0616038 蔡雨恩 AI HW4 Report

實作方法：因為在修機器學習時有寫過類似的作業，所以就將那次的作業做成基底，在修改一些資料處理與 random forest 等等的方面，就能完成這次作業。這次使用的 database 是 UCI Machine Learning Repository 上的 Heart failure clinical records Data Set，這次的 train 跟 test data 分割方式有 Holdout validation with the ratio 7:3 與 K-fold cross-validation with K = 3 這兩種方式，以下是程式執行的結果，n 代表執行 tree 的次數，k 代表樹的數量。

n=100, k=3

```
Holdout validation with the ratio 7:3
1. Accuracy: 1.0
2. Sensitivity(Recall): 1.0
3. Precision: 1.0
K-fold cross-validation with K = 3
1. Accuracy: 1.0
2. Sensitivity(Recall): 1.0
3. Precision: 1.0
```

n=5, k=60

```
Holdout validation with the ratio 7:3
1. Accuracy: 1.0
2. Sensitivity(Recall): 1.0
3. Precision: 1.0
```

```
K-fold cross-validation with K = 3
1. Accuracy: 1.0
2. Sensitivity(Recall): 1.0
3. Precision: 1.0
```

n=3, k=100

```
Holdout validation with the ratio 7:3
1. Accuracy: 1.0
2. Sensitivity(Recall): 1.0
3. Precision: 1.0
K-fold cross-validation with K = 3
1. Accuracy: 1.0
2. Sensitivity(Recall): 1.0
3. Precision: 1.0
```

n=1, k=300

```
Holdout validation with the ratio 7:3
1. Accuracy: 1.0
2. Sensitivity(Recall): 1.0
3. Precision: 1.0
K-fold cross-validation with K = 3
1. Accuracy: 1.0
2. Sensitivity(Recall): 1.0
3. Precision: 1.0
```

