人工智慧概論程式作業3報告

陳羿豐

實作決策樹

我使用 Python 語言來實作決策樹，因為我覺得 Python 是機器學習的標準語言。

程式碼放在附錄，使用方法是：

1. import cart 載入我的程式

2. tree = cart.trainCART(dataset, featureSet, numClasses, minSize) 會用輸入的資料來訓練，並把決策樹存到 tree 變數中

dataset 是二維陣列，dataset[i][j] 表示第 i 筆資料的第 j 個特徵，是浮點數，dataset[i][-1] 則是第 i 筆資料的分類結果，以整數表示，在 Python 中，第 -1 項是最後一項。

featureSet 表示要作為分類用的特徵，是一個整數陣列，整數值是特徵編號

numClasses 是分類結果的種類數，至少是 2。要求 0 ≤ dataset[i][-1] < numClasses

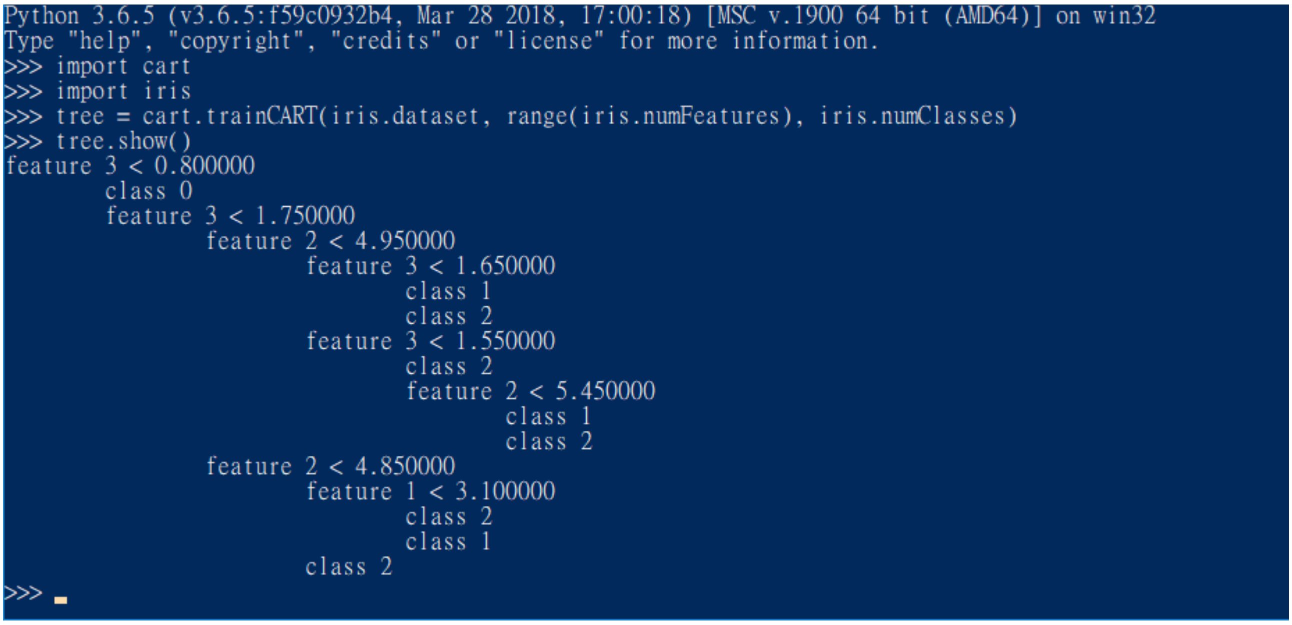
minSize 表示分支所需的最少資料數，如果沒有指定，則預設為 1，也就是不限最少資料數

3. tree.show() 顯示決策樹的結構

4. tree.predict(record) 對新的樣本作預測，回傳分類結果，以整數表示。record 是一維陣列，是一筆新資料。

執行圖中程式之前，請確認 cart.py、iris.py、iris.txt、cross200.py、cross200.txt、optical-digits.txt、opticaldigits.py 都在目前的工作目錄

