project 2 report

2017030464 한현진

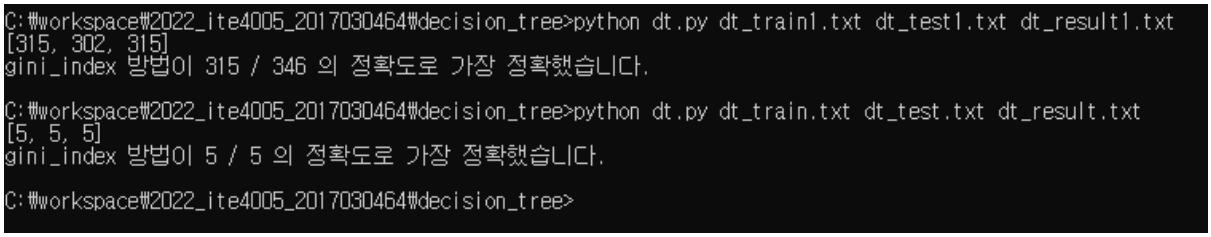
OS : Window 10 **언어버전** : Python 3.10.1

IDE : Pycharm Community Edition 2021.2.2 / 2021.3.2

Summary of your algorithm

decision tree 만들기 위한 gain 측정법으로 information_gain, gain_ratio, gini_index 방법을 다 사용했습니다. 그리고 decision_tree 구성과, test_file 마이닝 과정에서는 제 생각엔 dfs 알고리즘을 이용했습니다.

Instructions for compiling your source codes at TA's computer (e.g. screenshot) (Important!!)



파이썬 파일이라서 실행폴더까지 들어가서

python dt.py 트레이닝파일 테스트파일 결과파일

로 명령어를 치시면 될 것 같습니다.

Any other specification of your implementation and testing

데이터 파일들은 data폴더에 다 넣어주시고,

트레이닝파일: dt_train.txt 테스트파일: dt_test.txt 결과파일: dt_result.txt

로 파일명을 맞춰주셨으면 좋겠습니다. .txt 앞에 모두 동일한 숫자가 붙는 것 까지는 커버했습니다.

data 폴더 안에 정답파일: dt_answer.txt 도 결과파일과 동일한 숫자가 붙는다는 가정하에 무조건 들어가 있어야합니다. 없으면 gini_index로 측정한 방법이 기본적으로 선택되어 result 파일을 만듭니다. 제가 모든 측정법을 사용해서 decision tree를 만들어서, 정확도 비교를 위해서는 필요합니다.

numpy와 pandas 라이브러리를 이용했습니다. 다운 받아주시면 감사하겠습니다.

Detailed description of your codes

gini_py, gain_ratio.py, info.py

각 파일은 동일한 함수명으로 구현되어있고, 세부적으로만 다르기 때문에 공통된 부분은 묶고, 다른부분만 따로 작성하겠습니다.

- calc 함수 (gini.py)

```
def calc(df):
    data_size = len(df.values)
    dic = {}
    for data in df.values:
        if data[len(data) - 1] in dic:
            dic[data[len(data) - 1]] += 1
        else:
            dic[data[len(data) - 1]] = 1

    value = 1
    for i in range(len(dic)):
        p = list(dic.values())[i] / data_size
        p_squared = p ** 2
        value -= p_squared

    return value
```

인자로 df, pandas 에 의해 나온 data_frame 을 받아옵니다.
인자로 받아온 데이터프레임을 for 문을 돌면서, class_value 의 종류와 개수를 dic 변수(딕셔너리 자료형)로 파악하고,
두번째 for 문에서 gini_index 의 측정 계산식으로 측정해서 return 해줍니다.

- calc 함수 (gain_ratio.py, info.py)

```
value = 0
for i in range(len(dic)):
    p = list(dic.values())[i] / data_size
    log_2 = math.log(p, 2)
    value -= p * log_2
```

계산 식 부분은 gain_ratio 와 information_gain 의 방법이 다르지 않기 때문에 동일합니다.
math 를 사용하기 위해 math 라이브러리를 import 해줬습니다.
data_size 는 데이터프레임의 크기입니다.
p 는 각 class_value 가 어느정도 있는지 확률입니다.
p_squared 는 제곱, log_2 는 log 밑 2 P 값입니다.

