G15_HW2

Team Members:

Darpan Dodiya - dpdodiya
Shantanu Sharma - ssharm34
Shrijeet Joshi - sjoshi22

Solutions

Solution 1 : Decision Tree Construction

A] Entropy

Entropy of class attribute H(Class)
H(Class) = -7/16 * lg(7/16) - 9/16 * lg(9/16)
H(Class) = 0.988699

- Calculating IG for V1(Continuous attribute)

1. V1 <= 7 and V1 > 7

H(Class | V1 <= 7) = -(0/1)*log(0/1) - (1/1)*log(1/1) = 0
H(Class | V1 > 7) = -7/15* log(7/15) - 8/15*log(8/15) = 0.996792
H(Class | V1) = 0.934492
IG(Class | V1) = 0.054207

2. V1 <= 10 and V1 > 10

H(Class | V1 <= 10) =0
H(Class | V1 > 10) = 1
H(Class | V1) =0.875
IG(Class | V1) = 0.11369

3. V1 <= 11 and V1 > 11

H(Class | V1 <= 11) =0.918296
H(Class | V1 > 11) = 0.995727
H(Class | V1) =0.981209
IG(Class | V1) = 0.00749

4. V1 <= 13 and V1 > 13

H(Class | V1 <= 13) =0.811278
H(Class | V1 > 13) = 1
H(Class | V1) =0.95282
IG(Class | V1) = 0.03588

5. V1 <= 15 and V1 > 15

H(Class | V1 <= 15) =0.970951
H(Class | V1 > 15) = 0.99403
H(Class | V1) =0.986818
IG(Class | V1) = 0.001882

6. V1 <= 18 and V1 > 18

H(Class | V1 <= 18) =0.918296
H(Class | V1 > 18) = 1
H(Class | V1) =0.969361
IG(Class | V1) = 0.019338

7. V1 <= 20 and V1 > 20

H(Class | V1 <= 20) =0.863121
H(Class | V1 > 20) = 0.991076
H(Class | V1) =0.935096
IG(Class | V1) = 0.053604

8. V1 <= 22 and V1 > 22

H(Class | V1 <= 22) =0.811278
H(Class | V1 > 22) = 0.954434
H(Class | V1) =0.882856
IG(Class | V1) = 0.105843

9. V1 <= 27 and V1 > 27

H(Class | V1 <= 27) =0.918296
H(Class | V1 > 27) = 0.985228
H(Class | V1) =0.947579
IG(Class | V1) = 0.041121

10. V1 <= 30 and V1 > 30

H(Class | V1 <= 30) =0.970951
H(Class | V1 > 30) = 1

H(Class | V1) =0.981844
IG(Class | V1) = 0.006855

11. V1 <= 32 and V1 > 32

H(Class | V1 <= 32) =0.99403
H(Class | V1 > 32) = 0.970951
H(Class | V1) =0.986818
IG(Class | V1) = 0.001882

12. V1 <= 35 and V1 > 35

H(Class | V1 <= 35) =1
H(Class | V1 > 35) = 0.811278
H(Class | V1) =0.95282
IG(Class | V1) = 0.03588

13. V1 <= 37 and V1 > 37

H(Class | V1 <= 37) =0.995727
H(Class | V1 > 37) = 0.918296
H(Class | V1) =0.981209
IG(Class | V1) = 0.00749

14. V1 <= 40 and V1 > 40

H(Class | V1 <= 40) =1
H(Class | V1 > 40) = 0
H(Class | V1) =0.875
IG(Class | V1) = 0.113699

15. V1 <= 43 and V1 > 43

H(Class | V1 <= 43) =0.996792
H(Class | V1 > 43) = 0
H(Class | V1) =0.934492
IG(Class | V1) = 0.054207

16. V1 <= 50 and V1 > 50

H(Class | V1 <= 50) =0.996792
H(Class | V1 > 50) = 0
H(Class | V1) =0.934492
IG(Class | V1) = 0.054207

We can observe the highest IG on attribute split 10, 40. We select the leftmost point i.e 10 as best split for V1.

IG(V1) = 0.113699

- Calculating IG for V2

P(Class| V2 = Blue) = -⅝ log(⅝) -⅜ *log(⅜) = 0.9544
P(Class| V2 = White) = -2/8*log(2/8) -6/8*log(6/8) = 0.8113
P(Class | V2) = (0.5)*(0.9544) + (0.5)*(0.8113) = 0.8829
IG(V2) = 0.1058

- Calculating IG for V3

P(Class| V3 = Short) = -⅝ log(⅝) -⅜ *log(⅜) = 0.9544
P(Class| V3 = Long) = -2/8*log(2/8) -6/8*log(6/8) = 0.8113
P(Class | V3) = (0.5)*(0.9544) + (0.5)*(0.8113) = 0.8829
IG(V3) = 0.1058

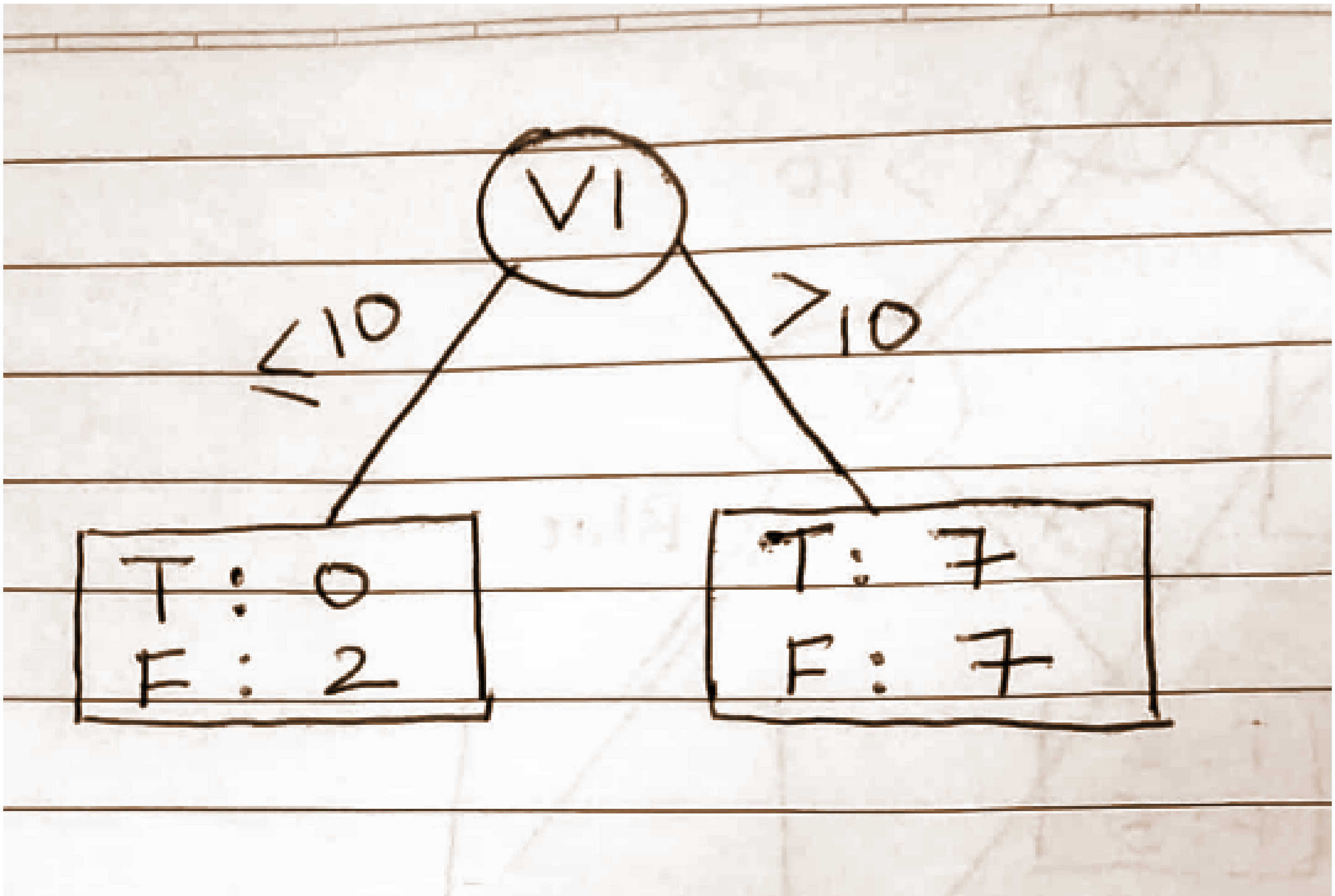
- Calculating IG for V4

P(Class| V4 = Cool) = -⅝ log(⅝) -⅜ *log(⅜) = 0.9544
P(Class| V4 = Hot) = -2/8*log(2/8) -6/8*log(6/8) = 0.8113
P(Class | V4) = (0.5)*(0.9544) + (0.5)*(0.8113) = 0.8829
IG(V4) = 0.1058

- Calculating IG for V5

P(Class| V5 = Low) = -⅝ log(⅝) -⅜ *log(⅜) = 0.9544
P(Class| V5 = High) = -2/8*log(2/8) -6/8*log(6/8) = 0.8113
P(Class | V5) = (0.5)*(0.9544) + (0.5)*(0.8113) = 0.8829
IG(V5) = 0.1058

Hence, we can see that highest IG = IG(V1). Hence, selecting split node is V1.



