بسمه تعالی

گزارش تکلیف عملی درخت تصمیم

امین اله زکی زاده

99123038

قسمت اول

برای انجام این آزمایش ستون ها ی فایل csv داده شده مقدار نداشتند به همین علت خود نام ستون ها را اضافه نمودم ، به همین دلیل دیتاست ها را نیز پیوست می نمایم.

برای ایجاد یک درخت تصمیم نیاز به یک تابع جستوجو گر برای پیدا نمودن تعداد زیر ویژگی ها داخل هر گروه ویژگی است که در نتیجه ی آن بتوان آنتروپی حساب کرد و بهترین گروه ویژگی را برای گره پیدا کرد . همچنین نیاز به محاسبه ی آنتروپی و IG داریم .

این دو گروه کار در فایل entropy_v1.py انجام می شود ه دارای محتویات زیر است:

```
import pandas as pd
import numpy as np


def search_csv(dtframe):
    whole_dict = {}
    for i in list(dtframe):
        # np.append(trn, train[i].to_numpy())
        # print(type(i))
        trn = (dtframe[i].to_numpy())
        trn_features, trn_count = np.unique(trn, return_counts=True)  # ,
return_index=True)
        thisdict = {'trn_features': trn_features, 'trn_count': trn_count}  #
, 'trn_index': trn_index}
        whole_dict[i] = thisdict  # saving features , sub_features and number
of sub-features like below
        # whole_dict = {'feature':{'trn_features': trn_features, 'trn_count':
trn_count} .... other features like this}
    return whole_dict


def search_class(dicti, dtfram):
    dtfram_tmp = pd.DataFrame.copy(dtfram)  # copy to stop main data get
manipulated
    lbl = dtfram_tmp.pop(dtfram_tmp.columns.tolist()[0]).to_numpy()  # pop
label column and cast to numpy
    for i in list(dtfram_tmp):  # i is the column feature
        frm = dtfram_tmp.pop(i).to_numpy()  # pop one column from temporary
dataframe and cast it to numpy
        numerator = np.array([], dtype='int64')
        for j in dicti[i]['trn_features']:  # get where in feature column, we
have specific sub_feature = j
            num = 0
```

```
            where_check = np.where(frm == j)
            where_check = np.array(where_check)
            # print(type(where_check))
            for h in where_check:  # we used two for-loop because the
where_check matrix has two dimensions
                for y in h:
                    # print('y=', y)
                    if lbl[y] == '<=50K':  # get the exact label for specific
sub_feature
                        num = num + 1  # only numbering labels '<=50K' others
can be fetched through the whole
                        # number of a feature
            numerator = np.append(numerator, num)  # putting each sub_feature
positive label counts in an array
        dicti[i]["num"] = numerator  # add array to the dictionary for
feature i
    return dicti



def info_gain(dictio, dtfra):
    ent = 0
    ent_wh = 0
    ent_whole = []
    inf_gane = []
    sm = np.sum(dictio['label']['trn_count'])  # adding both positive and
negetive numbers to get total rows in frame,
    # because we may have used some percentage of dataframe for train
    for i in dictio:  # for every feature in dictionary
        # print(dtfra.columns.tolist()[0])
        if i == dtfra.columns.tolist()[0]:  # if we want to calculate total
entropy we need our column to be label
            # print(dictio[i]['trn_count'])
            for j in dictio[i]['trn_count']:  # for every sub-feature counts
in feature i
                tmp = j / sm  # get the probability of sub-feature
                if tmp != 0:  # log2 may result nan
                    ent_wh = ent_wh + (tmp * np.log2(tmp)) * -1
            # print(ent_wh)
        else:  # if our dataframe column is not label , is a feature
            less_tn_50k = np.subtract(dictio[i]['trn_count'],
dictio[i]['num'])
            mr_tn_50k = dictio[i]['num']
            less_pr_50k = np.divide(less_tn_50k, dictio[i]['trn_count'])  #
probability of being below 50k for
            # every sub-feature
            # print(less_pr_50k)
            mr_pr_50k = np.divide(mr_tn_50k, dictio[i]['trn_count'])  #
probability of being more than 50k for
            # every sub-feature
            # print(mr_pr_50k)
            feature_probabe = np.divide(dictio[i]['trn_count'], sm)  # get
the probability of every sub-feature
            res_ls_pr = []
            res_mr_pr = []
            for el_ls in less_pr_50k:
                if el_ls != 0:
```

```
                    res_ls_pr = np.append(res_ls_pr, el_ls * np.log2(el_ls))
                else:
                    res_ls_pr = np.append(res_ls_pr, 0)

            for el_mr in mr_pr_50k:
                if el_mr != 0:
                    res_mr_pr = np.append(res_mr_pr, el_mr * np.log2(el_mr))
                else:
                    res_mr_pr = np.append(res_mr_pr, 0)

            each_ent = np.multiply(np.add(res_ls_pr, res_mr_pr), -1)  # get
entropy of a sub-feature

            ent = np.sum(np.multiply(feature_probabe, each_ent))  # get
entropy of a feature
            ent_whole = np.append(ent_whole, ent)  # appending feature
entropy to ent_whole array
    inf_gane = np.subtract(ent_wh, ent_whole)  # inf-gain array of train data
    # print(inf_gane)
    return inf_gane
```

تابع search_csv برای پیدا کردن انواع زیر ویژگی هاست که در دیکشنری whole_dict می ریزد و تابع search_class برای پیدا کردن تعداد هر کدام از آن ها و ریختن در همان دیکشنری است . تابع inf_gain ابتدا آنتروپی را برای یک دیتا فریم داده شده برای هر کلاس پیدا می کند سپس یک آرایه از information_gain ها برمیگرداند که برای هر هشت کلاس(البته برای کلاس های کمتر و بیشتر هم خود را منطبق می کند) مقدار دارد .

