Programming Assignment #2 Report

0716206 陳昱丞

Code Interpretation:

第一步先匯入資料。python 的 sklearn 裡面就有一些常用到的 datasets，例如 iris、wine 等。若不是用 sklearn 的就要另外從網路上下載並做一些前處理。之後把 dataset 切割成 training subset 和 validation subset，這裡我是用 sklearn 裡的 train_test_split，test_size = 0.3 就代表 training subset 佔 0.7、validation subset 佔 0.3，random_state 可以確保每次分割出來的 subset 不會一樣。

接下來就是實作 decision tree 的部分。這就像是條件比較多的二元樹。先寫一個叫 node 的 class，這個 class 的成員有 feature 陣列、與其對應的 target 陣列、node splitting 條件(attribute bagging、包含一個 feature index 和一個 threshold)、left node、right node。接著寫一個計算 gini 值的函式。這個函式會傳入兩個參數，feature 陣列和對應之 target 陣列。它依次取出每個 feature，將值排序後算出 target-1 個 threshold 值((v1+v2)/2, (v2+v3)/2, …, (vn-1+vn)/2)，依照每個 threshold 值切割 target 陣列，算出每個 gini 值。算完後選出最小的 gini 值，選擇此 gini 值對應之 feature 和 threshold 為 node splitting 之條件並回傳。

最後就可以遞迴建出整棵 decision tree 了。先建立 root node 並傳入 training subset 的 feature 陣列和 target 陣列，並將 root 傳入 tree 建構函式。每次遞迴，都用 gini 計算函式算出 splitting 條件，依照此條件分割出 left node 和 right node，再將這兩個 node 遞迴傳入 tree 建構函式，並將 depth 值加 1。若 depth 等於 5 或是分割出的 node 只有唯一一種 target 就 return。

Random forest 的部分就是建出 20 棵不同的 decision tree，這 20 棵 tree 算法部分完全相同，只是一開始 root 傳入的 feature 陣列、target 陣列不同。每棵樹傳入的 data 都是在 training subset 中隨機取樣，這裡我一樣用 train_test_split 切出佔比 0.7 的 data，而每次迴圈的 random_state 不同，確保每次取出的 data 不會完全一致。最後在 prediction 時採取 majority vote 的方式，但每棵樹的一票不完全等值，而是要乘上它的準確率。舉例來說，tree no.3 預測 target 為 2，而 tree no.3 預測 2 的準確率為 75%，那麼它的這一票就會是 1*0.75 = 0.75 分。最後 random forest 會選擇最高分的那個 target 為最終的 prediction 結果。

Observation:

我採用了三種 dataset 來做評估，分別是 iris、wine、ecoli。其中 iris 和 wine 是用 sklearn 裡面的，ecoli 是從 UCI Machine Learning Repository 下載的。我的 decision tree 和 random forest 的 depth 都是 5，其中 random forest 為 20 棵

樹做 tree bagging。算法部分都是照上個部分所述。每個 dataset 都執行三次，以下為執行結果(以 confusion matrix 和 total accuracy 呈現):

Iris:

decision tree:

[10 0 0]	[15 0 0]	[16 0 0]
[0 15 3]	[0 9 0]	[0 13 2]
[0 4 13]	[0 4 17]	[0 1 13]
accuracy: 84%	accuracy: 91%	accuracy: 93%

random forest:

[10 0 0]	[15 0 0]	[16 0 0]
[0 16 2]	[0 9 0]	[0 15 0]
[0 3 14]	[0 3 18]	[0 1 13]
accuracy: 89%	accuracy: 93%	accuracy: 98%

Wine:

decision tree:

[15 3 0]	[20 3 0]	[16 0 0]
[1 17 0]	[0 15 0]	[2 22 0]
[0 2 16]	[0 2 14]	[0 1 13]
accuracy: 89%	accuracy: 91%	accuracy: 94%

random forest:

[15 3 0]	[22 1 0]	[16 0 0]
[0 17 1]	[1 14 0]	[0 24 0]
[0 1 17]	[0 2 14]	[0 0 14]
accuracy: 91%	accuracy: 93%	accuracy: 100%

Ecoli:

decision tree:

[3 0 0 0 0 0 0 0]	[2 0 0 0 0 0 0 0]	[3 0 0 0 0 0 0 0]
[0 13 1 11 0 0 0 0]	[0 16 0 8 0 0 0 0]	[0 17 0 2 1 0 0 0]
[0 0 40 0 1 1 0 0]	[0 1 38 1 0 1 0 0]	[0 1 51 1 0 0 0 0]
[0 0 0 7 0 0 0 0]	[0 1 0 7 0 1 0 0]	[0 2 0 4 0 0 0 0]
[0 0 0 0 0 2 0 0]	[0 0 1 0 2 4 0 0]	[0 0 0 0 0 4 0 0]
[0 0 3 0 2 15 0 0]	[0 0 2 2 4 8 0 0]	[0 0 1 0 0 13 0 0]
[0 0 1 0 0 0 0 0]	[0 0 0 0 1 0 0 0]	[0 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0]	[1 0 0 0 0 0 0 0]	[0 0 1 0 0 0 0 0]

accuracy: 77%	accuracy: 72%	accuracy: 87%

random forest:

[3 0 0 0 0 0 0 0]	[2 0 0 0 0 0 0 0]	[3 0 0 0 0 0 0 0]
[0 15 1 9 0 0 0 0]	[0 19 0 5 0 0 0 0]	[0 15 0 5 0 0 0 0]
[0 0 41 0 0 1 0 0]	[0 0 40 0 0 1 0 0]	[0 1 52 0 0 0 0 0]
[0 0 0 7 0 0 0 0]	[0 2 0 6 0 1 0 0]	[0 2 0 4 0 0 0 0]
[0 0 0 0 0 2 0 0]	[0 0 1 0 1 5 0 0]	[0 0 0 0 0 4 0 0]
[0 0 3 0 0 17 0 0]	[0 2 3 2 0 9 0 0]	[0 0 1 0 0 13 0 0]
[0 0 1 0 0 0 0 0]	[0 0 0 0 0 1 0 0]	[0 0 0 0 0 0 0 0]
[0 0 1 0 0 0 0 0]	[1 0 0 0 0 0 0 0]	[0 0 0 0 0 1 0 0]
accuracy: 82%	accuracy: 76%	accuracy: 86%

從以上數據可以看出，random forest 的正確率普遍會比 decision tree 還要好一點。不論是樣本數小的 iris，或是樣本數、feature 數都較大的 ecoli 都呈現差不多的趨勢。這證明 random forest 用多棵樹隨機取樣預測並投票是有效果的。但缺點是犧牲了執行的效率，用 20 棵樹預測理論上就要比 decision tree 多花 20 倍的時間。

Experiments and Results:

以下分析之準確率皆為執行 5 次之平均。

Relative sizes of the training and validation subsets:

透過更改 train_test_split 的 test_size，就可以更改 training 和 validation 的比例。以下為 wine 在不同比例下的正確率:

Training size	Validation size	Decision tree	Random forest
0.9	0.1	94.33%	98%
0.7	0.3	91.33%	94.67%
0.5	0.5	92.67%	95%
0.3	0.7	85%	90.33%

由上表可以看出，當 training size 愈小、validation size 愈大時，正確率大致會下降。此結果符合預期，因為當 training 的資料愈多時，tree 看到的資料就更為全面，也就愈能做出正確的判斷。

Number of trees in the forest:

這次使用 target 和 feature 皆較複雜的 ecoli，較能看出變化:

Number of trees	5	10	15	20
Accuracy	78%	81.2%	83.6%	82.2%

由上表可以看出，用愈多棵樹進行投票，準確率大致會提升。而大概用 15 棵樹就可以達到最高的準確率。再用更多的樹將不會提升準確率，只會使執行效率更差。

Decision tree's depth:

將深度設為 tree 建構函式遞迴的中止條件可以限制 tree’s depth。理論上，tree 深度太小會造成 underfit，而深度過大會造成 overfit。以下為 wine 的 decision tree 在各深度的準確率:

depth	1	2	3	4	5	6	7	8
accuracy	56.4%	84.8%	90.2%	87.6%	90.4%	90.6%	87.8%	88.4%

由上表可看出，depth=1 時會明顯的 underfit，而 depth=2 就能做出很好的 node splitting，但還是稍稍 underfit；到 depth=7, 8 時會稍稍 overfit 但不明顯，是因為遞迴中止條件並不只有 depth，還有是當 node splitting 已經完全分出一類，也就是 gini 值等於 0 時，其遞迴也會中止。因此其深度根本無法達到太深，depth 超過一個臨界值就等同於無效了。

Rate of majority vote in random forest:

若將 20 棵樹都視為相等的效力，也就是 majority vote 中所謂的「一票」，這樣出來的結果，random forest 的正確率會和 decision tree 差不多、甚至更差，而不是預期的提升。這代表這組資料具有一定的雜亂程度，會容易讓每棵 decision tree 做出不同的選擇。而正確率較差的 tree 會影響到正確率較高的 tree，導致最後投出的 prediction 錯誤。因此，每棵樹不應該票票等值，而是要乘上它預測它要投的那個 target 的正確率。而此正確率可以在樹建出來時就順便算好，例如 tree bagging 時選擇 K 個 sample 進去 train，剩下的，也就是所謂的 out-of-bag samples，就拿來算正確率並存起來，等著 voting 時使用。

Extremely random forest:

每次 node splitting 時都選擇隨機一個 feature。要做到這件事只要將原本迭代 feature 的 for i in range(feature_size)改成 i = random.randint(0, feature_size-1)即可。這對於 feature 較多的 data 例如 ecoli，執行效率上會有大幅度的提升。然而隨機選取 feature 來做 splitting，找到的分法可能不會是最理想的，導致單一個 decision tree 的準確度變差。但 random forest 是所有 decision tree 的投票結果，準確度較差的 decision tree 拿到的票數也會較低，所以綜合來看，extremely random forest 的準確度和普通 random forest 不會差太多。以下數據為 wine 用 random feature splitting 的準確率:

Decision tree with random feature splitting	79%
Extremely random forest	98%

可以看出，單棵 decision tree 的準確率比起原本的 90 出頭大幅下降到了 79%，然而 random forest 依然能維持高檔的準確率。

Things I Have Learned:

這次作業中，我學會了如何建構一棵完整的 decision tree，並接著實作出 random forest。包含如何計算 gini、attribute bagging 找到最好的 node splitting 方法、forest 中利用 tree bagging 得出結果。最後是分析及比較各種 dataset、各種數據，並得出結論。

Remaining Questions:

剩下的問題主要是在 node splitting 的地方。我使用的 attribute bagging 方式一次只找一個 attribute，也就是只用一個條件來 splitting。但若是數據量龐大且複雜，有可能一個條件並不足以良好的切割，尤其是在 tree depth 或執行時間上有限制的時候。而若是一次使用多個條件進行 splitting，理論上能更有效率的找到更好的 splitting 方式、使 gini 值更低。

Ideas of Future Investigation:

不是每種 dataset 都能用 decision tree 和 random forest 做出最好的預測。還有許多 training model 可以用來嘗試，每種 dataset 都能依照其 feature、target 特性，找到一種最適合的 model。通常 training 需要大量的經驗和嘗試，才能讓準確率稍微上升一點點。而我們學習的目標就是能在盡量短的時間內找到最理想、能符合我們預期的 model。