Next splitting on Right subtree (V1 > 10)

Entropy of Class variable
P(T) = 7/14 = 0.5
P(F) = 7/14 = 0.5
H(Class) = -7/14log(7/14) - 7/14log(7/14) = 1

- Calculating IG for V2

P(Class| V2 = Blue) = -2/7*log(2/7) -5/7 *log(5/7) = 0.86312
P(Class| V2 = White) = -5/7*log(5/7) -2/7*log(2/7) = 0.86312
P(Class | V2) = (0.5)*(0.9544) + (0.5)*(0.8113) = 0.86312
IG(V2) = 0.13688

- Calculating IG for V3

P(Class| V3 = Long) = -2/7*log(2/7) -5/7 *log(5/7) = 0.86312
P(Class| V3 = Short) = -5/7*log(5/7) -2/7*log(2/7) = 0.86312
P(Class | V3) = (0.5)*(0.9544) + (0.5)*(0.8113) = 0.86312
IG(V3) = 0.13688

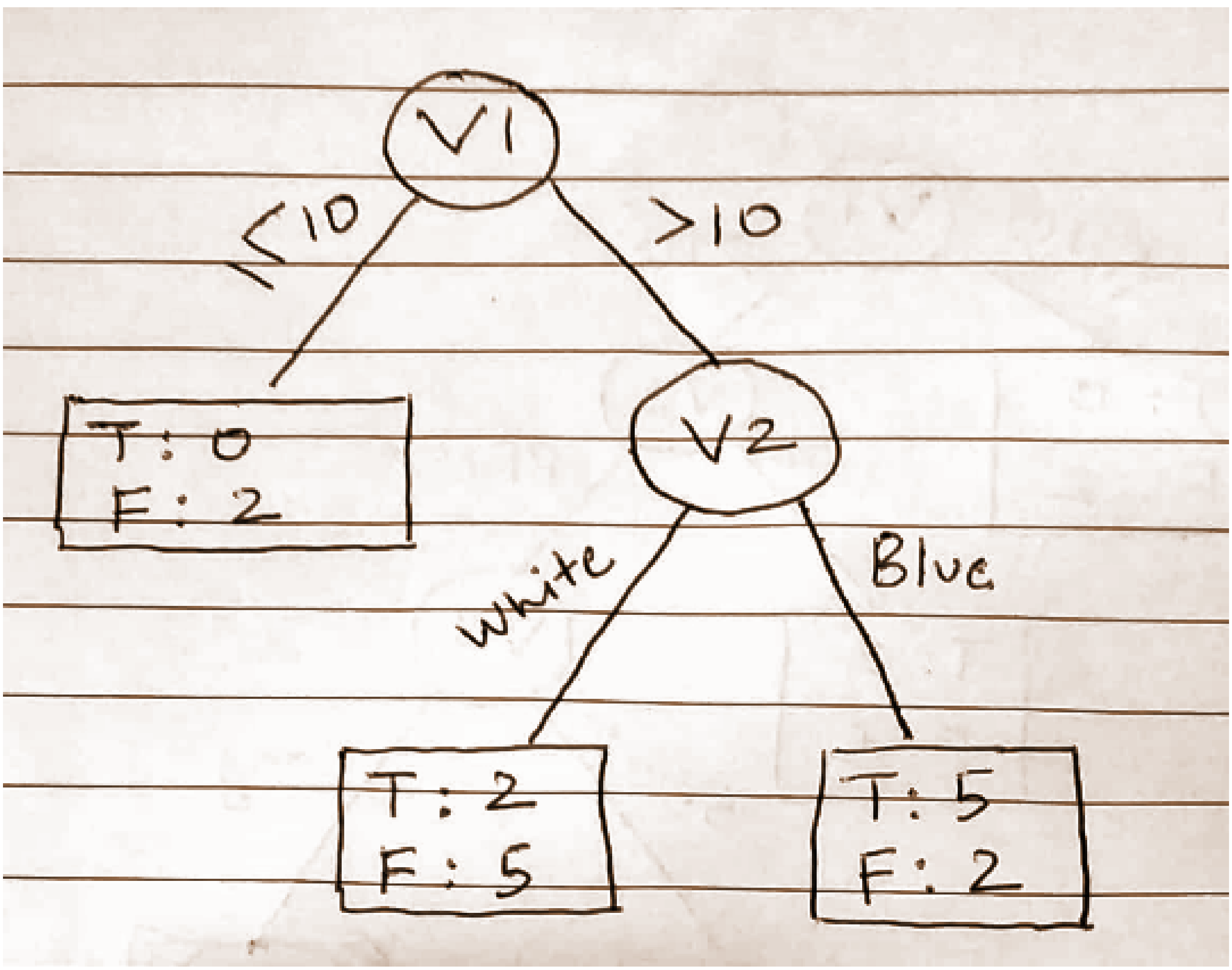
- Calculating IG for V4

P(Class| V4 = Long) = -2/7*log(2/7) -5/7 *log(5/7) = 0.86312
P(Class| V4 = Short) = -5/7*log(5/7) -2/7*log(2/7) = 0.86312
P(Class | V4) = (0.5)*(0.9544) + (0.5)*(0.8113) = 0.86312
IG(V4) = 0.13688

- Calculating IG for V5

P(Class| V5 = High) = -2/7*log(2/7) -5/7 *log(5/7) = 0.86312
P(Class| V5 = Low) = -5/7*log(5/7) -2/7*log(2/7) = 0.86312
P(Class | V5) = (0.5)*(0.9544) + (0.5)*(0.8113) = 0.86312
IG(V3) = 0.13688

Hence, we can see that IG is same for all attributes. Selecting leftmost attribute as split attribute i.e. V2



Calculating entropy for right subtree following “BLUE” branch after V2 split.

Calculating entropy of Class attribute
P(T) = 5/7
P(F) = 2/7

H(Class) = -2/7*log(2/7) - 5/7*log(5/7)
H(Class) = 0.8631

- Calculating IG for V3

P(Class| V3 = Long) = -2/3*log(2/3) -1/3 *log(1/3) = 0.9279
P(Class| V3 = Short) = -4/4*log(4/4) -0/4*log(0/4) = 0
P(Class | V3) = (0.9279)*(0.4285) + 0 = 0.3976
IG(V3) = 0.4654

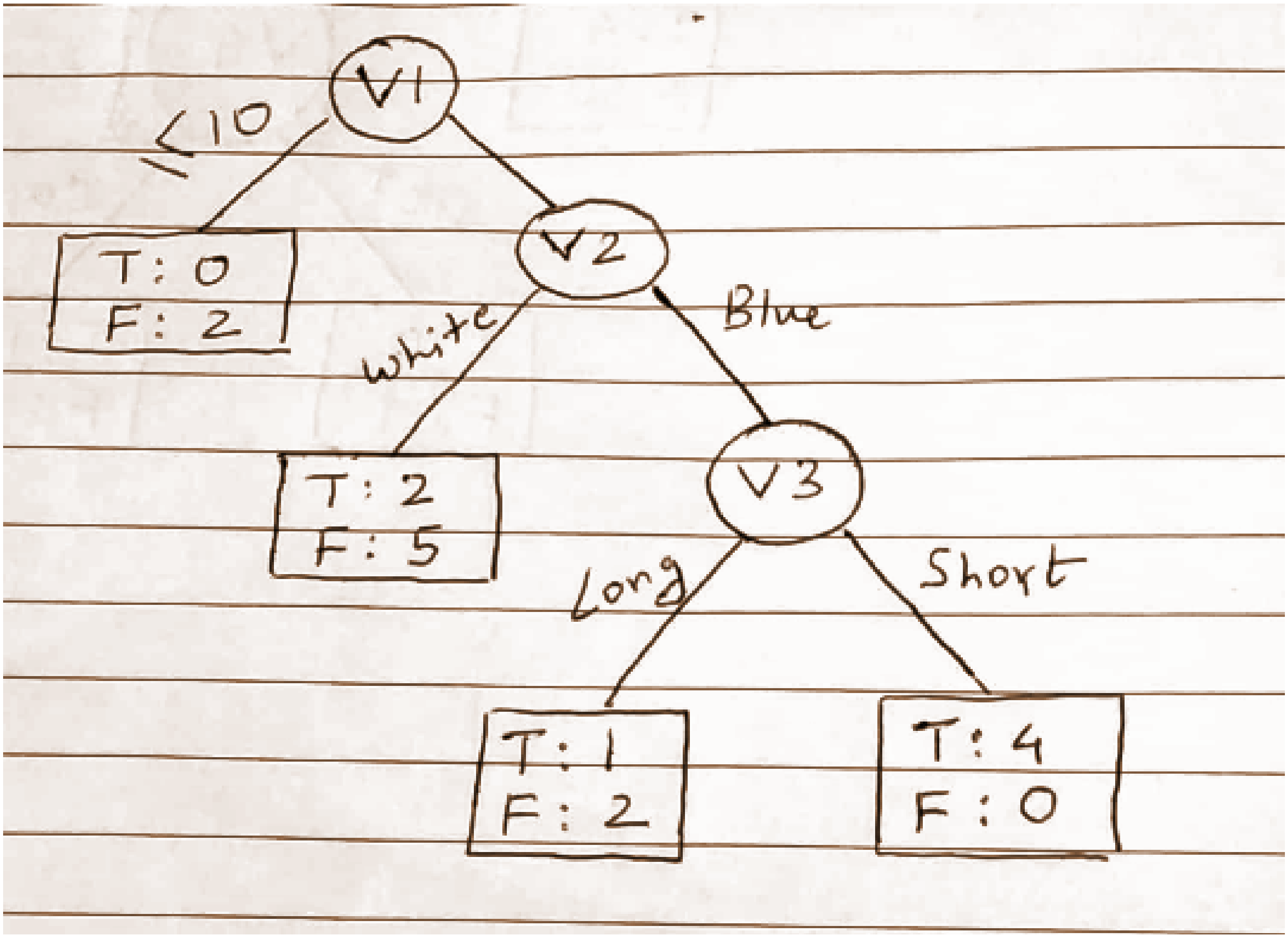
- Calculating IG for V4

P(Class| V4 = Cool) = -3/4*log(3/4) -1/4 *log(1/4) = 0.8112
P(Class| V4 = Hot) = -2/3*log(2/3) -1/3*log(1/3) = 0.91826
P(Class | V4) = (0.8112)*(0.5714) + (0.91826)*(0.4285) = 0.8569
IG(V3) = 0.006125

- Calculating IG for V5

P(Class| V5 = Low) = -3/4*log(3/4) -1/4 *log(1/4) = 0.8112
P(Class| V5 = High) = -2/3*log(2/3) -1/3*log(1/3) = 0.91826
P(Class | V5) = (0.8112)*(0.5714) + (0.91826)*(0.4285) = 0.8569
IG(V3) = 0.006125

Hence, we can see that IG of V3 is maximum. Hence, selecting V3 as split attributes.