یک تابع برای جستو جوی درخت برای پیدا کردن شاخه ی منطبق با یک داده تست نیاز داریم که داخل فایل evaluat_v1.py با محتویات زیر وجود دارد و نام آن تابع eval است که درخت به دست آمده و دیتا فریم تست یا validation را دریافت می کند و میزان دقت را با عددی بین 0 و 1 برمی گرداند . یک داده ی test را بر می دارد و کل آرایه ی درخت را به دنبال دیکشنری منطبق با کلاس و ویژگی آن می گردد . اگر در شاخه ای یک کلاس یک ویژگی یا یک کلاس در داده test برابر نباشد break می کند و شاخه دیگر را امتحان می کند . بعد از پیدا کردن شاخه منطبق break می کند و لیبل دیتاست test را با لیبل دیکشنری شاخه مقایسه می کند . اگر با هم برابر باشند is_eql یکی اضافه می شود در غیر این صورت is_neq یکی اضافه می شود . در نهایت نسبت is_eql به کل دقت مورد نظر را می دهد.

```
import pandas as pd
import numpy as np
import copy


def eval(test_frame, percent, tree):
    # percent is the percentage of test data to bring into evaluation
    is_eql = 0
    is_neq = 0

    temp_test = pd.DataFrame.copy(test_frame)
    # temp_test = temp_test.sample(frac=(percent / 100.))  # omit this
    # print(temp_test)
    for fr_index in np.arange(0, len(temp_test.index)):
        # print(fr_index)
        # print(temp_test.ndim)
        # print(fr_index)
        for branch in tree:
```

```
            brnch_compatbl = True
            for frac_branch in branch:

                if not (frac_branch == 'label' or frac_branch == 'depth' or
frac_branch == 'empty'
                        or frac_branch == 'vote' or frac_branch ==
'stp_prune' or frac_branch == 'prune_grp'):
                    brnch_compatbl = brnch_compatbl and
(temp_test.iloc[fr_index][frac_branch] == branch[frac_branch])
                    if not brnch_compatbl:  # check if the all branch's frac-
branch values are equal to test
                        break  # if false means that this branch is
incompatible , try another branch

            # after the compatibility was found that means we have
successfully ended the for frac-branch loop , it is
            # time to check if label is also equal or not
            if brnch_compatbl:
                # print(branch['label'])
                # print(temp_test.iloc[fr_index]['label'])
                if branch['label'] == temp_test.iloc[fr_index]['label']:
                    is_eql = is_eql + 1
                else:
                    is_neq = is_neq + 1

                break  # break the for branch loop if branch is compatible ,
because the desired branch is found

    return float(is_eql) / float(is_neq+is_eql)
```

برای پیدا نمودن درخت هم فایل tree_maker_v5.py وجود دارد که حاوی سه تابع است .

تابع find_root برای پیدا کردن ریشه ی درخت و initial کردن آرایه ی درخت نیاز است . یک درخت یک آرایه است که داخل آن گروه های لیبل خورده به صورت دیکشنری حاوی همه ی کلاس ها ی شاخه به عنوان key و ویژگی آن کلاس به عنوان value وجود دارد . محتویات دیکر دیکشنری یک شاخه شامل key های depth که عمق یک شاخه را نشان می دهد ، label که لیبل آن شاخه را نشان می دهد ، empty که در صورت یک بودن نشان می دهد اطلاعات کافی برای یک شاخه وجود نداشته و با voting لیبل خورده ، vote که یک بودن آن نشان دهنده آن است که یک شاخه تا حداکثر عمق رفته و لیبل های متفاوتی برای آن شاخه وجود داشته در نتیجه متداول ترین لیبل را خورده.

