Programming Assignment #4

0611031 資工 10 謝至恆

目標:

使用 random forests 實作 supervised learning，這次只要做分類問題的 learning，先實作 decision tree，要包括 induction part(使用 training data 建構 tree)，和 inference part (也就是驗算用的 samples)，只使用 real-valued attribute，使用 CART 方法來建構 tree。

解題思路:

根據不同的 attribute 分離 node，我們使用 Gini's impurity(將經過 attribute 後分類的資料，2 邊分別套入公式，全部相加後跟其他 attribute 比較)來檢查哪個 node 可以把資料分的最乾淨，優先選擇這個 attribute 來做上層的分類，就可以越快分類完成，使用這個方法一直循環下去就可以得到一個 decision tree。

接下來就要 build a forest，Random Forests(簡單來說，也就是利用 bagging 隨機【抽出後放回再繼續抽】的方式，建構一些 decision tree，最後再從這些 trees 中取平均，因為是抽出再放回，所以有可能會重複抽取，也比較不會出現完全一樣的組合，這樣一來就可以在現有的資料裡面創造出更多的資料組合也就是 dataset，這樣一來再 learning 上就有很大的優勢(增加 training 的數量)，希望可以得到一個比較折衷，且較正確的結果)，並使用 tree bagging(random select samples)和 attribute bagging(random select attribute)來觀察 forest 的結果。

CART:

a.決策樹生成：基於訓練數據生成決策樹，決策樹盡量大

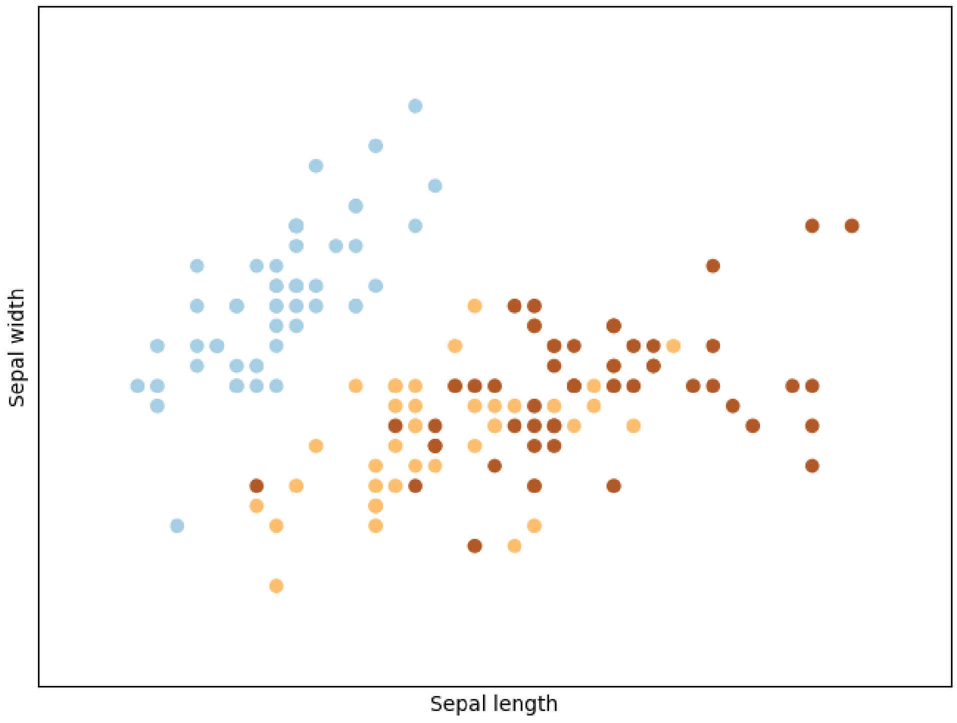
(使用 gini 係數來求得)