Tree terminates at “SHORT” branch. Calculating to split further on “LONG” branch

Calculating entropy of Class attribute
P(T) = 1/3
P(F) = 2/3
H(Class) = -⅓*log(⅓) - ⅔*log(⅔) = 0.91826

- Calculating IG for V4

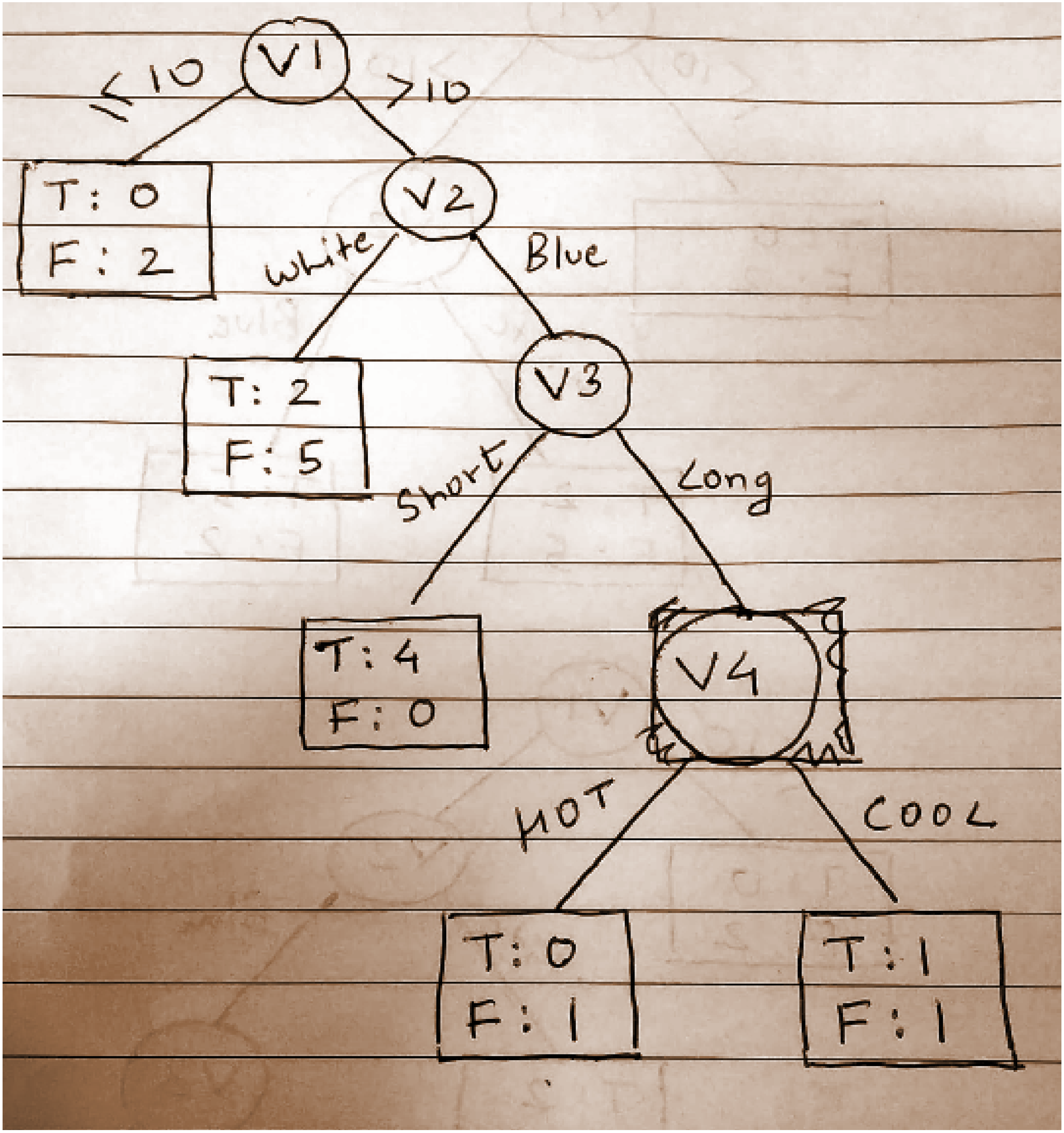
P(Class| V4 = Hot) = -1/1*log(1/1) -0/1 *log(0/1) = 0
P(Class| V4 = Cool) = -1/2*log(1/2) -1/2*log(1/2) = 1
P(Class | V4) = (0.33)*(0) + (0.66)*(1) = 0.6666
IG(V4) = 0.25162

- Calculating IG for V5

P(Class| V5 = High) = -1/1*log(1/1) -0/1 *log(0/1) = 0
P(Class| V5 = Low) = -1/2*log(1/2) -1/2*log(1/2) = 1
P(Class | V5) = (0.33)*(0) + (0.66)*(1) = 0.6666

IG(V5) = 0.25162

Hence, we can see that both attributes have same entropy. Selecting, V4 as it is the leftmost attribute.



We reached a depth of 4 hence stopping splitting procedure on the right subtree

Calculating entropy for left subtree following “WHITE” branch after V2 split.

Calculating entropy of the class attribute
P(T) = 2/7
P(F) = 5/7

H(Class) = -2/7*log(2/7) -5/7*log(5/7) = 0.8631

- Calculating IG for V3

P(Class| V3 = Long) = -1/4*log(1/4) -3/4 *log(3/4) = 0.81128
P(Class| V3 = Low) = -1/3*log(1/3) -2/3*log(2/3) = 0.9183
P(Class | V3) = 0.85714
IG(V3) = 0.00598

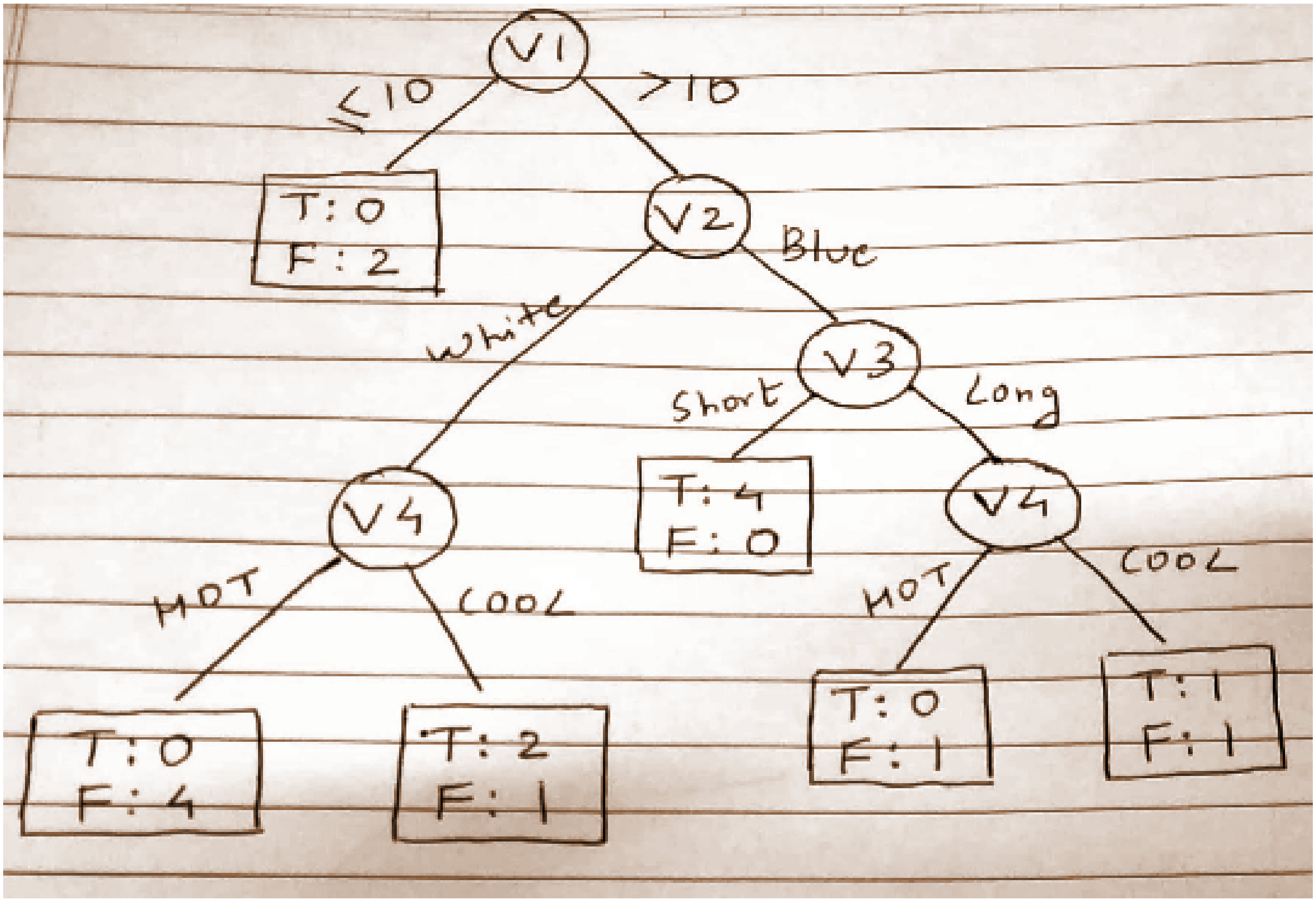
- Calculating IG for V4

P(Class| V4 = Hot) = -0/4*log(0/4) -4/4 *log(4/4) = 0
P(Class| V4 = Cool) = -1/3*log(1/3) -2/3*log(2/3) = 0.9183
P(Class | V4) = 0.39356
IG(V4) = 0.46956

- Calculating IG for V5

P(Class| V5 = High) = -0/3*log(0/3) -3/3 *log(3/3) = 0
P(Class| V5 = Low) = -2/4*log(2/4) -2/4*log(2/4) = 1
P(Class | V5) = 0.57143
IG(V3) = 0.29169

Hence, we can see that V4 has maximum IG. Hence, splitting using V4



Hence, we can see that tree terminates on “HOT” path. Calculating split along “COOL” path.

