

Indian Institute of Information Technology, Allahabad



Introduction to Machine Learning Decision Trees



By

Dr. Shiv Ram Dubey

Associate Professor

Computer Vision and Biometrics Lab

Department of Information Technology

Indian Institute of Information Technology, Allahabad

Web: <https://profile.iita.ac.in/srdubey/>

DISCLAIMER

The content (text, image, and graphics) used in this slide are adopted from many sources for academic purposes. Broadly, the sources have been given due credit appropriately. However, there is a chance of missing out some original primary sources. The authors of this material do not claim any copyright of such material.

Decision Trees

- Much more practical nonparametric learning algorithm
 - **Idea:** Can fit complex decision boundaries, but learns simpler ones first
- **Interpretable models:** Humans can look at the decision tree and understand what it is doing

Example: Diabetes Prediction



- Over the years, I've **collected data from lots of patients**, recording their physical information, their demographic information, habits, and done their lab work to diagnose diabetes.
- I'm wondering now: from all this data, could I **model the risk of other people** with similar characteristics having diabetes given all this other information about them?
- Eventually, we want to **explain our findings to patients**, and point out any behavioral changes that would mitigate their risk for diabetes.
- Even if the risk factors we find are non-modifiable, **insurance companies would be interested** in understanding and estimating this risk.
- Either way, it'd be great to have something that we can understand and interpret well!

Example: Diabetes Prediction

SEQN	RIDAGEYR	BMXWAIST	BMXHT	LBXTC	BMXLEG	BMXWT	BMXBMI	RIDRETH1	BPQ020	ALQ120Q	DMDDEDUC2	RIAGENDR	INDFMPPIR	LBXGH	DIABETIC
73557	69.0	100.0	171.3	167.0	39.2	78.3	26.7	Non-Hispanic Black	yes	1.0	high school graduate / GED	male	0.84	13.9	yes
73558	54.0	107.6	176.8	170.0	40.0	89.5	28.6	Non-Hispanic White	yes	7.0	high school graduate / GED	male	1.78	9.1	yes
73559	72.0	109.2	175.3	126.0	40.0	88.9	28.9	Non-Hispanic White	yes	0.0	some college or AA degree	male	4.51	8.9	yes
73562	56.0	123.1	158.7	226.0	34.2	105.0	41.7	Mexican American	yes	5.0	some college or AA degree	male	4.79	5.5	no
73564	61.0	110.8	161.8	168.0	37.1	93.4	35.7	Non-Hispanic White	yes	2.0	college graduate or above	female	5.0	5.5	no
73566	56.0	85.5	152.8	278.0	32.4	61.8	26.5	Non-Hispanic White	no	1.0	high school graduate / GED	female	0.48	5.4	no
73567	65.0	93.7	172.4	173.0	40.0	65.3	22.0	Non-Hispanic White	no	4.0	9th-11th grade	male	1.2	5.2	no
73568	26.0	73.7	152.5	168.0	34.4	47.1	20.3	Non-Hispanic White	no	2.0	college graduate or above	female	5.0	5.2	no
73571	76.0	122.1	172.5	167.0	35.5	102.4	34.4	Non-Hispanic White	yes	2.0	college graduate or above	male	5.0	6.9	yes
73577	32.0	100.0	166.2	182.0	36.5	79.7	28.9	Mexican American	no	20.0	Less than 9th grade	male	0.29	5.3	no
73581	50.0	99.3	185.0	202.0	42.8	80.9	23.6	Other or Multi-Racial	no	0.0	college graduate or above	male	5.0	5.0	no
73585	28.0	90.3	175.1	198.0	40.5	92.2	30.1	Other or Multi-Racial	no	4.0	some college or AA degree	male	2.26	5.0	no
73589	35.0	94.6	172.9	192.0	39.1	78.3	26.2	Non-Hispanic White	no	2.0	high school graduate / GED	male	1.74	5.5	no
73595	58.0	114.8	175.3	165.0	40.1	96.0	31.2	Other Hispanic	no	1.0	some college or AA degree	male	3.09	7.7	no
73596	57.0	117.8	164.7	151.0	35.3	104.0	38.3	Other or Multi-Racial	yes	1.0	college graduate or above	female	5.0	5.9	no
73600	37.0	122.9	185.1	189.0	48.1	126.2	36.8	Non-Hispanic Black	yes	2.0	high school graduate / GED	male	0.63	6.2	yes
73604	69.0	96.6	156.9	203.0	37.0	59.5	24.2	Non-Hispanic White	no	1.0	some college or AA degree	female	2.44	5.4	no
73607	75.0	130.5	169.6	161.0	36.5	111.9	38.9	Non-Hispanic White	yes	0.0	high school graduate / GED	male	1.08	5.0	no
73610	43.0	102.6	176.8	200.0	38.8	90.2	28.9	Non-Hispanic White	no	5.0	college graduate or above	male	2.03	4.9	no
73613	60.0	113.6	163.8	203.0	41.6	104.9	39.1	Non-Hispanic Black	yes	2.0	9th-11th grade	female	5.0	6.1	no
73614	55.0	90.9	167.9	256.0	43.5	60.9	21.6	Non-Hispanic White	no	0.0	high school graduate / GED	female	1.29	5.0	no
73615	65.0	100.3	145.9	166.0	30.0	55.4	26.0	Other Hispanic	yes	1.0	Less than 9th grade	female	1.22	6.3	yes

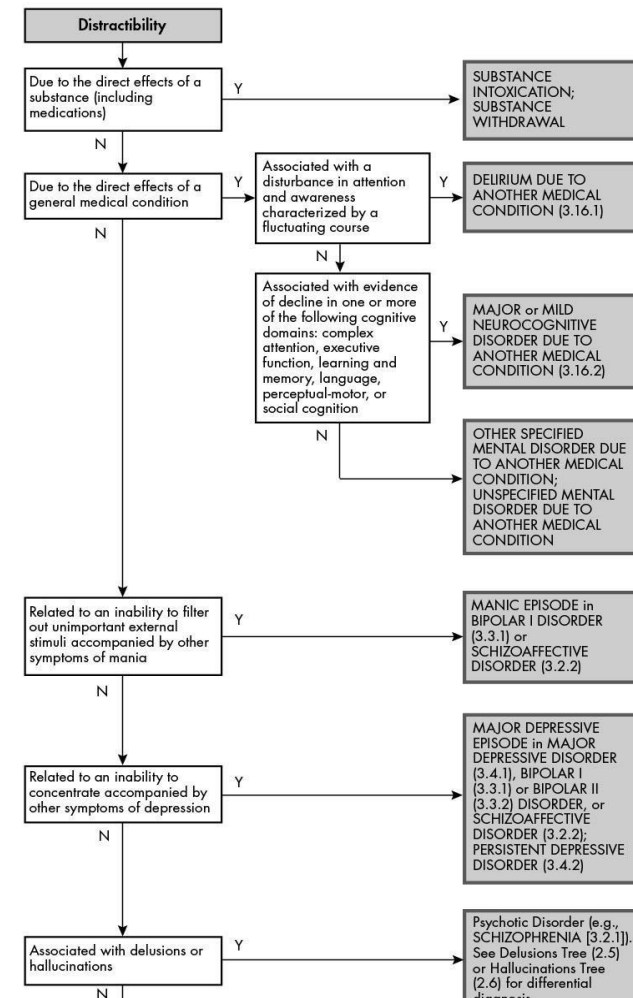
SEQN – Sequence number
RIDAGEYR – Age in years
BMXWAIST – Waist circumference
BMXHT – Height
LBXTC – Total cholesterol
BMXLEG – Leg length
BMXWT – Weight
BMXBMI – Body Mass Index
RIDRETH1 – Race/ethnicity
BPQ020 – Whether the participant is told they have high blood pressure
ALQ120Q – Alcohol consumption frequency
DMDDEDUC2 – Education level
RIAGENDR – Gender
INDFMPPIR – Family income-to-poverty ratio
LBXGH – Glycohemoglobin (HbA1c)
DIABETIC – Diabetes status

Inputs x_i

Labels y_i

Decision Trees

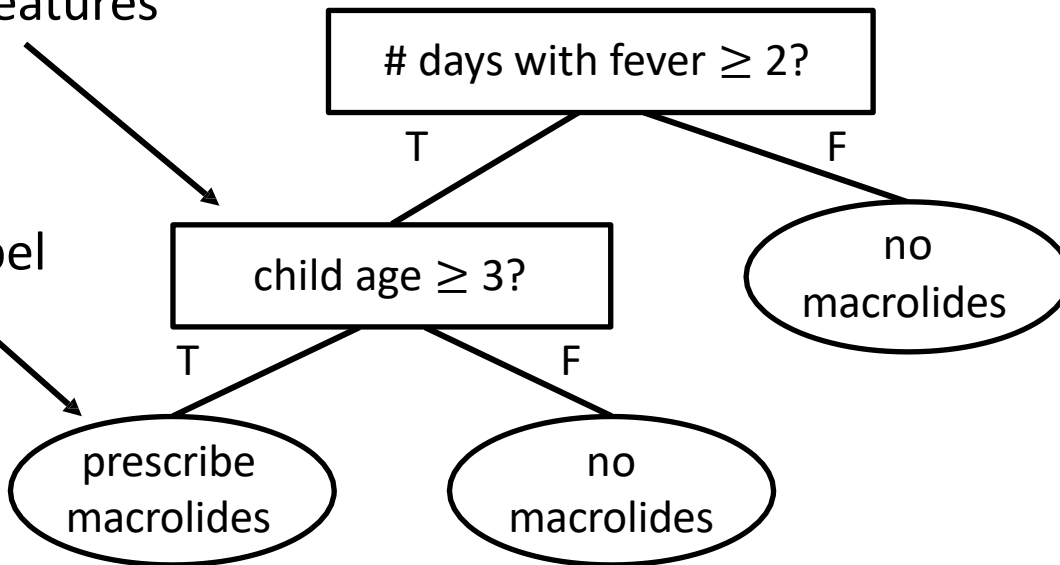
- A kind of flowchart based on tests
 - Commonly used in medicine
- “Explainable”, easy to mentally evaluate



Decision Trees

condition on features

predicted label



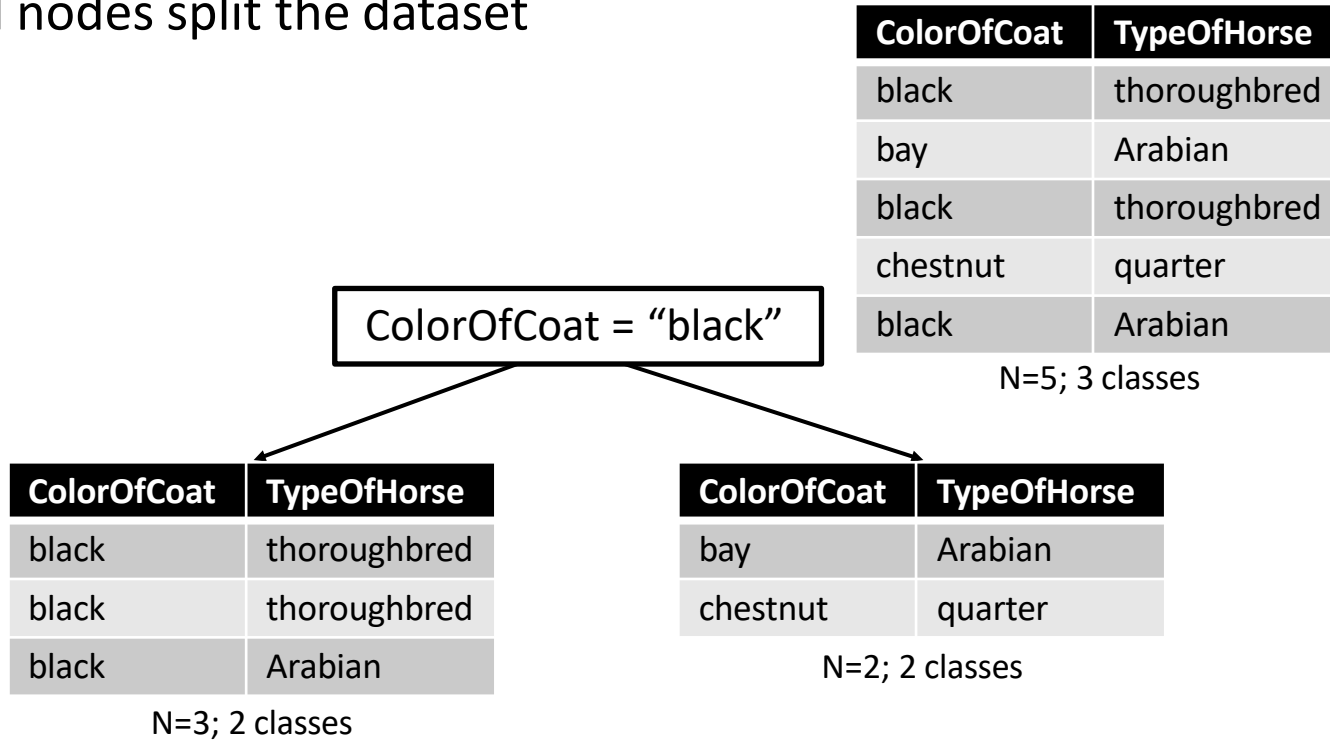
- Macrolides are a class of antibiotics to treat bacterial infections.
- Common examples include azithromycin, clarithromycin, and erythromycin.

Decision Trees

- Binary tree
- Each **internal node** has a Boolean condition that is a function of x
 - Typically reference a single feature x_j
 - **Real-valued feature:** Condition 1 ($x_j \geq t$) (where $t \in \mathbb{R}$)
 - **Categorical feature:** Condition 1 ($x_j = t$) (where $t \in \{1, \dots, k_j\}$ is a category)
- Each **leaf node** is a label
 - Can be either regression or classification
 - Can also be a probability distribution

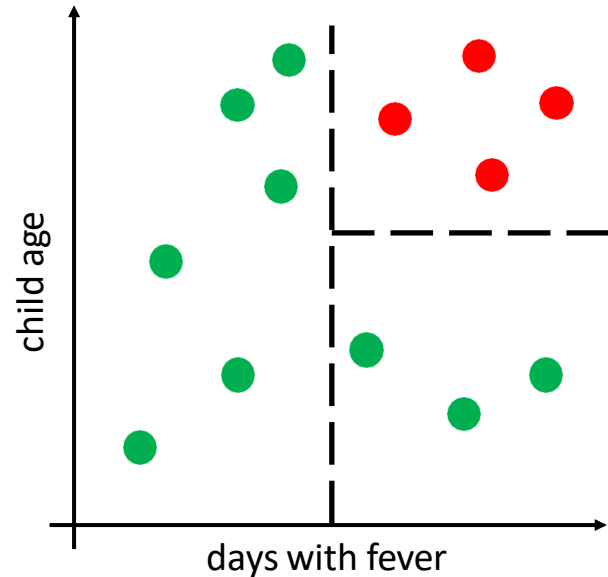
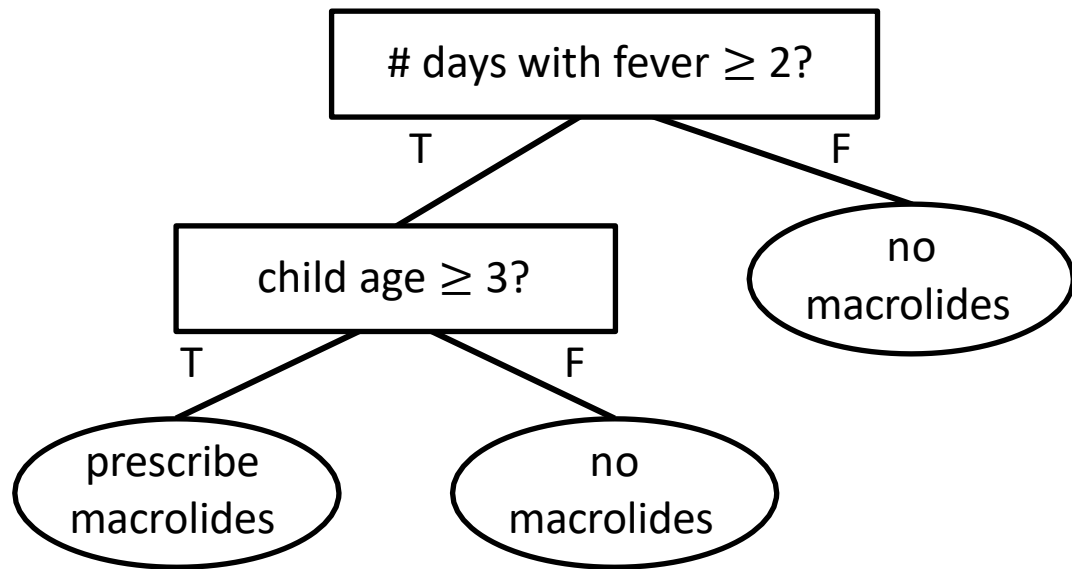
Intuition: Dataset Splitting