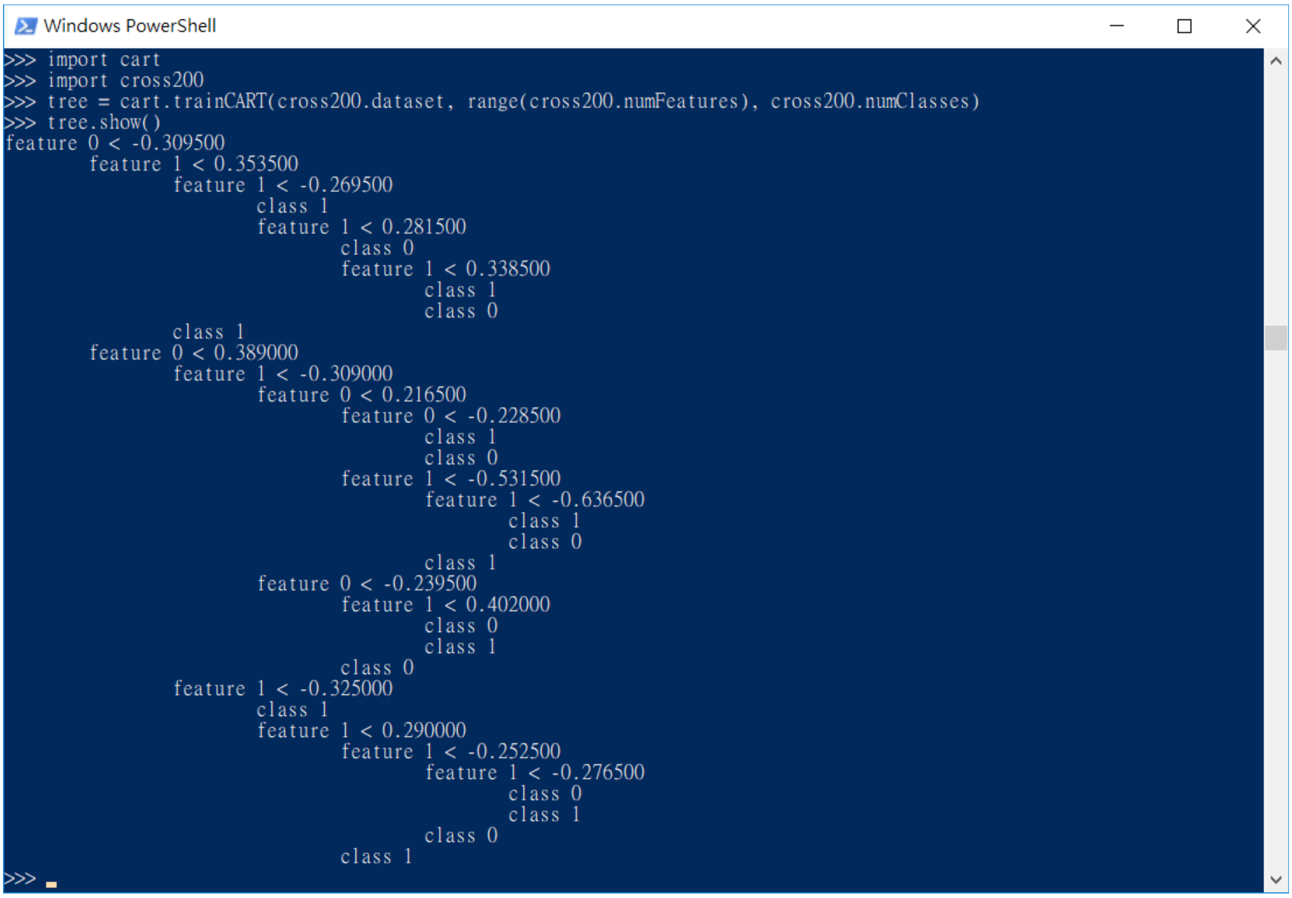
顯示 iris.txt 產生的決策樹



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import cart
>>> import iris
>>> tree = cart.trainCART(iris.dataset, range(iris.numFeatures), iris.numClasses)
>>> tree.show()
feature 3 < 0.800000
        class 0
        feature 3 < 1.750000
                feature 2 < 4.950000
                        feature 3 < 1.650000
                                class 1
                                class 2
                        feature 3 < 1.550000
                                class 2
                                feature 2 < 5.450000
                                        class 1
                                        class 2
                feature 2 < 4.850000
                        feature 1 < 3.100000
                                class 2
                                class 1
                        class 2
>>> 
```

其中 class 0 是 Iris-setosa，class 1 是 Iris-versicolor，class 2 是 Iris-virginica

顯示 cross200.txt 產生的決策樹