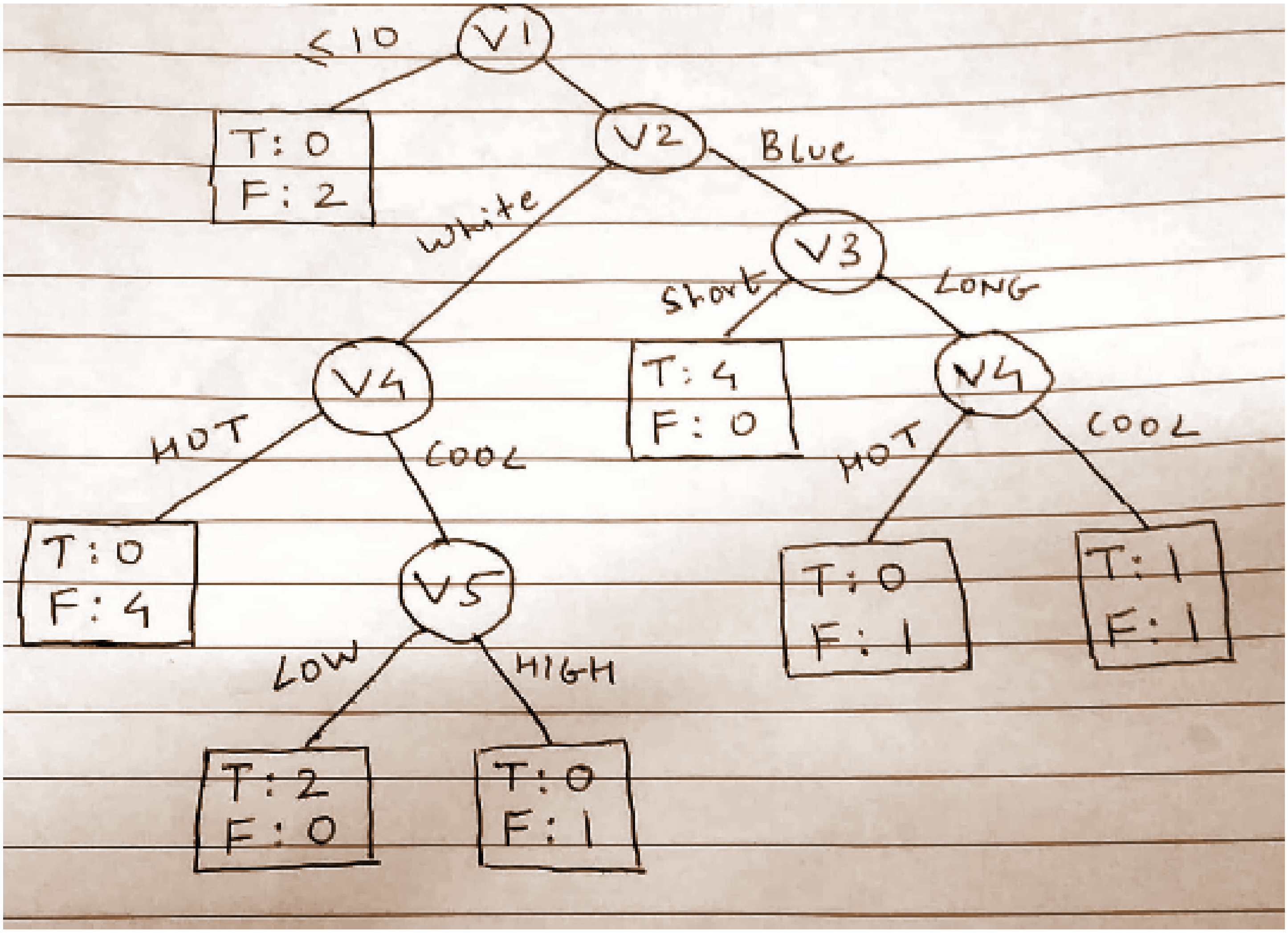
Calculating entropy of Class attribute
P(T) = 2/3
P(F) = 1/3
H(Class) = - ⅔*log(⅔) - ⅓*log(⅓)   = 0.91826

- Calculating IG for V3

P(Class| V3 = Short) = -1/1*log(1/1) -0/1 *log(0/1) = 0
P(Class| V3 =Long) = -1/2*log(1/2) -1/2*log(1/2) = 1
P(Class | V4) = (0.33)*(0) + (0.66)*(1) = 0.6666
IG(V4) = 0.25162

- Calculating IG for V5

P(Class| V5 = High) = -0/1*log(0/1) -1/1 *log(1/1) = 0
P(Class| V5 = Low) = -2/2*log(2/2) -0/2*log(0/2) = 0
P(Class | V5) = (0.33)*(0) + (0.66)*(0) =
IG(V5) = 0.91826
Hence, we can see V5 has maximum IG. Hence, selecting V5 as split attribute.



We have reached a depth of 4 and also tree has terminated. Hence, stopping splitting process.

B] GINI Index

LEVEL 1

Calculating GINI split for V1(Continuous attribute)

(<= 7, >7)

GINI(V1 <=7) = 1-0/1-1/1 = 0

GINI(V1 > 7) = 1-49/225-64/225 = 0.497778

GINI split(V1) = 0.466667

(<= 10, >10)

GINI(V1 <=10) = 1-0/4-4/4 = 0

GINI(V1 > 10) = 1-49/196-49/196 = 0.5

GINI split(V1) = 0.4375

(<= 11, >11)
GINI(V1 <=11) = 1-1/9-4/9 = 0.44444
GINI(V1 > 11) = 1-36/169-49/169 = 0.497041
GINI split(V1) = 0.487179

(<= 13, >13)
GINI(V1 <=13) = 1-1/16-9/16 = 0.375
GINI(V1 > 13) = 1-36/144-36/144 = 0.5
GINI split(V1) = 0.46875

(<= 15, >15)
GINI(V1 <=15) = 1-4/25-9/25 = 0.48
GINI(V1 > 15) = 1-25/121-36/121 = 0.495868
GINI split(V1) = 0.490909

(<= 18, >18)
GINI(V1 <=18) = 1-4/36-16/36 = 0.444444
GINI(V1 > 18) = 1-25/100-25/100 = 0.5
GINI split(V1) = 0.479167

(<= 20, >20)
GINI(V1 <=20) = 1-4/49-25/49 = 0.408163
GINI(V1 > 20) = 1-25/81-16/81 = 0.493827
GINI split(V1) = 0.456349
(<= 22, >22)
GINI(V1 <=22) = 1-4/64-36/64 = 0.375
GINI(V1 > 22) = 1-25/64-9/64 = 0.46875
GINI split(V1) = 0.421875

(<= 27, >27)
GINI(V1 <=27) = 1-9/81-36/81 = 0.44444
GINI(V1 > 27) = 1-16/49-9/49 = 0.489796
GINI split(V1) = 0.464286

(<= 30, >30)
GINI(V1 <=30) = 1-16/100-36/100 = 0.48
GINI(V1 > 30) = 1-9/36-9/36 = 0.5
GINI split(V1) = 0.4875

(<= 32, >32)
GINI(V1 <=32) = 1-25/121-36/121 = 0.495868
GINI(V1 > 32) = 1-4/25-9/25 = 0.48
GINI split(V1) = 0.490909

(<= 35, >35)
GINI(V1 <=35) = 1-36/144-36/144 = 0.5
GINI(V1 > 35) = 1-1/36-9/36 = 0.375

GINI split(V1) = 0.46875

(<= 37, >37)

GINI(V1 <=37) = 1-36/169-49/169 = 0.497041
GINI(V1 > 37) = 1-1/9-4/9 = 0.44444
GINI split(V1) = 0.487179

(<= 40, >40)

GINI(V1 <=40) = 1-49/196-49/196 = 0.5
GINI(V1 > 40) = 1-0/4-4/4 = 0
GINI split(V1) = 0.4375

(<= 43, >43)

GINI(V1 <=43) = 1-49/225-64/225 = 0.497778
GINI(V1 > 43) = 1-0/1-1/1 = 0
GINI split(V1) = 0.466667

(<= 50, >50)

GINI(V1 <=50) = 1-64/256-64/256 = 0.5
GINI(V1 > 50) = 1-0/0-0/0 = 1 (Ignoring Divide by zero error)
GINI split(V1) = 0.5

Hence, we will consider (<= 22, >22) as Best split for V1 attribute as it has lowest GINI split value

