Hw2 report

0616237 廖俊崴

1. data input:

import csv package 用 csv.Reader 讀 csv 檔，再把讀出來的 dictionary 轉成二維 list 儲存，把 X_train.csv 跟 Y_train.csv 存到同個 2 維 list data_arr 裡。

```
fp=open('X_train.csv','r',newline='')
xtestdict=csv.DictReader(fp)
f=open('Y_train.csv','r',newline='')
ytestdict=csv.DictReader(f)
#create 2d array ,include feature and outcome
data_arr=[]
row_len=0
#******
for row in xtestdict:
    data_arr.append([])
    for i in range(len(input_type)):
        data_arr[row_len].append(row[input_type[i]])
    row_len+=1
row_len=0
for row in ytestdict:
    data_arr[row_len].append(row[data_type[-1]]) #add category
    row_len+=1
f.close()
fp.close()
```

2. data preprocessing

先用 random.shuffle() shuffle data_arr 讓裡面的資料亂數排列，再把有 missing value 的資料 drop 掉，在用 trans_string_to_codestring()把 category 的 features 轉成用數字代表各種類別，再用 trans_continue_to_bounding()把 continue features 轉成用 15 個數字把連續資料 bounding 並代表在哪個區間的資料。

```
#shuffle data
np.random.shuffle(data_arr)

#delete miss value
arr_len=0
while arr_len<len(data_arr):
    a=0
    a=data_arr[arr_len].count(' ?')
    arr_len+=1
    if a>0:
        arr_len-=1
        del data_arr[arr_len]

#modified discrete data
trans_string_to_codestring(data_arr)

#modified continue date
trans_continue_to_bounding(data_arr,15)
```

3 .model construction:

Decision tree:

利用 crete_tree() create decision tree，先判斷是否到了 leaf，如果不是，用 best_attr,threshold = choose_attr(data_arr)選出要以此分類的 attribute 跟要用哪個數字當分類的 threshold，因為所有 attribute 都被轉成用數字表示，所以可以用 threshold 來區分，如果小於等於 threshold 就是左節點，大於 threshold 就是柚節點，詳細程式碼如下

```
feature_arr=['age','workclass','fnlwgt','education','education-num',
           'marital-status','occupation','relationship','race','sex',
           'capital-gain','capital-loss','hours-per-week','native-country']
input_type=['Id','age','workclass','fnlwgt','education','education-num',
           'marital-status','occupation','relationship','race','sex',
           'capital-gain','capital-loss','hours-per-week','native-country']
feature_continus_discrete_arr=['c','d','c','d','c','d','d','d','d','d','c','c','c','d']
data_type=['Id','age','workclass','fnlwgt','education','education-num',
```

```
          'marital-status','occupation','relationship','race','se
x',
          'capital-gain','capital-loss','hours-per-week','native-
country',
          'Category']

continue_feature_threshold={'age':37,'fnlwgt':178233,'education-n
um':10,'capital-gain':0,'capital-loss':0,'hours-per-week':40}
#判斷標準 中位數

answer_arr=['Id','Category']
class Node(object):
   def __init__(self,attribute,threshold):
      self.attr = attribute
      self.thres = threshold
      self.left = None
      self.right = None
      self.leaf = False
      self.predict = None
def trans_continue_to_bounding(data_arr,n):
    for i in range(1,len(feature_arr)+1):
      if feature_continus_discrete_arr[i-1]=='d':
         continue
      values=[int(row[i]) for row in data_arr]
      values.sort()
      size,rem=div_list_num(values,n)
      start=0
      end=size
      for row in data_arr:
         for j in range(n):
            start=j*size
            end=start+size-1
            if j==(n-1):
               end+=rem
            if int(row[i]) >= values[start] and  int(row[i]) <=va
lues[end]:
               row[i]=str(j)
```

```python
def div_list_num(values,n):
    size=0
    size+=math.floor(len(values)/n)
    rem=len(values)-size*n
    while rem > n:
        size+=math.floor(rem/n)
        rem=len(values)-size*n
    return size,rem


def trans_string_to_codestring(data_arr):
    for i in range(1,len(feature_arr)+1):
        if feature_continus_discrete_arr[i-1]=='c':
            continue
        codedict={}
        num=0
        for row in range(len(data_arr)):
            if data_arr[row][i] not in codedict.keys():
                codedict[data_arr[row][i]]=num
                num+=1
            data_arr[row][i]=str(codedict[data_arr[row][i]])

def compute_entropy(data_arr):
    arr_len=len(data_arr)
    labelcount={}
    for row in data_arr:
        tmp_category=row[-1]
        if tmp_category not in labelcount.keys():
            labelcount[tmp_category]=0
        labelcount[tmp_category]+=1
    entro=0
    #print(labelcount)
    for key in labelcount:
```

```
        pro=float(labelcount[key])/arr_len
        entro -=pro*math.log(pro,2)
    return entro
def find_most_category(classlist):
    classcount={}
    for vote in classlist:
        if vote not in classcount.keys():
            classcount[vote]=0
        classcount[vote]+=1
    sortedlist= sorted(classcount.items(),key=operator.itemgetter
(1),reverse=True)
    #print(type(sortedlist))
    return sortedlist[0][0]
def choose_thres(data_arr,attribute):
    if(feature_continus_discrete_arr[attribute-1]=='d'):
        values=[float(row[attribute]) for row in data_arr]
        values=set(values)
        values=list(values)
        values.sort()
        max_ig=float("-inf")
        thres_val=0
        for i in range(0,len(values)-1):
            thres=(values[i]+values[i+1])/2
            ig=info_gain(data_arr,attribute,thres)
            if ig>max_ig:
                max_ig=ig
                thres_val=thres
        return thres_val
    else:
        values=[float(row[attribute]) for row in data_arr]
        return np.median(values)

def info_gain(data_arr,attr,threshold):
    sub1=[row for row in data_arr if float(row[attr])<=threshold]
    #print("sub1len:",len(sub1))
    sub2=[row for row in data_arr if float(row[attr])>threshold]
    #print("sub2len:",len(sub2))
    ig=compute_entropy(data_arr)-remainder(data_arr,[sub1 ,sub2])
```

```
    return ig

def remainder(data_arr,data_subsets):
        num=len(data_arr)
        rem=float(0)
        for sub in data_subsets:
            rem += float(len(sub)/num)*compute_entropy(sub)
        return rem
def choose_attr(data_arr):
    max_info_gain=float('-inf')
    best_attr=None
    threshold=0
    for attr in range(1,len(feature_arr)+1 ):
        thres=choose_thres(data_arr,attr)
    ig = info_gain(data_arr,attr,thres)
    if ig > max_info_gain:
        max_info_gain=ig
        best_attr=attr
        threshold=thres
    return best_attr,threshold
def create_tree(data_arr):
    #print("now arrlen:",len(data_arr))
    classlist=[row[-1] for row in data_arr]
    if classlist.count(classlist[0])==len(classlist):
        leaf = Node(None,None)
        leaf.leaf=True
        leaf.predict=classlist[0]
        return leaf
    best_attr,threshold = choose_attr(data_arr)
    attrlist=[row[best_attr] for row in data_arr ]
    if attrlist.count(attrlist[0])==len(attrlist):
        leaf = Node(None,None)
        leaf.leaf=True
        f=classlist.count('0')
        t=classlist.count('1')
        if t>f:
            leaf.predict='1'
        else:
```

```
            leaf.predict='0'
        return leaf
    #print("choose success")
    tree=Node(best_attr,threshold)
    sub1=[row for row in data_arr if float(row[best_attr]) <= thres
hold]
    sub2=[row for row in data_arr if float(row[best_attr]) > thresh
old]
    #print("sub1len:",len(sub1))
    #print("sub2len:",len(sub2))
    #print("attribute:",feature_arr[best_attr-1],'threshold:',thre
shold)
    tree.left = create_tree(sub1)
    tree.right = create_tree(sub2)
```

random forest:

用 crete_forest()建，qautity 代表建幾顆 tree，data_size 代表要建一顆 tree 所需資料量，回傳由一堆 tree 的 root 組成的 list。

```
def create_forest(data_arr,quantity,sub_data_size):
    forest=[]
    for i in range(quantity):
        sub_arr=random.sample(data_arr,sub_data_size)
        forest.append(create_tree(sub_arr))
    return forest
```

4validation

實作在 main function 裡，基本上就是調整傳進 create_tree 或 create_forest 裡的參數，跟 result 的計算要稍微改一下，更加詳細情形在 hw2-1.ipynb 裡。