تابع labeler تنها برای لیبل زدن root در صورت امکان استفاده می شود و برای گره های بعدی به صورت داخلی در تابع continue_tree برای لیبل زدن مابقی گره ها استفاده می شود.

```
# differece to v2 is that in v2 branches was more than data in train frame so
i though its reason is that we have
# branches  even at places with no sub-feature , assume that we are at a node
, when we want to make a sub-feature
# we even include those sub-features that are excluded in that node because
of data shortage at a specific node ,
```

```
# so in v3 we do not include such sub-features

import pandas as pd
import numpy as np
from entropy_v1 import info_gain
from entropy_v1 import search_class
from entropy_v1 import search_csv
import copy


def labeler(dataframe_random, tree):
    for i in tree:
        temp_frame = pd.DataFrame.copy(dataframe_random)
        if i['label'] is None:
            for j in i:

                if not ((j == 'depth') or (j == 'label') or (j == 'empty') or
(j == 'vote') or (j == 'stp_prune')
                        or (j == 'prune_grp')):
                    temptemp_frame = temp_frame[temp_frame[j] == i[j]]  #
give us the rows which have the same
                    # sub-feature as we desire
                    if not temptemp_frame.empty:
                        temp_frame = temptemp_frame
                        temp_frame = temp_frame.drop(columns=j)
                        i['empty'] = 0
                    else:
                        i['empty'] = 1

        arr = temp_frame.pop('label').to_numpy()

        if i['empty'] == 1:  # if we saw that no sufficient data is available
for specific sub-feature, then vote
            if np.array(np.where(arr == '<=50K')).shape[1] < (len(arr) -
np.array(np.where(arr == '<=50K')).shape[1]):
                i['label'] = '>50K'
            else:
                i['label'] = '<=50K'
        else:  # check if all labels are specific
            if np.array(np.where(arr == '<=50K')).shape[1] == len(arr):
                i['label'] = '<=50K'
            elif np.array(np.where(arr == '>50K')).shape[1] == len(arr):
                i['label'] = '>50K'
            else:
                i['label'] = None
    return tree


def find_root(datafrm_random):
    tree = np.array([])
    dicti = search_csv(datafrm_random)  # do the entropy things here
    dicti = search_class(dicti, datafrm_random)
    inf_g = info_gain(dicti, datafrm_random)
    rt_place = np.argmax(inf_g)  # find the maximum index that shows us the
root

    for i in dicti[datafrm_random.columns.tolist()[rt_place +
```

```
1]]['trn_features']:  # +1 is because the first column
        # is label
        tree = np.append(tree, {'depth': 1, 'label': None, 'empty': None,
'vote': 0, 'stp_prune': 0, 'prune_grp': 0,
                                datafrm_random.columns.tolist()[rt_place +
1]: i})  # making the tree root with
        # this structure : {'depth': 1, 'label': None, 'empty': None, 'vote':
0, 'stp_prune': 0, 'prune_grp': 0,
        #                   feature : sub-feature}
    tree = labeler(datafrm_random, tree)
    return tree


def continue_tree(random_frame, tree):  # after the root is found ( the tree
has been initiated ) , continue the tree
    max_loop = len(random_frame.columns.tolist()) - 1

    cont_or_not = True
    loop = 0
    while (cont_or_not):
        loop = loop + 1
        # print(loop)
        num_of_branch = 0
        ind_delete = np.array([], dtype='int')  # indices(branches) to be
deleted after a branch is expanded
        for i in tree:

            num_of_branch = num_of_branch + 1
            # print(num_of_branch)
            temp_frame = pd.DataFrame.copy(random_frame)
            if i['label'] is None:
                ind_delete = np.append(ind_delete, np.argwhere(tree == i))
                for j in i:  # for every sub-branch in a branch

                    if not ((j == 'depth') or (j == 'label') or (j ==
'empty') or (j == 'vote') or (j == 'stp_prune')
                            or (j == 'prune_grp')):

                        temptemp_frame = temp_frame[temp_frame[j] == i[j]]  #
pick rows in which have the
                        # same sub-feature
                        if not temptemp_frame.empty:
                            temp_frame = temptemp_frame
                            temp_frame = temp_frame.drop(columns=j)
                dicti = search_csv(temp_frame)  # do the following to see
which feature is good enough to be
                # the next sub-branch
                dicti = search_class(dicti, temp_frame)
                inf_g = info_gain(dicti, temp_frame)
                rt_place = np.argmax(inf_g)

                for b in dicti[temp_frame.columns.tolist()[rt_place +
1]]['trn_features']:

                    temp_i = copy.deepcopy(i)

                    alpha = temp_frame.columns.tolist()[rt_place + 1]
```

```
                    lab_temp_frame = pd.DataFrame.copy(temp_frame)
                    temp_i[alpha] = b
                    lab_temptemp_frame = lab_temp_frame[lab_temp_frame[alpha]
== b]
                    if not lab_temptemp_frame.empty:  # see if the branch
created is empty to get voted for labling
                        lab_temp_frame = lab_temptemp_frame
                        lab_temp_frame = lab_temp_frame.drop(columns=alpha)
                        temp_i['empty'] = 0
                    else:
                        temp_i['empty'] = 1

                    lab_arr = lab_temp_frame['label'].to_numpy()
                    where_lab_arr_up = np.array(np.where(lab_arr ==
'<=50K')).shape[1]
                    where_lab_arr_dn = np.array(np.where(lab_arr ==
'>50K')).shape[1]
                    len_lab_arr = len(lab_arr)
                    if temp_i['empty'] == 1:  # voting for empty
                        if where_lab_arr_up < where_lab_arr_dn:
                            temp_i['label'] = '>50K'
                        else:
                            temp_i['label'] = '<=50K'
                    else:
                        if where_lab_arr_up == len_lab_arr:  # ending place
of a branch if this happens
                            temp_i['label'] = '<=50K'
                        elif where_lab_arr_dn == len_lab_arr:
                            temp_i['label'] = '>50K'
                        else:
                            temp_i['label'] = None
                            if loop == max_loop - 1:  # the data have
multiple labels at the deepest place, then vote
                                temp_i['vote'] = 1

                                if where_lab_arr_up < where_lab_arr_dn:
                                    temp_i['label'] = '>50K'
                                else:
                                    temp_i['label'] = '<=50K'

                    temp_i['depth'] = temp_i['depth'] + 1

                    tree = np.append(tree, temp_i)

        # print((ind_delete))
        tree = np.delete(tree, ind_delete)

        cont_or_not = False
        for i in tree:
            cont_or_not = cont_or_not or (i['label'] is None)  # uf there are
still None labels , continue
        # print(cont_or_not)
        # print(tree)
        # print('tree length is :', len(tree))
    np.save('./tree.npy', tree, allow_pickle=True)  # edited

    return tree
```

```
#
# train = pd.read_csv("adult.train.10k.discrete.csv")
#
# random_train = train.sample(frac=1)
#
# tree = []
# tree = find_root(random_train, tree)
# print(tree)
# continue_tree(random_train, tree)
```

در نهایت برای گزارش های خواسته شده در تکلیف یک فایل taklif.py ایجاد شده که توابع بالا را با درصد های متفاوت داده train آموزش می دهد و روی کل داده های train و test درخت به دست آمده را تست می کند و در خروجی داده های خواسته شده را print می کند .

```
import pandas as pd
import numpy as np
from tree_maker_v5 import continue_tree
from tree_maker_v5 import find_root
from evaluat_v1 import eval


def process(train_percent):
    test_eval = np.array([])
    train_eval = np.array([])
    for i in np.arange(0, 5):
        print('round', i, 'with percentage', train_percent, 'wait until
result')
        train = pd.read_csv("adult.train.10k.discrete.csv")
        train = train.sample(frac=train_percent / 100.)
        # random_train = train.sample(frac=1)
        test = pd.read_csv('adult.test.10k.discrete.csv')

        # making tree
        tree = find_root(train)
        # print(tree)
        tree = continue_tree(train, tree)
        lentgh = len(tree)
        train_eval = np.append(train_eval, eval(train, 100, tree))
        test_eval = np.append(test_eval, eval(test, 100, tree))
        print('round ', i, 'with percentage', train_percent)
        print('train_eval is', train_eval[i], 'test_eval', test_eval[i])
        print('length of tree is ', lentgh)
        print('depth of tree is', tree[-1]['depth'])
    print('average train_eval with percentage', train_percent, 'is',
np.divide(np.sum(train_eval), 5))
    print('average test_eval percentage', train_percent, 'is',
np.divide(np.sum(test_eval), 5))


def total():
    for i in ([25, 35, 45, 55, 65, 75, 100]):
        process(i)
```

```
total()
```