b.決策樹剪枝：用驗證數據集對已生成的樹進行剪枝並選擇最優子樹，用損失函數最小作為剪枝的標準

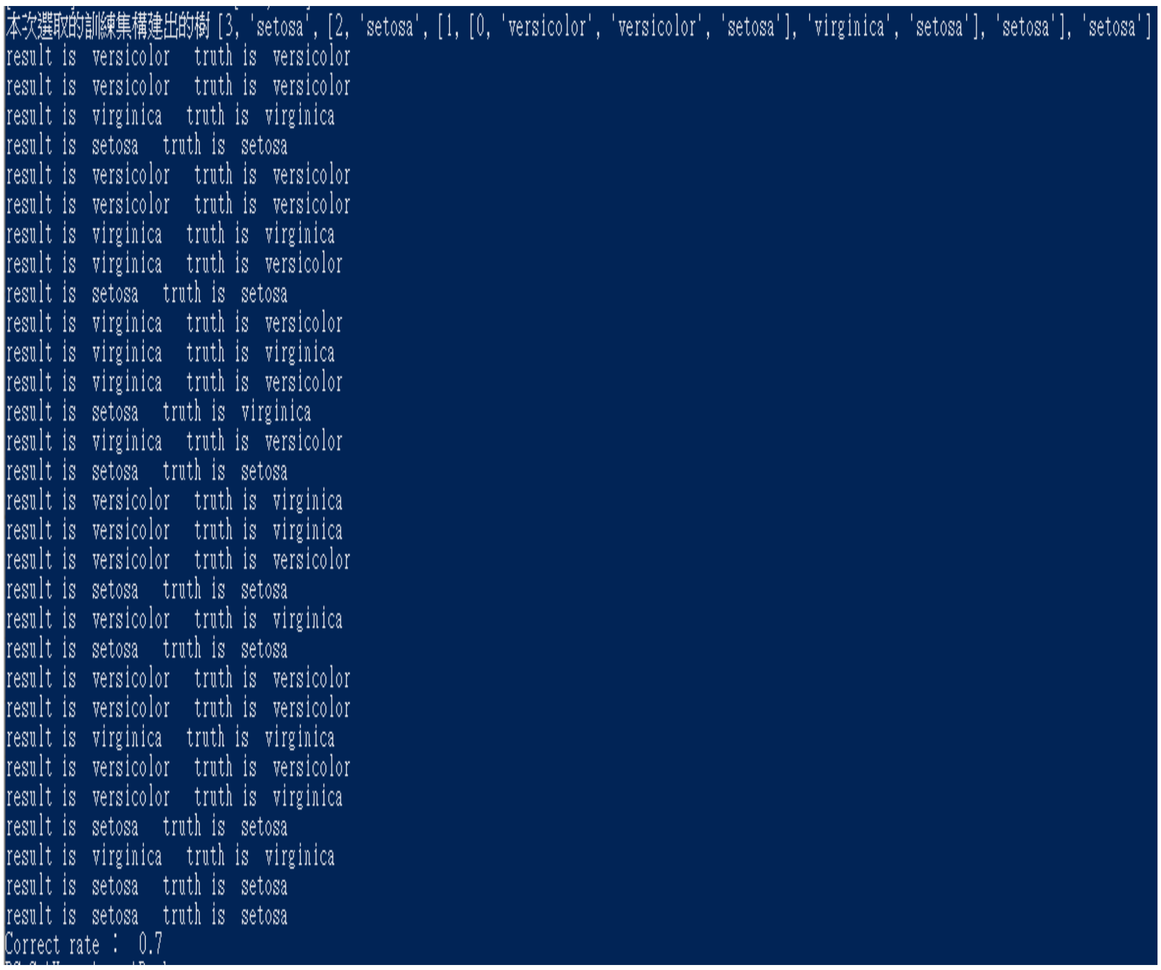
採用 CCP（代價複雜度）剪枝方法。代價複雜度選擇節點表面誤差率增益值最小的非葉子節點，刪除該非葉子節點的左右子節點，若有多個非葉子節點的表面誤差率增益值相同小，則選擇非葉子節點中子節點數最多的非葉子節點進行剪枝。

實作:

這次的作業我是使用 python 來做，因為這方面的 code 我比較不熟悉，上網查了許多資料，發現 python 的資料最豐富也比較詳細，加上普遍 code 比較短衣些，所以我也就用 python 了，我是使用-鳶尾花 iris 的 dataset，(聽說這個比較簡單?)
裡面有 3 種不同的鳶尾花，裡面有 150 筆資料，4 種特徵(分別為 萼片(sepal)之長與寬以及花瓣(petal)之長與寬)。



