Summary algorithm

Decision tree는 dictionary 형태로 구현하였고 Key값에는 attribute를 value 값에는 자식 node를 저장하였습니다.
수업시간에 배운 Decision Tree의 여러 measure 중 Test 결과 Gini index method 가 가장 accuracy가 높아 Gini index로 선택 하였습니다.
또한 Decision Tree의 각 level 마다 기본 값인 None 을 추가해 data에 해당하는 분기가 없더라도 분류 될 수 있도록 하였습니다.
None 값의 value는 Test 결과 accuracy가 가장 높은 현재 level 중 수가 가장 많은 값으로 선정하였습니다.

Detail Each function

프로그램 실행 시 시작되는 메인 함수

```
def main():
    argv = sys.argv

    # 디버깅 용
    training_file = 'dt_train1.txt'
    test_file = 'dt_test1.txt'
    output_file = 'dt_result1.txt'

    if len(argv) != 1:
        training_file = argv[1]
        test_file = argv[2]
        output_file = argv[3]

    training = pd.read_csv(training_file, sep='\t')
    test = pd.read_csv(test_file, sep='\t')

    target_attribute = training.columns[-1]
    tree = make_decision_tree(training, training.columns[:-1], target_attribute)

    # classification = [classify_by_info(test.iloc[idx], tree) for idx in range(len(test))]
    classification = [classify_by_gini(test.iloc[idx], tree) for idx in range(len(test))]

    result = pd.DataFrame(test)
    result[target_attribute] = classification

    result.to_csv(output_file, sep='\t', index=False)
```

IDE 에서 실행 할 수 있도록 기본 변수들을 저장하였고 Terminal에서 실행 시 입력 받은 인자들을 변수에 저장하였습니다.

make_decision_tree 함수를 호출해 Training data 로 decision tree를 만들고
classify_by_gini 함수를 호출해 해당 tree를 통해 test data를 classify 합니다.

Gini를 계산하는 함수

```
# Gini Index measure 계산
def get_gini(target):
    _, count = np.unique(target, return_counts=True)
    count = count / sum(count)

    return 1 - sum(p * p for p in count)
```

정해진 attribute로 분기했을 때 gini index measure를 계산하는 함수

Input parameter

- Dataset : 현재 level의 data
- Split_atribute : 분기할 attribute
- Target_attribute : 목표 값

```
def gini_index(dataset, split_attribute, target_attribute):
    value = set(np.unique(dataset[split_attribute]))
    total = len(dataset)

    # 현재 level 의 gini 계산
    before_gini = get_gini(dataset[target_attribute])

    best_split = []
    diff_gini = 0
    # value 의 갯수에 따라 여러 가지 경우의 수를 다 split 해본다.
    # ex) value 가 4개 -> 1/3, 2/2
    for i in range(1, (len(value) // 2) + 1):
        combinations = map(set, itertools.combinations(value, i))
        for comb in combinations:
            counter = value - comb
            child_a = dataset[dataset[split_attribute].isin(comb)]
            child_b = dataset[dataset[split_attribute].isin(counter)]
            child_gini = (len(child_a) / total) * get_gini(child_a[target_attribute]) + \
                         (len(child_b) / total) * get_gini(child_b[target_attribute])

            result = before_gini - child_gini
            if result > diff_gini:
                diff_gini = result
                best_split = [tuple(comb), tuple(counter)]

    return best_split, diff_gini
```

Gini index의 경우 다음 분기를 binary로 나눠야 하기 때문에 가지고 있는 attribute 값들 중 best combination을 찾아 가장 높은 gini index 값과 해당 조합을 return 합니다.

정해진 split attribute를 따라 Child data들을 구하는 함수
Input parameter

- Dataset : 현재 level의 data들
- Split: 분기에 쓰이는 attribute
- Comb: gini index에서 분기할 때의 binary 조합

```
# 정해진 split 에 따라 child 를 구하는 함수
def split_data(dataset, split, comb=None):
    result = []

    # Information Gain / Gain Ratio
    if comb is None:
        value = np.unique(dataset[split])
        for val in value:
            result.append(dataset[dataset[split] == val])
        return result, value

    # Gini index
    else:
        for c in comb:
            result.append(dataset[dataset[split].isin(c)])

    # split attribute 을 구할 때 combination 을 구해서 value 를 return 할 필요 없다
    return result
```