- find_test_attribute 함수 (gini.py, info.py)

```
def find_test_attribute(df):
    attributes = df.columns
    data_set = df.values
    data_size = len(data_set)
    compare_value_list = []
    compare_df_list_list = []
    for i in range(len(attributes) - 1):

        attribute_values = []
        for value in df.values:
            if value[i] not in attribute_values:
                attribute_values.append(value[i])

        compare_df_list = []
        for value in attribute_values:
            compare_df_list.append(df.loc[df[attributes[i]] == value, :])

        calc_result = 0
        for compare_df in compare_df_list:
            calc_result += (len(compare_df.values) / data_size) * calc(compare_df)

        compare_value_list.append(calc_result)
        compare_df_list_list.append(compare_df_list)

    minindex = np.argmin(compare_value_list)

    # print(compare_value_list)
    # print(attributes[minindex])
    # print(compare_df_list_list[minindex])

    return attributes[minindex], compare_df_list_list[minindex]
```

우선 이론에 의하면 gain 이라 함은 보통 부모노드쪽에서 즉, 분류하기 전에 information_gain 이나 gini_index 를 구하고, 각 attribute 들로 분류하여 또 각 info_gain, gini_index 를 구해서 부모의 것과 빼서 가장 큰 것을 고르는 것으로 배웠지만 어차피 부모노드의 각 측정값은 다 똑같을 것이기에 그냥 자식노드 측정값중에 가장 큰 것을 선택하여 해당 attribute 를 test_attribute 로 선택하게 했습니다.

인자로 df, pandas 에 의해 나온 data_frame 을 받아옵니다.
attributes 는 데이터프레임의 attribute 리스트입니다.
data_set 은 그냥 보기 편해서 만들었습니다. 데이터프레임은 한줄씩 비교하거나하는게 아직 제가 익숙하지 못해서 그런지 어렵더라구요
data_size 는 데이터 개수입니다.

compare_value_list 는 가장 작은 값을 찾아야되니까 그 값들을 임시로 넣어두는 것입니다.
compare_df_list_list 는 compare_df_list 를 저장하기 위한 리스트입니다.

attribute 의 종류만큼만 확인하는데, class_label 은 test_attribute 가 될 수 없으니 attribute 의 종류 수 – 1 개 만큼 range 를 잡고 for 문을 돌려줍니다.

각 attribute 별로 for 문이 돌아가는데, 각 attribute 에 어떤 attribute_value 가 있을 지 모르니까, attribute_values 라는 리스트를 만들어서 모든 데이터를 확인하여 value 를 파악합니다

후에 compare_df_list 라는 리스트를 만들어주는데, 이 리스트는 현재 선택된 attribute 를 기준으로 나눴을 때 나눠지는 data frame 들을 저장하는 리스트입니다.

compare_df_list 를 다 만들었다면 모든 compare_df 의 측정값을 계산하여, 이론시간에 배운 자식노드들의 측정값을 계산하는 방법으로 값을 계산해줍니다.
이 값들을 다 compare_value_list 에 넣고 가장 바깥쪽 포문을 빠져나왔을 땐,

각 attribute 로 나눴을 때 측정된 측정값과, compare_df_list 가 담긴 list 가 완성되었을 테니, 측정값들 중 가장 작은 것의 인덱스를 알아내서 어떤 attribute 로 나눈 것인지, 해당 attribute 로 나눴을 때 나눠지는 데이터 프레임들은 무엇인지 return 해줍니다.

- find_test_attribute 함수 (gain_ratio.py)

```
calc_result = 0
split_info = 0
for compare_df in compare_df_list:
    calc_result += (len(compare_df.values) / data_size) * calc(compare_df)
    p = len(compare_df.values) / data_size
    log_2 = math.log(p, 2)
    split_info -= p * log_2
compare_value_list.append(calc_result / split_info)
compare_df_list_list.append(compare_df_list)
```

gain_ratio 는 따로 또 splitinfo 도 계산해줘야해서 이론에 나온 식대로 썼습니다.

node_class.py (Node 클래스)

decision tree 를 형성하고, 테스트 데이터의 class value 를 찾아주는 파일입니다.

```
def __init__(self, test_attribute=None, cf_attribute_value=None, class_value=None):
    # 뭘로 나뉠 건가
    self.test_attribute = test_attribute
    # 나뉘고 난 뒤 해당하는 값이 뭔가
    self.cf_attribute_value = cf_attribute_value
    # 클래스 레이블의 값이 뭔가
    self.class_value = class_value
    # 자식노드정보
    self.childnodes = []
```

각 노드는

test_attribute: 어떤 attribute 로 나뉠 건지

cf_attribute_value: child 노드라면, 어떤 부모의 attribute_value 로 나뉘어서 어떤 attribute_value 를 가졌는지

class_value: 해당 레벨의 노드에선 class_value 는 어떤 값을 갖는지

childnodes: child 노드가 있다면 그 노드는 무엇인지 담는 리스트

변수를 지니고 있습니다.