根據上網查到的 source code 來做改寫。
先將 data load 進來，取前 120 做 traing data，30 個做 validation data，用來剪枝。

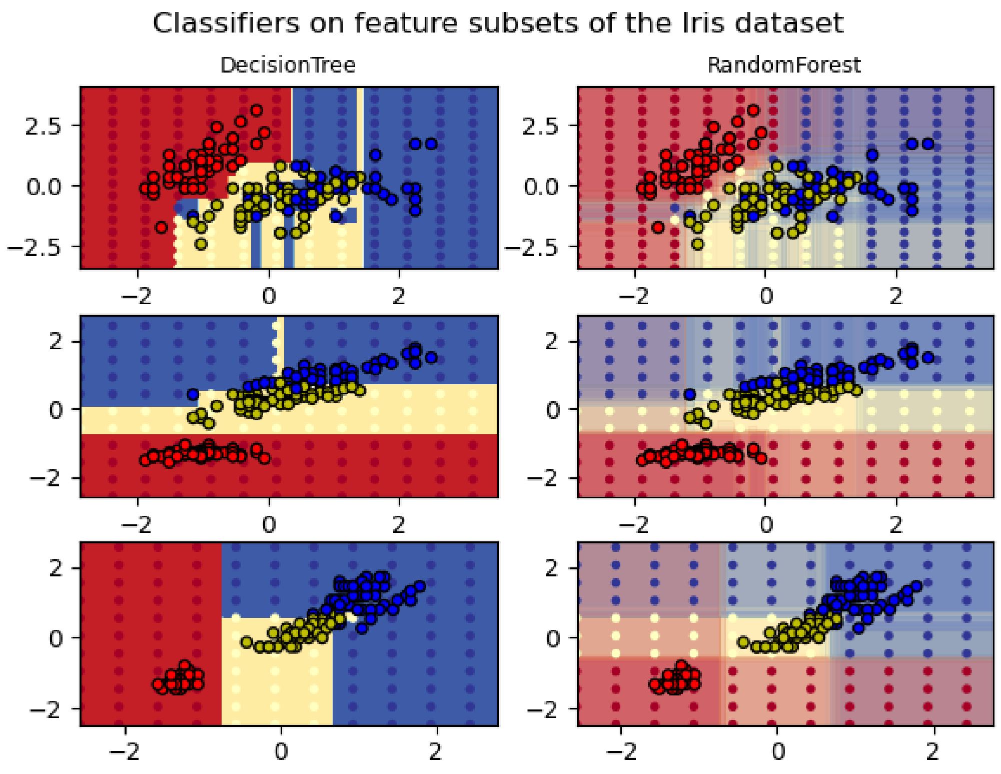


```
本次選取的訓練集構建出的樹 [3, 'setosa', [2, 'setosa', [1, [0, 'versicolor', 'versicolor', 'setosa'], 'virginica', 'setosa'], 'setosa'], 'setosa']
result is  versicolor   truth is  versicolor
result is  versicolor   truth is  versicolor
result is  virginica   truth is  virginica
result is  setosa   truth is  setosa
result is  versicolor   truth is  versicolor
result is  versicolor   truth is  versicolor
result is  virginica   truth is  virginica
result is  virginica   truth is  versicolor
result is  setosa   truth is  setosa
result is  virginica   truth is  versicolor
result is  virginica   truth is  virginica
result is  virginica   truth is  versicolor
result is  setosa   truth is  virginica
result is  virginica   truth is  versicolor
result is  setosa   truth is  setosa
result is  versicolor   truth is  virginica
result is  versicolor   truth is  virginica
result is  versicolor   truth is  versicolor
result is  setosa   truth is  setosa
result is  versicolor   truth is  virginica
result is  setosa   truth is  setosa
result is  versicolor   truth is  versicolor
result is  versicolor   truth is  versicolor
result is  virginica   truth is  virginica
result is  versicolor   truth is  versicolor
result is  versicolor   truth is  virginica
result is  setosa   truth is  setosa
result is  virginica   truth is  virginica
result is  setosa   truth is  setosa
result is  setosa   truth is  setosa
Correct rate :  0.7
```