5.result

```
        attribe: native-country the value<=thres 1.5 go to left nod
e
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value>=thres 0.5 go to right node
```

```
in the leaf the prediction is: 0
attribe: native-country the value>=thres 1.5 go to right node
attribe: native-country the value<=thres 18.5 go to left node
attribe: native-country the value<=thres 4.5 go to left node
attribe: native-country the value<=thres 2.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value>=thres 1.5 go to right node
attribe: native-country the value<=thres 18.5 go to left node
attribe: native-country the value<=thres 4.5 go to left node
attribe: native-country the value>=thres 2.5 go to right node
attribe: native-country the value<=thres 3.5 go to left node
in the leaf the prediction is: 0
decision tree with holdout validation result:
confusion matrix
true positive: 4753  false positive: 1570
false negative: 11 true negative: 5
Accuracy: 0.750591575958353
Precision: 0.751700142337498
Recall: 0.9976910159529807
```

random forest has2000 tree with 50 datas with holdout validation re
sult:
confusion matrix
true positive: 4764 false positive: 1575
false negative: 0 true negative: 0
Accuracy: 0.751538097491718
Precision: 0.751538097491718
Recall: 1.0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value>=thres 0.5 go to right node
in the leaf the prediction is: 0
attribe: native-country the value>=thres 1.5 go to right node
attribe: native-country the value<=thres 18.5 go to left node
attribe: native-country the value<=thres 4.5 go to left node
attribe: native-country the value<=thres 2.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value>=thres 1.5 go to right node

```
attribe: native-country the value<=thres 18.5 go to left node
attribe: native-country the value<=thres 4.5 go to left node
attribe: native-country the value>=thres 2.5 go to right node
attribe: native-country the value<=thres 3.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 2.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 2.5 go to left node
attribe: native-country the value>=thres 0.5 go to right node
attribe: native-country the value<=thres 1.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 2.5 go to left node
attribe: native-country the value>=thres 0.5 go to right node
attribe: native-country the value>=thres 1.5 go to right node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 2.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 2.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 2.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 2.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 2.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 2.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value>=thres 2.5 go to right node
attribe: native-country the value<=thres 4.5 go to left node
attribe: native-country the value<=thres 3.5 go to left node
in the leaf the prediction is: 0
```

```
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value>=thres 0.5 go to right node
in the leaf the prediction is: 0
attribe: native-country the value>=thres 1.5 go to right node
attribe: native-country the value<=thres 4.5 go to left node
attribe: native-country the value<=thres 2.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value<=thres 1.5 go to left node
attribe: native-country the value<=thres 0.5 go to left node
in the leaf the prediction is: 0
attribe: native-country the value>=thres 1.5 go to right node
attribe: native-country the value<=thres 4.5 go to left node
attribe: native-country the value>=thres 2.5 go to right node
attribe: native-country the value<=thres 3.5 go to left node
in the leaf the prediction is: 0

decision tree with k-fold result:
confusion matrix
true positive: 4756.0  false positive: 1570.3333333333333
false negative: 8.0 true negative: 4.666666666666667
```

```
Accuracy: 0.7510122521954041
Precision: 0.7517782812582329
Recall: 0.998320738874895

ranndom forest has2000 tree with 50 datas with k-fold validation re
sult:
confusion matrix
true positive: 5276.333333333333  false positive: 1768.0
false negative: 0.0 true negative: 0.0
Accuracy: 0.7490181233142479
Precision: 0.7490181233142479
Recall: 1.0
```

6.comparsion&result:

這次作業我做得不夠好，如果有更多時間我應該會把我的 decision tree 改成多節點樹，因為要做 2 元樹，我把所有 feature 用數字表示，所以可能被分到同個節點的資料其實之間也沒有相關性，只是數字剛好同時小於\大於 threshold 而已，而且從我的 prediction result 可以看出來因為我的 feature 不會越來越少，導致可能都只用同一個 feature 來判斷，而且改成多節點樹，深度最多等於 feature 數量而已，總體來說，我對這次作業並不太滿意，希望下次可以更好，不過網路上的資料真的很難找，超大部分都是用 sklearn 的，讓我花了好長時間找資料。

7.kaggle submission