- make_tree 함수

```
def make_tree(self, df, cf_attribute_value, measure):
    self.cf_attribute_value = cf_attribute_value
    calc_value = 0
    if measure == 'gini':
        calc_value = gini.calc(df)
    elif measure == 'gain_ratio':
        calc_value = gain_ratio.calc(df)
    elif measure == 'info':
        calc_value = info.calc(df)
    dic = {}
    for data in df.values:
        if data[len(data) - 1] in dic:
            dic[data[len(data) - 1]] += 1
        else:
            dic[data[len(data) - 1]] = 1

    maxindex = np.argmax(list(dic.values()))
    self.class_value = list(dic.keys())[maxindex]
    if calc_value == 0 or len(df.columns) == 2:
        return self
    test_attribute = ''
    cf_df_list = []
    if measure == 'gini':
        test_attribute, cf_df_list = gini.find_test_attribute(df)
    elif measure == 'gain_ratio':
        test_attribute, cf_df_list = gain_ratio.find_test_attribute(df)
    elif measure == 'info':
        test_attribute, cf_df_list = info.find_test_attribute(df)
    self.test_attribute = test_attribute

    for cf_df in cf_df_list:
        cf_df_cf_attribute_value = cf_df[test_attribute].values[0]
        del cf_df[test_attribute]
        child_node = Node()
        child_node = child_node.make_tree(cf_df, cf_df_cf_attribute_value, measure)
        self.childnodes.append(child_node)
    return self
```

인자로 데이터 프레임과, 부모 노드의 test_attribute로부터 분류되어 나온 test_attribute의 value값, 어떤 측정방법을 썼는지 measure를 받습니다.

cf_attribute_value는 인자로 받은걸 그대로 받는데, root노드는 부모가 없어서 dt.py에서 불러올 때, None 입니다. 나머지는 코드 보면 아시겠지만 재귀함수로 make_tree를 만든거라서, 값이 제대로 들어가서 받아질 것입니다.

calc_value는 각 방법대로 calc함수로 계산해주면 됩니다.
간혹, test 데이터들 중에, decision tree의 트리대로 분류대로 가다가 없는 데이터도 있을 때가 있어서, 노드들마다, 그 때의 df의 class_value가 가장 많은 것을 class_value로 정해줍니다.

만약 calc_value가 0이거나 (아예 한쪽에 몰린경우) df 의 column의 수가 2인경우(데이터 프레임의 수가 2개라는 것은 attribute와 class label 로 두개라는 것이므로 더 이상 분류 불가능) 자신을 return 해줍니다.

하지만 그것이 아니라면 더 분류할 가능성이 있다는 것이므로, 분류를 해주는데,
이 때 어떤 test_attribute를 선택할 것인지 각 방법의 find_test_attribute 함수를 이용해서 test_attribute와 해당 attribute로 나뉘어진 데이터프레임을 알아냅니다.

나뉘어진 데이터프레임을 cf_df_list로 받는데 이 데이터프레임을 이용하여 재귀적으로 make_tree 함수를 만들어서 childnode들을 채워줍니다. dfs 방식으로 만들어지는 것 같습니다. 근데 이때 cf_df는 test_attribute에 의해 나뉘어진 상태이니, 포문안의 첫번째 코드로, 각 cf_df에서의 test_attribute 의 값으로 어떤 값을 가지는지 알아내고, 다음 다음 게속 포문으로 내려갈 때, test_attribute를 column으로 게속 가지고있으면, 분류기준으로 게속 선택될 수 있으므로, 해당 column을 삭제해준채로 재귀를 합니다.