把每個 attribute 使用 Gini 方式計算出的資料給予的資訊量，選取最大的來做劃分，以此類推來得到 decision tree。

Random forest 的部分，採用 tree bagging 的方式就是先從所有資料隨機抽取 120 個資料(有可能選到重複的資料)做 training data，再抽 30 個 validation data，來建構 decision tree，取 30 個深度為 3 的 tree，做平均來得到一個 tree。Attribute bagging 就是隨機抽 1 個 attribute 來生成 trees 再求平均，這樣就可以得到一個學到比較少 noise 但又可以保證 signal 趨於大部分 sample 的 tree。

程式結果



在第一行中，僅使用萼片寬度和長度特徵構建分類器，在第二行中僅使用花瓣長度和萼片長度構建分類器，在第三行中僅使用花瓣寬度和花瓣長度構建分類器。可以看出 第 3 行>第 2 行>第 1 行的分類結果

心得；

這次作業的觀念實際上在課堂上就大部分都已經了解了，不過要實作還是有點不知道該怎麼下手，上網查了很多資料，這次的作業比較難的部分我覺得是將上網查的 code 看懂並改成作業要求的形式。不過看到最後結果有跑出來還蠻有成就感的，decision tree 的部分本身不難，random forest 的實作就比較複雜了，雖然書上原理好像蠻簡單的，但實際要做的程式碼我看了蠻久才比較了解。
這次作業讓我更加理解上課的內容，任我對 Learning 的辦法了解了更多。

參考資料

決策樹- Cart + iris 數據集+ python 實現

https://blog.csdn.net/weixin_43909872/article/details/86027350

演算法-隨機森林分類(RandomForestClassifier)

http://www.taroballz.com/2019/05/25/ML_RandomForest_Classifier/

The iris 鳶尾花資料集
https://machine-learning-python.kspax.io/datasets/ex3_the_iris_dataset
Plot the decision surfaces of ensembles of trees on the iris dataset
https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_iris.html

决策树分类鸢尾花数据集 python 实现

https://blog.csdn.net/zhoulei124/article/details/88994445?utm_medium=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase&depth_1-utm_source=distribute.pc_relevant.none-task-blog-BlogCommendFromMachineLearnPai2-1.nonecase

Decision Tree 實作和 **Random Forest(**觀念**)**
https://medium.com/@jacky308082/machine-learning%E4%B8%8B%E7%9A%84decision-tree%E5%AF%A6%E4%BD%9C%E5%92%8Crandom-forest-%E8%A7%80%E5%BF%B5-%E4%BD%BF%E7%94%A8python-3a94cef2ce6f

