_dict[majority.index[i]]:
                final = majority.index[i]
                cnt = major_dict[final]
    return final
```

입력 받은 table 내에서의 majority class label을 계산하는 함수입니다. 만일 majority 가 중복된다면 중복된 class label 중 전체 training data set 내에서 개수가 가장 적은 class label을 선택합니다. 개수가 더 많은 value로 쏠리는 information gain의 단점을 보완하기 위한 선택입니다.

5) Decision tree

```
# build decision tree
def decision_tree(table, attributes, depth, major_dict):
    label = table.columns[-1]
    label_val = table[label].unique()
    total = table[label].count()
    attr_cnt = len(attributes)
    # 해당 노드의 데이터들이 하나의 class label을 갖게된 경우
    if len(label_val) == 1:
        return label_val[0]
    # branch로 쪼갤 attribute가 없는 경우 -> majority vote
    if attr_cnt == 1:
        return majority_vote(table, label, major_dict)
    # 해당 노드에 데이터 수가 두개 이하인 경우 -> majority vote (pre-pruning)
    if total <= 2:
        return majority_vote(table, label, major_dict)
    # 해당 node의 전체 데이터 entropy 계산
    info_d = 0
    for i in table[label].value_counts():
        pi = i/total
        info_d -= pi*np.log2(pi)
```

Decision tree를 build하는 함수입니다. 입력으로 해당 노드에 있는 data들의 table, 아직 test attribute로 선택되지 않은 attribute들, 해당 노드의 depth, majority 계산에 필

요한 dictionary를 받습니다. 우선 종료조건 세가지를 확인합니다.

1. 만일 해당 노드의 모든 데이터들이 하나의 class label을 갖고 있으면 그 class label을 리턴하고 종료합니다.
2. 만일 해당 노드에 입력으로 들어온 attribute가 한 개 뿐이라면 majority vote를 통해 class label을 결정하고 리턴합니다.
3. 만일 해당 노드 내의 데이터 수가 2개 이하라면 overfitting을 막기 위해 majority vote를 하고 branch를 내리지 않습니다. (pre-pruning)

위의 경우가 모두 아니라면 information gain을 계산하기 위해 필요한 전체 데이터에 대한 entropy를 계산합니다.

```
# test attribute를 선택
selected_attr, selected_gain = select_attribute(
    table, attributes, label_val, info_d, label)
# information gain이 0.1보다 작고 tree의 깊이가 1보다 깊은 경우 -> majority vote (pre-pruning)
if selected_gain < 0.1 and depth > 1:
    return majority_vote(table, label, major_dict)
tree = {selected_attr: {}}
# test attribute를 제외한 나머지 attribute들
not_selected_attrs = np.delete(
    attributes, np.where(attributes == selected_attr))
# test attribute를 기준으로 branch를 내리도록 decision_tree()를 재귀적으로 호출
for i in table[selected_attr].unique():
    selected_table = table[table[selected_attr] == i]
    branch = decision_tree(
        selected_table, not_selected_attrs, depth+1, major_dict)
    tree[selected_attr][i] = branch
return(tree)
```

select_attribute() 함수를 이용해 information gain이 가장 큰 test attribute와 information gain값을 계산합니다. 여기서 한가지 종료조건을 확인합니다.

만일 information gain이 0.1보다 작은 동시에 depth가 1보다 크다면 majority vote를 한 뒤 그 결과를 리턴 합니다. (pre-pruning)

여기서 종료되지 않았다면 test attribute를 기준으로 branch를 내리는 과정을 진행합니다. Test attribute를 기준으로 데이터들을 나누고 test attribute를 제외한 나머지 attribute들과 나눠진 데이터들을 가지고 재귀적으로 decision_tree()함수를 호출합니다. 호출한 decision_tree()함수가 종료되어 tree를 리턴하면 그 리턴된 tree를 branch로 연결합니다. Tree의 구성은 python의 dictionary를 이용했습니다.

6) Classify

```python
# 생성된 tree를 이용해 test data를 classify함
def classify(tree, data, major_result):
    while True:
        # tree가 하나의 class value로만 이루어진 경우
        if type(tree) == str:
            return major_result
        attr = list(tree.keys())[0]
        if attr in data:
            if data[attr] in tree[attr]:
                # leaf node이면 그 value를 리턴
                if type(tree[attr][data[attr]]) == str:
                    return tree[attr][data[attr]]
                tree = tree[attr][data[attr]]
            # test data의 attribute value가 존재하지 않는 경우 전체 dataset의 majority vote결과를 리턴
            else:
                return major_result
        # data에 없는 attribute가 tree의 test attribute인 경우  -> 전체 data에서의 major voting값을 리턴함
        else:
            return major_result
```

Class label이 없는 데이터를 입력 받아 decision tree를 이용해 class label을 분류하는 함수입니다. 입력으로 class label을 분류할 데이터, decision tree, 전체 training data set 에서의 majority class label을 받습니다. 만일 tree가 하나의 class label만 부여하도록 구성된다면 바로 majority class label을 리턴합니다. while문으로 decision tree를 순회하면서 class label이 지정되면 그 데이터는 해당 class label로 분류합니다. 순회 도중 data에 없는 attribute가 tree의 test attribute이거나, data가 갖고 있는 attribute value가 tree에서 분류되어 있지 않다면 그 데이터는 majority class label로 분류합니다.

3. 컴파일 방법

Python 3.9.1 버전으로 작성한 프로그램입니다.
테스트는 windows10 내의 command prompt를 사용하여 진행했습니다.

1) numpy library와 pandas library를 사용하기 때문에 각 라이브러리를 설치해야 합니다.
   - pip install numpy
   - pip install pandas
2) dt.py와 data 폴더, test 폴더를 같은 디렉토리 안에 넣습니다.
3) Data 폴더 안에는 training 파일과 test 파일을 넣습니다.
4) Test 폴더 안에는 dt_test.exe 파일과 answer 파일을 넣습니다.
5) "python dt.py <training filename> <test filename> <output filename>"의 형식으로 실행시킬 수 있습니다.

   Ex) `python dt.py dt_train1.txt dt_test1.txt dt_result1.txt`

4. 실행 결과

1) dt_train.txt, dt_test.txt로 실행했을 때의 테스트 결과

```
C:\Users\yds04\Documents\4학년 1학기\데이터 사이언스\과제\assignment2\test>
dt_test.exe dt_answer.txt dt_result.txt
5 / 5
```

Accuracy: 100%

2) dt_train1.txt, dt_test1.txt로 실행했을 때의 테스트 결과

```
C:\Users\yds04\Documents\4학년 1학기\데이터 사이언스\과제\assignment2\test>
dt_test.exe dt_answer1.txt dt_result1.txt
316 / 346
```

Accuracy: 91.3%

5. 특이사항

- dt.py를 실행시키면 output file에 입력할 최종 결과물을 표준출력합니다.
- 결과 파일은 test 폴더 안에 생성되도록 하였습니다.
- dt_train.txt, dt_test.txt로 실행한 결과 파일과 dt_train1.txt, dt_test1.txt로 실행한 결과 파일을 git에 함께 제출하였습니다.