Section B

Wednesday, January 25, 2023 12:23 PM

Steps of ML: (1) Learn (2) Predict (3) Evaluate

Alg. 4: Decision Tree

```
def h(x⃗):
    return h_recurse(root, x⃗)
def h_recurse(node, x⃗):
    if node.type == "Leaf":
        return node.vote
    else:
        next = branches[X_m]
        return h_recurse(next, x⃗)
```

recursive

```
class Node:
    str type      // "Leaf" or "Internal"
    y vote        // label for a leaf node
    {} branches   // map from attribute values to Node objects
    int m         // attribute for internal node
```

```
def train(D):
    root = train_recurse(D)
    store root

def train_recurse(D'):
    Let p = new Node()
```

Base Case If (a) all labels in D' are the same

(b) D' is empty

(c) for each attribute in D', all values are identical

p.type = "Leaf"

p.vote = majority_vote(D')

return p

Recursive Case Otherwise

p.type = "Internal"

p.m = best attribute according to $\boxed{\text{splitting criterion}}$

$= \underset{m \in \{1,\dots,M\}}{\operatorname{argmax}} \text{ splitting\_criterion}(D', m)$

store attribute on which to split

(1) accuracy = 1 − (error)

(2) Gini Gain

(3) Mutual Information

for each value v of attribute X_m:

$D_{X_m=v} = \{(\vec{x}, y) \in D' : X_m = v\}$ select a partition of D'

child_v = train_recurse($D_{X_m=v}$) recursion

p.branches[v] = child_v add a branch w/ label v

return p

similar to Decision Stump, but we recursively subdivide the dataset

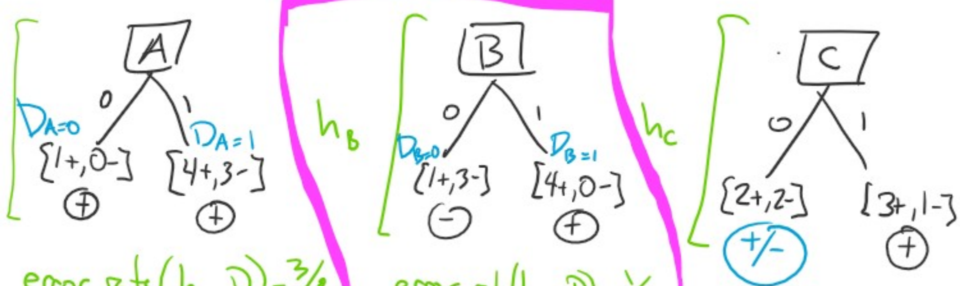
p.branches[v] = child$_v$ ← add a branch w/label v
return p

Example: Decision Tree Learning w/ Accuracy as Splitting Criterion

create root: D has [5+, 3-]

y	A	B	C
−	1	0	0
−	1	0	1
−	1	0	0
+	0	0	1
+	1	1	0
+	1	1	1
+	1	1	0
+	1	1	1

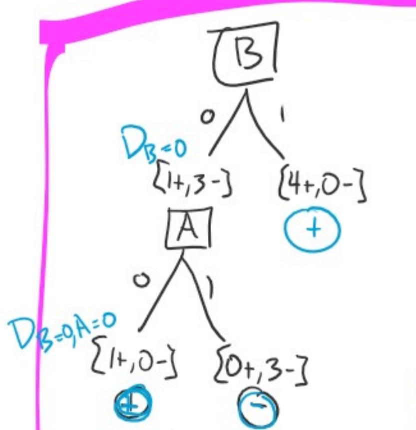
$D_{B=0}$ $D_{B=1}$



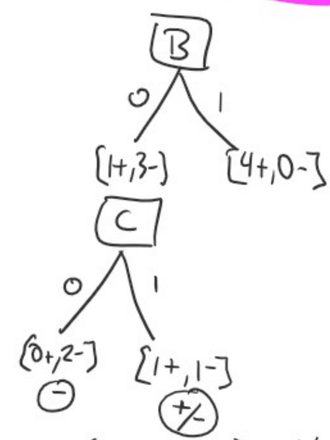
error_rate(h_A, D) = 3/8

error_rate(h_B, D) = 1/8

error_rate(h_C, D) = 3/8 regardless of tie



error(h_{BA}, $D_{B=0}$) = 0/4



error(h_{BC}, $D_{B=0}$) = 1/4

Mutual Information

Given a random variable Y over K classes {1, ..., K}

Def: Entropy $H(Y; D) = -\sum_{k=1}^{K} \underbrace{P(Y=k)}_{\text{prob of die landing on side k}} \log_2 P(Y=k)$

For DT: $P(Y=k) = \frac{\# Y=k}{|D|}$

(informal): "how impure are the labels from Y"

Def: a set of values is pure if all are the same

"how much randomness there is in Y"

(for DT): want to reduce entropy of r.v. we are trying to predict how much randomness there is in ?

(for DT): want to reduce entropy of r.v. we are trying to predict

Def: Mutual Information (for binary attribute X_m)

$$I(Y, X_m; D) = \underbrace{H(Y;D)}_{\text{entropy at parent}} - \underbrace{\left(P(X_m=0)\,\underbrace{H(Y;D_{X_m=0})}_{\text{entropy at left child}} + P(X_m=1)\,\underbrace{H(Y;D_{X_m=1})}_{\text{entropy at right child}}\right)}_{\text{weighted entropy of children}}$$

For DT:

$$\boxed{P(X_m=v) = \frac{\# X_m = v}{|D|}}$$

OH

$$P(A,C) = \underline{P(A)}\,\underline{P(C)}$$

$$P(A=a, C=c) = P(A=a)P(C=c)$$

$\forall a, c$

$P(A=a_1) = 0.1 + 0.05 + 0.15$

$P(A=a_2) =$ " "

$P(A=a_3) =$ " "

$P(C=c_1) =$

$=$

$P(C=c_3) =$

A \ C	c_1	c_2	c_3
a_1			
a_2			
a_3			

def h_recurse(node, myx)

if ..

else :

[ ] m] == 0 :

class Node

~~branches~~

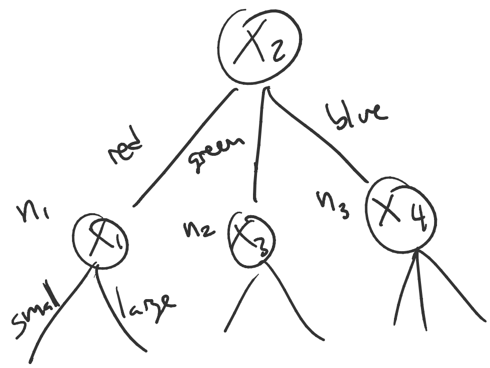
left_child

right_child

right_child

```
else:
    if myx[node.m] == 0:
        next = node.left_child
    else:
        next = node.right_child
```

branches = { red : n_1,
green : n_2,
blue : n_3 }



branches[red]

$$P(B=b_1 \mid A=a_2, C=c_1) = \frac{p(b_1, a_2, c_4)}{\sum_{b \in \{b_1, \ldots, b_3\}} p(b, a_2, c_4)}$$