نتیجه ی چاپ شده به صورت زیر است :

```
round 0 with percentage 25 wait until result
round  0 with percentage 25
train_eval is 0.8932 test_eval 0.8107594228833463
length of tree is  662
depth of tree is 8
round 1 with percentage 25 wait until result
round  1 with percentage 25
train_eval is 0.902 test_eval 0.8047987442538401
length of tree is  651
depth of tree is 8
round 2 with percentage 25 wait until result
round  2 with percentage 25
train_eval is 0.9004 test_eval 0.809361608614438
length of tree is  626
depth of tree is 8
round 3 with percentage 25 wait until result
round  3 with percentage 25
train_eval is 0.8988 test_eval 0.8031575411488076
length of tree is  699
depth of tree is 8
round 4 with percentage 25 wait until result
round  4 with percentage 25
train_eval is 0.9072 test_eval 0.8055555555555556
length of tree is  645
depth of tree is 8
average train_eval with percentage 25 is 0.90032
average test_eval percentage 25 is 0.8067265744911974
```

```
round 0 with percentage 35 wait until result
round  0 with percentage 35
train_eval is 0.8911428571428571 test_eval 0.8156004861341288
length of tree is  863
depth of tree is 8
round 1 with percentage 35 wait until result
round  1 with percentage 35
train_eval is 0.8908571428571429 test_eval 0.811327003753588
length of tree is  829
depth of tree is 8
round 2 with percentage 35 wait until result
round  2 with percentage 35
train_eval is 0.8877142857142857 test_eval 0.8071689346166612
length of tree is  839
depth of tree is 8
round 3 with percentage 35 wait until result
round  3 with percentage 35
```

train_eval is 0.9008571428571429 test_eval 0.8096077567210225
length of tree is 823
depth of tree is 8
round 4 with percentage 35 wait until result
round 4 with percentage 35
train_eval is 0.89 test_eval 0.8149314144366989
length of tree is 896
depth of tree is 8
average train_eval with percentage 35 is 0.8921142857142857
average test_eval percentage 35 is 0.81 172711913242

round 0 with percentage 45 wait until result
round 0 with percentage 45
train_eval is 0.8871111111111111 test_eval 0.8067606254129046
length of tree is 1006
depth of tree is 8
round 1 with percentage 45 wait until result
round 1 with percentage 45
train_eval is 0.8842222222222222 test_eval 0.8069476712025666
length of tree is 1039
depth of tree is 8
round 2 with percentage 45 wait until result
round 2 with percentage 45
train_eval is 0.8811111111111111 test_eval 0.8066121185316232
length of tree is 1046
depth of tree is 8
round 3 with percentage 45 wait until result
round 3 with percentage 45
train_eval is 0.8888888888888888 test_eval 0.8118238443885915
length of tree is 1031
depth of tree is 8
round 4 with percentage 45 wait until result
round 4 with percentage 45
train_eval is 0.8848888888888888 test_eval 0.817047817047817
length of tree is 1037
depth of tree is 8
average train_eval with percentage 45 is 0.8852444444444444
average test_eval percentage 45 is 0.8098384153167005

round 0 with percentage 55 wait until result
round 0 with percentage 55
train_eval is 0.8852727272727273 test_eval 0.8148472571991678
length of tree is 1200
depth of tree is 8
round 1 with percentage 55 wait until result
round 1 with percentage 55
train_eval is 0.8747272727272727 test_eval 0.819934282584885
length of tree is 1225
depth of tree is 8
round 2 with percentage 55 wait until result
round 2 with percentage 55

train_eval is 0.8858181818181818 test_eval 0.8200589970501475
length of tree is 1145
depth of tree is 8
round 3 with percentage 55 wait until result
round 3 with percentage 55
train_eval is 0.8845454545454545 test_eval 0.8101613080215078
length of tree is 1260
depth of tree is 8
round 4 with percentage 55 wait until result
round 4 with percentage 55
train_eval is 0.8847272727272727 test_eval 0.8182615367751837
length of tree is 1206
depth of tree is 8
average train_eval with percentage 55 is 0.8830181818181819
average test_eval percentage 55 is 0.8166526763261783
round 0 with percentage 65 wait until result
round 0 with percentage 65
train_eval is 0.8878461538461538 test_eval 0.8124455100261552
length of tree is 1387
depth of tree is 8
round 1 with percentage 65 wait until result
round 1 with percentage 65
train_eval is 0.88 test_eval 0.8179143510951291
length of tree is 1386
depth of tree is 8

round 0 with percentage 65 wait until result
round 0 with percentage 65
train_eval is 0.8878461538461538 test_eval 0.8124455100261552
length of tree is 1387
depth of tree is 8
round 1 with percentage 65 wait until result
round 1 with percentage 65
train_eval is 0.88 test_eval 0.8179143510951291
length of tree is 1386
depth of tree is 8
round 2 with percentage 65 wait until result
round 2 with percentage 65
train_eval is 0.8832307692307693 test_eval 0.8127450980392157
length of tree is 1348
depth of tree is 8
round 3 with percentage 65 wait until result
round 3 with percentage 65
train_eval is 0.8838461538461538 test_eval 0.8110894097222222
length of tree is 1337
depth of tree is 8
round 4 with percentage 65 wait until result

round 4 with percentage 65
train_eval is 0.882 test_eval 0.8183304272013949
length of tree is 1353
depth of tree is 8
average train_eval with percentage 65 is 0.8833846153846153
average test_eval percentage 65 is 0.8145049592168234

round 0 with percentage 75 wait until result
round 0 with percentage 75
train_eval is 0.8817333333333334 test_eval 0.81671012603216
length of tree is 1490
depth of tree is 8
round 1 with percentage 75 wait until result
round 1 with percentage 75
train_eval is 0.8768 test_eval 0.8149751136117723
length of tree is 1543
depth of tree is 8
round 2 with percentage 75 wait until result
round 2 with percentage 75
train_eval is 0.8805333333333333 test_eval 0.8149913344887348
length of tree is 1482
depth of tree is 8
round 3 with percentage 75 wait until result
round 3 with percentage 75
train_eval is 0.8809333333333333 test_eval 0.8203703703703704
length of tree is 1540
depth of tree is 8
round 4 with percentage 75 wait until result
round 4 with percentage 75
train_eval is 0.8782666666666666 test_eval 0.8121119703735976
length of tree is 1548
depth of tree is 8
average train_eval with percentage 75 is 0.8796533333333333
average test_eval percentage 75 is 0.8158317829753271