이 때, cf_df의 데이터 수 자체가 적으면 overfitting 될 것을 염려해서 개수제한을 해서 (prepruning) overfitting을 막아볼까 했는데, 그러면 train1 test1 result1 answer1 에서 결과가 나빠지기만해서 그냥 안했습니다. (데이터 개수를 1개로 제한해도 성능이 나빠지기만함)

- mining 함수

```
def mining(self, test_data, attributes, measure):
    if len(self.childnodes) == 0:
        test_data[len(test_data)-1] = self.class_value
        return test_data
    else:
        for i in range(len(attributes)):
            if self.test_attribute == attributes[i]:
                for child in self.childnodes:
                    if test_data[i] == child.cf_attribute_value:
                        if measure == 'gini':
                            test_data = child.mining(test_data, attributes, measure)
                        elif measure == 'gain_ratio':
                            test_data = child.mining(test_data, attributes, measure)
                        elif measure == 'info':
                            test_data = child.mining(test_data, attributes, measure)
                        return test_data
                test_data[len(test_data) - 1] = self.class_value
                return test_data
```

test_data: 그냥 데이터 한 개로 인자를 받아옵니다.
attributes: training, test 할 데이터의 attribute를 받아옵니다.
measure: 어떤 측정 방법을 사용했는지 알기위한 인자입니다.

그냥 트리를 dfs 방법으로 재귀적으로 쭉 내려가서 class_value를 정해줍니다.

만약 자식노드가 없다면, 즉, 더 분류할 수 없다면 그냥 해당 노드의 class_value를 테스트 데이터의 class_value로 정해주고 test_data를 return 해 줍니다.
만들어진 트리의 노드들을 계속 확인하면서, 어떤 attribute로 분류될 노드인지 확인하고 만약 자식노드가 있다면, 즉 더 분류할 수 있는 노드라면,
해당 노드의 test_attribute 면에서 test_data가 가지는 값이 그 자식노드의 cf_attribute_value로 있다면 분류가 될 수 있는 test_data이니까 재귀적으로 mining 함수를 통해서 분류를 이어나가 주고, 끝났을 때 바로 return을 해줍니다.
없다면 분류가 될 수 없으니까 위의 return에 걸리지 않고 그렇게 되면, 자식노드에 들어가지 못하고 현재노드에서 분류가 끝난 것이니까 그냥 현재 노드의 class_value를 테스트 데이터의 class_value로 정해주고 test_data를 return 해 줍니다.

dt.py (main)

- main 함수

```
# 입력 받아오는 부분
args = sys.argv[1:]
try:
    train_file = open('./data/' + args[0], 'r', encoding='utf-8')
except FileNotFoundError:
    print("*** train 파일이 없습니다. ***\n")
    return 0
try:
    test_file = open('./data/' + args[1], 'r', encoding='utf-8')
except FileNotFoundError:
    print("*** test 파일이 없습니다. ***\n")
    return 0
result_file = open('./data/' + args[2], 'w', encoding='utf-8')

result_file_gini = open('./data/gini_' + args[2], 'w', encoding='utf-8')
result_file_gain_ratio = open('./data/gain_ratio_' + args[2], 'w', encoding='utf-8')
result_file_info = open('./data/info_' + args[2], 'w', encoding='utf-8')

training_rdr = csv.reader(train_file, delimiter='\t')
test_rdr = csv.reader(test_file, delimiter='\t')
```

데이터 파일을 불러오는 코드입니다.

training 데이터와, test 데이터 그리고 결과를 저장할 result 데이터 파일을 불러옵니다.

result 파일은 decision tree의 measure로 information_gain과 gain_ratio, gini_index를 사용하여 가장 괜찮은 결과를 보이는 것을 선택하기 위해 우선 세가지로 저장을 해뒀습니다.

```
# 트레이닝 데이터 프레임 만드는 과정
training_data_set = []
for line in training_rdr:
    training_data_set.append(line)
training_attributes = training_data_set.pop(0)
training_data = {}
for attribute in training_attributes:
    training_data[attribute] = []
for data in training_data_set:
    for i in range(len(data)):
        training_data[training_attributes[i]].append(data[i])
training_df = pd.DataFrame(training_data)
```

트레이닝 데이터 프레임을 만드는 과정입니다. pandas 를 이용했습니다.

```
# gini_index 트리
dt_gini = node_class.Node()
dt_gini = dt_gini.make_tree(training_df, None, 'gini')

# gain_ratio 트리
dt_gain_ratio = node_class.Node()
dt_gain_ratio = dt_gain_ratio.make_tree(training_df, None, 'gain_ratio')

# information 트리
dt_info = node_class.Node()
dt_info = dt_info.make_tree(training_df, None, 'info')
```