```
>>> import cart
>>> import cross200
>>> tree = cart.trainCART(cross200.dataset, range(cross200.numFeatures), cross200.numClasses)
>>> tree.show()
feature 0 < -0.309500
        feature 1 < 0.353500
                feature 1 < -0.269500
                        class 1
                        feature 1 < 0.281500
                                class 0
                                feature 1 < 0.338500
                                        class 1
                                        class 0
                class 1
        feature 0 < 0.389000
                feature 1 < -0.309000
                        feature 0 < 0.216500
                                feature 0 < -0.228500
                                        class 1
                                        class 0
                                feature 1 < -0.531500
                                        feature 1 < -0.636500
                                                class 1
                                                class 0
                                        class 1
                        feature 0 < -0.239500
                                feature 1 < 0.402000
                                        class 0
                                        class 1
                                class 0
                feature 1 < -0.325000
                        class 1
                        feature 1 < 0.290000
                                feature 1 < -0.252500
                                        feature 1 < -0.276500
                                                class 0
                                                class 1
                                        class 0
                                class 1
>>>
```

其中 class 0 是 1，class 1 是 2

optical-digits.txt 產生的決策樹太大了，無法在這裡呈現，不過可以用這個指令輸出

```
import cart
import opticaldigits
tree = cart.trainCART(opticaldigits.dataset, range(opticaldigits.numFeatures),
opticaldigits.numClasses)
tree.show()
```

建立隨機森林 (Random Forest)

隨機森林的建立方法是建立好幾個決策樹，每個決策樹使用的特徵都是隨機指定的，大約有總共特徵數量的開根號個。

執行方法

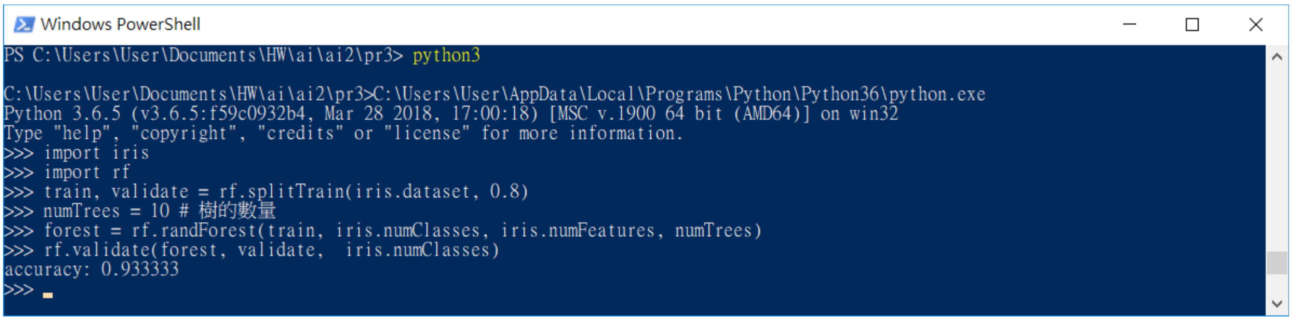
請確認 rf.py 在目前作用中的資料夾

```
import iris
import rf
train, validate = rf.splitTrain(iris.dataset, 0.8) # train data 佔 80%
numTrees = 10 # 樹的數量
forest = rf.randForest(train, iris.numClasses, iris.numFeatures, numTrees)
rf.validate(forest, validate,  iris.numClasses)
```

如果要處理其他的資料，就把 iris 改成別的名字

結果

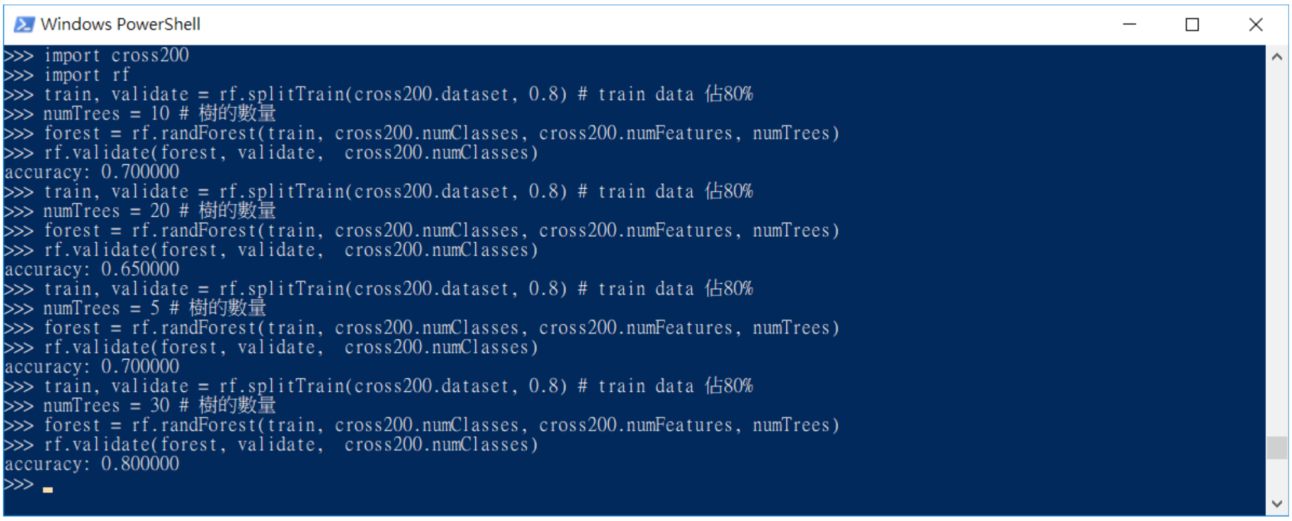
iris