- Internal nodes split the dataset



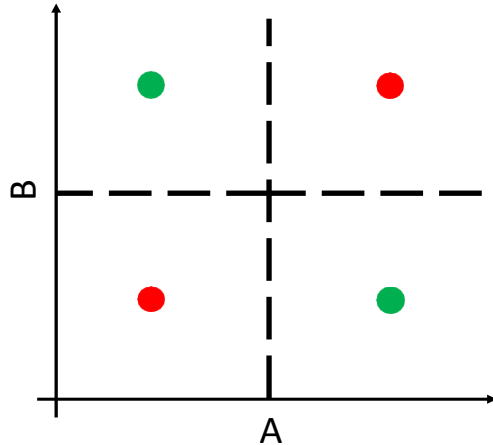
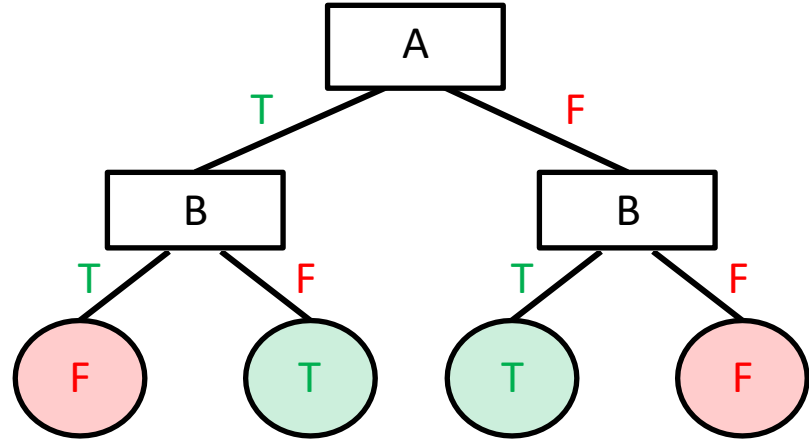
Visualizing the Model Family

- Axis-aligned decision boundaries



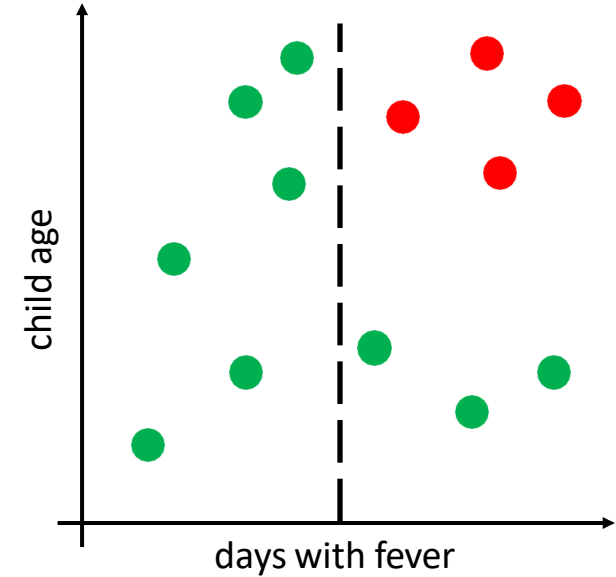
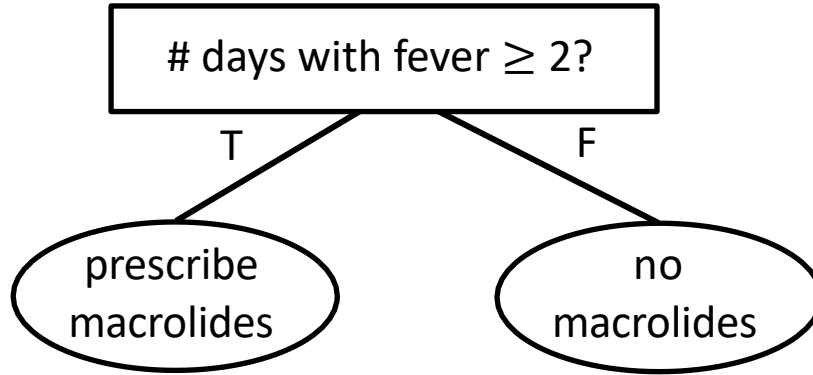
Example: Decision Trees and XOR

A	B	A XOR B
T	T	F
T	F	T
F	T	T
F	F	F



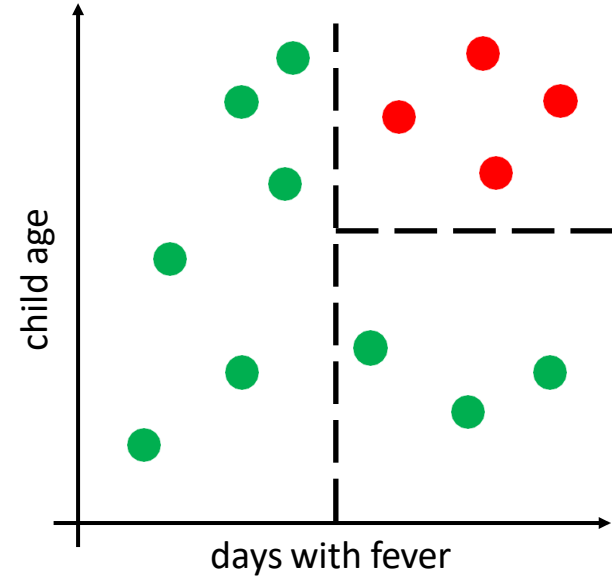
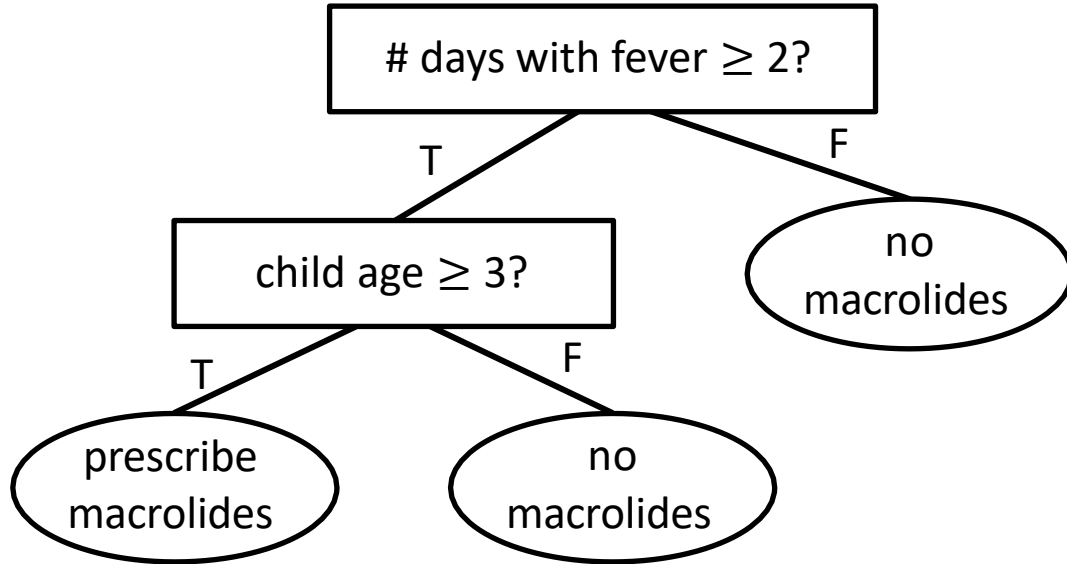
Decision Boundary and Depth

- Complexity increases with depth



Decision Boundary and Depth

- Complexity increases with depth



Decision Tree Learning Algorithms

- Similar to kNN, traditional decision tree learning algorithms do not fit in the loss minimization framework
 - **Computing the optimal decision tree is NP complete**
 - Recent work has tried to devise more efficient algorithms
- Instead, they are heuristically constructed in a top-down fashion.
- **Older algorithms:** ID3, C4.5, ...
- **Most popular current algorithm is CART**
 - “Classification And Regression Trees”
 - (Mostly) implemented in SciKit Learn

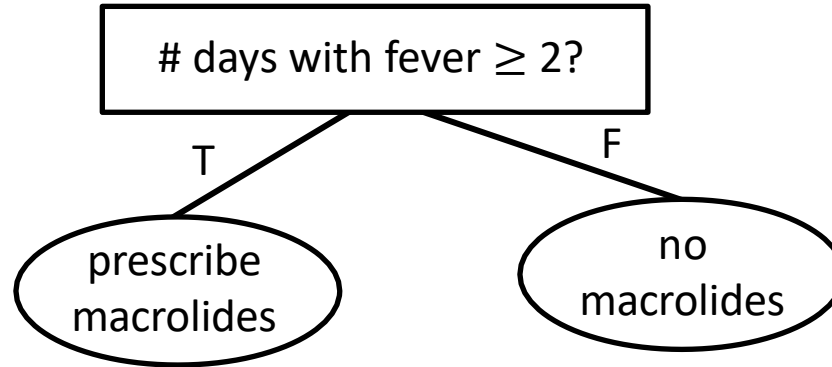
Learning Algorithm

no
macrolides

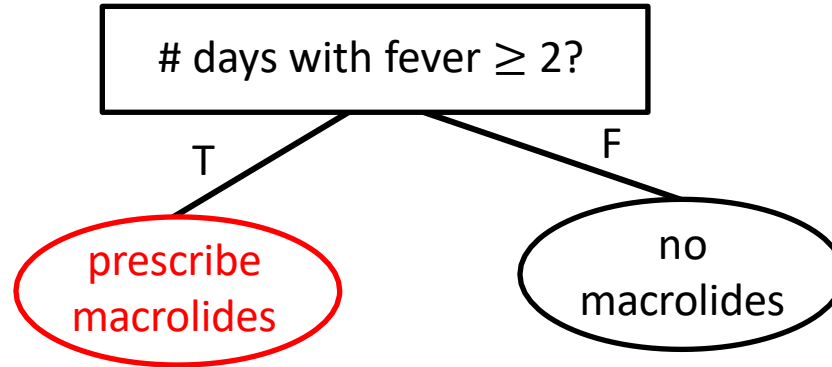
Learning Algorithm

no
macrolides

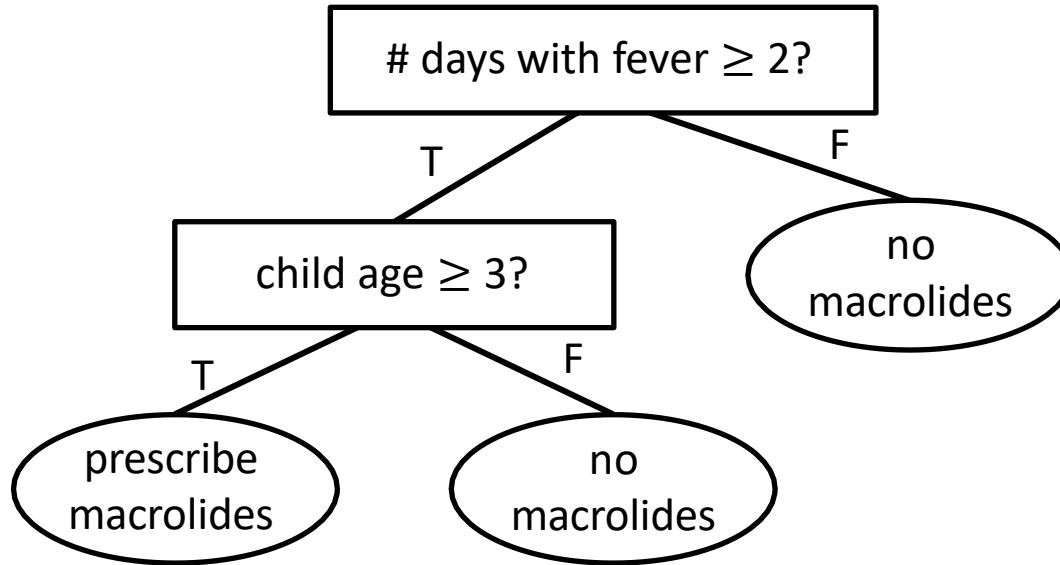
Learning Algorithm



Learning Algorithm



Learning Algorithm



Learning Algorithm

- Let $Z[C] = \{ (x, y) \in Z \mid C(x, y) \}$ be the subset of Z where C holds

```
def LearnTree( $Z$ ):
```

```
    if all labels in  $Z$  are the same and equal  $y$ :
```

```
        return LeafNode( $y$ )
```

```
     $(j, t) \leftarrow \text{BestSplit}(Z)$ 
```

```
     $T_{\text{left}} \leftarrow \text{LearnTree}(Z[x_j \geq t])$ 
```

```
     $T_{\text{right}} \leftarrow \text{LearnTree}(Z[x_j < t])$ 
```

```
    return InternalNode( $j, t, T_{\text{left}}, T_{\text{right}}$ )
```

Z

Learning Algorithm

- Let $Z[C] = \{ (x, y) \in Z \mid C(x, y) \}$ be the subset of Z where C holds

def LearnTree(Z):

if all labels in Z are the same and equal y :

return LeafNode(y)

$(j, t) \leftarrow \text{BestSplit}(Z)$

$T_{\text{left}} \leftarrow \text{LearnTree}(Z[x_j \geq t])$

$T_{\text{right}} \leftarrow \text{LearnTree}(Z[x_j < t])$

return InternalNode($j, t, T_{\text{left}}, T_{\text{right}}$)

Z

days with fever ≥ 2 ?