Calculating GINI split for V2

GINI(Blue) = 1-25/64-9/64 = 0.46875
GINI(White) = 1-4/64-36/64 = 0.375
GINI split(V2) = 0.421875

Calculating GINI split for V3

GINI(Long) = 1-4/64-36/64 = 0.375
GINI(Short) = 1-25/64-9/64= 0.46875
GINI split(V3) = 0.421875

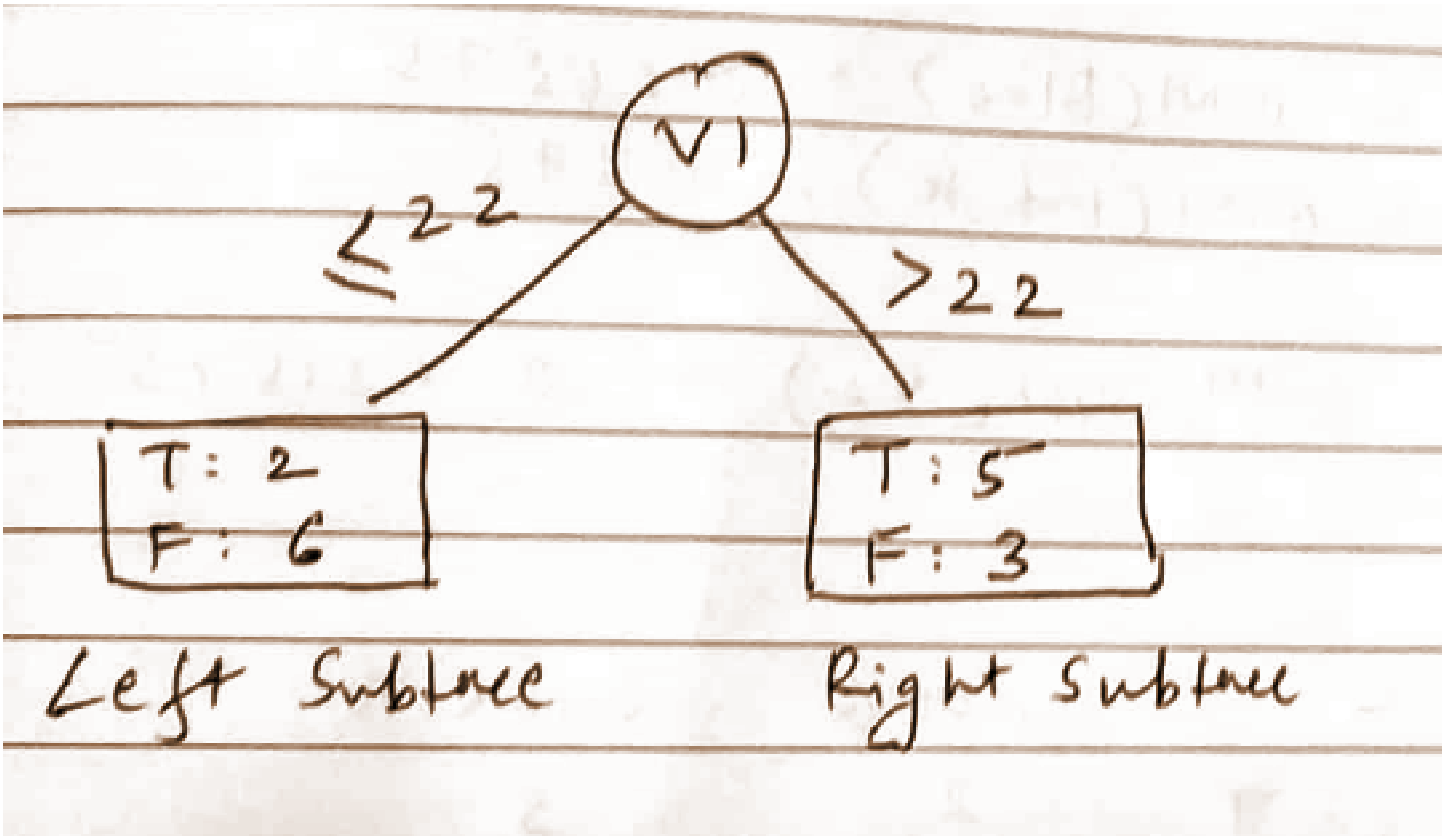
Calculating GINI split for V4

GINI(Cool) = 1-25/64-9/64 = 0.46875
GINI(Hot) = 1-4/64-36/64 = 0.375
GINI split(V4) = 0.421875

Calculating GINI split for V5

GINI(Low) = 1-25/64-9/64 = 0.46875
GINI(High) = 1-4/64-36/64 = 0.375
GINI split(V5) = 0.421875

As all attributes have equal GINI split value, considering leftmost attribute as split attribute i.e V1



LEVEL 2

A] LEFT Subtree where V1 <= 22

Calculating GINI split for V2

GINI(Blue) = 1-4/16-4/16 = 0.5

GINI(White) = 1-0/16-16/16 = 0

GINI split(V2) = 0.25

Calculating GINI split for V3

GINI(Short) = 1-4/16-4/16 = 0.5

GINI(Long) = 1-0/16-16/16 = 0

GINI split(V5) = 0.25

Calculating GINI split for V4

GINI(Hot) = 1-1/16-9/16 = 0.375

GINI(Cool) = 1-1/16-9/16 = 0.375

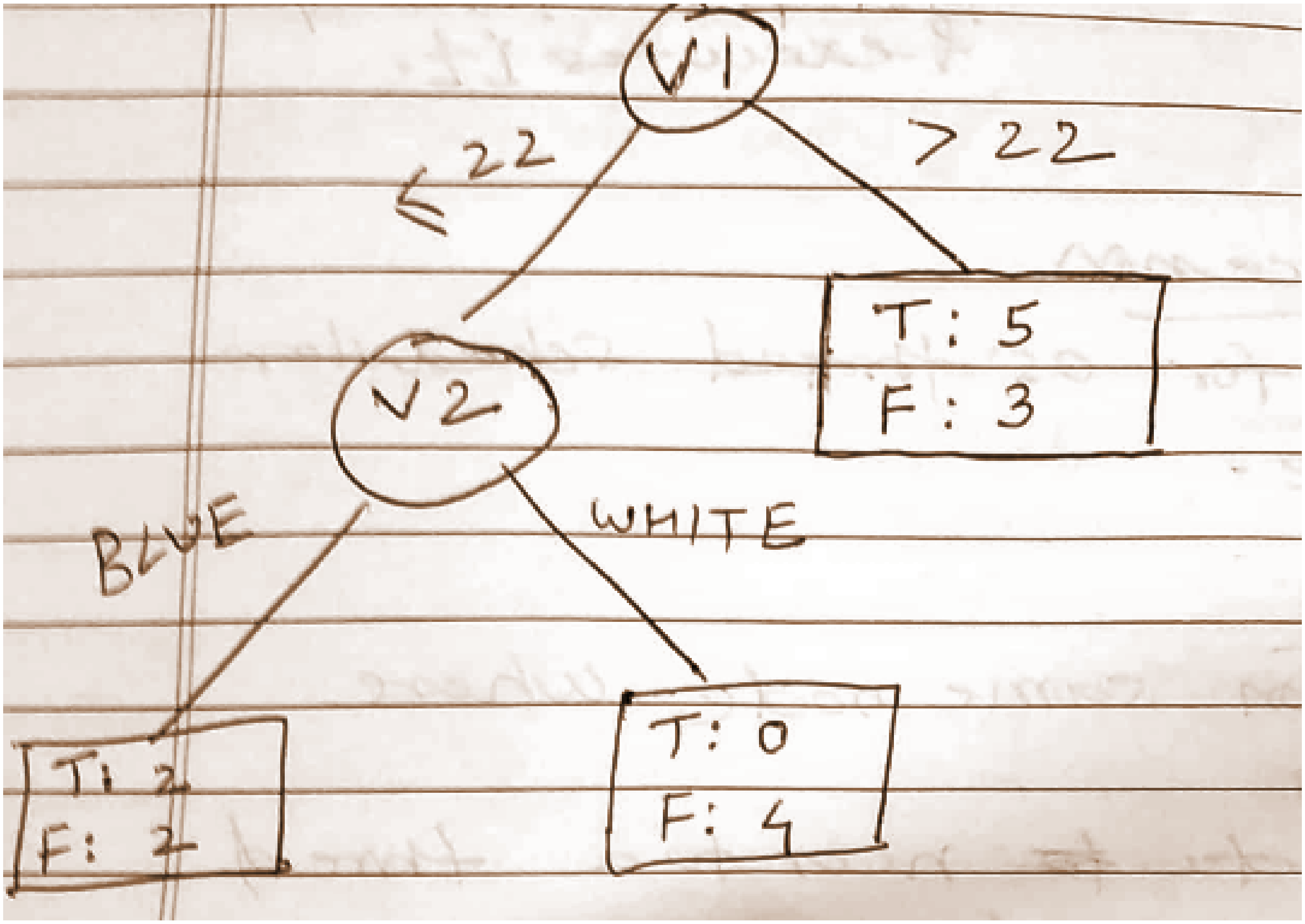
GINI split(V4) = 0.375

Calculating GINI split for V5

GINI(High) = 1-4/64-36/64 = 0.375

GINI(Low) = 1-0/0-0/0 = 1
GINI split(V5) = 0.25   (Ignoring Divide by zero error)

Considering V2 as split attribute as it has least GINI split value and is leftmost.