round 0 with percentage 100 wait until result
round 0 with percentage 100
train_eval is 0.8754 test_eval 0.8186136903476571
length of tree is 1900
depth of tree is 8
round 1 with percentage 100 wait until result

```
round  1 with percentage 100
train_eval is 0.8754 test_eval 0.8186136903476571
length of tree is  1900
depth of tree is 8
round 2 with percentage 100 wait until result
round  2 with percentage 100
train_eval is 0.8754 test_eval 0.8186136903476571
length of tree is  1900
depth of tree is 8
round 3 with percentage 100 wait until result
round  3 with percentage 100
train_eval is 0.8754 test_eval 0.8186136903476571
length of tree is  1900
depth of tree is 8
round 4 with percentage 100 wait until result
round  4 with percentage 100
train_eval is 0.8754 test_eval 0.8186136903476571
length of tree is  1900
depth of tree is 8
average train_eval with percentage 100 is 0.8754
average test_eval percentage 100 is 0.8186136903476571
```

در بالا مشاهده شد که تا 45 درصد داده دقت 80 درصد بود اما با افزایش تعداد داده ها درخت فقط بزرگتر شد از لحاظ طول و دقت بر روی داده های train تغییری نکرد.

قسمت دوم : انجام هرس

برای هرس کردن به این صورت عمل شده که به علت طولانی بودن عملیات evaluate و گرفتن زمان فراوان برای پیدا کردن دقت درخت بعد از حذف هر گره ، برای انجام عمل هرس چهار ستون از کلاس ها و 90 درصد ردیف ها حذف شدند و یک درخت بر اساس آن ها تشکیل شد و عمل هرس برای آن ها صورت گرفت .

به همان روش گفته شده در کلاس یعنی گرفتن کاندید و سپس حذف یک گره و پیدا کردن validation بر روی درخت یافت شده و سپس حذف گروهی با بیشترین مقدار دقت ایجاد شده جلو رفتم که نتیجه ی آن فایلی به صورت زیر و با نام pruning.py شد .

در آن سه تابع وجود دارد .

تابع prune_session برای پیدا کردن بهترین گره برای حذف استفاده می شود و در خروجی مقدار دقت بر روی درخت هرس شده ی به دست آمده و خود آن درخت را بر می گرداند .

تابع node_counter برای شمردن تعداد گره های درخت هرس شده در هر بار به دست آمدن یک درخت هرس شده به منظور رسم نمودار های خطا بر روی validation و train و test است .

در نهایت تابع prune از تابع prune_session تا جایی که 0.5 درصد برتری در validation دیده شود عمل pruning را انجام می دهد و در هر بار گره ها را می شمرد و بر روی train و test میزان خطا را بر روی یک آرایه ذخیره می کند. در نهایت هم نمودار های خواسته شده را رسم می کند و درخت هرس شده را برمی گرداند .

در برنامه زیر 25 درصد از داده های train برای validation انتخاب شده است . برنامه تا جایی پیش میرود که خطای داده های validation به اندازه 5 درصد نسبت به حذف گره آخر بهتر عمل کرده باشد .(با توجه به

while total_val_eval < (prn_eval - .005):

داخل تابع prune)