Learning Algorithm

- Let $Z[C] = \{ (x, y) \in Z \mid C(x, y) \}$ be the subset of Z where C holds

```
def LearnTree( $Z$ ):
```

```
    if all labels in  $Z$  are the same and equal  $y$ :
```

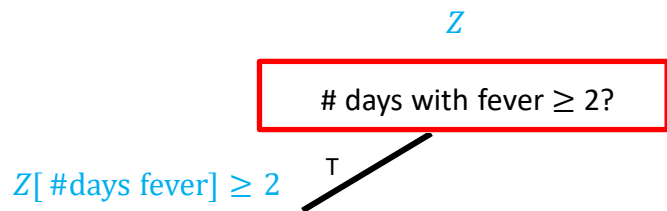
```
        return LeafNode( $y$ )
```

```
     $(j, t) \leftarrow \text{BestSplit}(Z)$ 
```

```
     $T_{\text{left}} \leftarrow \text{LearnTree}(Z[x_j \geq t])$ 
```

```
     $T_{\text{right}} \leftarrow \text{LearnTree}(Z[x_j < t])$ 
```

```
    return InternalNode( $j, t, T_{\text{left}}, T_{\text{right}}$ )
```



Learning Algorithm

- Let $Z[C] = \{ (x, y) \in Z \mid C(x, y) \}$ be the subset of Z where C holds

def LearnTree(Z):

if all labels in Z are the same and equal y :

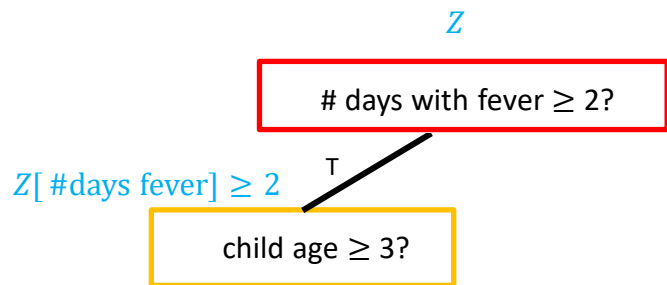
return LeafNode(y)

 (j, t) $\leftarrow$ BestSplit(Z)

$T_{\text{left}} \leftarrow$ LearnTree($Z[x_j \geq t]$)

$T_{\text{right}} \leftarrow$ LearnTree($Z[x_j < t]$)

return InternalNode($j, t, T_{\text{left}}, T_{\text{right}}$)



Learning Algorithm

- Let $Z[C] = \{ (x, y) \in Z \mid C(x, y) \}$ be the subset of Z where C holds

```
def LearnTree( $Z$ ):
```

```
    if all labels in  $Z$  are the same and equal  $y$ :
```

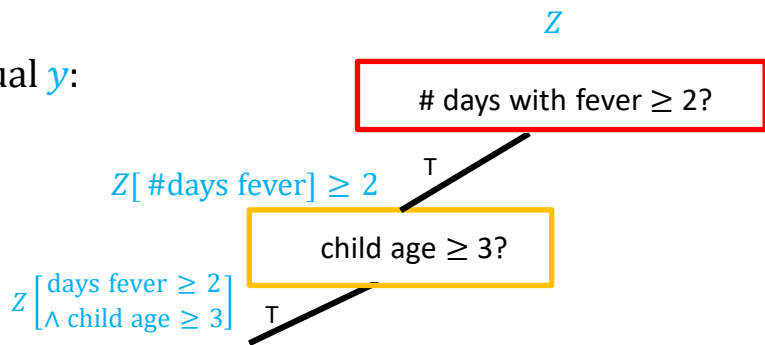
```
        return LeafNode( $y$ )
```

```
    ( $j, t$ )  $\leftarrow$  BestSplit( $Z$ )
```

```
     $T_{\text{left}} \leftarrow \text{LearnTree}(Z[x_j \geq t])$ 
```

```
     $T_{\text{right}} \leftarrow \text{LearnTree}(Z[x_j < t])$ 
```

```
    return InternalNode( $j, t, T_{\text{left}}, T_{\text{right}}$ )
```



Learning Algorithm

- Let $Z[C] = \{ (x, y) \in Z \mid C(x, y) \}$ be the subset of Z where C holds

def LearnTree(Z):

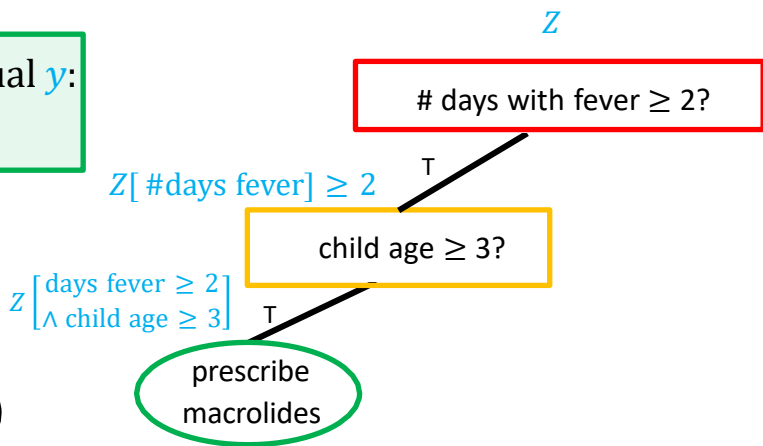
if all labels in Z are the same and equal y :
return LeafNode(y)

$(j, t) \leftarrow \text{BestSplit}(Z)$

$T_{\text{left}} \leftarrow \text{LearnTree}(Z[x_j \geq t])$

$T_{\text{right}} \leftarrow \text{LearnTree}(Z[x_j < t])$

return InternalNode($j, t, T_{\text{left}}, T_{\text{right}}$)



Learning Algorithm

- Let $Z[C] = \{ (x, y) \in Z \mid C(x, y) \}$ be the subset of Z where C holds

```
def LearnTree( $Z$ ):
```

```
    if all labels in  $Z$  are the same and equal  $y$ :
```

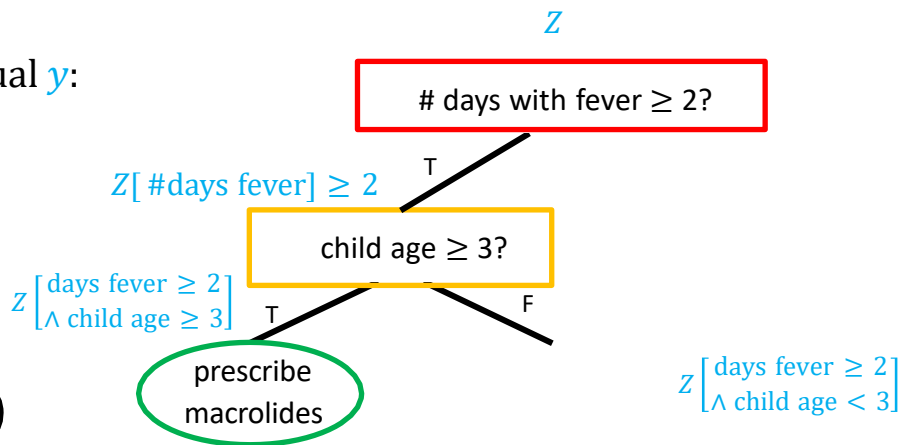
```
        return LeafNode( $y$ )
```

```
    ( $j, t$ )  $\leftarrow$  BestSplit( $Z$ )
```

```
     $T_{\text{left}} \leftarrow \text{LearnTree}(Z[x_j \geq t])$ 
```

```
     $T_{\text{right}} \leftarrow \text{LearnTree}(Z[x_j < t])$ 
```

```
    return InternalNode( $j, t, T_{\text{left}}, T_{\text{right}}$ )
```



Learning Algorithm

- Let $Z[C] = \{ (x, y) \in Z \mid C(x, y) \}$ be the subset of Z where C holds

def LearnTree(Z):

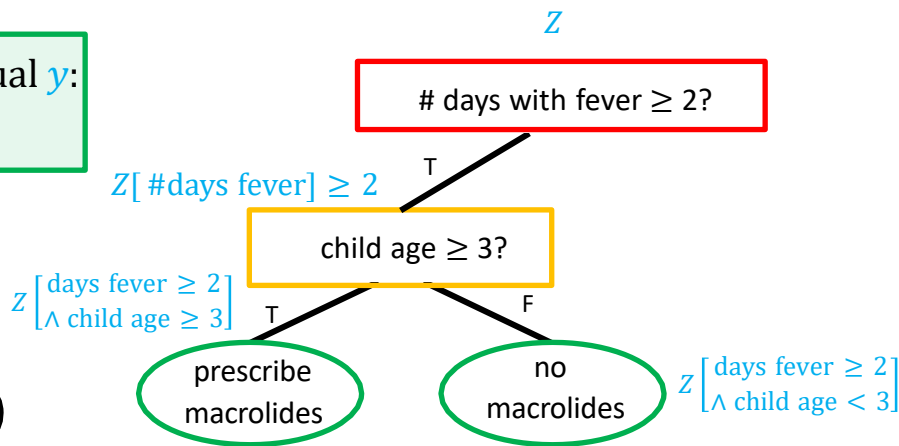
if all labels in Z are the same and equal y :
return LeafNode(y)

$(j, t) \leftarrow \text{BestSplit}(Z)$

$T_{\text{left}} \leftarrow \text{LearnTree}(Z[x_j \geq t])$

$T_{\text{right}} \leftarrow \text{LearnTree}(Z[x_j < t])$

return InternalNode($j, t, T_{\text{left}}, T_{\text{right}}$)



Learning Algorithm

- Let $Z[C] = \{ (x, y) \in Z \mid C(x, y) \}$ be the subset of Z where C holds

def LearnTree(Z):

if all labels in Z are the same and equal y :

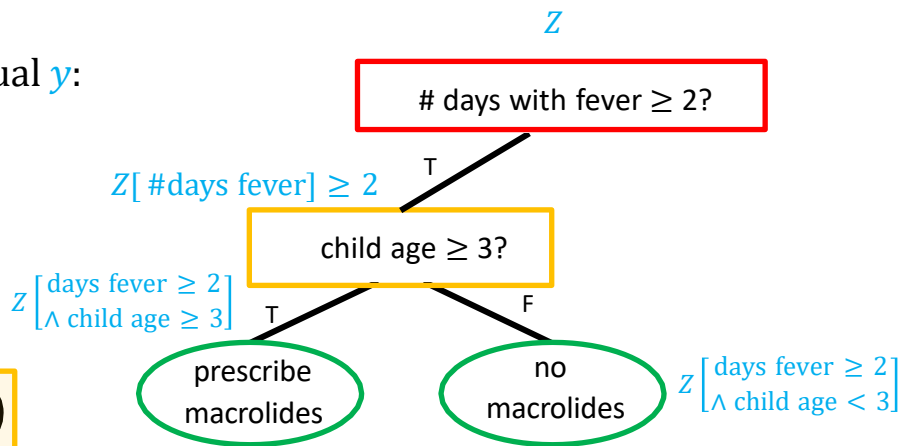
return LeafNode(y)

$(j, t) \leftarrow \text{BestSplit}(Z)$

$T_{\text{left}} \leftarrow \text{LearnTree}(Z[x_j \geq t])$

$T_{\text{right}} \leftarrow \text{LearnTree}(Z[x_j < t])$

return InternalNode($j, t, T_{\text{left}}, T_{\text{right}}$)



Learning Algorithm

- Let $Z[C] = \{ (x, y) \in Z \mid C(x, y) \}$ be the subset of Z where C holds

def LearnTree(Z):

if all labels in Z are the same and equal y :

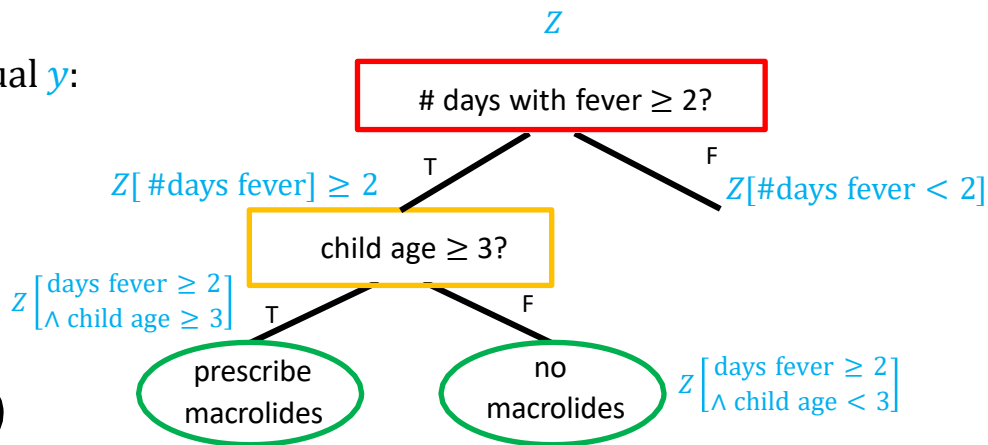
return LeafNode(y)

$(j, t) \leftarrow \text{BestSplit}(Z)$

$T_{\text{left}} \leftarrow \text{LearnTree}(Z[x_j \geq t])$

$T_{\text{right}} \leftarrow \text{LearnTree}(Z[x_j < t])$

return InternalNode($j, t, T_{\text{left}}, T_{\text{right}}$)



Learning Algorithm

- Let $Z[C] = \{ (x, y) \in Z \mid C(x, y) \}$ be the subset of Z where C holds

def LearnTree(Z):

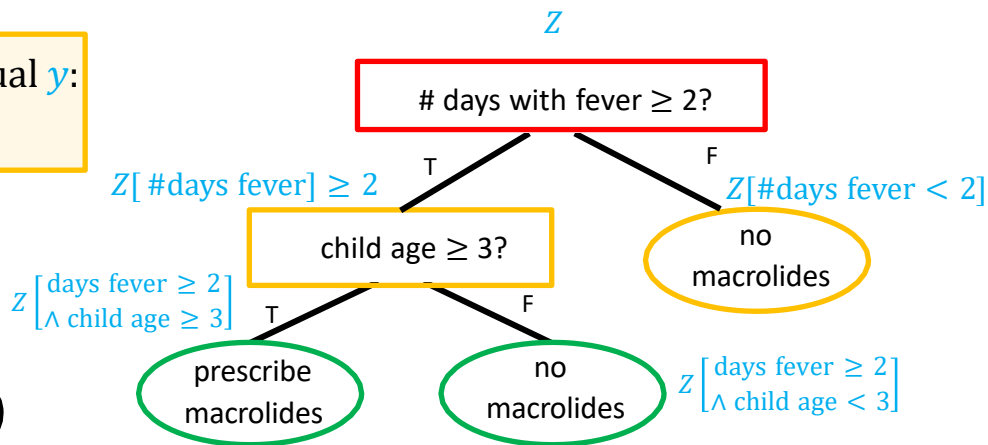
if all labels in Z are the same and equal y :
return LeafNode(y)

$(j, t) \leftarrow \text{BestSplit}(Z)$

$T_{\text{left}} \leftarrow \text{LearnTree}(Z[x_j \geq t])$

$T_{\text{right}} \leftarrow \text{LearnTree}(Z[x_j < t])$

return InternalNode($j, t, T_{\text{left}}, T_{\text{right}}$)



Learning Algorithm

- Let $Z[C] = \{ (x, y) \in Z \mid C(x, y) \}$ be the subset of Z where C holds

def LearnTree(Z):

if all labels in Z are the same and equal y :

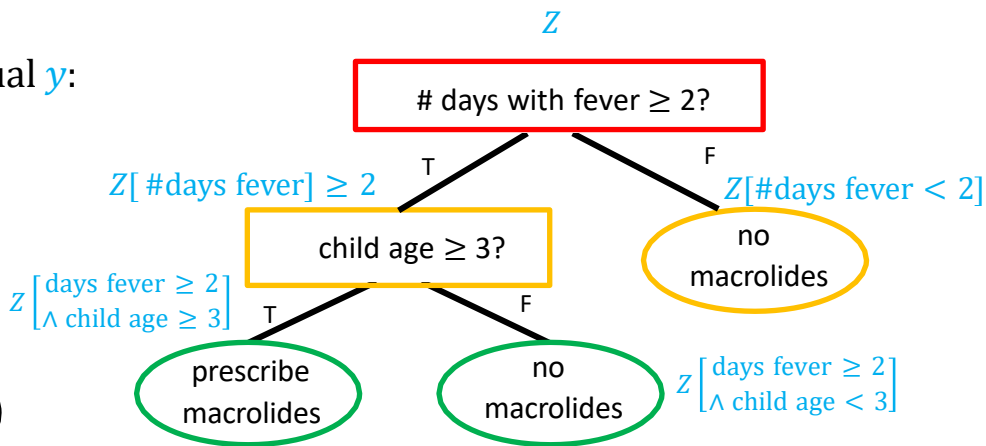
return LeafNode(y)