현재 단계 에서 major, minor 값을 찾는 함수들

```
# 현재 단계의 target 중 가장 많은 값을 찾는 함수
def find_major(classification, count):
    major = 0
    result = classification[0]
    for cnt, target in zip(count, classification):
        if major < cnt:
            result = target
            major = cnt
    return result

# 현재 단계의 target 중 가장 적은 값을 찾는 함수
def find_minor(classification, count):
    minor = math.inf
    result = classification[0]
    for cnt, target in zip(count, classification):
        if minor >= cnt:
            result = target
            minor = cnt
    return result
```

Decision tree 를 재귀적으로 만드는 함수
Input Parameter

- Dataset : 분류할 data
- Split_attributes : 현재 level에서 분류할 수 있는 attribute들
- Target_attribute : 목표 값 ex) car evaluation

```
def make_decision_tree(dataset, split_attributes, target_attribute):
    classification, count = np.unique(dataset[target_attribute], return_counts=True)

    # target attribute 가 1개로 나와서 모두 분류된 경우
    if len(classification) == 1:
        # print('classify clear')
        return classification[0]

    major = find_major(classification, count)
    minor = find_minor(classification, count)

    # 만약 더이상 분기할 attribute 가 없으면 target_attribute 중 가장 major 한 값을 return 한다
    # 일종의 예외 처리
    if len(split_attributes) == 0:
        # print('no split attributes')
        return major
        # return minor
```

가장 먼저 현재 level 의 target_attribute의 가장 많은 값(major)과 가장 적은 값(minor)를 구합니다.

만약 더 이상 분기할 attributes가 없으면 major 값을 return합니다
→ 다만 주어진 test data의 경우 이런 case는 없었습니다.

```
    split = ''
    gain = 0
    best_comb = []
    for attr in split_attributes:
        # info = information_gain(dataset, attr, target_attribute)
        # info = gain_ratio(dataset, attr, target_attribute)
        comb, info = gini_index(dataset, attr, target_attribute)
        if gain < info:
            gain = info
            split = attr
            best_comb = comb
```

현재 level에서 가장 분류하기 가장 적당한 attribute를 gini index 방식으로 찾습니다.

```
    # Gini Index
    children = split_data(dataset, split, best_comb)
    tree = {}
    for child, comb in zip(children, best_comb):
        next_split_attributes = split_attributes
        # attribute 의 value 가 무조건 binary 형태, 따라서 현재 나눈 attribute 가 child tree 에도 있을 수 있다
        if len(comb) == 1:
            next_split_attributes = next_split_attributes.drop(split)

        tree[comb] = make_decision_tree(child, next_split_attributes, target_attribute)
        tree[None] = major
        # tree[None] = minor

    return {split: tree}
    # Gini Index End
```

앞에서 찾은 split attribute와 해당 조합으로 자식 node를 생성합니다.
Gini index에서는 binary로 나눠야 하기 때문에 attribute 값들이 여러가지 일 수 있습니다.
따라서 현재 level에서 분기한 attribute가 자식 level들에서도 쓰일 수 있기 때문에 해당 조합이 1인 경우에만 해당 attribute를 제외시킵니다.

subtree 재귀적으로 만든 뒤 Default 값인 None Value를 추가합니다.
마지막으로 현재 level에서의 split atrribute와 subtree를 return 합니다.

만든 decision tree를 통해 test data들을 classify 하는 함수

```
def classify_by_gini(item, tree):
    if type(tree) != dict:
        return tree

    attr = list(tree.keys())[0]
    child = tree[attr]
    feature = item[attr]

    for key in child.keys():
        if key is not None:
            if feature in key:
                feature = key
                break
    if child.get(feature) is None:
        subtree = child[None]
    else:
        subtree = child[feature]

    return classify_by_gini(item, subtree)
```

만약 입력받은 tree가 dict 형태가 아닌 경우 leaf node 이므로 그 값을 반환합니다
Attr : 현재 level에서 분기에 쓰이는 attribute
Child: 해당 attribute로 가는 subtree
Feature: test data에서 주어진 값

Key 가 tuple일 수 있기 때문에 주어진 feature를 포함되는 key 값으로 변경
만약 feature 값이 child에 없다면 default 값을 있다면 해당 subtree를 이용해 함수를 재귀적으로 호출

Instruction for compile

제출한 source code는 .py 형태이므로 python3 명령어를 이용해 실행하면 됩니다.
Ex) python3 dt.py dt_train.txt dt_test.txt dt_result.txt