تابع prune هم از prune_session استفاده می کند و یک دور گره های انتهایی را امتحان می کند در صورتی که بهترین validation را حساب کند و 0.5 درصد بهتر عمل کند نسبت به قبل آن هرس را اعمال می کند تا وقتی که این شرط زیر پا گذاشته شود. سپس بر اساس آرایه های nodes و val_eval و test_eval و train_eval عمل رسم انجام می شود که در هر مرحله اندازه ی خطا و تعداد نود ها در آن ها ذخیره شده بود .

```
import pandas as pd
import numpy as np
from tree_maker_v5 import continue_tree
from tree_maker_v5 import find_root
from evaluat_v1 import eval
from matplotlib import pyplot as plt


def prune_session(tmptree, train_data, val_data):
    tree = np.copy(tmptree)  # for comparison to be done elementwize must be
ndarray
    # prn_grp = 0
    eval_val_grp = []  # each group of pruning will put its evaluation on
validation in here
    app_brnch_grp = []  # saving representative branches here
    app_tree_grp = []  # every tree before eval is saved here
    for branch in tree:
        # eval_val = eval(validate, 100, tree)

        if branch['stp_prune'] == 0:
            # prn_grp = prn_grp + 1  # grouping same kinds of nodes for
deleting ,if validation changes a lot when this
            # group is deleted ( group means one node)
            temp_prnd_tr = np.copy(tree)
            # temp_prnd_tr = np.delete(temp_prnd_tr, temp_prnd_tr == branch)
# how an element is deleted by value
            # temp_prnd_tr[temp_prnd_tr == branch]['stp_prune'] = 1
            deletion_grp = np.full(len(tree), False)  # every branch
concluded in pruning group will put its true,false
            # of being in group here
            # to get deleted by np.delete afterwards

            branch_dict_key = np.array([])
            for br in branch:  # find the keys ( nodes ) of a branch to
compare with each temp_prnd_tr node
                if not (br == 'label' or br == 'depth' or br == 'empty'
                        or br == 'vote' or br == 'stp_prune' or br ==
'prune_grp'):
                    branch_dict_key = np.append(branch_dict_key, br)

            # app_brnch = {}  # the branch that must get appende to the
temp_tree in representation of
            # deleted branches
            temp_bran = dict.copy(branch)
            dict.pop(temp_bran, branch_dict_key[-1])  # removing last item of
```

```
branch to use for new tree eval
            app_brnch = temp_bran  # append branch equals the branch without
last feature
            app_brnch['label'] = None
            app_brnch['depth'] = app_brnch['depth'] - 1
            app_brnch['empty'] = 0
            app_brnch_grp = np.append(app_brnch_grp, app_brnch)  # save main
branch to add to final pruned tree

            for brch in temp_prnd_tr:  # process of deleting similar branches
                if brch['stp_prune'] == 0:
                    dont_check_last_node = 0
                    last_node = brch['depth']

                    for sub_brch in brch:

                        if not (sub_brch == 'label' or sub_brch == 'depth' or
sub_brch == 'empty'
                                or sub_brch == 'vote' or sub_brch ==
'stp_prune' or sub_brch == 'prune_grp'):
                            if sub_brch ==
branch_dict_key[dont_check_last_node]:  # check the keys to be equal # edited

                                if not dont_check_last_node == last_node - 1:
# do not check value of last key

                                    if not brch[sub_brch] ==
branch[branch_dict_key[dont_check_last_node]]:
                                        # if values of key are not equal then
try next branch
                                        break
                                    # else:
                                    # app_brnch[sub_brch] =
branch[branch_dict_key[dont_check_last_node]]  # this is
                                    # created many times

                                elif dont_check_last_node == last_node - 1:
                                    deletion_grp =
np.logical_or(deletion_grp, temp_prnd_tr == brch)

                                    tmp_brch = dict.copy(brch)
                                    tmp_brch['stp_prune'] = 1
                                    # tmp_brch['prune_grp'] = prn_grp
                                    tree[np.array(np.where(tree == brch))[0,
0]] = tmp_brch
                                    # tree[tree == brch]['stp_prune'] = 1
###################################wont work
                                    # tree[tree == brch]['prune_grp'] =
prn_grp  # starting with 1 not 0########wont work

                                    # turning stp_prune in main tree to 1
                                    # numbering prune_group
                                    # delete from temp_tree to calculate
                                    # break
################################### edited no need to break
                                dont_check_last_node = dont_check_last_node +
1
```

```
                            else:  # if keys are not equal try another branch
                                break
                    # do not put anything here for break to work in upper else

            temp_prnd_tr = np.delete(temp_prnd_tr, deletion_grp)
            # add a branch with removed node
            temp_prnd_tr = np.append(temp_prnd_tr, app_brnch)

            # label it by voting by train data

            for r in temp_prnd_tr:
                temp_frame = pd.DataFrame.copy(train_data)
                if r['label'] is None:
                    for t in r:

                        if not ((t == 'depth') or (t == 'label') or (t ==
'empty') or (t == 'vote') or (
                                t == 'stp_prune') or (t == 'prune_grp')):
                            temptemp_frame = temp_frame[temp_frame[t] ==
r[t]]
                            if not temptemp_frame.empty:  # redundant
                                temp_frame = temptemp_frame
                                temp_frame = temp_frame.drop(columns=t)
                            #     r['empty'] = 0
                            # else:
                            #     r['empty'] = 1

                    arr = temp_frame.pop('label').to_numpy()  # edited
                    if np.array(np.where(arr == '<=50K')).shape[1] < (
                            len(arr) - np.array(np.where(arr ==
'<=50K')).shape[1]):
                        r['label'] = '>50K'
                    else:
                        r['label'] = '<=50K'
            print(len(temp_prnd_tr))
            app_tree_grp = np.append(app_tree_grp, {'key': temp_prnd_tr})  #
save tree
            # eval must happen here on validation data
            eval_val_grp = np.append(eval_val_grp, eval(val_data, 100,
temp_prnd_tr))
            # save eval in eval_val_grp
    max_eval = np.max(eval_val_grp)
    tree = app_tree_grp[np.argmax(eval_val_grp)]['key']
    for branch in tree:  # make it available for another pruning session
        branch['stp_prune'] = 0
        branch['prune_grp'] = 0
    return max_eval, tree


def node_counter(tree_aftr_prn):
    branch_keeper = np.array([])

    nod_num = 0
    for bra in tree_aftr_prn:
        rnd_cnt = 0
        branch_maker = np.array([])
```