```
PS C:\Users\User\Documents\HW\ai\ai2\pr3> python3

C:\Users\User\Documents\HW\ai\ai2\pr3>C:\Users\User\AppData\Local\Programs\Python\Python36\python.exe
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import iris
>>> import rf
>>> train, validate = rf.splitTrain(iris.dataset, 0.8)
>>> numTrees = 10 # 樹的數量
>>> forest = rf.randForest(train, iris.numClasses, iris.numFeatures, numTrees)
>>> rf.validate(forest, validate,  iris.numClasses)
accuracy: 0.933333
>>> _
```

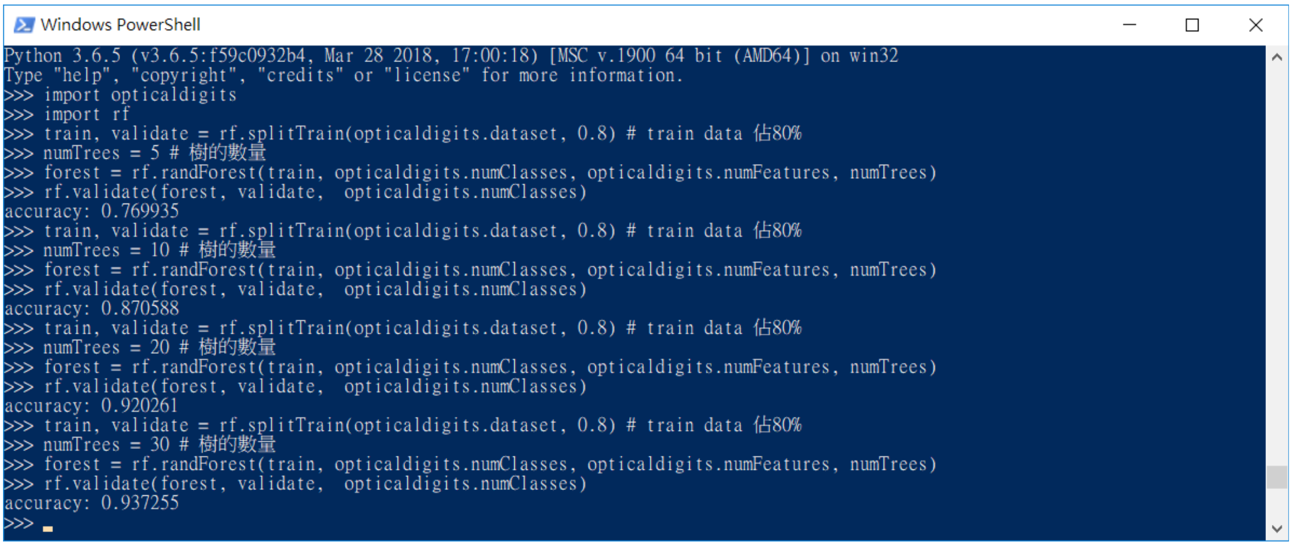
cross200