Appendix:

```
import numpy as np
import pandas as pd
from sklearn import datasets, metrics
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import random

# iris data
def iris():
    iris = datasets.load_iris()
    # splitting data to training set and validation set
    x_train, x_test, y_train, y_test = train_test_split(iris.data, iris.target, test_size=0.3)
    return x_train, x_test, y_train, y_test
# wine data
def wine():
    wine = datasets.load_wine()
    x_train, x_test, y_train, y_test = train_test_split(wine.data, wine.target, test_size=0.3)
    return x_train, x_test, y_train, y_test
# ecoli data
def ecoli():
   # data preprocessing
    ecoli = pd.read_csv('ecoli.data', header=None)
    ecoli_data = []
    ecoli_target = []
    for i in range(len(ecoli)):
        a = ecoli[0][i].split()
        li = [float(i) for i in a[1:8]]
        ecoli_data.append(li)
        ecoli_target.append(a[8])

    ecoli_data = np.array(ecoli_data)

    echoli_target_t = list(set(ecoli_target))
    e = []
    for i in ecoli_target:
```

```
        for j in range(len(echoli_target_t)):
            if echoli_target_t[j] == i:
                e.append(j)
                break

    ecoli_target = np.array(e)
    x_train, x_test, y_train, y_test = train_test_split(ecoli_data, ecoli_target, test_size=0.3)
    return x_train, x_test, y_train, y_test

x_train, x_test, y_train, y_test = wine()        # choose which dataset to use
feature_size = len(x_train[0])                    # get number of feature
target_size = len(set(y_train))                   # get number of target

# tree node
class node:
    def __init__(self,feature,target):
        self.feature=feature
        self.target=target
        self.attr_ind=-1
        self.attr_val=-1
        self.left=None
        self.right=None
        self.attr=-1

    def get_attr(self):
        # calculate which target has the most samples in this node,
        # this node represent this target in prediction
        global target_size
        cnt = []
        for i in range(target_size):
            cnt.append(self.target.tolist().count(i))

        m = -1
        ind = -1
        for i in range(len(cnt)):
            if cnt[i] > m:
                m = cnt[i]
                ind = i
```

```
        self.attr = ind


    def is_terminate(self):
      # if this node doesn't need to split (gini == 0), terminate
        global target_size
        cnt = []
        for i in range(target_size):
            cnt.append(self.target.tolist().count(i))

        if cnt.count(0) == len(cnt) - 1:
            for i in range(len(cnt)):
                if cnt[i] != 0:
                    self.attr = i
                    break
            return True
        return False



def calc_gini(li,li_y):
  # do attribute bagging, try all feature and threshold, calculate and return the splitting condition
    global feature_size
    l = len(li_y)
    g_min = 10000
    g_ind = -1
    g_m = 10000

    if li.size == 0:
        return 0,0

    #i = random.randint(0,feature_size-1)        # extremely random forest, select attribute randomly
    for i in range(feature_size):                # random forest, iterate all attribute
        attr = li[:,i]
        tmp_attr = sorted(attr)
        m_li = []
        for j in range(len(attr)-1):
```

```
            m_li.append((tmp_attr[j]+tmp_attr[j+1])/2)      # sort and calculate all threshold
        # splitting and calculate gini
        for m in m_li:
            y_right = []
            y_left = []
            for j in range(l):
                if li[j][i] > m:
                    y_right.append(li_y[j])
                else:
                    y_left.append(li_y[j])

            gini_right = 0
            gini_left = 0
            if len(y_right) != 0:
                gini_right = 1 - ((y_right.count(0)/len(y_right))**2 + (y_right.count(1)/len(y_right))**2 +
(y_right.count(2)/len(y_right))**2)
            else:
                continue
            if len(y_left) != 0:
                gini_left = 1 - ((y_left.count(0)/len(y_left))**2 + (y_left.count(1)/len(y_left))**2 +
(y_left.count(2)/len(y_left))**2)
            else:
                continue
            gini = len(y_right) * gini_right + len(y_left) * gini_left
            if gini < g_min:
                g_min = gini
                g_ind = i
                g_m = m

    return g_ind, g_m       # return splitting condition (attribute and threshold)

# tree construction
def tree(n,depth):
    if n.is_terminate():      # terminate condition: gini == 0
        return

    if depth == 5:       # terminate condition: depth
        n.get_attr()
```

```
            return

        x = n.feature
        y = n.target
        a,m = calc_gini(x,y)    # get splitting condition
        n.attr_ind = a
        n.attr_val = m
        # node splitting
        y_right = []
        x_right = []
        y_left = []
        x_left = []
        for i in range(len(y)):
            if x[i][a] > m:
                y_right.append(y[i])
                x_right.append(x[i])
            else:
                y_left.append(y[i])
                x_left.append(x[i])

        x_left = np.array(x_left)
        y_left = np.array(y_left)
        x_right = np.array(x_right)
        y_right = np.array(y_right)

        node_left = node(x_left,y_left)
        node_right = node(x_right,y_right)
        n.left = node_left
        n.right = node_right
        # recursive call tree construction (construct left and right subtrees)
        tree(n.left, depth+1)
        tree(n.right, depth+1)

root = node(x_train,y_train)
tree(root,0)

# predict a value
def evaluation(root,t):
```

```
    curr = root
    ans = -1
    while curr != None:
        if t[curr.attr_ind] > curr.attr_val:
            ans = curr.attr
            curr = curr.right
        else:
            ans = curr.attr
            curr = curr.left

    return ans

correct = 0
pred = []
for i in range(len(y_test)):    # put all validation data into prediction: decision tree
    result = evaluation(root,x_test[i])
    pred.append(result)
    if result == y_test[i]:
        correct += 1

print('Decision Tree:')
print(confusion_matrix(y_test, pred))
print(classification_report(y_test, pred))

# calculate rate of vote
def confusion_matrix_element_precision(cm, i):
    try:
        cm2 = cm[i].tolist()
    except:
        return 0
    acc = 0
    if sum(cm2) != 0:
        acc = cm2[i]/sum(cm2)
    return acc

# random forest tree bagging
trees = []
ac_score = []
```

```
# construct 20 trees in a forest
for i in range(20):
    # randomly select 0.7 from training subset
    num = random.randint(1,1000)
    x_train1, x_test1, y_train1, y_test1 = train_test_split(x_train, y_train, test_size=0.3, random_state=num)
    root = node(x_train1,y_train1)
    tree(root,0)
    trees.append(root)
    pred = []
    for j in range(len(y_test1)):
        result = evaluation(root,x_test1[j])
        pred.append(result)
    # calculate out-of-bag and store rate of voting
    cm = confusion_matrix(y_test1, pred)
    ele = []
    for j in range(target_size):
        ele.append(confusion_matrix_element_precision(cm, j))
        ac_score.append(ele)

# put all validation data into prediction: random forest
rf_pred = []
for i in range(len(y_test)):
    vote = []
    for r in trees:       # voting
        res = evaluation(r,x_test[i])
        vote.append(res)
    #rf_pred.append(max(vote))    # majority vote
    # majority vote with different rate
    k = [0] * target_size
    for j in range(target_size):
        for v in range(20):
            if vote[v] == j:
                k[j] += ac_score[v][j]
    m = -1
    ind = -1
    for j in range(target_size):
        if k[j] > m:
            m = k[j]
```

```
                ind = j
        rf_pred.append(ind)


print('Random Forest:')
print(confusion_matrix(y_test, rf_pred))
print(classification_report(y_test, rf_pred))
```