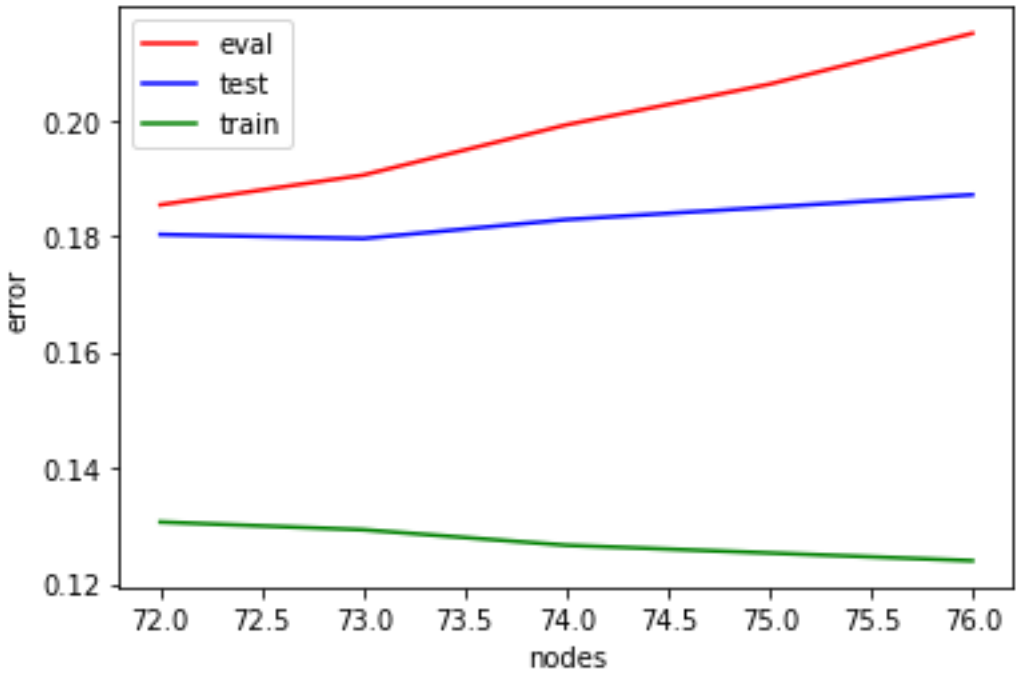
```
        for sub_bra in bra:
            if not (sub_bra == 'label' or sub_bra == 'depth' or sub_bra ==
'empty'
                    or sub_bra == 'vote' or sub_bra == 'stp_prune' or sub_bra
== 'prune_grp'):
                rnd_cnt = rnd_cnt + 1
                # if not rnd_cnt == 1:
                branch_maker = np.append(branch_maker, bra[sub_bra])
                # print(rnd_cnt)
                # print(branch_maker)
                is_there = 0
                for br_arry in branch_keeper:
                    if np.array_equal(br_arry['key'], branch_maker):
                        is_there = 1
                if is_there == 0:
                    # print(list(branch_maker.shape)[0])
                    branch_keeper = np.append(branch_keeper, {'key':
branch_maker})
        # print(branch_keeper)

    nod_num = len(branch_keeper) - len(tree_aftr_prn) + 1
    print('number of nodes:', nod_num)
    return nod_num


def prune(tree, train_data, val_data, test_data):
    total_val_eval = 0.
    prn_eval = 1.
    nodes = np.array([])
    # nodes = np.append(nodes, node_counter(tree))
    train_eval = np.array([])
    test_eval = np.array([])
    val_eval = np.array([])
    tmp_tree = np.array([])
    rnd = 0
    # total_val_eval = eval(val_data, 100, tree)
    # val_eval = np.append(val_eval, 1. - total_val_eval)
    # test_eval = np.append(test_eval, 1. - eval(test_data, 100, tree))
    # train_eval = np.append(train_eval, 1. - eval(train_eval, 100, tree))
    while total_val_eval < (prn_eval - .005):  # if working 0.05 percent
better continue pruning
        rnd = rnd + 1
        print('round', rnd, 'of pruning')
        # if total_val_eval < (prn_eval - .005):
        if rnd == 1:
            total_val_eval = eval(val_data, 100, tree)
        else:
            total_val_eval = prn_eval  # eval(val_data, 100, tree)
        val_eval = np.append(val_eval, 1. - total_val_eval)
        test_eval = np.append(test_eval, 1. - eval(test_data, 100, tree))
        train_eval = np.append(train_eval, 1. - eval(train_data, 100, tree))
        nodes = np.append(nodes, node_counter(tree))

        prn_eval, tmp_tree = prune_session(tree, train_data, val_data)
        if total_val_eval < (prn_eval - .005):
            tree = tmp_tree
            np.save('prntree.npy', tree, allow_pickle=True)  # edited
```

```
    np.save('val_eval.npy', val_eval, allow_pickle=True)
    np.save('test_eval.npy', test_eval, allow_pickle=True)
    np.save('train_eval.npy', train_eval, allow_pickle=True)
    np.save('nodes.npy', nodes, allow_pickle=True)

    # plotting results
    plt.figure()
    plt.plot(nodes, val_eval, 'r', label='eval')
    plt.plot(nodes, test_eval, 'b', label='test')
    plt.plot(nodes, train_eval, 'g', label='train')
    plt.xlabel('nodes')
    plt.ylabel('error')
    plt.legend()
    plt.show()
    return tree



train = pd.read_csv("adult.train.10k.discrete.csv")
train = train.sample(frac=.1)
train = train.drop(columns=[' race', ' sex', ' native-country', '
workclass'])
validate = train.sample(frac=.25)
train = train.drop(index=validate.index.to_numpy())
# random_train = train.sample(frac=1)
test = pd.read_csv('adult.test.10k.discrete.csv')
test = test.drop(columns=[' race', ' sex', ' native-country', ' workclass'])
test = test.sample(frac=.1)

# making tree
tree = find_root(train)
# print(tree)
tree = continue_tree(train, tree)
# print('nodes1:', node_counter(tree))
# prune_session(tree, train, validate)
# print(tree)
# print('nodes2:', node_counter(tree))

# tree = np.load('tree.npy', allow_pickle=True)
prune(tree, train, validate, test)

# total_val_eval = eval(validate, 100, tree)
```

نمودار زیر در نهایت به دست می آید که از جایی شروع شده که دقت روی داده های validation شروع به کم شدن کرده .

در نمودار زیر مشاهده می شود دقت همچنان در حال افزایش از 72 گره به بعد روی داده های train را نشان می دهد در حالی که از 72 گره به بعد برای داده های validate و از 73گره به بعد برای داده های test شاهد کم شدن دقت و افزایش خطا هستیم.

در نتیجه می توان گفت از گره 73 به بعد شاهد overfitting هستیم.



همین عمل را با انتخاب 25 درصد داده های validation از روی دیتاست های test تکرار می کنیم. این بار برای مشاهده ی کل کمی قبل تر از overfitting را هم نشان دهد پس خط زیر را :

```
while total_val_eval < (prn_eval - .005):
```

به زیر تغییر می دهیم:

```
while total_val_eval < (prn_eval + .003):
```

و تغییر شرط زیر:

```
if total_val_eval < (prn_eval - .005):
```

به زیر:

```
if total_val_eval < (prn_eval + .003):
```

و خطوط زیر را که داده validation را از train بر می داشت :