```
>>> import cross200
>>> import rf
>>> train, validate = rf.splitTrain(cross200.dataset, 0.8) # train data 佔80%
>>> numTrees = 10 # 樹的數量
>>> forest = rf.randForest(train, cross200.numClasses, cross200.numFeatures, numTrees)
>>> rf.validate(forest, validate,  cross200.numClasses)
accuracy: 0.700000
>>> train, validate = rf.splitTrain(cross200.dataset, 0.8) # train data 佔80%
>>> numTrees = 20 # 樹的數量
>>> forest = rf.randForest(train, cross200.numClasses, cross200.numFeatures, numTrees)
>>> rf.validate(forest, validate,  cross200.numClasses)
accuracy: 0.650000
>>> train, validate = rf.splitTrain(cross200.dataset, 0.8) # train data 佔80%
>>> numTrees = 5 # 樹的數量
>>> forest = rf.randForest(train, cross200.numClasses, cross200.numFeatures, numTrees)
>>> rf.validate(forest, validate,  cross200.numClasses)
accuracy: 0.700000
>>> train, validate = rf.splitTrain(cross200.dataset, 0.8) # train data 佔80%
>>> numTrees = 30 # 樹的數量
>>> forest = rf.randForest(train, cross200.numClasses, cross200.numFeatures, numTrees)
>>> rf.validate(forest, validate,  cross200.numClasses)
accuracy: 0.800000
>>> _
```

為什麼樹有 20 個的時候，準確度最糟？

optical-digits



```
Python 3.6.5 (v3.6.5:f59c0932b4, Mar 28 2018, 17:00:18) [MSC v.1900 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import opticaldigits
>>> import rf
>>> train, validate = rf.splitTrain(opticaldigits.dataset, 0.8) # train data 佔80%
>>> numTrees = 5 # 樹的數量
>>> forest = rf.randForest(train, opticaldigits.numClasses, opticaldigits.numFeatures, numTrees)
>>> rf.validate(forest, validate,  opticaldigits.numClasses)
accuracy: 0.769935
>>> train, validate = rf.splitTrain(opticaldigits.dataset, 0.8) # train data 佔80%
>>> numTrees = 10 # 樹的數量
>>> forest = rf.randForest(train, opticaldigits.numClasses, opticaldigits.numFeatures, numTrees)
>>> rf.validate(forest, validate,  opticaldigits.numClasses)
accuracy: 0.870588
>>> train, validate = rf.splitTrain(opticaldigits.dataset, 0.8) # train data 佔80%
>>> numTrees = 20 # 樹的數量
>>> forest = rf.randForest(train, opticaldigits.numClasses, opticaldigits.numFeatures, numTrees)
>>> rf.validate(forest, validate,  opticaldigits.numClasses)
accuracy: 0.920261
>>> train, validate = rf.splitTrain(opticaldigits.dataset, 0.8) # train data 佔80%
>>> numTrees = 30 # 樹的數量
>>> forest = rf.randForest(train, opticaldigits.numClasses, opticaldigits.numFeatures, numTrees)
>>> rf.validate(forest, validate,  opticaldigits.numClasses)
accuracy: 0.937255
>>> _
```

本來想說，optical-digits 是空間資料，決策樹應該會表現得很糟才是，不過在這裡，卻有 90%以上的精準度，很神奇

附錄：

程式碼

cart.py

用 CART 演算法建立決策樹

```
class CART:
    def __init__(self):
        self.left = None
        self.right = None
        self.feature = 0
        self.threshold = 0.0
        self.classify = -1 # -1 means not leaf, >= 0 means output class
    def leaf(self, output):
        self.classify = output
        return self
    def branch(self, left, right, feature, threshold):
        self.left = left
        self.right = right
        self.feature = feature
        self.threshold = threshold
        return self
    def show(self, indent = 0):
        if self.classify >= 0:
            print ("%sclass %d" % ("\t" * indent, self.classify))
        else:
            print ("%sfeature %d < %f" % ("\t" * indent, self.feature, self.threshold))
            self.left.show(indent+1)
            self.right.show(indent+1)
    def predict(self, record):
        if self.classify >= 0:
            return self.classify
        if record[self.feature] < self.threshold:
            return self.left.predict(record)
        else:
            return self.right.predict(record)