각 트리를 만드는 과정 코드입니다.

```
# 테스트 데이터 프레임 만드는 과정
test_data_set = []
for line in test_rdr:
    test_data_set.append(line)
test_attributes = test_data_set.pop(0)
test_data = {}
for attribute in test_attributes:
    test_data[attribute] = []
for data in test_data_set:
    for i in range(len(data)):
        test_data[test_attributes[i]].append(data[i])
test_df = pd.DataFrame(test_data)
test_df[training_attributes[len(training_attributes) - 1]] = None
```

테스트 데이터 프레임 만드는 과정 코드입니다.

```
test_data_set_gini = test_df.values
test_data_set_gain_ratio = test_df.values
test_data_set_info = test_df.values
for i in range(len(training_attributes)):
    if i == len(training_attributes) - 1:
        result_file_gini.write(training_attributes[i])
        result_file_gain_ratio.write(training_attributes[i])
        result_file_info.write(training_attributes[i])
    else:
        result_file_gini.write(training_attributes[i] + '\t')
        result_file_gain_ratio.write(training_attributes[i] + '\t')
        result_file_info.write(training_attributes[i] + '\t')
result_file_gini.write('\n')
result_file_gain_ratio.write('\n')
result_file_info.write('\n')
```

attribute 들을 데이터프레임의 자료를 for 문을 굴리면서 result_file 에 작성하기 번거로워서 그냥 첫줄에 attribute 들을 미리 써놨습니다.

```
# gini_index 트리 결과로 마이닝 하는 과정
for i in range(len(test_data_set_gini)):
    test_data_set_gini[i] = dt_gini.mining(test_data_set_gini[i], training_attributes, 'gini')
    for j in range(len(training_attributes)):
        if j == len(training_attributes) - 1:
            result_file_gini.write(test_data_set_gini[i][j])
        else:
            result_file_gini.write(test_data_set_gini[i][j] + '\t')
    result_file_gini.write('\n')

# gain_ratio 트리 결과로 마이닝 하는 과정
for i in range(len(test_data_set_gain_ratio)):
    test_data_set_gain_ratio[i] = dt_gain_ratio.mining(test_data_set_gain_ratio[i], training_attributes,
                                                       'gain_ratio')
    for j in range(len(training_attributes)):
        if j == len(training_attributes) - 1:
            result_file_gain_ratio.write(test_data_set_gain_ratio[i][j])
        else:
            result_file_gain_ratio.write(test_data_set_gain_ratio[i][j] + '\t')
    result_file_gain_ratio.write('\n')

# info_index 트리 결과로 마이닝 하는 과정
for i in range(len(test_data_set_info)):
    test_data_set_info[i] = dt_info.mining(test_data_set_info[i], training_attributes, 'info')
    for j in range(len(training_attributes)):
        if j == len(training_attributes) - 1:
            result_file_info.write(test_data_set_info[i][j])
        else:
            result_file_info.write(test_data_set_info[i][j] + '\t')
    result_file_info.write('\n')

train_file.close()
```

각 방법을 이용하여 만든 트리로 테스트 데이터의 class value 를 찾아서 각 result_file 에 작성하는 코드입니다. 알고리즘 코드는 다른 py 파일에 적어뒀습니다.

```
train_file.close()
test_file.close()
result_file_gini.close()
result_file_gain_ratio.close()
result_file_info.close()

gini_value = check(args[2], 'gini')
if gini_value == -1:
    result_file_gini = open('./data/gini_' + args[2], 'r', encoding='utf-8')
    for line in result_file_gini:
        result_file.write(line)
    result_file_gini.close()
    print("answer 파일이 없는 관계로 각 방법 비교를 하지 못해, gini_index 방법을 선택했습니다.")
    print()
    result_file.close()
    return

gain_ratio_value = check(args[2], 'gain_ratio')
info_value = check(args[2], 'info')
values = [gini_value, gain_ratio_value, info_value]
print(values)
```