$(j, t) \leftarrow \text{BestSplit}(Z)$

$T_{\text{left}} \leftarrow \text{LearnTree}(Z[x_j \geq t])$

$T_{\text{right}} \leftarrow \text{LearnTree}(Z[x_j < t])$

return InternalNode($j, t, T_{\text{left}}, T_{\text{right}}$)



Learning Algorithm

- Let $Z[C] = \{ (x, y) \in Z \mid C(x, y) \}$ be the subset of Z where C holds

def LearnTree(Z):

if all labels in Z are the same and equal y :

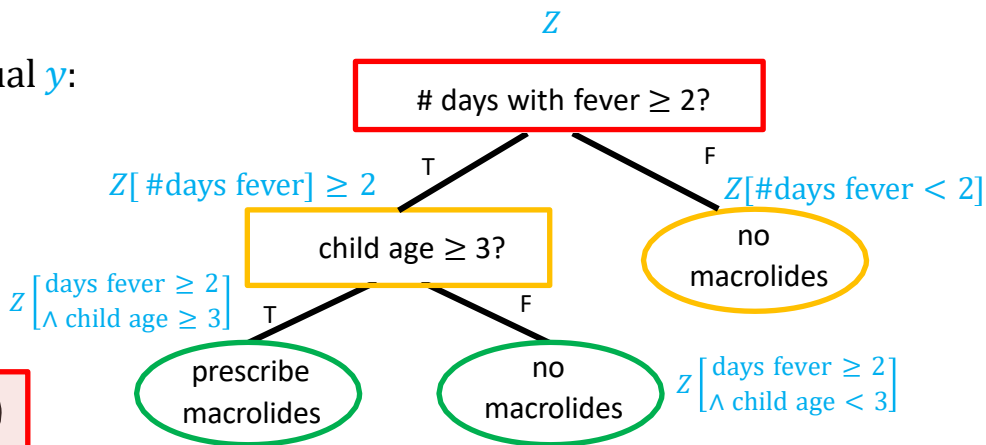
return LeafNode(y)

$(j, t) \leftarrow \text{BestSplit}(Z)$

$T_{\text{left}} \leftarrow \text{LearnTree}(Z[x_j \geq t])$

$T_{\text{right}} \leftarrow \text{LearnTree}(Z[x_j < t])$

return InternalNode($j, t, T_{\text{left}}, T_{\text{right}}$)



Learning Algorithm

- Let $Z[C] = \{ (x, y) \in Z \mid C(x, y) \}$ be the subset of Z where C holds

def LearnTree(Z):

if all labels in Z are the same and equal y :

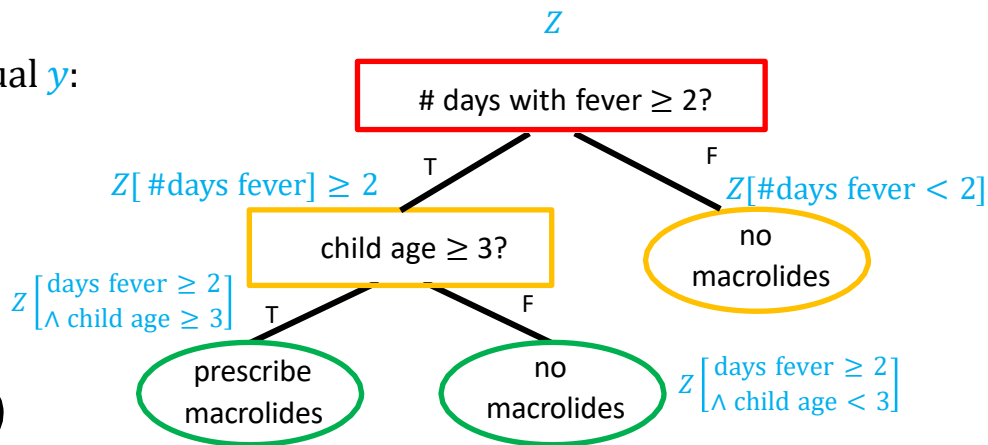
return LeafNode(y)

$(j, t) \leftarrow \text{BestSplit}(Z)$

$T_{\text{left}} \leftarrow \text{LearnTree}(Z[x_j \geq t])$

$T_{\text{right}} \leftarrow \text{LearnTree}(Z[x_j < t])$

return InternalNode($j, t, T_{\text{left}}, T_{\text{right}}$)



Learning Algorithm

- Let $Z[C] = \{ (x, y) \in Z \mid C(x, y) \}$ be the subset of Z where C holds

def LearnTree(Z):

if all labels in Z are the same and equal y :

return LeafNode(y)

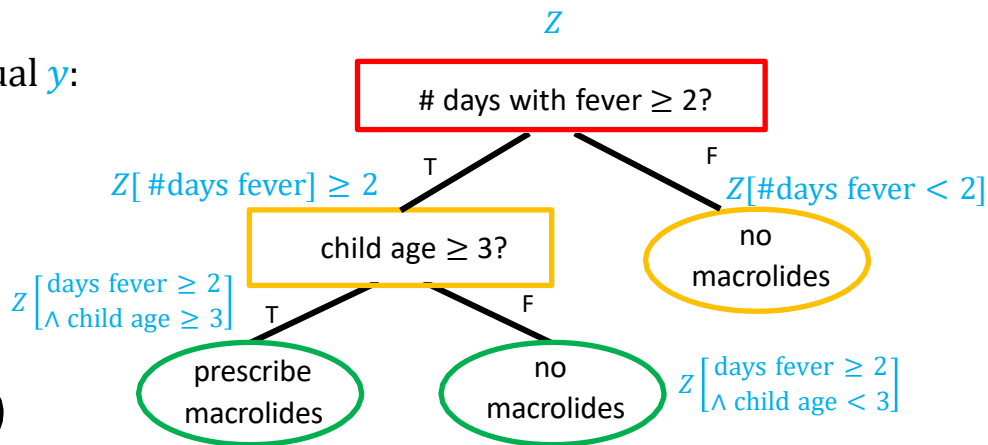
How to
choose the
best split?

$(j, t) \leftarrow \text{BestSplit}(Z)$

$T_{\text{left}} \leftarrow \text{LearnTree}(Z[x_j \geq t])$

$T_{\text{right}} \leftarrow \text{LearnTree}(Z[x_j < t])$

return InternalNode($j, t, T_{\text{left}}, T_{\text{right}}$)



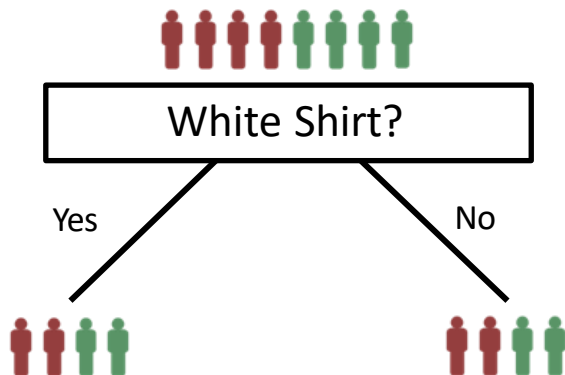
Choosing Splits

- What is the “best” split (j, t) for the current dataset Z ?
- **Intuition**
 - Want the tree to be as small as possible
 - Larger tree $\rightarrow$ more parameters $\rightarrow$ higher variance
 - To create a leaf node, need $Z[x_j \geq t]$ and $Z[x_j < t]$ to have “pure” labels
- **Strategy:** Choose the split (j, t) that results in the “purest” labels
 - Called **maximum information gain**

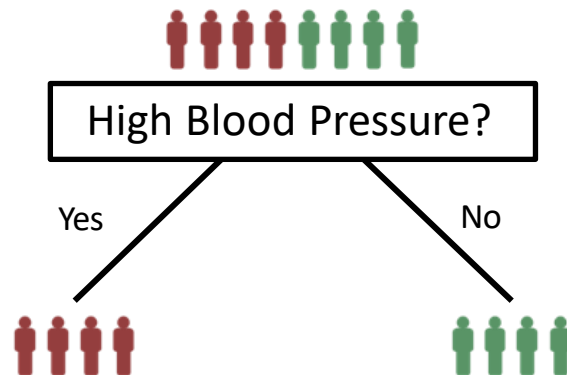
Search Space of Candidate Splits

- When choosing the “best” split (j, t) , what splits to consider?
 - Assuming x_j is a real-valued feature (we will discuss categorical features later)
- **Search space**
 - All features $j \in \{1, \dots, d\}$
 - All values $t \in \{x_{ij} \mid x_{ij} \in Z\}$
- The choices of t induce all possible splits of Z

Choosing Splits

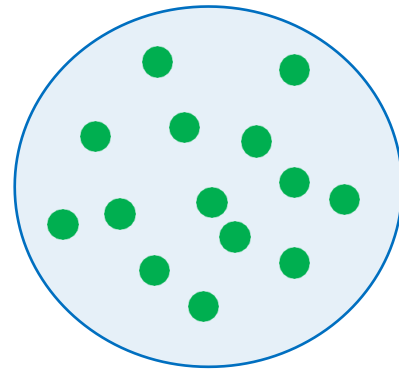
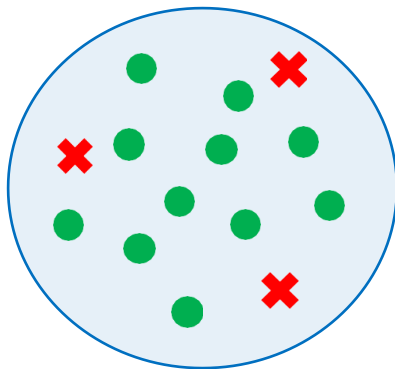
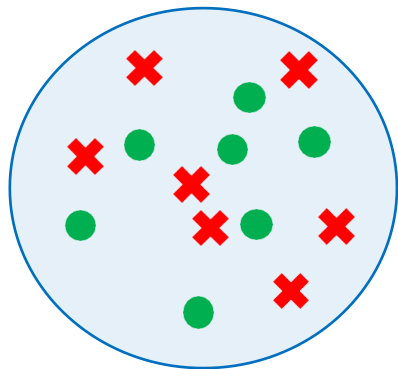
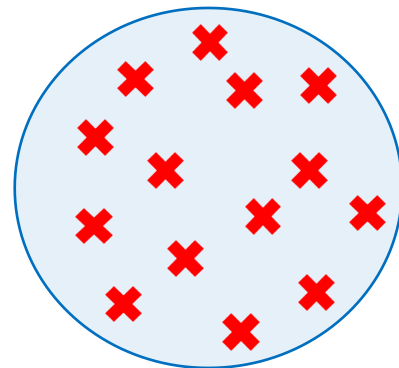
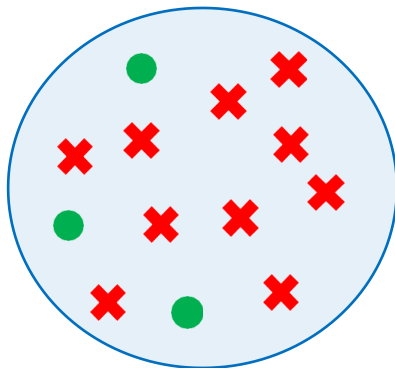
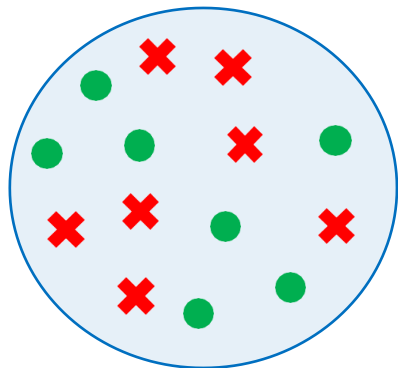


more impure



less impure

Choosing Splits



← maximally impure

→ minimally impure

Quantifying Impurity

- Given a random variable Y with domain $\{1, \dots, k\}$, its **entropy** is

$$H(Y) = - \sum_{y=1}^k P(Y = y) \log_2 P(Y = y)$$

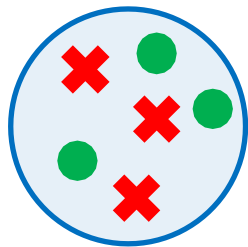
- Given a dataset Z , we define the random variable Y by

$$P(Y = y) = \frac{1}{n} \sum_{i=1}^n \mathbf{1}(y = y_i)$$

- Now, we define the entropy of Z to be $H(Z) = H(Y)$

Quantifying Impurity

$$H(Y) = - \sum_{y=1}^k P(Y = y) \log_2 P(Y = y)$$



$$H(Z) = -\frac{1}{2} \log \frac{1}{2} - \frac{1}{2} \log \frac{1}{2} = 1$$

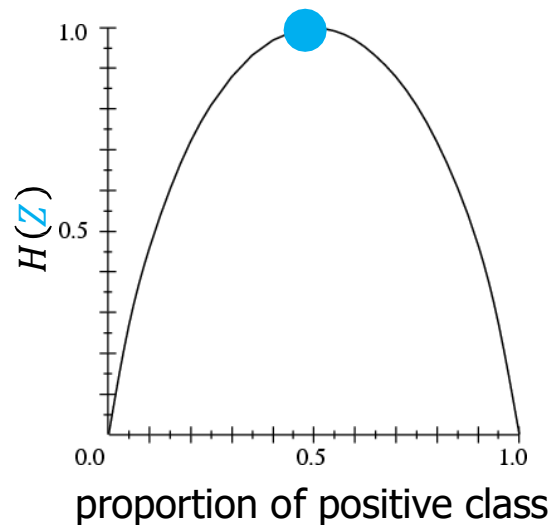
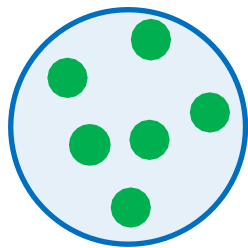


Image: Tom Mitchell

Quantifying Impurity

$$H(Y) = - \sum_{y=1}^k P(Y = y) \log_2 P(Y = y)$$



$$H(\textcolor{teal}{Z}) = -1 \log 1 = 0$$

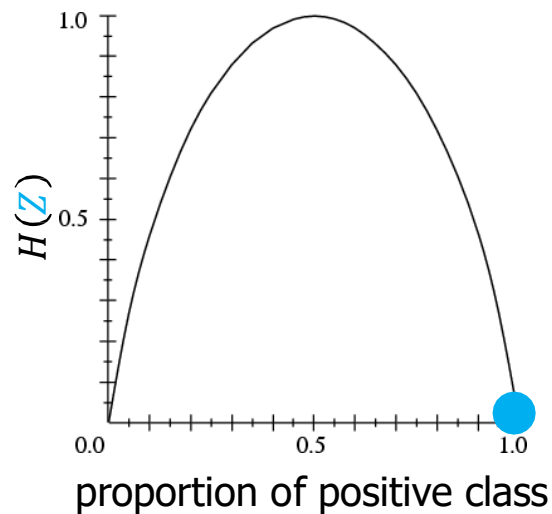


Image: Tom Mitchell

Choosing Splits

- Choose splits that most **reduce** impurity (i.e., entropy)
- Quantified by the **information gain**:

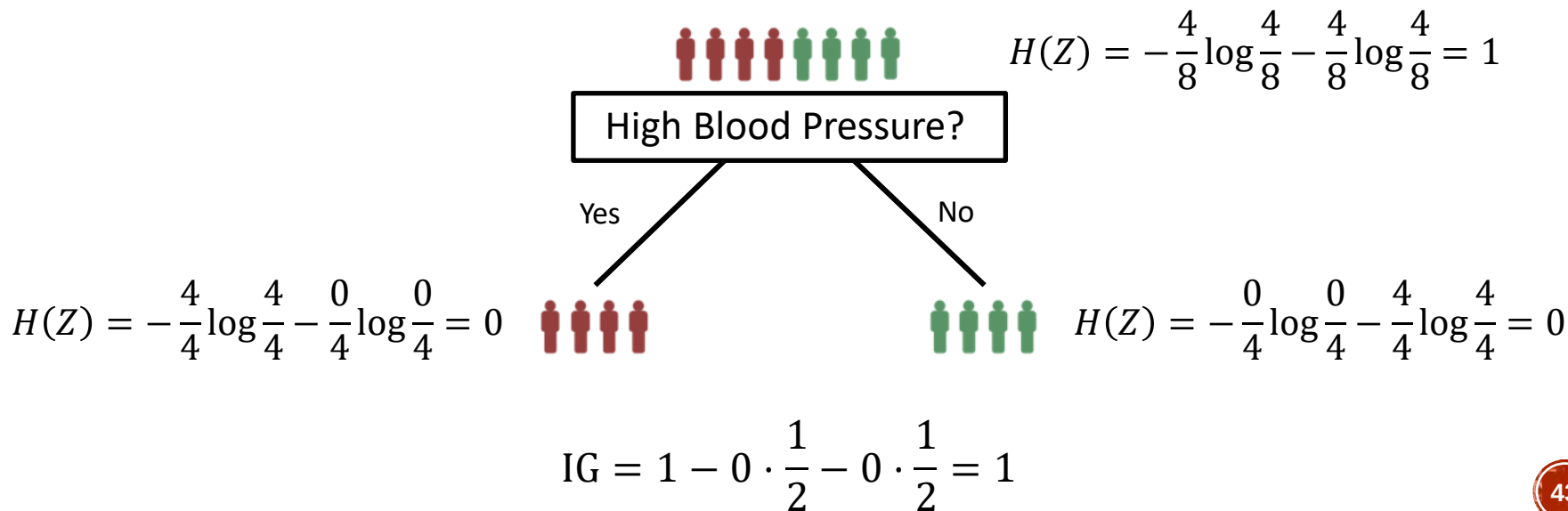
$$\text{IG}(\mathbf{Z}, j, t) = H(\mathbf{Z}) - H(\mathbf{Z}[x_j \geq t])P(x_j \geq t) - H(\mathbf{Z}[x_j < t])P(x_j < t)$$

- Here, $P(C) = \frac{1}{n} \sum_{i=1}^n 1(x_i \text{ satisfies } C)$ is the fraction of examples in $\mathbf{Z}$ that satisfy condition C
- Information gain is higher if split reduces impurity by more

Choosing Splits

$$H(Z) = - \sum_{y=1}^k P(Y = y) \log_2 P(Y = y)$$

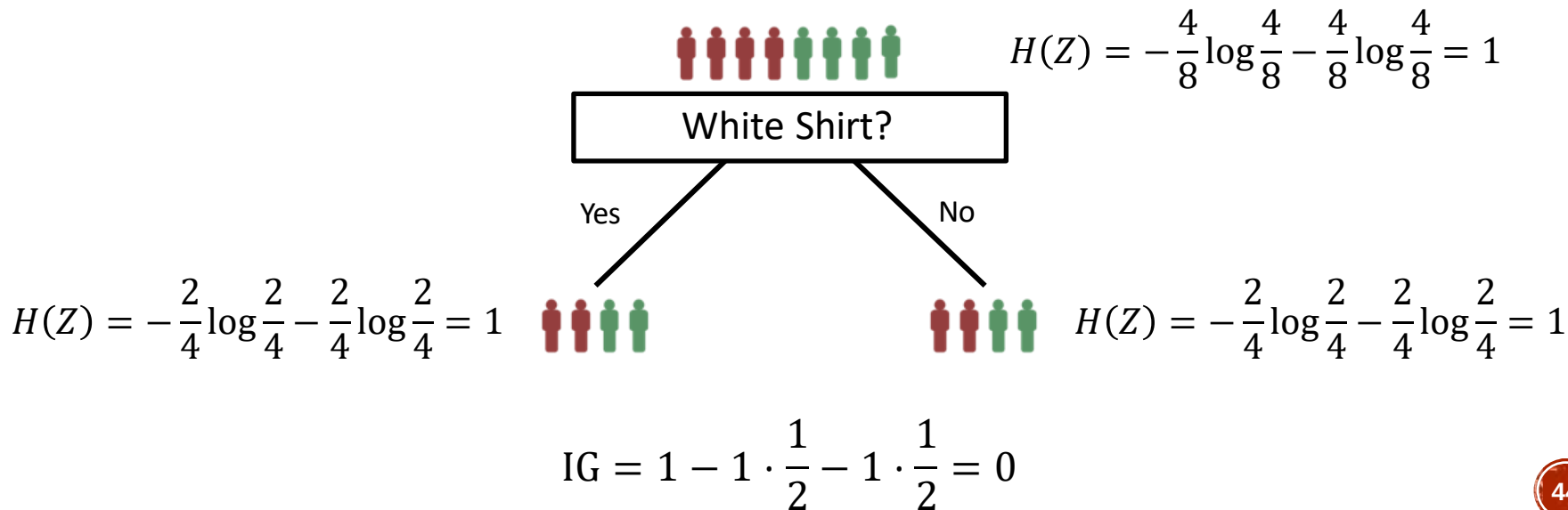
$$\text{IG}(Z, j, t) = H(Z) - H(Z[x_j \geq t])P(x_j \geq t) - H(Z[x_j < t])P(x_j < t)$$



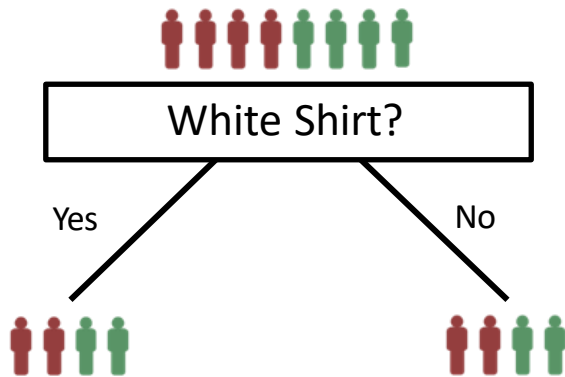
Choosing Splits

$$H(Z) = - \sum_{y=1}^k P(Y = y) \log_2 P(Y = y)$$

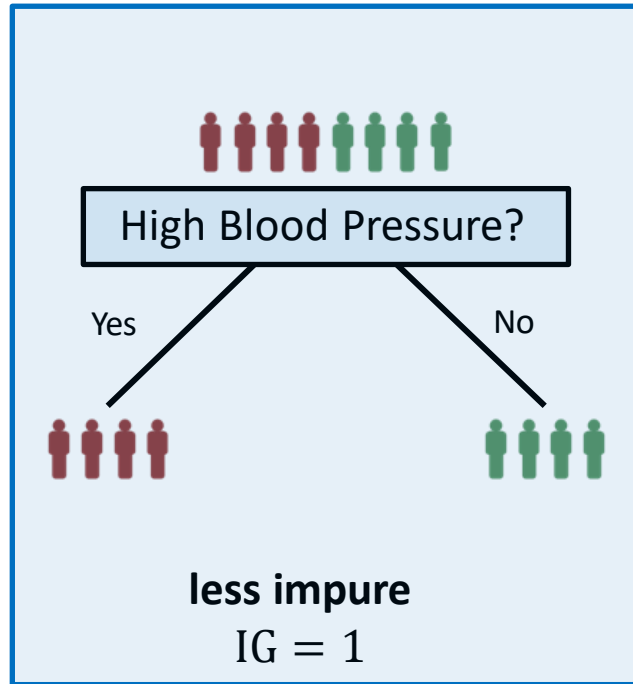
$$\text{IG}(Z, j, t) = H(Z) - H(Z[x_j \geq t])P(x_j \geq t) - H(Z[x_j < t])P(x_j < t)$$



Choosing Splits



more impure
 $IG = 0$



less impure
 $IG = 1$

best feature

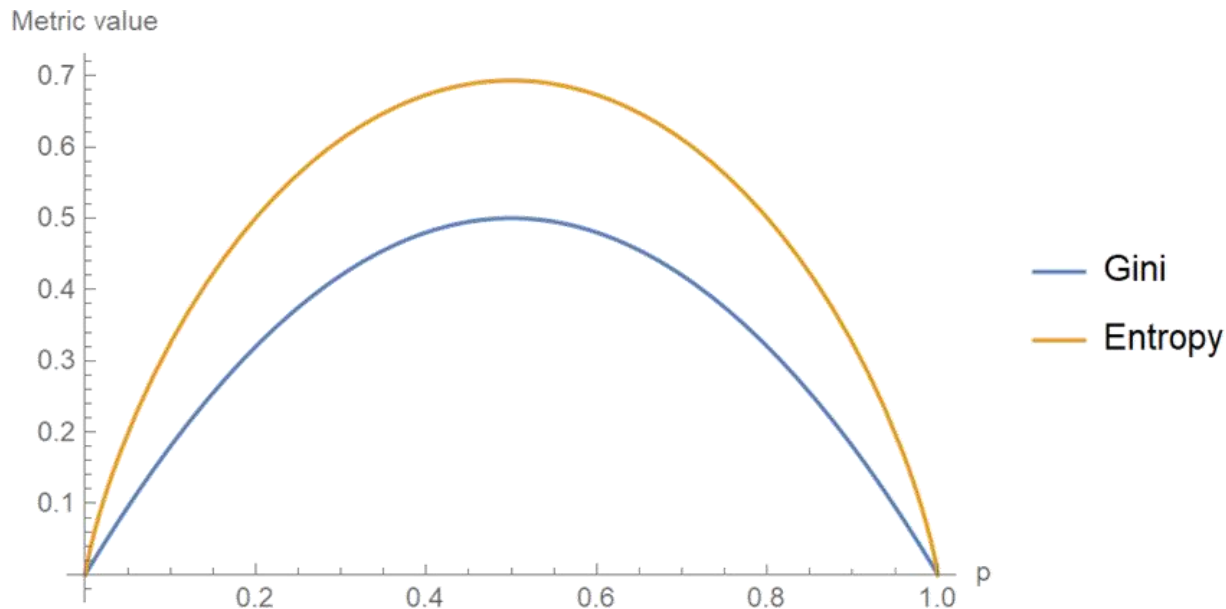
Alternative to Entropy: Gini Index

- Given a random variable Y with domain $\{1, \dots, k\}$, its **Gini index** is

$$\text{Gini}(Y) = \sum_{y=1}^k P(Y = y)(1 - P(Y = y))$$

- Define Gini (Z) as before
 - Assume we label examples randomly according to label distribution of Z
 - Then, this is the probability that a random example is incorrectly labeled
- More computationally efficient than entropy

Alternative to Entropy: Gini Index



Categorical Features

- **Structure of a split:**

- Parameters (j, t) , where j is a feature index and t is a category
- Branches are $x_j = t$ (left) and $x_j \neq t$ (right)

- Use the alternative information gain

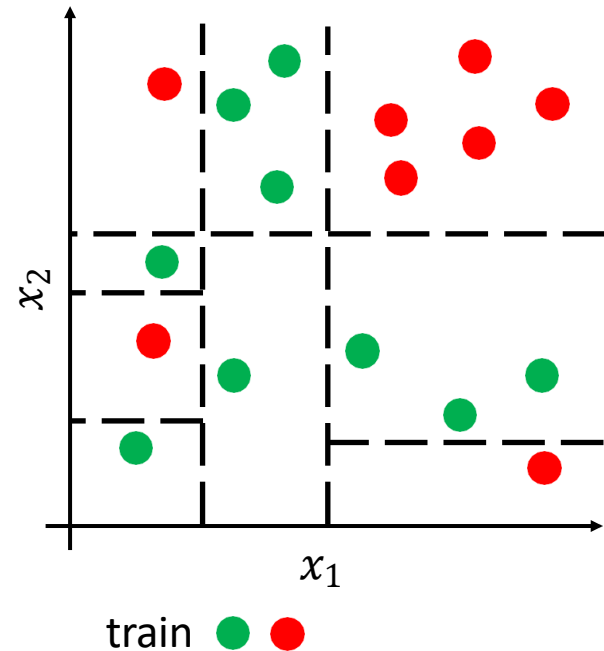
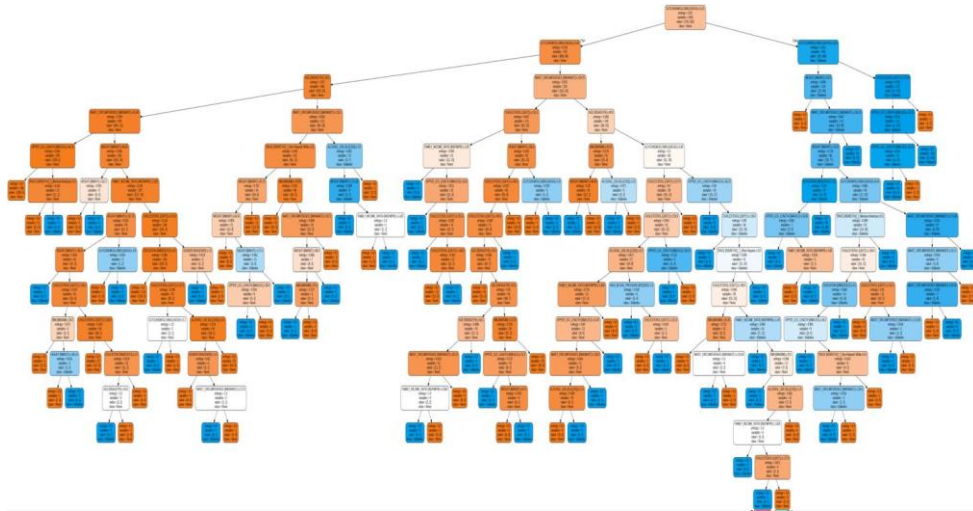
$$\text{IG}(Z, j, t) = H(Z) - H(Z[x_j = t])P(x_j = t) - H(Z[x_j \neq t])P(x_j \neq t)$$

Bias-Variance Tradeoff

- On the diabetes dataset:
 - **Training accuracy:** 100%
 - **Test accuracy:** 82.8%
- The training accuracy is **always** 100%!
 - Because we grow trees until each leaf node is pure

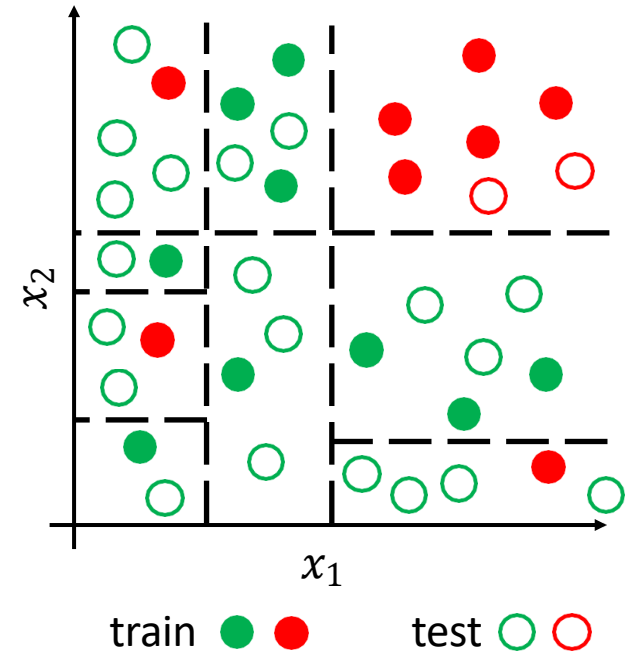
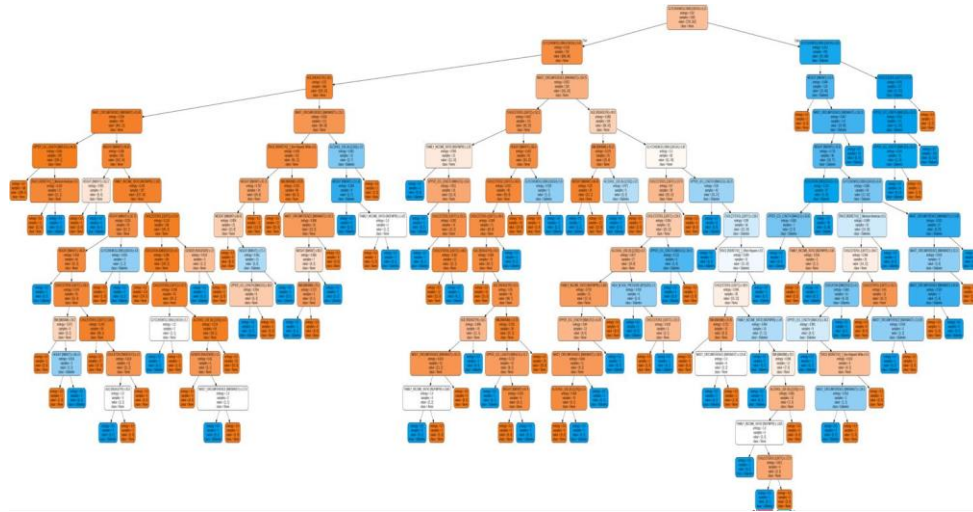
Bias-Variance Tradeoff