B] RIGHT Subtree where V1 > 22

Calculating GINI split for V2
GINI(Blue) = 1-9/16-1/16 = 0.375
GINI(White) = 1-4/16-4/16 = 5
GINI split(V2) = 0.4375

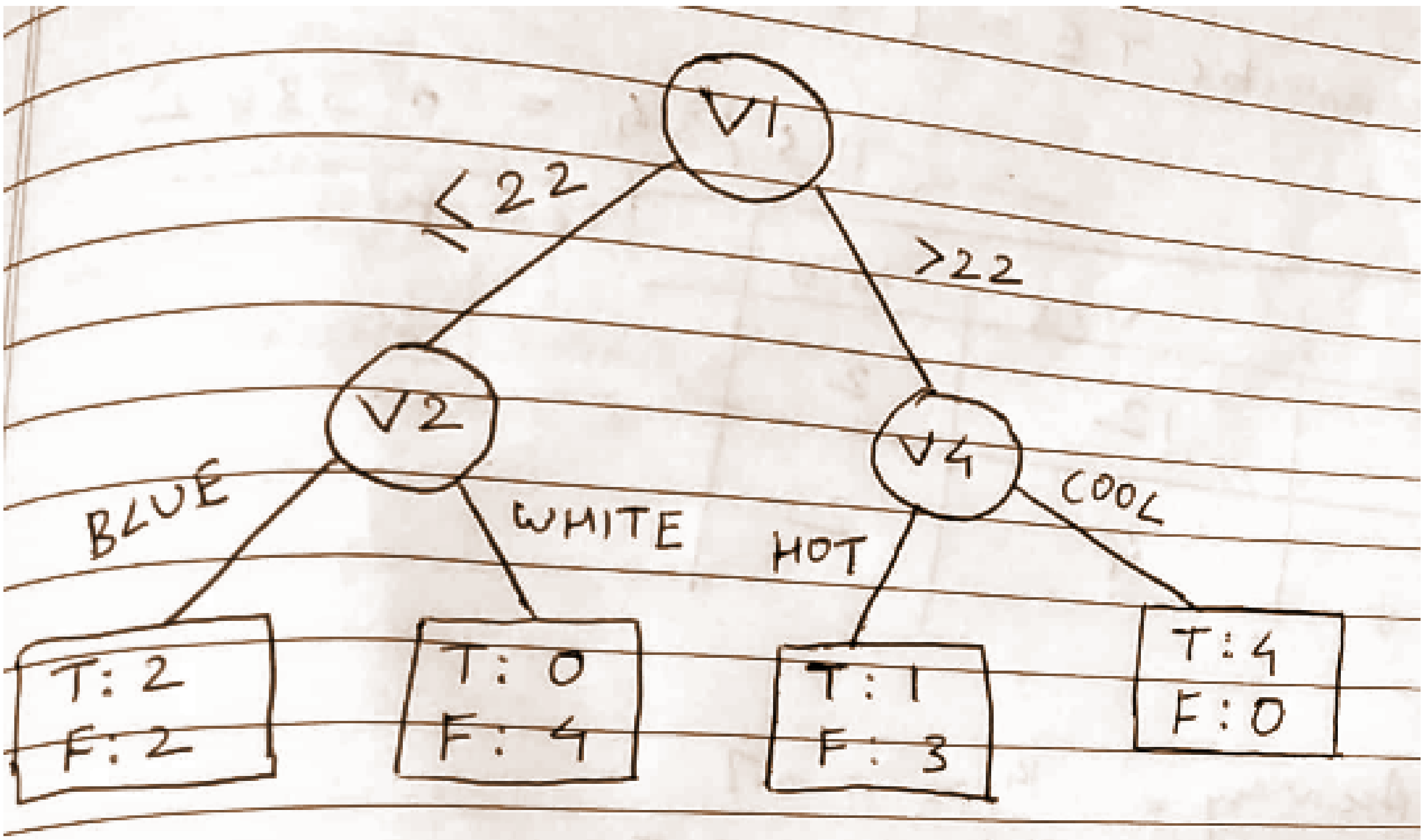
Calculating GINI split for V3
GINI(Short) = 1-9/16-1/16 = 0.375
GINI(Long) = 1-4/16-4/16 = 5
GINI split(V3) = 0.4375

Calculating GINI split for V4

GINI(Cool) = 1-16/16-0/16 = 0
GINI(Hot) = 1-1/16-9/16 = 0.375
GINI split(V4) = 0.1875

Calculating GINI split for V5
GINI(High) = 1-0/0-0/0 = 1 (Ignoring Divide by zero error)
GINI(Low) = 1-25/64-9/64 = 0.4687
GINI split(V5) = 0.4687

Considering V4 as split attribute as it has least GINI split value.



C]

1. Tree are different in the way they are constructed. Entropy based Decision Tree uses Information Gain as a value to select split nodes while GINI index based Decision Tree uses GINI_split value to select the split nodes.
2. Entropy based Decision Tree splits by small partitions at locations where it can reach a decision on at least one branch of the Tree. GINI based Decision Tree splits in the interior of the dataset i.e it favours larger partitions.
3. Continuous attribute V1 is best split at value 10 for Entropy based Decision Tree and at value 22 for GINI based Decision Tree.
4. Entropy based Decision Tree is complex when compared to Gini index based Decision Tree.

Examples of data objects classified differently by the two Decision Trees -->

1. 11 BLUE SHORT HOT HIGH
   Entropy : True
   Gini : Cannot reach to a decision

2. 40 BLUE LONG COOL LOW
   Entropy : Cannot reach to a decision
   Gini : True

D]

Entropy based Decision Tree will perform better on the Training Dataset.
Entropy based Decision Tree error = 2/16
Gini index based Decision Tree error = 5/16
Entropy based Decision Tree overfits the data as Training error is quite small.

Here, we do not know the test data set so we cannot compare the performance of the trees on the test data. However, we can predict that the GINI index based Decision Tree may perform better on the test data by considering the complexity of both the trees.

Solution 2 : Evaluation Measures

a)
Optimistic error = (2 + 2 + 4) / (34) = 4 / 17 = **0.2352**

Pessimistic error = [8 + (0.5 * 7)] / (34) = **0.3382**

b)
Confusion matrix based on the decision tree and csv file →

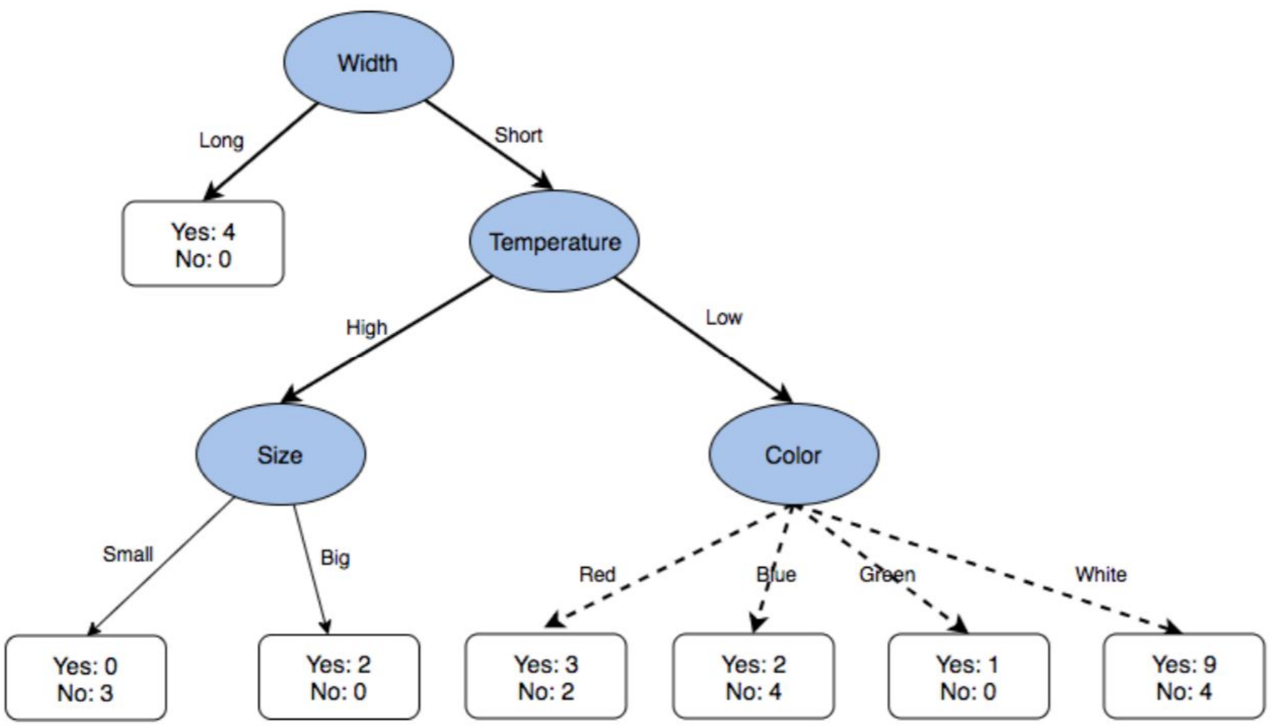
	Predicted Class		
Actual Class		Yes	No
	Yes	12	2
	No	4	2