def trainCART(data, features, numClasses, minSize = 1):
    n = len(data)
    minGini = n + 1
    featureToUse = None
    threshold = 0.0
    classCount = [0] * numClasses
    # count number of each class
    for rec in data:
        output = rec[-1]
        classCount[output] += 1
    # if all data belong to one class, return a leaf node
    for c in range(len(classCount)):
        if classCount[c] == n:
            return CART().leaf(c)
    for f in features:
        countSmall = [0] * len(classCount)
        countBig = classCount[0:]
        sortedData = sorted(data, key = lambda rec: rec[f])
        # test each threshold
        for i in range(1, len(data)):
            # count each output class
            output = sortedData[i-1][-1]
            countSmall[output] += 1
            countBig[output] -= 1
            # check feature value
            prev = sortedData[i-1][f]
            current = sortedData[i][f]
            # if previous value is the same, it cannot be a threshold
            if prev == current:
                continue
            # calculate Gini index
            gini = computeGini(countSmall, i) * i + computeGini(countBig, n-i) * (n-i)
            if gini <= minGini:
                minGini = gini
                featureToUse = f
                threshold = (prev + current) / 2.0
    # no way to split
    if featureToUse == None or len(data) < minSize:
        m = 0
        choose = 0
        for c in range(numClasses):
            if classCount[c] > m:
                m = classCount[c]
                choose = c
        return CART().leaf(choose)
    # split data
    left = []
    right = []
    for rec in data:
        if rec[featureToUse] < threshold:
            left.append(rec)
        else:
            right.append(rec)
    leftTree = trainCART(left, features, numClasses, minSize)
    rightTree = trainCART(right, features, numClasses, minSize)
    return CART().branch(leftTree, rightTree, featureToUse, threshold)

def computeGini(count, size):
    return 1.0 - sum(nc * nc for nc in count) / float(size * size)
```

iris.py

處理 iris.txt 的資料

```
numFeatures = 4
numClasses = 3
```

```
def convertClassName(name):
    classes = {
        "Iris-setosa": 0 ,
        "Iris-setosa*": 0, # I don't know why the dataset provided by teacher has this
        "Iris-versicolor": 1,
        "Iris-virginica": 2
    }
    return classes[name]

file = open("iris.txt", "r")
dataset = []
for line in file.readlines():
    line = line.rstrip()
    if line == "": continue # skip empty line
    record = line.split(',')
    out = record[-1]
    record[-1] = convertClassName(out)
    for i in range(numFeatures):
        record[i] = float(record[i])
    dataset.append(record)
```

cross200.py

處理 cross200 的資料

```
numFeatures = 2
numClasses = 2
def convertClassName(name):
    classes = {
        "1": 0,
        "2": 1,
    }
    return classes[name]

file = open("cross200.txt", "r")
dataset = []
for line in file.readlines():
    line = line.strip()
    if line == "": continue # skip empty line
    record = line.split()
    out = record[-1]
    record[-1] = convertClassName(out)
    for i in range(numFeatures):
        record[i] = float(record[i])
    dataset.append(record)
```

opticaldigits.py

處理 optical-digits.txt 的資料

```
numFeatures = 64
numClasses = 10
```

```
def convertClassName(name):
    classes = {
        "0": 0,
        "1": 1,
        "2": 2,
        "3": 3,
        "4": 4,
        "5": 5,
        "6": 6,
        "7": 7,
        "8": 8,
        "9": 9
    }
    return classes[name]

file = open("optical-digits.txt", "r")
dataset = []
for line in file.readlines():
    line = line.strip()
    if line == "": continue # skip empty line
    record = line.split(',')
    out = record[-1]
    record[-1] = convertClassName(out)
    for i in range(numFeatures):
        record[i] = float(record[i])
    dataset.append(record)
```

rf.py

Random Forest 的實作和計算成效的程式

```
import cart
import random
import math

# returns tuple (train data, validate data)
def splitTrain(dataset, trainRatio = 0.8):
        n = len(dataset)
        train = []
        validate = list(range(n))
        trainSize = int(len(dataset) * trainRatio)
        for i in range(trainSize):
                r = random.randint(0, n-1-i)
                train.append(validate[r])
                validate[r] = validate[n-1-i]
        validate = validate[0 : n-trainSize]
        train = [dataset[i] for i in train]
        validate = [dataset[i] for i in validate]
        return (train, validate)
```

```
def randForest(dataset, numClasses, numFeatures, numTrees):
    forest = []
    featCount = int(math.sqrt(numFeatures))
    for i in range(numTrees):
        attributes = random.sample(range(numFeatures), featCount)
        tree = cart.trainCART(dataset, attributes, numClasses)
        forest.append(tree)
    return forest

def predict(forest, record, numClasses):
    count = [0] * numClasses
    for tree in forest:
        ans = tree.predict(record)
        count[ans] += 1
    m = max(count)
    for i in range(numClasses):
        if count[i] == m:
            return i

def validate(forest, validate, numClasses):
    total = len(validate)
    correct = 0
    for record in validate:
        ans = predict(forest, record, numClasses)
        if ans == record[-1]:
            correct += 1
    print ("accuracy: %f" % (float(correct)/total))
```