- Grow trees until each leaf node is pure



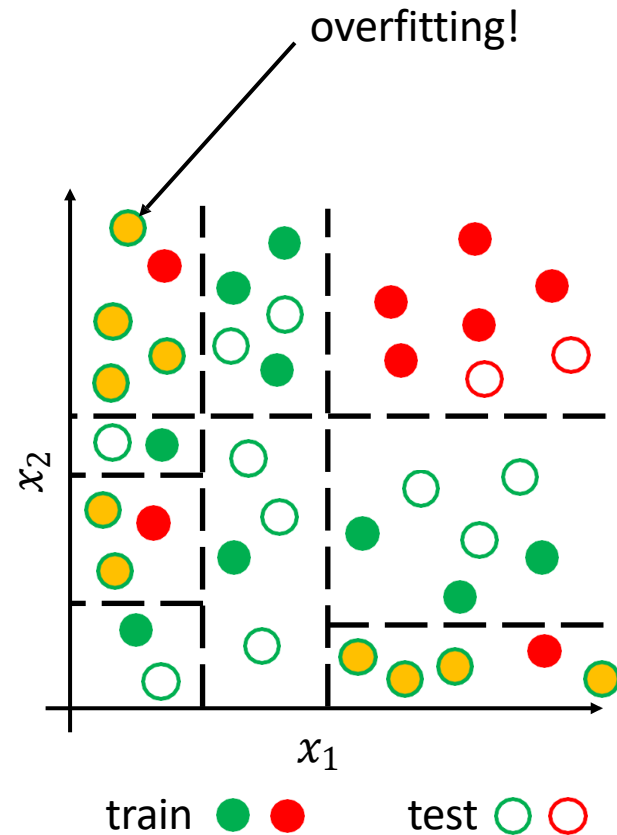
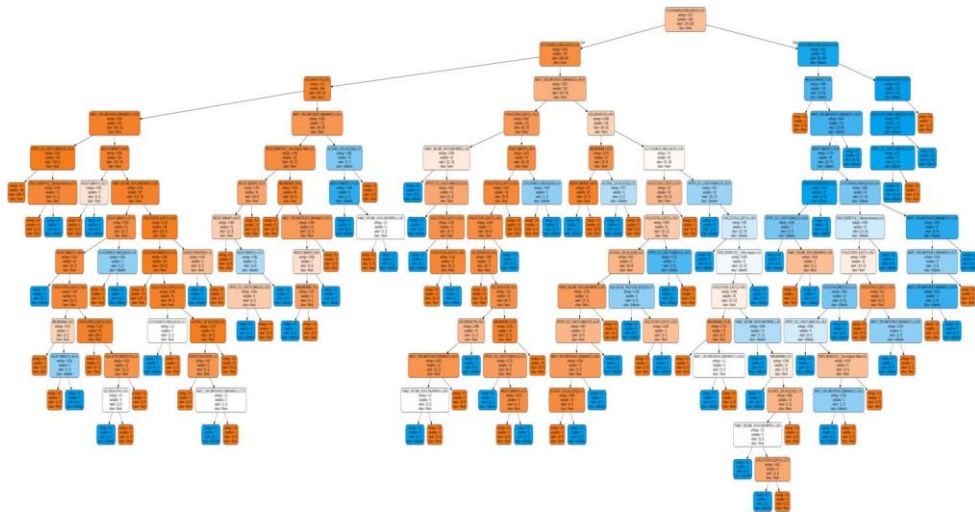
Bias-Variance Tradeoff

- Grow trees until each leaf node is pure



Bias-Variance Tradeoff

- Grow trees until each leaf node is pure

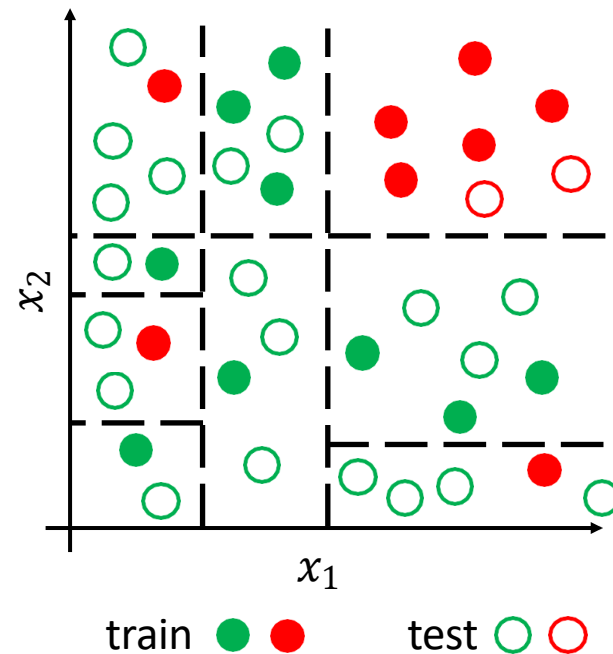
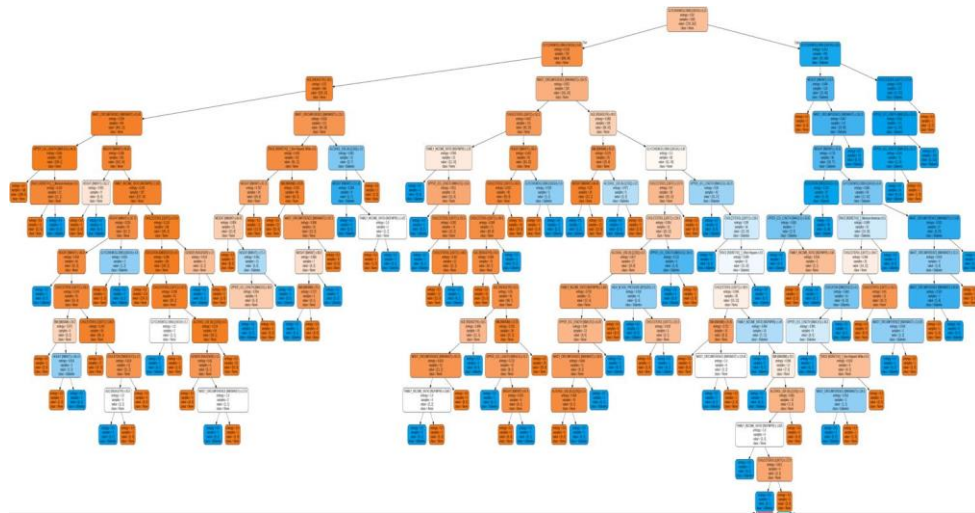


Bias-Variance Tradeoff

- **Recall:** Overfitting happens because we fit “noise” in the data
- **Avoiding overfitting**
 - Gather more data
 - Remove features
 - **Regularization:** Bias towards simpler models
- **Idea:** “Prune” decision tree to reduce its size
 - Smaller decision tree → fewer parameters → lower model capacity

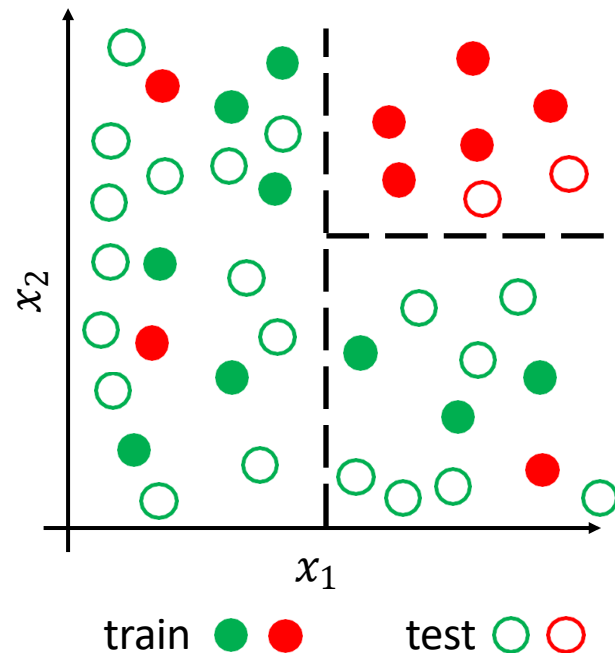
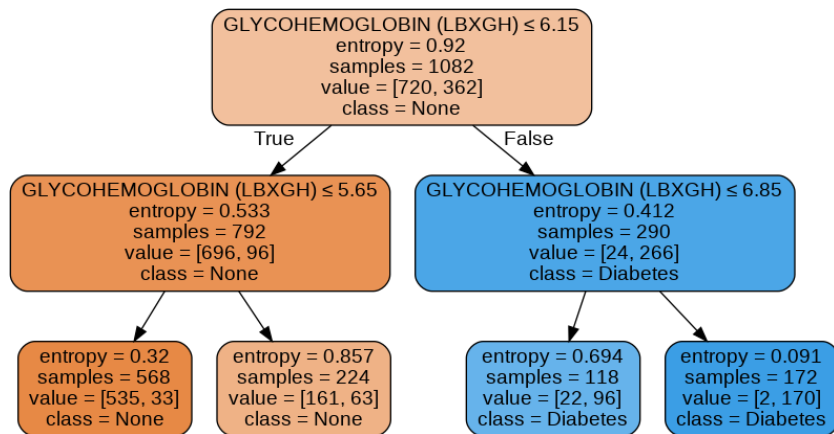
Bias-Variance Tradeoff

- Prune decision tree to simplify



Bias-Variance Tradeoff

- Prune decision tree to simplify



Decision Tree Pruning Strategies

- **Training hyperparameters**

- **Examples:** Max depth, min samples per leaf, nonzero impurity threshold
- If stopping early, use mode of labels for leaf (or mean for classification)
- Can be chosen using cross-validation, but may require significant effort

- **Post pruning**

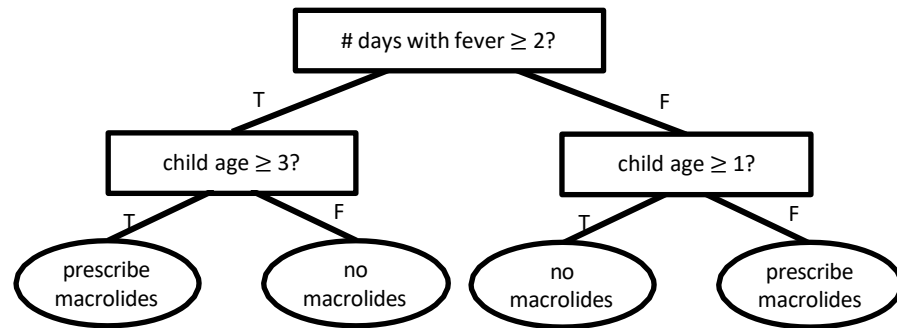
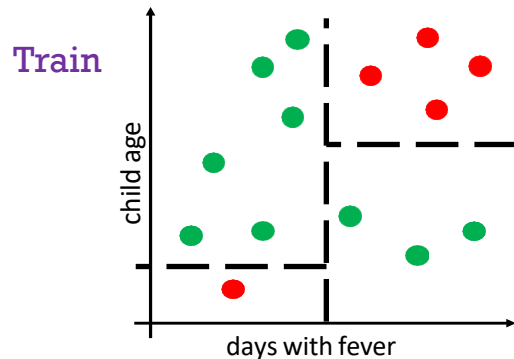
- **Step 1:** Grow full decision tree
- **Step 2:** Prune unhelpful nodes based on **validation dataset**

Decision Tree: Pruning

```
def PruneTree( $T, Z_{\text{train}}, Z_{\text{val}}$ ):  
    for each internal node  $N$  of  $T$ :  
         $T_N \leftarrow \text{Replace} \left( T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])) \right)$   
         $g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$   
     $N_0 \leftarrow \arg \max_N g_N$   
    if  $g_{N_0} > 0$ :  
        return PruneTree( $T_{N_0}, Z_{\text{train}}, Z_{\text{val}}$ )  
    else:  
        return  $T$ 
```

Decision Tree: Pruning

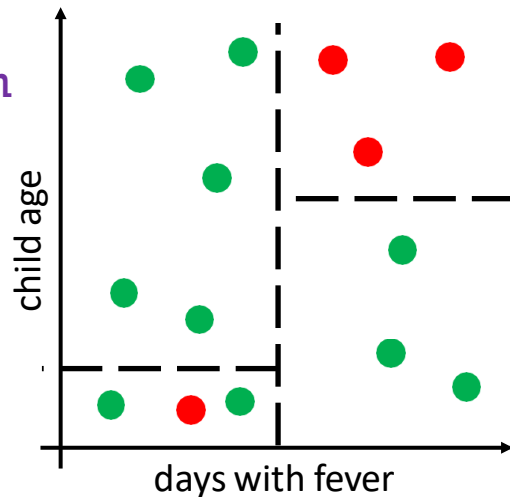
```
def PruneTree ( $T, Z_{\text{train}}, Z_{\text{val}}$ ):  
    for each internal node  $N$  of  $T$ :  
         $T_N \leftarrow \text{Replace} \left( T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])) \right)$   
         $g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$   
     $N_0 \leftarrow \arg \max_N g_N$   
    if  $g_{N_0} > 0$ :  
        return PruneTree ( $T_{N_0}, Z_{\text{train}}, Z_{\text{val}}$ )  
    else:  
        return  $T$ 
```



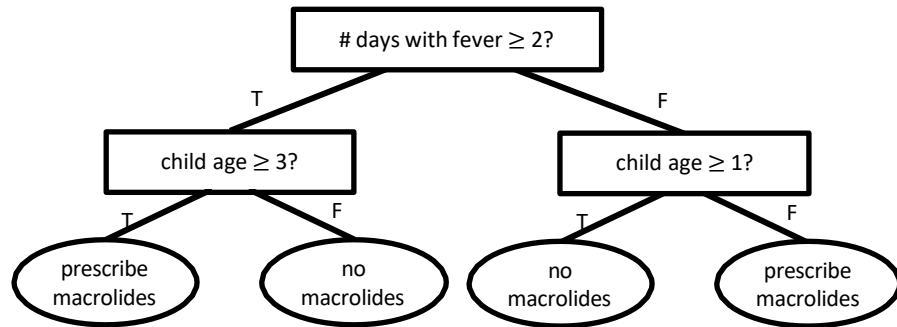
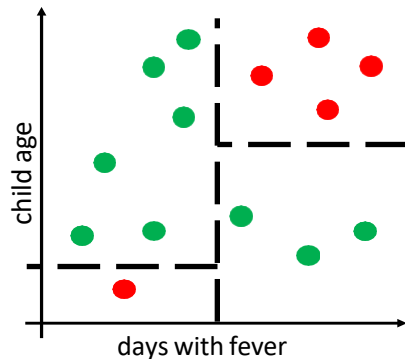
Decision Tree: Pruning

```
def PruneTree ( $T, Z_{\text{train}}, Z_{\text{val}}$ ):
    for each internal node  $N$  of  $T$ :
         $T_N \leftarrow \text{Replace} \left( T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])) \right)$ 
         $g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$ 
     $N_0 \leftarrow \arg \max_N g_N$ 
    if  $g_{N_0} > 0$ :
        return PruneTree ( $T_{N_0}, Z_{\text{train}}, Z_{\text{val}}$ )
    else:
        return  $T$ 
```