Code:

```
from sklearn import datasets
import math
import numpy as np



def getInformationEntropy(arr,leng):

    return -(arr[0]/leng*math.log(arr[0]/leng if arr[0]>0 else 1)+
arr[1]/leng*math.log(arr[1]/leng if arr[1]>0 else 1)+
arr[2]/leng*math.log(arr[2]/leng if arr[2]>0 else 1))

def discretization(index):

    feature1 = np.array([iris.data[:,index],iris.target]).T
    feature1 = feature1[feature1[:,0].argsort()]

    counter1 = np.array([0,0,0])
    counter2 = np.array([0,0,0])

    resEntropy = 100000
    for i in range(len(feature1[:,0])):

        counter1[int(feature1[i,1])] = counter1[int(feature1[i,1])] + 1
        counter2 = np.copy(counter1)

        for j in range(i+1,len(feature1[:,0])):

            counter2[int(feature1[j,1])] =    counter2[int(feature1[j,1])] + 1
            if i != j and j != len(feature1[:,0])-1:

                sum = (i+1)*getInformationEntropy(counter1,i+1) +
(j-i)*getInformationEntropy(counter2-counter1,j-i) +                    (length-j-
1)*getInformationEntropy(np.array(num)-counter2,length-j-1)
                if sum < resEntropy:
                    resEntropy = sum
                    res = np.array([i,j])
    res_value = [feature1[res[0],0],feature1[res[1],0]]
    print(res,resEntropy,res_value)
    return res_value

def getRazors():
    a = []
    for i in range(len(iris.feature_names)):
        print(i)
        a.append(discretization(i))

    return np.array(a)

#隨機抽取 120 的 training data set 和 30 的 validation data set
def divideData():
    completeData = np.c_[iris.data,iris.target.T]
    np.random.shuffle(completeData)
    trainData = completeData[range(int(length*0.8)),:]
    testData = completeData[range(int(length*0.8),length),:]
    return [trainData,testData]

def getEntropy(counter):

    res = 0
    denominator = np.sum(counter)
    if denominator == 0:
        return 0
    for value in counter:
        if value == 0:
            continue
        res += value/denominator * math.log(value/denominator if value>0 and
denominator>0 else 1)
    return -res


def findMaxIndex(dataSet):
    maxIndex = 0
    maxValue = -1
    for index,value in enumerate(dataSet):
```

```
        if value>maxValue:
            maxIndex = index
            maxValue = value
    return maxIndex



def recursion(featureSet,dataSet,counterSet):

    if(counterSet[0]==0 and counterSet[1]==0 and counterSet[2]!=0):
        return iris.target_names[2]
    if(counterSet[0]!=0 and counterSet[1]==0 and counterSet[2]==0):
        return iris.target_names[0]
    if(counterSet[0]==0 and counterSet[1]!=0 and counterSet[2]==0):
        return iris.target_names[1]

    if len(featureSet) == 0:
        return iris.target_names[findMaxIndex(counterSet)]
    if len(dataSet) == 0:
        return []

    res = 1000
    final = 0
    for feature in featureSet:
        i = razors[feature][0]
        j = razors[feature][1]
        set1 = []
        set2 = []
        set3 = []
        counter1 = [0,0,0]
        counter2 = [0,0,0]
        counter3 = [0,0,0]
        for data in dataSet:
            index = int(data[-1])
            if data[feature]< i :
                set1.append(data)
                counter1[index] = counter1[index]+1
            elif data[feature] >= i and data[feature] <=j:
                set2.append(data)
                counter2[index] = counter2[index]+1
            else:
                set3.append(data)
                counter3[index] = counter3[index]+1


        a =( len(set1)*getEntropy(counter1) +
len(set2)*getEntropy(counter2) +                  len(set3)*getEntropy(counter3) )/
len(dataSet)

        if a<res :
            res = a
            final = feature


    #返回被選中的特徵的下標
    #sequence.append(final)
    featureSet.remove(final)
    child = [0,0,0,0]
    child[0] = final
    child[1] = recursion(featureSet,set1,counter1)
    child[2] = recursion(featureSet,set2,counter2)
    child[3] = recursion(featureSet,set3,counter3)

    return child


def judge(data,tree):

    root = "unknow"
    while(len(tree)>0):
        if isinstance(tree,str) and tree in iris.target_names:
            return tree
        root = tree[0]
        if(isinstance(root,str)):
            return root
        if isinstance(root,int):
            if data[root]<razors[root][0] and tree[1] != [] :
                tree = tree[1]
            elif tree[2] != [] and (tree[1]==[] or (data[root]>=razors[root][0] and
data[root]<=razors[root][1])):
                tree = tree[2]
            else :
                tree = tree[3]
    return root


if __name__ == '__main__':

    iris = datasets.load_iris()
    num = [0,0,0]
    for row in iris.data:
        num[int(row[-1])] = num[int(row[-1])] + 1

    length = len(iris.target)