TP = {8,10,9,11,12,13,14,15,16,19,20,21}
FN = {17, 18}
FP = {2, 3, 4, 7}
TN = {5, 6}

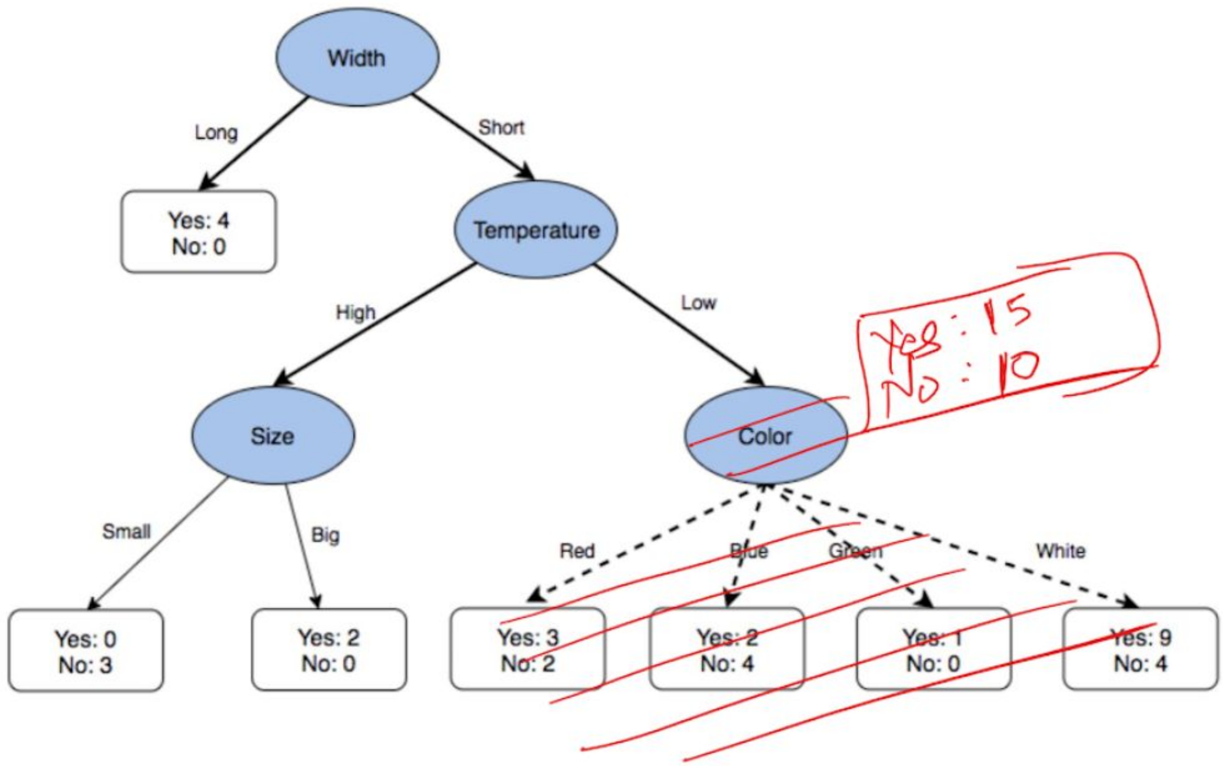
Accuracy = (TP + TN)/(TP + TN + FP + FN) = 14/20 = **0.7**
Error Rate = 1 - Accuracy = **0.3**
Precision = TP/(TP + FP) = 12/16 = **0.75**
Recall = TP/(TP + FN) = 12/14 = **0.85**
F1 score = (2*R*P) / (R + P) = 24/30 = **0.8**

Solution 3 : Decision Tree Pruning

The tree before pruning:



The tree after pruning:



a)

Before splitting
Errors = 0 + 0 + 0 + 10 = 10

Optimistic training classification error before splitting: 10/34 = 29.4%

After splitting
Errors = 2 + 2 + 0 + 4 = 8

Optimistic training classification error after splitting: 8/34 = 23.5%

No, the node shouldn't be pruned to minimize optimistic error rate, as the classification error after splitting is actually improving.

b)

Before splitting
Pessimistic training classification error before splitting: (10 + 0.8*4) /34 = 38.8%

After splitting
Errors = 2 + 2 + 0 + 4 = 8
Pessimistic training classification error after splitting: (8 + 0.8 * 7) / 34 = 40%

Yes, the node should be pruned to minimize pessimistic error rate, as the classification error after splitting does not improve.

c)

An example tree is demonstrated after pruning the color node.

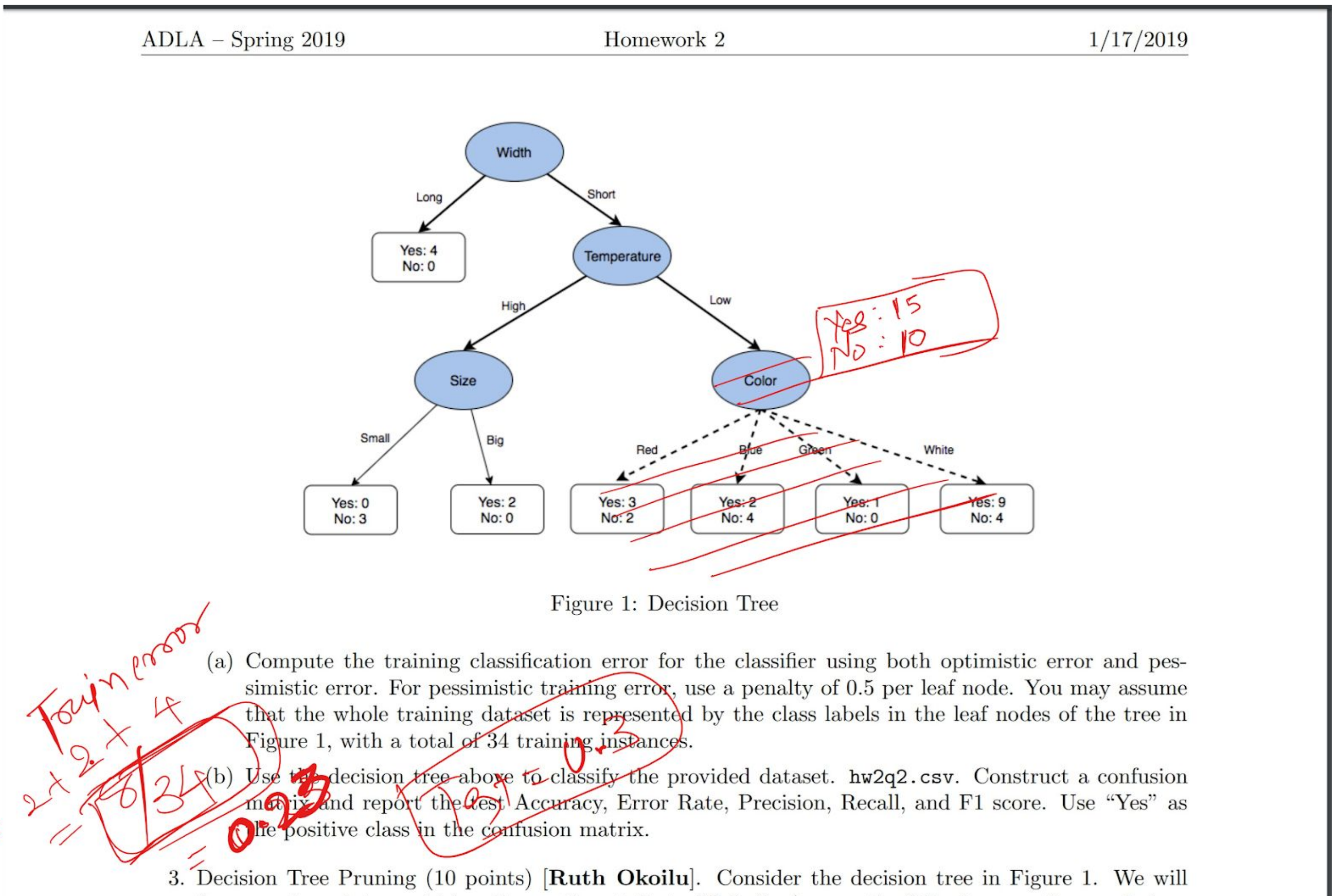

ADLA – Spring 2019 Homework 2 1/17/2019

Figure 1: Decision Tree

Train error = 2 + 2 + 4 = 8/34 = 0.23

(a) Compute the training classification error for the classifier using both optimistic error and pessimistic error. For pessimistic training error, use a penalty of 0.5 per leaf node. You may assume that the whole training dataset is represented by the class labels in the leaf nodes of the tree in Figure 1, with a total of 34 training instances.

Test = 0.3

(b) Use the decision tree above to classify the provided dataset. hw2q2.csv. Construct a confusion matrix and report the test Accuracy, Error Rate, Precision, Recall, and F1 score. Use "Yes" as the positive class in the confusion matrix.

3. Decision Tree Pruning (10 points) [**Ruth Okoilu**]. Consider the decision tree in Figure 1. We will focus on the sub-tree which splits on the attribute "Color". Answer the following questions and show

The error rate on the test data is recalculated as follows:

Width	Temper	Size	Color	Label	Label Before Prunir	Classification Error	**Label After Pruning**	**Classification Error**
Long	Low	Small	White	No	Yes	Yes	**Yes**	**Yes**
Short	Low	Big	Red	No	Yes	Yes	**Yes**	**Yes**
Short	Low	Big	Red	No	Yes	Yes	**Yes**	**Yes**
Short	Low	Big	Blue	No	No	No	**Yes**	**Yes**
Short	Low	Small	Blue	No	No	No	**Yes**	**Yes**
Short	Low	Big	White	No	Yes	Yes	**Yes**	**Yes**
Long	Low	Big	Blue	Yes	Yes	No	**Yes**	**No**
Long	Low	Big	Red	Yes	Yes	No	**Yes**	**No**
Long	Low	Big	Blue	Yes	Yes	No	**Yes**	**No**
Long	Low	Small	Red	Yes	Yes	No	**Yes**	**No**
Long	Low	Small	Red	Yes	Yes	No	**Yes**	**No**
Long	Low	Small	White	Yes	Yes	No	**Yes**	**No**
Short	Low	Big	Green	Yes	Yes	No	**Yes**	**No**
Short	Low	Big	Red	Yes	Yes	No	**Yes**	**No**
Short	High	Big	Blue	Yes	Yes	No	**Yes**	**No**
Short	Low	Small	Blue	Yes	No	Yes	**Yes**	**No**
Short	High	Small	Red	Yes	No	Yes	**No**	**Yes**
Short	Low	Small	Red	Yes	Yes	No	**Yes**	**No**
Short	High	Big	Green	Yes	Yes	No	**Yes**	**No**
Short	Low	Big	White	Yes	Yes	No	**Yes**	**No**
					Misclassified labels before pruning	6	**Misclassified labels after pruning**	7
					Error rate	0.3	**Error rate**	0.35

7 data points were mis-identified after pruning so the **Test error rate is: 7/20 = 35%**

The error rate before pruning was 6/20 = 30%

As it can be seen, the error rate before pruning was 30% and after the color node was removed, that is, after the decision tree's complexity was reduced, it actually jumped to 35%. Implying, the complex tree with color node was actually better.