Validation



Train



Decision Tree: Pruning

```
def PruneTree ( $T, Z_{\text{train}}, Z_{\text{val}}$ ):
```

for each internal node N of T :

$T_N \leftarrow \text{Replace} \left(T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])) \right)$

$g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$

$N_0 \leftarrow \arg \max_N g_N$

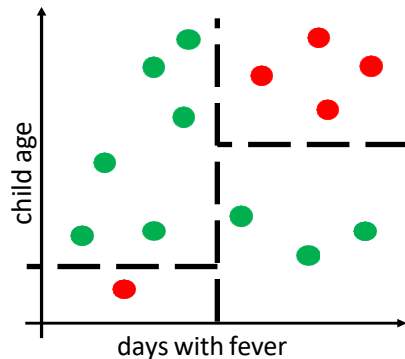
if $g_{N_0} > 0$:

return $\text{PruneTree}(T_{N_0}, Z_{\text{train}}, Z_{\text{val}})$

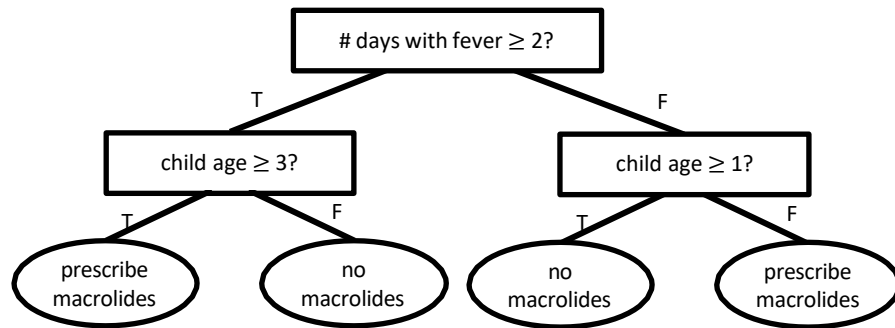
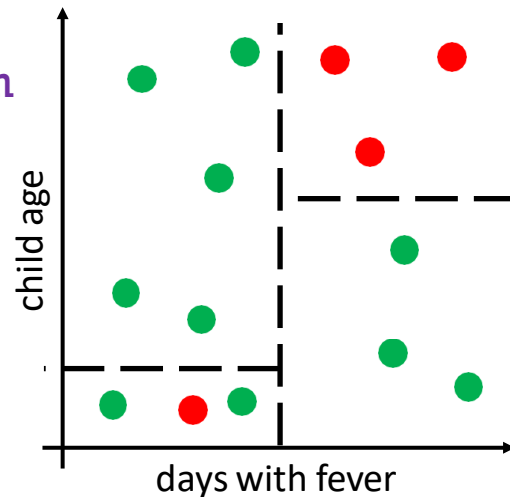
else:

return T

Train



Validation



Decision Tree: Pruning

def PruneTree ($T, Z_{\text{train}}, Z_{\text{val}}$):

for each internal node N of T :

$T_N \leftarrow \text{Replace} \left(T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])) \right)$

$g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$

$N_0 \leftarrow \arg \max_N g_N$

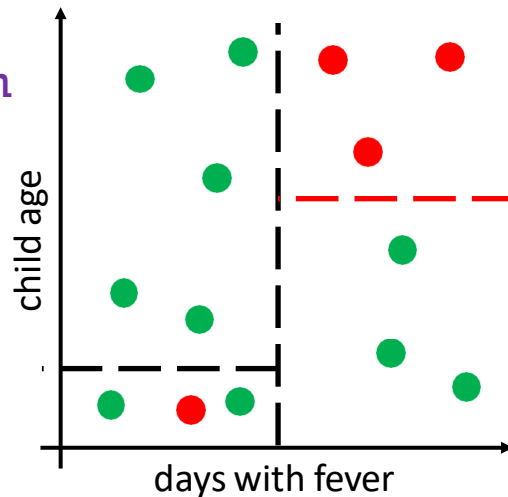
if $g_{N_0} > 0$:

return PruneTree ($T_{N_0}, Z_{\text{train}}, Z_{\text{val}}$)

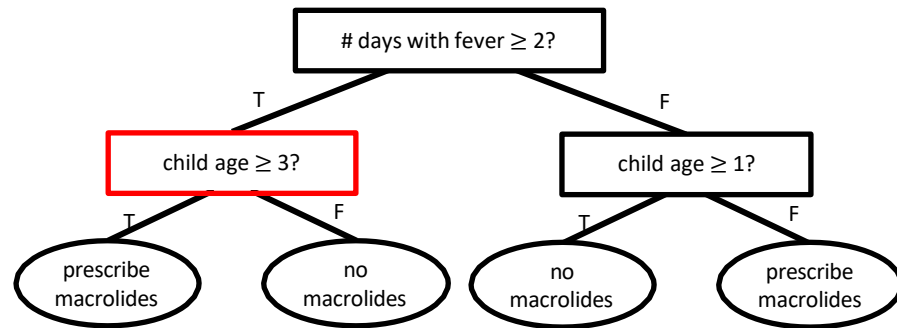
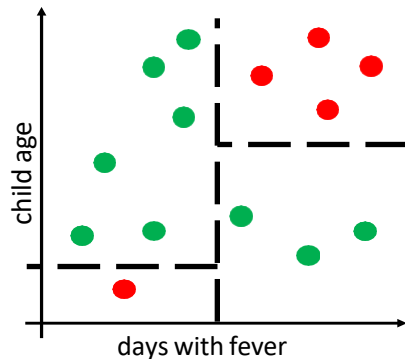
else:

return T

Validation



Train



Decision Tree: Pruning

def PruneTree ($T, Z_{\text{train}}, Z_{\text{val}}$):

for each internal node N of T :

$T_N \leftarrow \text{Replace} \left(T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])) \right)$

$g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$

$N_0 \leftarrow \arg \max_N g_N$

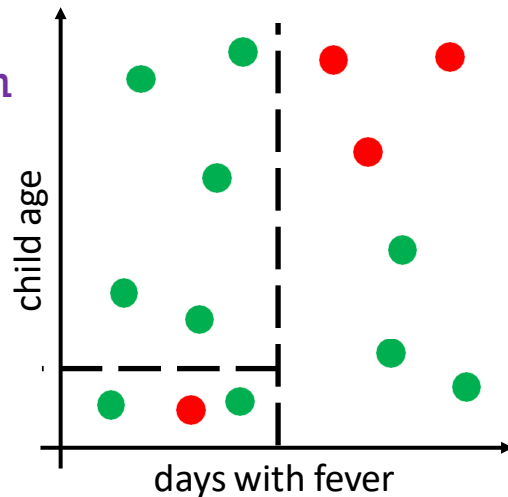
if $g_{N_0} > 0$:

return PruneTree ($T_{N_0}, Z_{\text{train}}, Z_{\text{val}}$)

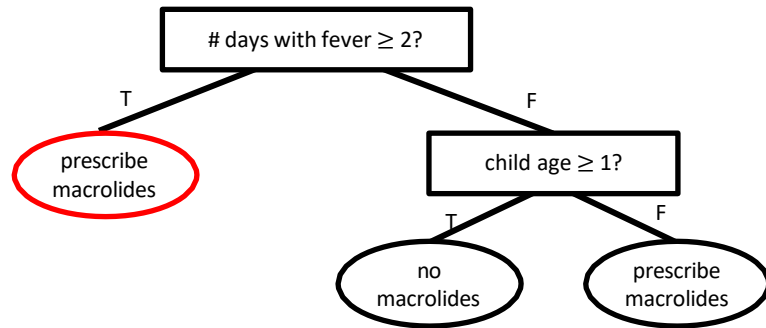
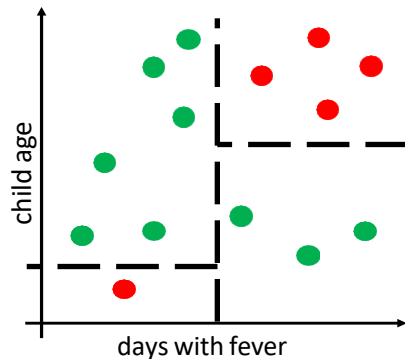
else:

return T

Validation



Train

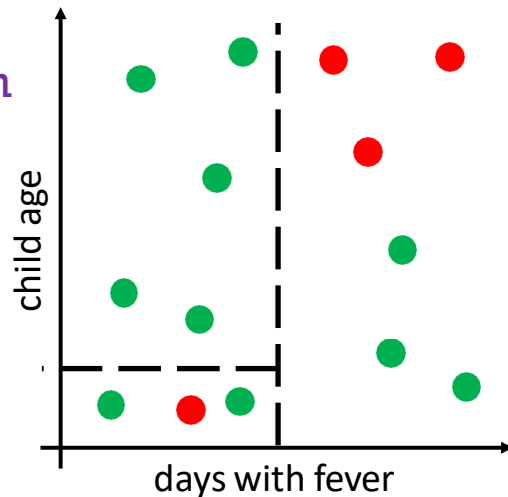


Decision Tree: Pruning

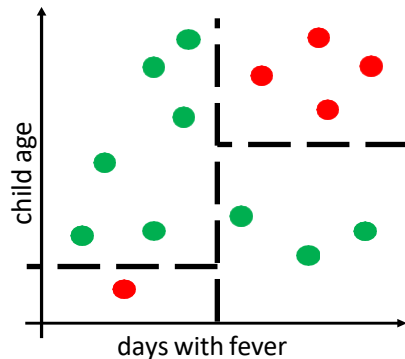
```

def PruneTree ( $T, Z_{\text{train}}, Z_{\text{val}}$ ):
    for each internal node  $N$  of  $T$ :
         $T_N \leftarrow \text{Replace} (T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])))$ 
         $g_N \leftarrow \text{Loss} (T, Z_{\text{val}}) - \text{Loss} (T_N, Z_{\text{val}})$ 
     $N_0 \leftarrow \arg \max_N g_N$ 
    if  $g_{N_0} > 0$ :
        return PruneTree ( $T_{N_0}, Z_{\text{train}}, Z_{\text{val}}$ )
    else:
        return  $T$ 
    
```

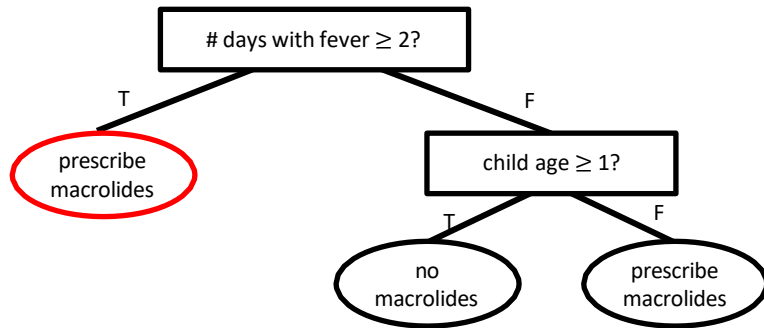
Validation



Train



$$g = \frac{2}{14} - \frac{5}{14}$$

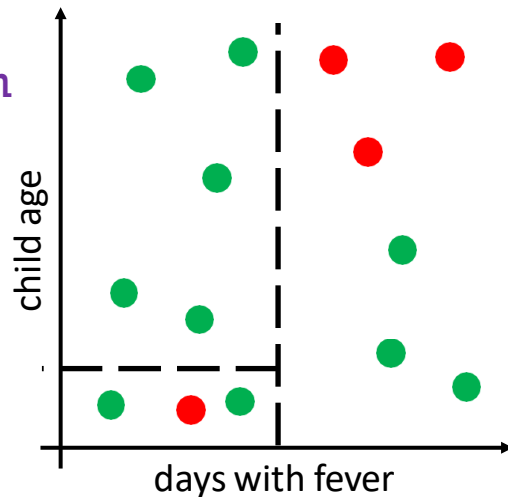


Decision Tree: Pruning

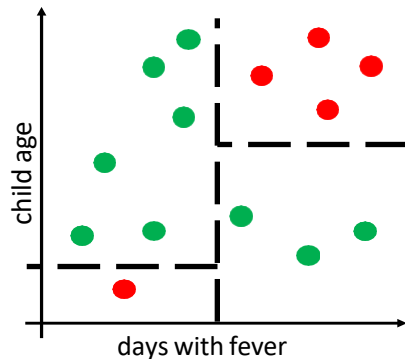
```

def PruneTree ( $T, Z_{\text{train}}, Z_{\text{val}}$ ):
    for each internal node  $N$  of  $T$ :
         $T_N \leftarrow \text{Replace} (T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])))$ 
         $g_N \leftarrow \text{Loss} (T, Z_{\text{val}}) - \text{Loss} (T_N, Z_{\text{val}})$ 
     $N_0 \leftarrow \arg \max_N g_N$ 
    if  $g_{N_0} > 0$ :
        return PruneTree ( $T_{N_0}, Z_{\text{train}}, Z_{\text{val}}$ )
    else:
        return  $T$ 
    
```

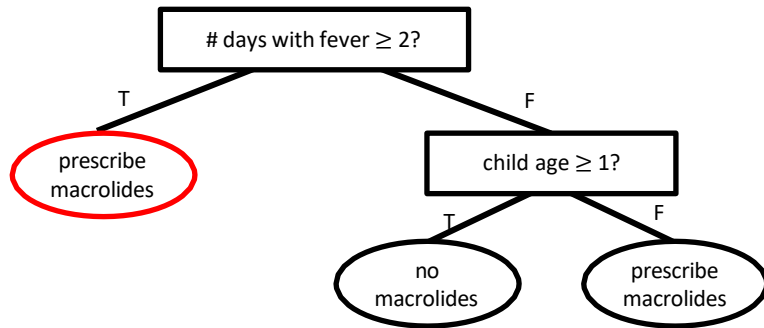
Validation



Train



$$g = -\frac{3}{14}$$



Decision Tree: Pruning

def PruneTree ($T, Z_{\text{train}}, Z_{\text{val}}$):

for each internal node N of T :

$T_N \leftarrow \text{Replace} \left(T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])) \right)$

$g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$

$N_0 \leftarrow \arg \max_N g_N$

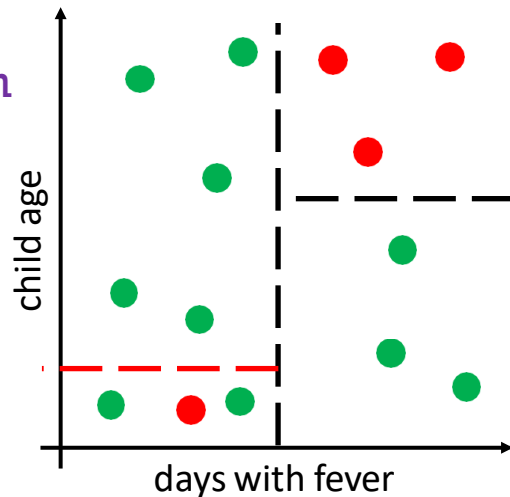
if $g_{N_0} > 0$:

return PruneTree ($T_{N_0}, Z_{\text{train}}, Z_{\text{val}}$)