트레이닝 데이터와 테스트 데이터는 모두 사용했고, 각 result 파일들도 다 작성을 완료하여 닫아줍니다. 그리고 check 함수를 통해서 각 방법의 정확도를 파악합니다. 만약 answer 파일이 없다면 그냥 gini_index 의 방법으로 나온 결과를 result 파일에 복사합니다.

```
maxindex = np.argmax(list(values))
if maxindex == 0:
    result_file_gini = open('./data/gini_' + args[2], 'r', encoding='utf-8')
    for line in result_file_gini:
        result_file.write(line)
    result_file_gini.close()
    print("gini_index 방법이 " + str(gini_value) + ' / ' + str(len(test_data_set)) + " 의 정확도로 가장 정확했습니다.")
elif maxindex == 1:
    result_file_gain_ratio = open('./data/gain_ratio_' + args[2], 'r', encoding='utf-8')
    for line in result_file_gain_ratio:
        result_file.write(line)
    result_file_gain_ratio.close()
    print("gain_ratio 방법이 " + str(gain_ratio_value) + ' / ' + str(len(test_data_set)) + "의 정확도로 가장 정확했습니다.")
elif maxindex == 2:
    result_file_info = open('./data/info_' + args[2], 'r', encoding='utf-8')
    for line in result_file_info:
        result_file.write(line)
    result_file_info.close()
    print("information_gain 방법이 " + str(info_value) + ' / ' + str(len(test_data_set)) + "의 정확도로 가장 정확했습니다.")
result_file.close()
```

정확도가 가장 높은 방법을 선택하고 명세에서 요구하는 형식의 result_file 에 해당 방법을 사용한 result_file 을 복사합니다.

- check 함수

```
def check(result_file_name, measure):
    i = -5
    num = ''
    if not result_file_name[i].isdigit():
        try:
            answer_file = open('./data/dt_answer.txt', 'r', encoding='utf-8')
        except FileNotFoundError:
            print("*** answer 파일이 없습니다. ***")
            return -1
        try:
            result_file = open('./data/' + measure + '_dt_result.txt', 'r', encoding='utf-8')
        except FileNotFoundError:
            print("*** " + measure + "_result 파일이 없습니다. ***\n")
            return 0
    else:
        while result_file_name[i].isdigit():
            num = result_file_name[i] + num
            i -= 1
        try:
            answer_file = open('./data/dt_answer' + num + '.txt', 'r', encoding='utf-8')
        except FileNotFoundError:
            print("*** answer 파일이 없습니다. ***")
            return -1
        try:
            result_file = open('./data/' + measure + '_dt_result' + num + '.txt', 'r', encoding='utf-8')
        except FileNotFoundError:
            print("*** " + measure + "_result 파일이 없습니다. ***\n")
            return 0

    answer_rdr = csv.reader(answer_file, delimiter='\t')
    result_rdr = csv.reader(result_file, delimiter='\t')
```

answer file 과 result file 을 불러옵니다.

```
answer_rdr = csv.reader(answer_file, delimiter='\t')
result_rdr = csv.reader(result_file, delimiter='\t')

answer_data_set = []
for line in answer_rdr:
    answer_data_set.append(line)
answer_attributes = answer_data_set.pop(0)
answer_data = {}
for attribute in answer_attributes:
    answer_data[attribute] = []
for data in answer_data_set:
    for i in range(len(data)):
        answer_data[answer_attributes[i]].append(data[i])
answer_df = pd.DataFrame(answer_data)

result_data_set = []
for line in result_rdr:
    result_data_set.append(line)
result_attributes = result_data_set.pop(0)
result_data = {}
for attribute in result_attributes:
    result_data[attribute] = []
for data in result_data_set:
    for i in range(len(data)):
        result_data[result_attributes[i]].append(data[i])
result_df = pd.DataFrame(result_data)
```

answer 데이터와, result 데이터를 데이터 프레임으로 만들어줍니다.

```
answer_data_set = answer_df.values
result_data_set = result_df.values

value = 0
for i in range(len(result_data_set)):
    if result_data_set[i].all() == answer_data_set[i].all():
        value += 1

answer_file.close()
result_file.close()

return value
```

비교하여 결과를 반환합니다.

아래 check_test_program 함수도 있지만, 이 함수는 같이 주신 test 파일의 정확도와 유사한 파일을 만들어서 check 함수를 만들기 위해 작성한 거의 같은 코드이고, 실제 사용되지 않는 코드입니다. check_test_program 을 해본결과 check 함수의 경우 test.exe 파일과 동일한 결과를 내는 것으로 판단됩니다. (모든 행과 열이 똑같고 순서까지 같아야 같은 데이터로 인식한다는 점 – answer 데이터나 result 데이터의 순서를 바꾸면 결과가 달라짐, 달라지는 결과도 check 함수와 test 파일의 결과가 같음)