Thus, the original tree with color node was not overfitting. Even though complexity was decreased after pruning, the error rate increased. Also the difference between the two error rates (30% and 35%) isn't high enough to be classified as overfitting.

(Below is alternate explanation if we *compare test error with train error* to measure overfitting. Overfitting occurs when there's complex model with small training error but with high testing error. In the decision tree give, the training error is 0.23. However, the testing error, with/without color node is 0.3 and 0.35 respectively which is much higher compared to training error. Thus the tree is more complex than necessary and the training error doesn't provide good estimate on previously unseen records. Thus, it may be concluded that the model was overfitting.)

Solution 4 : 1-NN, Evaluation, Cross Validation

4(A) Distance Matrix is as following :

```
> dm <- as.matrix(dist(m))
> dm
          1         2         3         4         5         6         7         8         9
1  0.000000 32.745229 24.627221  9.848858 33.559648 13.601471 10.198039  5.830952 27.540879
2 32.745229  0.000000  8.139410 41.500000  2.236068 45.500000 42.547033 35.514082  5.590170
3 24.627221  8.139410  0.000000 33.533565  9.013878 37.529988 34.503623 27.613403  3.162278
4  9.848858 41.500000 33.533565  0.000000 42.547033  4.000000  2.236068  6.082763 36.585516
5 33.559648  2.236068  9.013878 42.547033  0.000000 46.542991 43.500000 36.623080  6.020797
6 13.601471 45.500000 37.529988  4.000000 46.542991  0.000000  3.605551 10.049876 40.577087
7 10.198039 42.547033 34.503623  2.236068 43.500000  3.605551  0.000000  7.615773 37.503333
8  5.830952 35.514082 27.613403  6.082763 36.623080 10.049876  7.615773  0.000000 30.700163
9 27.540879  5.590170  3.162278 36.585516  6.020797 40.577087 37.503333 30.700163  0.000000
> 
```

4(B)
(i) Hold out test using last 4 data set:

Table 1: 1-NN

ID	x_1	x_1	y
1	35.0	15.0	-
2	2.5	11.0	-
3	10.5	12.5	+
4	44.0	11.0	+
5	1.5	13.0	-
6	48.0	11.0	+
7	45.0	13.0	-
8	38.0	10.0	+
9	7.5	13.5	-

highlighted is the test data set,
If we use the distance matrix shared above and 1 NN classifier, predicted classes will be

Id	Closest Neighbour	Distance	Class (Predicted)	Class (Actual)
6	4	4	+	+
7	4	2.23	+	-
8	1	5.83	-	+
9	3	3.16	+	-

Confusion Matrix

	Predicted(Yes)	Predicted(No)
Actual (Yes)	TP=1	FN=1
Actual(No)	FP=2	TN=0

Testing Accuracy

Accuracy = TP + TN / Total = (1 + 0) / 4 = 25%

(ii) 3 Fold cross validation using (3,6,9) (1,4,7) (2,5,8)

Test=(3,6,9), Train=(1,2,4,5,7,8)

Id	Closest Neighbour	Distance	Class (Predicted)	Class (Actual)
3	2	8.14	-	+
6	7	3.6	-	+
9	2	5.6	-	-

Confusion Matrix 1

	Predicted(Yes)	Predicted(No)
Actual (Yes)	TP=0	FN=2
Actual(No)	FP=0	TN=1

Test=(1,4,7), Train=(2,3,5,6,8,9)

Id	Closest Neighbour	Distance	Class (Predicted)	Class (Actual)
1	8	5.83	+	-
4	6	4.00	+	+

7	6	3.60	+	-

Confusion Matrix 2

	Predicted(Yes)	Predicted(No)
Actual (Yes)	TP=1	FN=0
Actual(No)	FP=2	TN=0

Test=(2,5,8), Train=(1,3,4,6,7,9)

Id	Closest Neighbour	Distance	Class (Predicted)	Class (Actual)
2	9	5.59	-	-
5	9	6.02	-	-
8	1	5.83	-	+

Confusion Matrix 3

	Predicted(Yes)	Predicted(No)
Actual (Yes)	TP=0	FN=1
Actual(No)	FP=0	TN=2

Confusion Matrix for k-fold cross-validation is generally created by summing up all individual confusion matrices.

Thus, summing up Confusion Matrix 1, 2 and 3 defined above,

3-fold Confusion Matrix

	Predicted(Yes)	Predicted(No)
Actual (Yes)	TP=1	FN=3
Actual(No)	FP=2	TN=3

3-fold Accuracy

Accuracy = TP + TN / Total = (1 + 3) / 9 = 44%

(iii) LOOCV

Id	Closest	Distance	Predicted	Actual
1	8	5.83	+	-
2	5	2.23	-	-
3	9	3.16	-	+
4	7	2.23	-	+
5	2	2.23	-	-
6	7	3.6	-	+
7	4	2.23	+	-
8	1	5.83	-	+
9	3	3.16	+	-

Confusion Matrix

	Predicted(Yes)	Predicted(No)
Actual (Yes)	TP=0	FN=4
Actual(No)	FP=3	TN=2

Testing Accuracy

Accuracy = TP + TN / Total = (0 + 2) / 9 = 22%

4 C LOOCV vs Simple

The dataset has 10 +ve and 10-ve instances, and the new algorithm chooses the majority element. This means that if we perform LOOCV,
CASE 1 : If the left out element belongs to +ve class , it means that -ve will be in majority. Hence according to our algorithm we shall assign the majority class to it => we assign -ve class.
CASE 2: If we pick an element belonging to -ve class , we will assign it +ve class.

Accuracy = (TP+TN)/(TP+TN+FP+FN)
With our experiment we shall get TP=0 and TN=0 , our classification will lead to either FP or FN. Hence accuracy for our experiment will be 0.

Solution 5 : R Programming

5(e) Below are the confusion matrix for all the experiments. I correct classifications are on the diagonal. Rest all values are misclassification.

> **euclidean_result**
$`confmat`

	Reference			
Prediction	1	2	3	4
1	8	5	7	6
2	0	9	1	0
3	0	0	6	0
4	1	1	1	5

$accuracy
Accuracy= 0.56

> **cosine_result**
$`confmat`

	Reference			
Prediction	1	2	3	4
1	9	0	1	2
2	0	15	1	1
3	0	0	13	1
4	0	0	0	7

$accuracy
Accuracy = 0.88

> **conf_result**
$`confmat`

	Reference			
Prediction	1	2	3	4
1	9	0	0	1
2	0	14	1	1
3	0	0	14	1
4	0	1	0	8

$accuracy
Accuracy= 0.9

> **dt_result**
$`confmat`
Reference
Prediction 1 2 3 4
1 8 6 7 5
2 0 8 0 0
3 0 1 8 1
4 1 0 0 5

$accuracy
Accuracy = 0.58

> **dt_cv_result**
$`confmat`
Reference
Prediction 1 2 3 4
1 8 6 7 5
2 0 8 0 0
3 0 1 8 1
4 1 0 0 5

$accuracy
Accuracy = 0.58
Simple Kfold validation gives accuracy of .46 but if we tune the parameters using tuneLengh=20, we can achieve accuracy = .58 which is equal to

1. **Overall accuracy** : knn_classifier_confidence performs the best with accuracy=0.9
2. **Misclassification** : knn_classifier_confidence performs the best as it has least misclassifications.

Sno	Experiment	#Misclassification	Ratio
1	**knn_classifier(euclidean)**	22	.44
2	**knn_classifier(cosine)**	6	.12
3	**knn_classifier_confidence**	5	.10
4	**dtree**	21	.42
5	**dtree_cv**	21	.42

3. **Class Wise Misclassification and Model Performance** :

KNN_Eucledian:

Reference

Prediction	1	2	3	4
1	8	5	7	6
2	0	9	1	0
3	0	0	6	0
4	1	1	1	5

Class 3 performs the best as it has 0 misclassifications in
Class 1 performs the worst with 18 wrong calls out of 26.

KNN_cosine

Reference

Prediction	1	2	3	4
1	9	0	1	2
2	0	15	1	1
3	0	0	13	1
4	0	0	0	7

Class 4 performs the best as it has 0 misclassifications
Class 1 performs worst with 3 mis predictions

KNN_Confidence

Reference

Prediction	1	2	3	4
1	9	0	0	1
2	0	14	1	1
3	0	0	14	1
4	0	1	0	8

Class 3 performs the best as it has 1 Misclassification from 15 instances
Class 2 has max mis classification =2

DTree and DTree_CV (Both models have same results)

Reference

Prediction	1	2	3	4
1	8	6	7	5
2	0	8	0	0
3	0	1	8	1
4	1	0	0	5

Class 2 performs the best as it has 0 misclassifications.
Class 1 has max misclassification = 18