    [trainData,testData] = divideData()

    razors = getRazors()

    tree = recursion(list(range(len(iris.feature_names))),
trainData,[np.sum(trainData[:,-1]==0),            np.sum(trainData[:,-
1]==1),np.sum(trainData[:,-1]==2)])
    print("本次選取的訓練集構建出的樹",tree)
    index = 0
    right = 0
    for data in testData:
        result = judge(testData[index],tree)
        truth = iris.target_names[int(testData[index][-1])]

        print("result is ",result ,"    truth is ",truth)
        index = index + 1
        if result == truth:
            right = right + 1
    print("Correct rate  ：  ",right/index)
    print(__doc__)

import numpy as np
import matplotlib.pyplot as plt
from matplotlib.colors import ListedColormap

from sklearn.datasets import load_iris
from sklearn.ensemble import (RandomForestClassifier)
from sklearn.tree import DecisionTreeClassifier

# Parameters
n_classes = 3
n_estimators = 30
cmap = plt.cm.RdYlBu
plot_step = 0.02    # fine step width for decision surface contours
plot_step_coarser = 0.5    # step widths for coarse classifier guesses
RANDOM_SEED = 13    # fix the seed on each iteration

# Load data
iris = load_iris()

plot_idx = 1

models = [DecisionTreeClassifier(max_depth=None),
             RandomForestClassifier(n_estimators=n_estimators),
                                          ]

for pair in ([0, 1], [0, 2], [2, 3]):
    for model in models:
        # We only take the two corresponding features
        X = iris.data[:, pair]
        y = iris.target

        # Shuffle
        idx = np.arange(X.shape[0])
        np.random.seed(RANDOM_SEED)
        np.random.shuffle(idx)
```

```
X = X[idx]
y = y[idx]

# Standardize
mean = X.mean(axis=0)
std = X.std(axis=0)
X = (X - mean) / std

# Train
model.fit(X, y)

scores = model.score(X, y)
# Create a title for each column and the console by using str() and
# slicing away useless parts of the string
model_title = str(type(model)).split(
        ".")[-1][:-2][:-len("Classifier")]

model_details = model_title
if hasattr(model, "estimators_"):
        model_details += " with {} estimators".format(
                len(model.estimators_))
print(model_details + " with features", pair,
            "has a score of", scores)

plt.subplot(3, 2, plot_idx)
if plot_idx <= len(models):
        # Add a title at the top of each column
        plt.title(model_title, fontsize=9)

# Now plot the decision boundary using a fine mesh as input to a
# filled contour plot
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.arange(x_min, x_max, plot_step),
                                      np.arange(y_min, y_max, plot_step))

# Plot either a single DecisionTreeClassifier or alpha blend the
# decision surfaces of the ensemble of classifiers
```

```
        if isinstance(model, DecisionTreeClassifier):
            Z = model.predict(np.c_[xx.ravel(), yy.ravel()])
            Z = Z.reshape(xx.shape)
            cs = plt.contourf(xx, yy, Z, cmap=cmap)
        else:
            # Choose alpha blend level with respect to the number
            # of estimators
            # that are in use (noting that AdaBoost can use fewer estimators
            # than its maximum if it achieves a good enough fit early on)
            estimator_alpha = 1.0 / len(model.estimators_)
            for tree in model.estimators_:
                Z = tree.predict(np.c_[xx.ravel(), yy.ravel()])
                Z = Z.reshape(xx.shape)
                cs = plt.contourf(xx, yy, Z, alpha=estimator_alpha, cmap=cmap)

        # Build a coarser grid to plot a set of ensemble classifications
        # to show how these are different to what we see in the decision
        # surfaces. These points are regularly space and do not have a
        # black outline
        xx_coarser, yy_coarser = np.meshgrid(
            np.arange(x_min, x_max, plot_step_coarser),
            np.arange(y_min, y_max, plot_step_coarser))
        Z_points_coarser = model.predict(np.c_[xx_coarser.ravel(),
                                               yy_coarser.ravel()]
                                               ).reshape(xx_coarser.shape)
        cs_points = plt.scatter(xx_coarser, yy_coarser, s=15,
                                c=Z_points_coarser, cmap=cmap,
                                edgecolors="none")

        # Plot the training points, these are clustered together and have a
        # black outline
        plt.scatter(X[:, 0], X[:, 1], c=y,
                    cmap=ListedColormap(['r', 'y', 'b']),
                    edgecolor='k', s=20)
        plot_idx += 1    # move on to the next plot in sequence

plt.suptitle("Classifiers on feature subsets of the Iris dataset", fontsize=12)
plt.axis("tight")
plt.tight_layout(h_pad=0.2, w_pad=0.2, pad=2.5)
plt.show()
```