從結果發現不管 n 與 k 的值如何改變準確度都是 100%，我推測應該是因為 data 的資料數太少了(只有 300)。

```
import csv
import math
import random

featuretype = [1, 0, 1, 0, 1, 0, 0, 0 , 0 ,0 ,1,1,1,0]
featurenum = 14

def numorstring(x):
    if(x.isdigit()):
        return int(x)
    else:
        return x

def readfile(pathx,pathy):
    linesx = list(csv.reader(open(pathx)))
    linesy = list(csv.reader(open(pathy)))

    dataset = []

    for i in range(1,len(linesx)):
        row = linesx[i]
        thisRow = []
        for j in range(1,len(row)):
            thisRow.append(row[j])
        thisRow.append(i-1)
        thisRow.append(int(linesy[i][-1]))

        dataset.append(thisRow)

    return dataset


def readfile2(pathx):
    linesx = list(csv.reader(open(pathx)))

    dataset = []

    for i in range(1,len(linesx)):
```

```
            row = linesx[i]
            thisRow = []
            for j in range(1,len(row)):
                thisRow.append(numorstring(row[j]))
            thisRow.append(i+22791)
            thisRow.append(1234567)

            dataset.append(thisRow)

    return dataset

def countnum(ds, i, thres):
    ans = {}


    if(i == -1 or featuretype[i] == 0):
        for j in range(len(ds)):
            if ds[j][i] not in ans :
                ans[ds[j][i]] = 0
            ans[ds[j][i]] += 1
    else:
        for j in range(len(ds)):
            c = 0
            if float(ds[j][i]) >= thres:
                c = 1

            if c not in ans:
                ans[c] = 0
            ans[c] += 1
    return ans

def splitdata(dstrain, feature, thres):

    ansDict = {}
    ansList = []
```

```
    if(feature == -1 or featuretype[feature] == 0):

        for i in range(len(dstrain)):

            c = dstrain[i][feature]

            if c not in ansDict:
                ansDict[c] = []
            ansDict[c].append(dstrain[i])

    else:
        for i in range(len(dstrain)):

            if( float(dstrain[i][feature]) >= thres):
                c = 1
            else:
                c = 0

            if c not in ansDict:
                ansDict[c] = []

            ansDict[c].append(dstrain[i])


    for key in ansDict:
        ansList.append(ansDict[key])

    return ansList




def entropy(dstrain):
    ccc = countnum(dstrain, -1, 0)
    answer = 0
    for key in ccc :
        pt = ccc[key] / len(dstrain)
        answer -= pt * math.log(pt,2)
```

```
    return answer

def remainder(dstrain, k, thres):
    dss = splitdata(dstrain, k , thres)
    ccc = countnum(dstrain, k, thres)
    answer = 0

    if(featuretype[k] == 0):

        for ds in dss:
            key = ds[0][k]
            pt = ccc[key] / len(dstrain)
            answer += pt * entropy(ds)

    else:

        for ds in dss:
            key = float(ds[0][k])
            if(key >= float(thres)):
                key = 1
            else:
                key = 0
            pt = ccc[key] / len(dstrain)
            answer += pt * entropy(ds)

    return answer

def entropyfeature(dstrain, k, thres):
    dss = splitdata(dstrain, k ,thres)
    ccc = countnum(dstrain, k ,thres)
    answer = 0

    if(featuretype[k] == 0):

        for ds in dss:
            key = ds[0][k]
            pt = ccc[key] / len(dstrain)
            answer -= pt * math.log(pt,2)
```

```
        else:
            for ds in dss:
                key = float(ds[0][k])
                if(key >= float(thres)):
                    key = 1
                else:
                    key = 0
                pt = ccc[key] / len(dstrain)
                answer -= pt * math.log(pt,2)
        return answer


def getanswer(dstrain):
        ccc = countnum(dstrain, -1, 0)
        if(1 not in ccc):
            return 0
        if(0 not in ccc):
            return 1
        if(ccc[0] >= ccc[1]):
            return 0
        else:
            return 1

def getThres(dstrain, k):
        if(featuretype[k] == 0):
            return 876543
        mn = float(dstrain[0][k])
        mx = float(dstrain[0][k])
        #print(mn)
        #print(mx)
        for i in range(len(dstrain)):
            if(float(dstrain[i][k]) > mx):
                mx = float(dstrain[i][k])
            if(float(dstrain[i][k]) < mn):
                mn = float(dstrain[i][k])

        return (mn + mx) / 2
```

```
def buildtree(dstrain, depth):
    node = {}
    #node=> isleaf, ds, decision, children,feature, thres
    node["ds"] = dstrain
    node["answer"] = getanswer(dstrain)
    node["children"] = []
    node["isleaf"] = 0

    key = 0
    for i in range(len(dstrain)):
        if(dstrain[i][-1] != dstrain[0][-1]):
            key = 1
    if(key == 0 or depth >= 75):
        node["isleaf"] = 1
        return node

    Hall = entropy(dstrain)
    R = []
    HF = []
    G = []
    GR = []
    T = []
    for i in range(featurenum):
        T.append( getThres(dstrain,i) )
        R.append(remainder(dstrain, i, T[i]))
        HF.append(entropyfeature(dstrain, i,T[i]))
        G.append(Hall - R[i])
        if(HF[i] == 0 ):
            GR.append(0)
        else:
            GR.append(G[i] / HF[i])

    node["feature"] = -1

    for i in range(featurenum):
        if(node["feature"] == -1 or GR[i] > GR[node["feature"]]):
            node["feature"] = i
```

```
                node["thres"] = T[i]

        dsChild = splitdata(dstrain, node["feature"],node["thres"])

        for dsC in dsChild:
            node["children"].append(buildtree(dsC, depth+1))

        return node


def query(data,node):
        if(node["isleaf"] == 1):
            # print("leaf! return",node["answer"])
            return node["answer"]

        if( featuretype[node["feature"]] == 0 ):
            # print("categorical feature",node["feature"])
            # print("data feature =", data[node["feature"]])
            for ch in node["children"]:
                if(ch["ds"][0][node["feature"]] == data[node["feature"]]):
                    return query(data,ch)
            # print("no match! return",node["answer"])
            return node["answer"]
        else:
            # print("continuous feature",node["feature"])
            # print("threshold =",node["thres"])
            # print("data feature =", data[node["feature"]])
            for ch in node["children"]:
                if(ch["ds"][0][node["feature"]] >= node["thres"]) ==
(data[node["feature"]] >= node["thres"]):
                    return query(data,ch)
            # print("no match! return",node["answer"])
            return node["answer"]


def getTreeSamples(ds,k):
        ans = []
        dsCopy = list(ds)
```

```
    for i in range(k):
        x = random.randrange(len(dsCopy))
        ans.append(dsCopy.pop(x))
    return ans

def solve(dstrain, dstest,mode):
    n = 1 #tree number (forest size)
    k = len(dstrain)//10

    vote = {}
    for i in range(len(dstest)):
        vote[dstest[i][-2]] = ({0:0,1:0})

    for i in range(n):
        tree = buildtree(getTreeSamples(dstrain,k), 0)

        for j in range(len(dstest)):
            # print(dstest[j])
            vote[dstest[j][-2]][query(dstest[j],tree)] += 1



    if(mode ==0 ):
        ac = 0
        wa = 0


        cm = [ [0,0], [0,0] ]
        for i in range(len(dstest)):
            nowid = dstest[i][-2]
            ans = 0
            if(vote[nowid][0] >= vote[nowid][1]):
                ans = 0
            else:
                ans = 1
            if( ans == dstest[i][-1]):
                ac+=1
            else:
                wa+=1
```

```
                cm[dstest[i][-1]][ans] += 1

            return cm

        elif(mode==1):
            #print("Id,Category")
            for i in range(len(dstest)):
                nowid = dstest[i][-2]
                ans = 0
                if(vote[nowid][0] >= vote[nowid][1]):
                    ans = 0
                else:
                    ans = 1
                #print( str(nowid) + "," + str(ans))

def splittraintest(ds, ratio):
    dstrain = list(ds)
    dstest = []
    testSize = int(len(ds) * (1-ratio))
    for i in range(testSize):
        x = random.randrange(len(dstrain))
        dstest.append(dstrain.pop(x))

    return dstrain,dstest

def k_fold(ds,k):
    ansA = []
    ansB = []
    dsCopy = list(ds)

    onesize = int(len(ds) / k)

    for i in range(k):

        nowTrain = list(ds)
        nowTest = []

        for j in range(onesize):
```

```
                x = random.randrange(len(dsCopy))
                nowTest.append(dsCopy[x])
                nowTrain.remove(dsCopy[x])
                dsCopy.pop(x)

        ansA.append(nowTrain)
        ansB.append(nowTest)
    return ansA,ansB

def main():
    #random.seed("56789")

    dsall = readfile("./heart_failure.csv","./result.csv")
    for i in range(len(dsall)):
        for row in dsall[i]:
            if i != 0:
                row = float(row)
    #nowmode = 2
    #if(nowmode == 0):
    dstrain, dstest = splittraintest(dsall, 0.7)
    random.shuffle(dstrain)
    random.shuffle(dstest)
    cm = solve(dstrain, dstest,0)
    print("Holdout validation with the ratio 7:3")
    print("1. Accuracy:",(cm[0][0] + cm[1][1]) / len(dstest))
    print("2. Sensitivity(Recall):",(cm[0][0]) / (cm[0][0] + cm[0][1]))
    print("3. Precision:",(cm[0][0])/(cm[0][0] + cm[1][0]))

    #elif(nowmode == 1):
    #    dstrain = dsall
    #    dstest = readfile2("./X_test.csv")
    #    random.shuffle(dstrain)
    #    random.shuffle(dstest)
    #    solve(dstrain, dstest,1)
    #elif(nowmode == 2):
    trainList,testList = k_fold(dsall, 3)
    cmS = [ [0,0], [0,0] ]
    for dstrain,dstest in zip(trainList,testList):
```

```
            cm = solve(dstrain,dstest,0)
            for i in range(2):
                for j in range(2):
                    cmS[i][j] += cm[i][j] / 3
                #print(cm)

        print("K-fold cross-validation with K = 3")
        print("1. Accuracy:",(cmS[0][0] + cmS[1][1]) / len(testList[0]))
        print("2. Sensitivity(Recall):",(cmS[0][0]) / (cmS[0][0] + cmS[0][1]))
        print("3. Precision:",(cmS[0][0])/(cmS[0][0] + cmS[1][0]))



main()
```