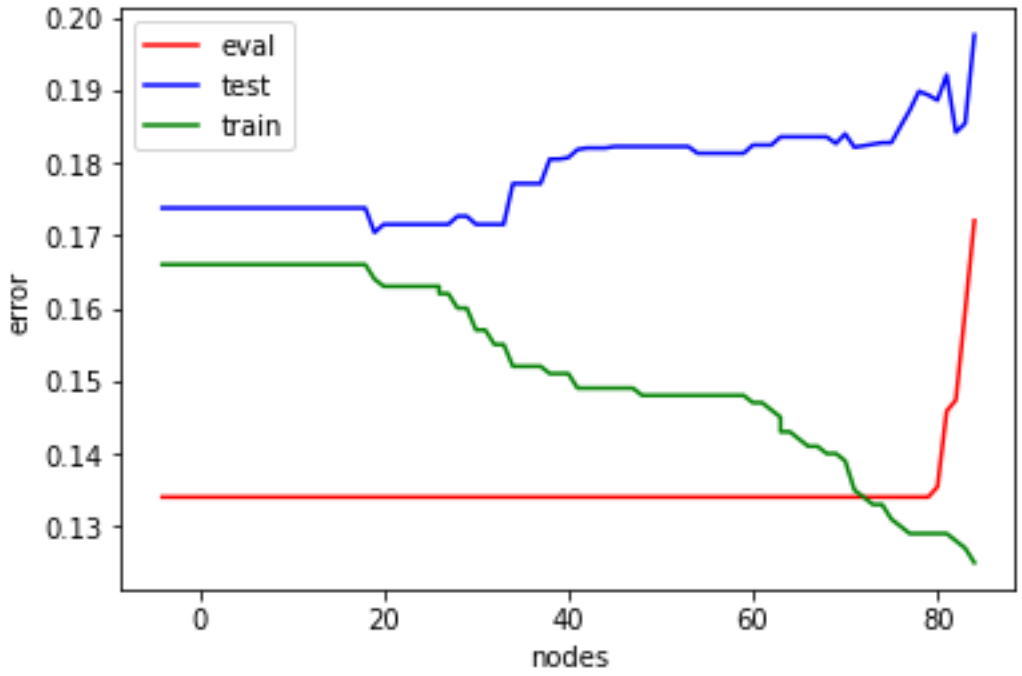
```
train = pd.read_csv("adult.train.10k.discrete.csv")
train = train.sample(frac=.1)
train = train.drop(columns=[' race', ' sex', ' native-country', '
workclass'])
validate = train.sample(frac=.25)
train = train.drop(index=validate.index.to_numpy())
# random_train = train.sample(frac=1)
test = pd.read_csv('adult.test.10k.discrete.csv')
test = test.drop(columns=[' race', ' sex', ' native-country', ' workclass'])
test = test.sample(frac=.1)
```

به زیر تغییر می دهیم:

```
train = pd.read_csv("adult.train.10k.discrete.csv")
train = train.sample(frac=.1)
```

```
train = train.drop(columns=[' race', ' sex', ' native-country', '
workclass'])
# random_train = train.sample(frac=1)
test = pd.read_csv('adult.test.10k.discrete.csv')
test = test.drop(columns=[' race', ' sex', ' native-country', ' workclass'])
test = test.sample(frac=.1)
validate = test.sample(frac=.1)
test = test.drop(index=validate.index.to_numpy())
```

با تکرار دوباره بالا نمودار زیر را داریم :

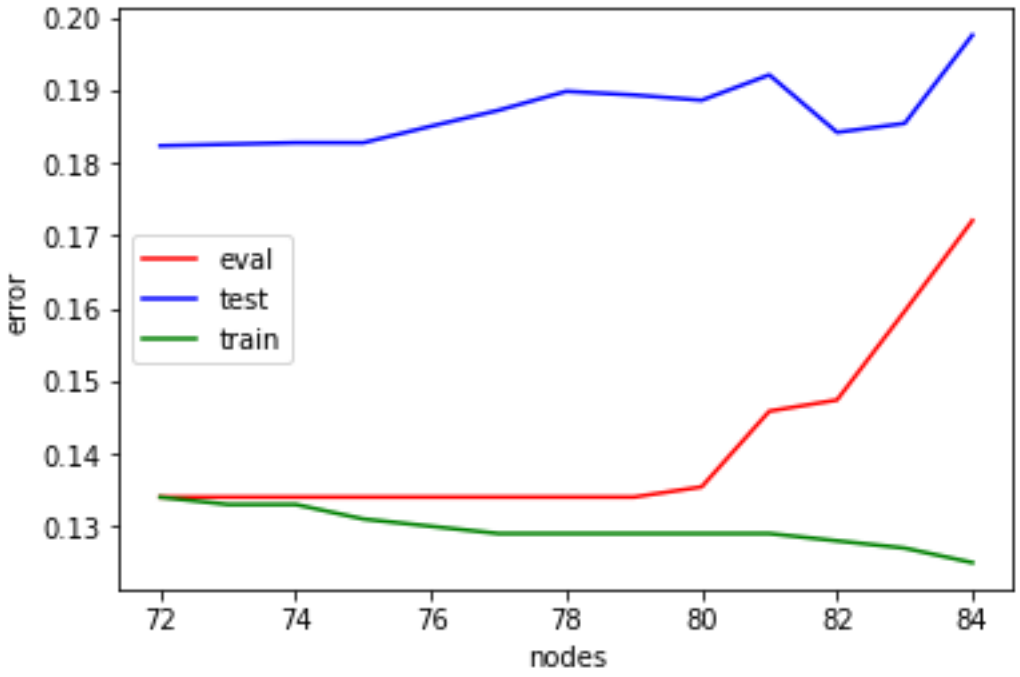


اگر فقط چند آرایه ی اول را برداریم و رسم کنیم :

```
nodes = np.array([])
val_eval = np.array([])
test_eval = np.array([])
train_eval = np.array([])
nodes = np.load('nodes.npy',allow_pickle=True)
val_eval = np.load('val_eval.npy', allow_pickle=True)
train_eval = np.load('train_eval.npy', allow_pickle=True)
test_eval = np.load('test_eval.npy', allow_pickle=True)
val_eval = np.load('val_eval.npy', allow_pickle=True)
nodes = nodes[0:15]
train_eval = train_eval[0:15]
test_eval = test_eval[0:15]
val_eval = val_eval[0:15]

plt.plot(nodes, val_eval, 'r', label='eval')
plt.plot(nodes, test_eval, 'b', label='test')
plt.plot(nodes, train_eval, 'g', label='train')
plt.xlabel('nodes')
```

```
plt.ylabel('error')
plt.legend()
plt.show()
```

نتیجه :



نتیجه در گره 79 نشان می دهد دیتای validation دچار افزایش خطا می شود . این افزایش خطا را در گره 75 در داده های test می توان مشاهده نمود . در نتیجه می توان گفت overfitting بعد از گره 75 به بعد دیده می شود.