Machine Learning 1 : C-M-004 Homework 4

- Data

x1	x2	y
F	12	F
F	14	Y
T	13	Y
T	16	F

The best discretization for x2 from an information gain perpective is based on which threshold (if a threshold is θ, then x2 ≤θ is 0, else

- (i) 12, (ii) 13, (iii) 14, (iv) 16 (10 points) Based on entropy, the first split will be based on:
- (i) x1, (ii) x2, (iii) Doesn't matter (10 points)

Answers

x1Entropy at the root node

Number of F : 2
Number of Y : 2

Entropy at root node = $-\sum_0^n p_i log(p_i)$ = $-\frac{2}{4}log(\frac{2}{4})$ - $-\frac{2}{4}log(\frac{2}{4})$

```
print('Entropy at node: ',-(2/4*math.log2(2/4))-(2/4*math.log2(2/4)))
```

```
Entropy at node:  1.0
```

Finding the entropy gain when split by x1

If we split the node by x1, the resulting nodes will be
Node 1 : [FY]
Node 2 : [YF]

Calculating the entropt of individual nodes:

Node 1: $-\frac{1}{2}log(\frac{1}{2})$-$-\frac{1}{2}log(\frac{1}{2})$
Node 2: $-\frac{1}{2}log(\frac{1}{2})$-$-\frac{1}{2}log(\frac{1}{2})$

```
Entropy gain 1.0
```

Finding the entropy when split by x2

```
import math
```

Here we can have 4 possibilities

- When 12 is used as threshold, x2 becomes [T,F,F,F], Hence ouput is classified as [F] and [YYF]
- When 14 is used as threshold, x3 becomes [T,T,T,F],hence output is classified as [FYY] and [F]
- When 13 is used as threshold, x3 becomes [T,F,T,F],hence output is classified as [FY] and [YF]
- When 16 is used as threshold, x4 becomes [T,T,T,T],hence output is classified as [FYYF] and []

Finding all four possible entropy gains

- Case 1, when 12 is threshold

```
n1=-(1/1*math.log2(1/1))
n2=-(2/3*math.log2(2/3))-(1/3*math.log2(1/3))
print('Entropy of Node 1: ',n1)
print('Entropy of Node 2: ',n2)
print('Entropy gain', ((1/4)*n1)+((3/4)*n2))
```

```
Entropy of Node 1:  -0.0
Entropy of Node 2:  0.9182958340544896
Entropy gain 0.6887218755408672
```

- Case 1, when 14 is threshold

```
n1=-(1/3*math.log2(1/3))-(2/3*math.log2(2/3))
n2=-(1/1*math.log2(1/1))
print('Entropy of Node 1: ',n1)
print('Entropy of Node 2: ',n2)
print('Entropy gain', ((3/4)*n1)+((1/4)*n2))
```

```
Entropy of Node 1:  0.9182958340544896
```

```
Entropy of Node 1:  1.0
Entropy of Node 2:  1.0
Entropy gain 1.0
```

- Case 1, when 16 is threshold

```
n1=-(2/4*math.log2(2/4))-(2/4*math.log2(2/4))
print('Entropy of Node 1: ',n1)
print('Entropy gain', ((4/4)*n1))
```

```
Entropy of Node 1:  1.0
Entropy gain 1.0
```

Answers and Conclusions

- The best discretization for x2 based on the above result is based on the value 12 and 14. When the split is by considering 12 as threshold or 14 as threshold, we get the lowest entropy gain.
- When split is based on x2, with 12 and 14 we get the nodes with lower entropy. Hence, the split would be the best based on x2. When splitting by other parameters it creates a

2. Is a random forest of random forests a good idea?

- No, Absolutely not
- Yes, of course
- Maybe, varies from case to case
- None of the above

Null Hypothesis

A random forest contains a lot of low performers whose outputs are then aggregated together to give the final output. Here, it is ensured that the different

```
# Creating a decision dree classifier object
from sklearn.tree import DecisionTreeClassifier
dtClassifier=DecisionTreeClassifier(random_state=1)

# Using K Fold cross validation
k_fold=KFold(n_splits=5,random_state=1,shuffle=True)
scores=[]
for train_index,test_index in k_fold.split(inputs):
    x_train=inputs[train_index]
    y_train=outputs[train_index]
    x_test=inputs[test_index]
    y_test=outputs[test_index]

    dtClassifier.fit(x_train,y_train)
    scores.append(dtClassifier.score(x_test,y_test))
print(sum(scores)/len(scores))
# Creating a random forest
random_forest=BaggingClassifier(base_estimator=dtClassifier,n_estimators=20,rand
#K Fold on Random forest
# Using K Fold cross validation

k_fold=KFold(n_splits=5,random_state=1,shuffle=True)
scores=[]
for train_index,test_index in k_fold.split(inputs):
    x_train=inputs[train_index]
    y_train=outputs[train_index]
    x_test=inputs[test_index]
    y_test=outputs[test_index]

    random_forest.fit(x_train,y_train)
    scores.append(random_forest.score(x_test,y_test))
print(sum(scores)/len(scores))

# Creating a forest of forests
random_forest_1=BaggingClassifier(base_estimator=random_forest,n_estimators=20,r
```

```
    x_test=inputs[test_index]
    y_test=outputs[test_index]

    random_forest_2.fit(x_train,y_train)
    scores.append(random_forest_2.score(x_test,y_test))
print(sum(scores)/len(scores))
```

```
0.9297314081664337
0.9437820214252446
0.9578481602235678
0.9508306163639186
```

From the above testing, it can be seen that the accuracy is increasing for a forest of forest. Hence, rejecting Null hypothesis.

One creating a forest of forest of forest, it can be seen that the accuracy has dropped a little, but it is still more than the random forest itself. Hence, it can be that the Bagging Classifier tries to create independent models which are low performing random forest classifiers and then it finally creates a deeper and wider tree which are low performing classifiers.

C. Maybe, varies from case to case