

```

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(numorstring(row[j]))
        thisRow.append(i-1)
        thisRow.append(int(linesy[i][-1]))

        dataset.append(thisRow)

    return dataset

```

numorstring will check if the input type is int, if it is int then return int, if it is not int then return string

readfile will combine X\_train.csv and X\_test.csv. Place features in the front and place "Id" and "Category" at the back.

```

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

```

readfile2 will read y\_train.csv. Place features in the front and place "Id" and a

random number at the back, so that readfile and readfile2 will return datasets which have the same number of columns.

```
def splitraintest(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
```

splitraintest will split dataset into two datasets (train and test) with ratio 7:3

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

    onsize = int(len(ds) / k)

    for i in range(k):
        nowTrain = list(ds)
        nowTest = []

        for j in range(onsize):
            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
```

k\_fold will split dataset into two datasets (train and test) with ratio  $1-1/k : 1/k$  and repeat the same move k times.

```
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 ds[j][i] >= thres:
                c = 1

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

If the ith feature is categorical, countnum will count how many times does each type in the ith feature appear.

If the ith feature is continuous, countnum will count how many values in the ith feature is bigger than the threshold and how many values in the ith feature is smaller.

```
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( 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
```

If the feature is categorical, splitdata will put the row that have the same type in

“feature” column into the list.

If the feature is continuous, splitdata will first compare the value with the threshold and then put the row into the list.

```
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
```

entropy will compute the entropy.

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

If the feature is categorical, remainder will compute the remainder.

If the feature is continuous, remainder will change the value bigger than threshold into 1, smaller than threshold into 0, then compute the remainder.

```

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

```

If the feature is categorical, entropyfeature will compute the entropy of each feature. If the feature is continuous, entropyfeature will change the value that is bigger than the threshold into 1 and smaller than the threshold into 0, then compute the entropy of each feature.

```

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 = dstrain[0][k]
    mx = dstrain[0][k]

    for i in range(len(dstrain)):
        if(dstrain[i][k] > mx):
            mx = dstrain[i][k]
        if(dstrain[i][k] < mn):
            mn = dstrain[i][k]

    return (mn + mx) / 2

```

getanswer will return 0 if there are more 0 than 1, return 1 if there are more 1 than 0  
getThres will find the maximum and minimum value in the kth feature and return

(maximum plus minimum) multiply by 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
```

buildtree will build a tree by computing information gain ratio. Split the dataset by comparing values in the feature that has the biggest information gain ratio. Each node will record if it is leaf, the dataset it has, the answer, its children, its feature and its threshold.

```
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"]
```

query will return the answer of the node if the node is leaf, if the node isn't leaf it will keep running the query function again until there is no match and then will return the answer of the node

```
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
```

getTreeSamples will get k rows from the dataset. It will be used to create random forest.

```

def solve(dstrain, dtest, mode):
    n = 100 #tree number (forest size)
    k = len(dstrain)//10

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

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

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

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

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

        return cm

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

```

solve will create n trees and each have k rows of values. If more trees predict the answer is 1 then we predict the answer is 1, if more trees predict the answer is 0 then we predict the answer is 0. Then record our prediction and the real answer so that we can show the confusion matrix later. If mode = 1, solve will print the id and prediction so that I can submit the result to Kaggle.



```

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

    dsall = readfile("./X_train.csv", "./y_train.csv")

    nowmode = 2
    if(nowmode == 0):
        dstrain, dptest = splittraintest(dsall, 0.7)
        random.shuffle(dstrain)
        random.shuffle(dptest)
        cm = solve(dstrain, dptest, 0)
        print("Holdout validation with the ratio 7:3")
        print("1. Confusion matrix")
        print("      Prediction")
        print("      <=50k      >50k")
        print("Target <=50k      ", cm[0][0], "      ", cm[0][1])
        print("      >50k      ", cm[1][0], "      ", cm[1][1])
        print("2. Accuracy:", (cm[0][0] + cm[1][1]) / len(dptest))
        print("3. Sensitivity(Recall):", (cm[0][0]) / (cm[0][0] + cm[0][1]))
        print("4. Precision:", (cm[0][0]) / (cm[0][0] + cm[1][0]))

    elif(nowmode == 1):
        dstrain = dsall
        dptest = readfile2("./X_test.csv")
        random.shuffle(dstrain)
        random.shuffle(dptest)
        solve(dstrain, dptest, 1)

    elif(nowmode == 2):
        trainList, testList = k_fold(dsall, 3)
        cmS = [ [0,0], [0,0] ]
        for dstrain, dptest in zip(trainList, testList):
            cm = solve(dstrain, dptest, 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. Confusion matrix")
        print("      Prediction")
        print("      <=50k      >50k")
        print("Target <=50k      ", cmS[0][0], "      ", cmS[0][1])
        print("      >50k      ", cmS[1][0], "      ", cmS[1][1])
        print("2. Accuracy:", (cmS[0][0] + cmS[1][1]) / len(testList[0]))
        print("3. Sensitivity(Recall):", (cmS[0][0]) / (cmS[0][0] + cmS[0][1]))
        print("4. Precision:", (cmS[0][0]) / (cmS[0][0] + cmS[1][0]))

```

main function has three modes, each mode does similar things. The difference between the modes is that mode 0 use Holdout validation with ration 7:3, mode 1 will print the id and prediction of the answer so that I can submit the result to Kaggle, mode 2 use K-fold cross-validation with K = 3. Each mode will first read the test data and shuffle the data. Then, put the test data into solve function. Last, mode 0 and 2 will print the results like confusion matrix, accuracy, sensitivity, and precision.

### Holdout validation with the ratio 7:3

#### 1. Confusion matrix

		Prediction	
		<=50k	>50k
Target	<=50k	4817	341
	>50k	650	1029

2. Accuracy: 0.8550533859880064

3. Sensitivity(Recall): 0.9338891043039937

4. Precision: 0.8811048106822754

```

K-fold cross-validation with K = 3
1. Confusion matrix

                        Prediction
                        <=50k    >50k
Target <=50k          5354      355
        >50k          777       1111

2. Accuracy: 0.853626431486113
3. Sensitivity(Recall): 0.9310125115848007
4. Precision: 0.8822593039850697













```

Because there are too many lines, I show the prediction and reasoning in another file.

Conclusion:

使用 random forest 的時候，調整 tree 的個數跟 sample 的數量就能提高準確率，非常有趣

Kaggle:

#	Team Name	Notebook	Team Members	Score 🏆	Entries	Last
1	ML Faker			0.86313	26	6h
2	ML Huni 相赫哥 你不要再打了 ...			0.86040	43	15h
📍	↑ 助教：超越不過去可惡啊啊啊啊			0.85870		
3	buganshie			0.85563	2	1d
4	tim310579			0.85392	16	42m
5	Tseng			0.85255	2	11d
6	herOray			0.84744	3	4h
7	kaiiiz			0.84573	5	1d
8	Tsai-yuen			0.84436	2	15h
<b>Your Best Entry ↑</b> Your submission scored 0.84436, which is an improvement of your previous score of 0.84095. Great job! <a href="#">Tweet this!</a>						
9	haha			0.84368	1	2d
10	samuelyutt			0.84300	12	5h
11	yx deng			0.84266	5	5h
12	J Chen			0.84197	2	4d