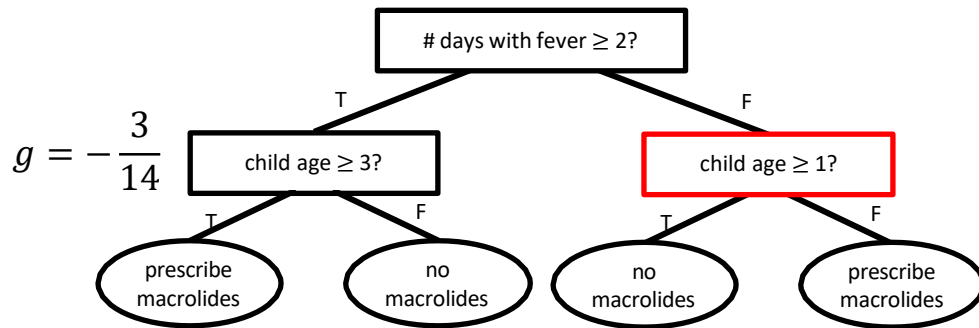
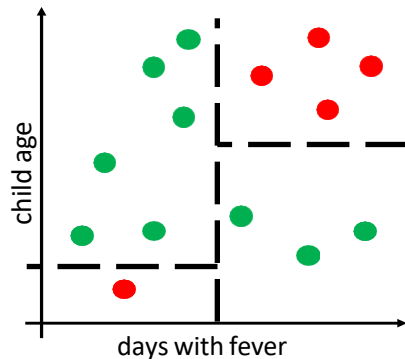
else:

return T

Validation



Train



Decision Tree: Pruning

def PruneTree ($T, Z_{\text{train}}, Z_{\text{val}}$):

for each internal node N of T :

$T_N \leftarrow \text{Replace} \left(T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])) \right)$

$g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$

$N_0 \leftarrow \arg \max_N g_N$

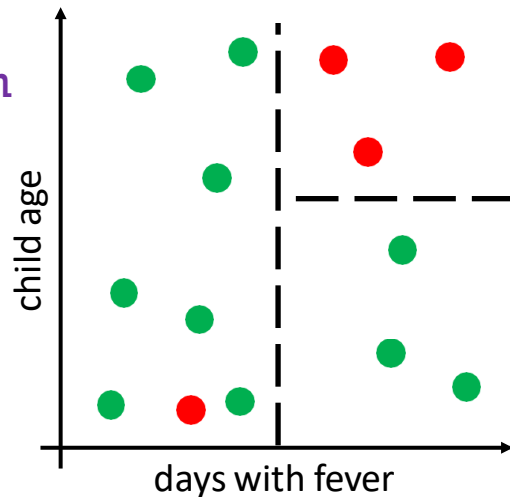
if $g_{N_0} > 0$:

return PruneTree ($T_{N_0}, Z_{\text{train}}, Z_{\text{val}}$)

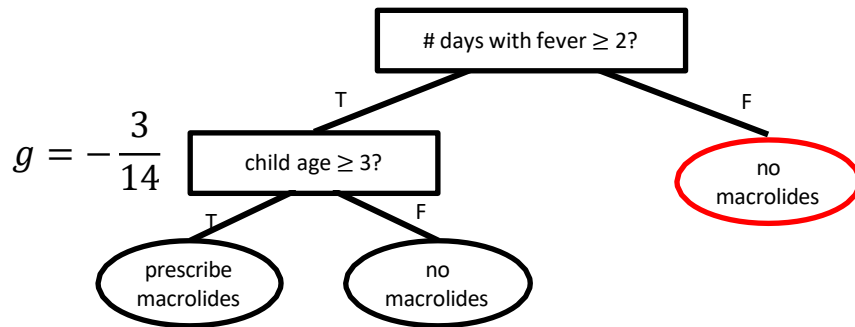
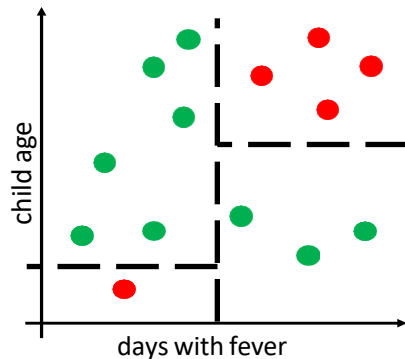
else:

return T

Validation



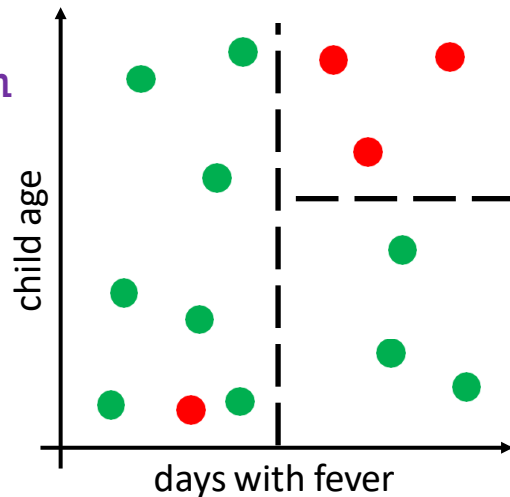
Train



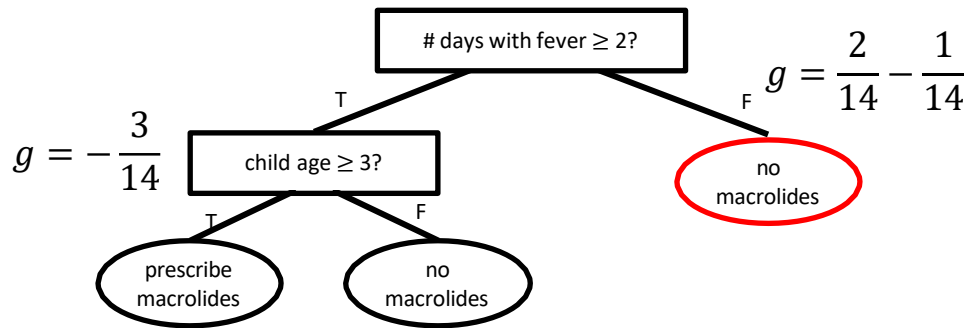
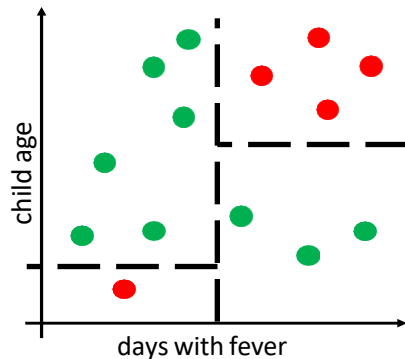
Decision Tree: Pruning

```
def PruneTree ( $T, Z_{\text{train}}, Z_{\text{val}}$ ):
    for each internal node  $N$  of  $T$ :
         $T_N \leftarrow \text{Replace} (T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])))$ 
         $g_N \leftarrow \text{Loss} (T, Z_{\text{val}}) - \text{Loss} (T_N, Z_{\text{val}})$ 
     $N_0 \leftarrow \arg \max_N g_N$ 
    if  $g_{N_0} > 0$ :
        return PruneTree ( $T_{N_0}, Z_{\text{train}}, Z_{\text{val}}$ )
    else:
        return  $T$ 
```

Validation



Train

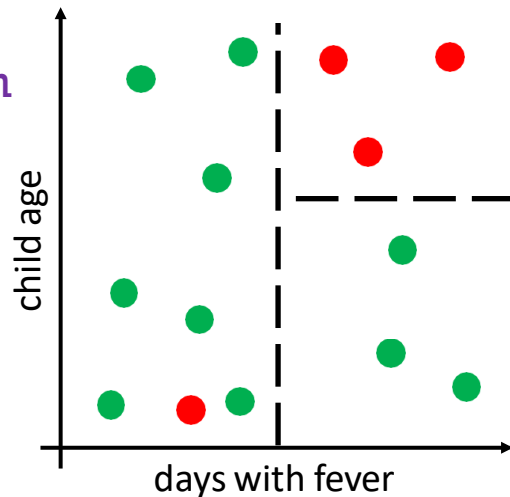


Decision Tree: Pruning

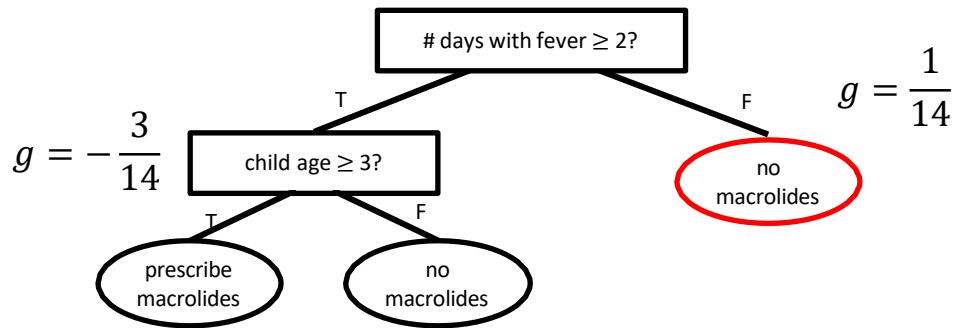
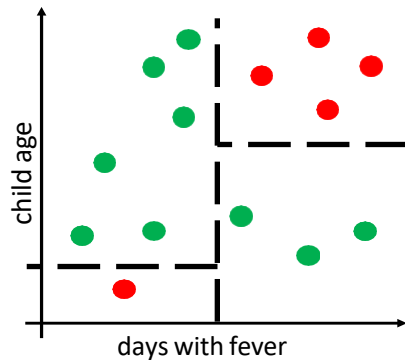
```

def PruneTree ( $T, Z_{\text{train}}, Z_{\text{val}}$ ):
    for each internal node  $N$  of  $T$ :
         $T_N \leftarrow \text{Replace} \left( T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])) \right)$ 
         $g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$ 
     $N_0 \leftarrow \arg \max_N g_N$ 
    if  $g_{N_0} > 0$ :
        return PruneTree ( $T_{N_0}, Z_{\text{train}}, Z_{\text{val}}$ )
    else:
        return  $T$ 
    
```

Validation



Train



Decision Tree: Pruning

def PruneTree ($T, Z_{\text{train}}, Z_{\text{val}}$):

for each internal node N of T :

$T_N \leftarrow \text{Replace} \left(T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])) \right)$

$g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$

$N_0 \leftarrow \arg \max_N g_N$

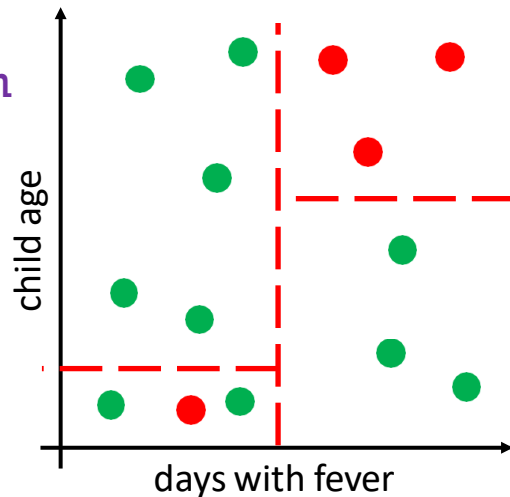
if $g_{N_0} > 0$:

return PruneTree ($T_{N_0}, Z_{\text{train}}, Z_{\text{val}}$)

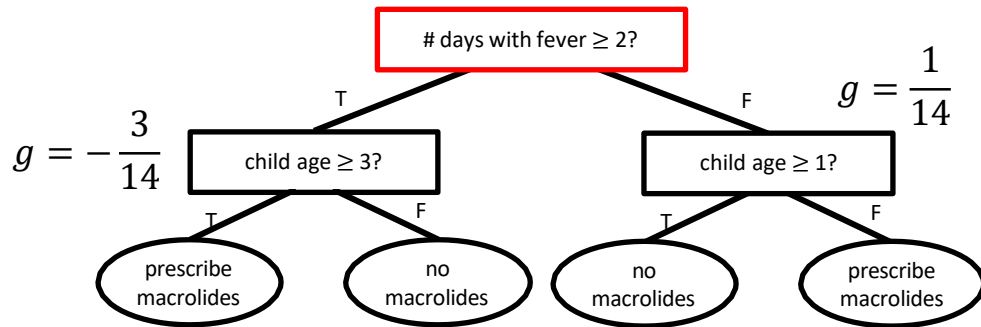
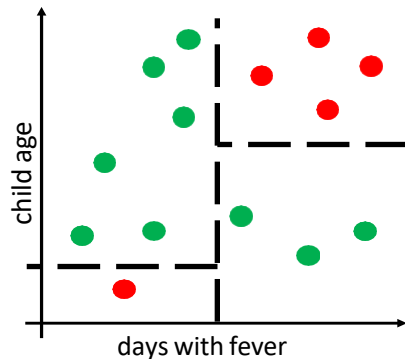
else:

return T

Validation



Train



Decision Tree: Pruning

def PruneTree ($T, Z_{\text{train}}, Z_{\text{val}}$):

for each internal node N of T :

$T_N \leftarrow \text{Replace} \left(T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])) \right)$

$g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$

$N_0 \leftarrow \arg \max_N g_N$

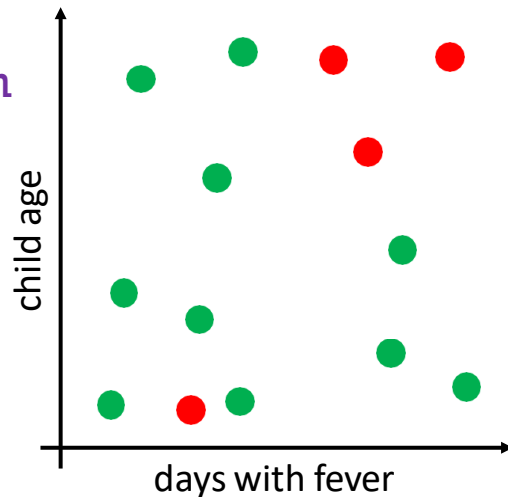
if $g_{N_0} > 0$:

return PruneTree ($T_{N_0}, Z_{\text{train}}, Z_{\text{val}}$)

else:

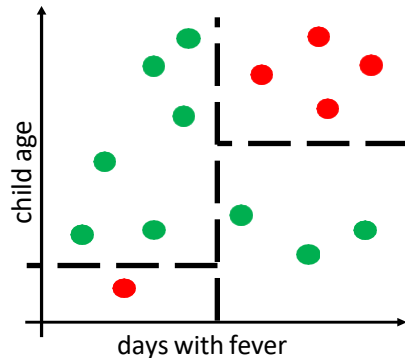
return T

Validation



no
macrolides

Train

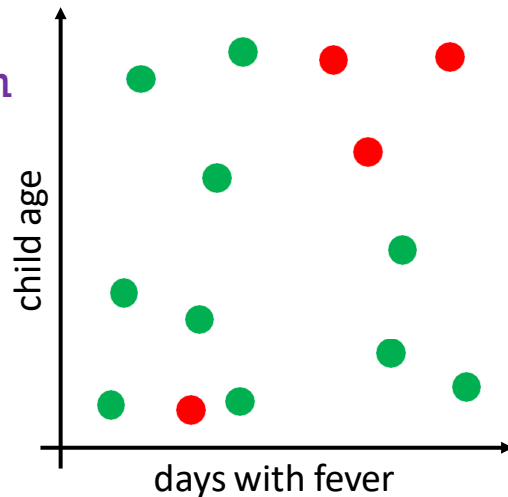


Decision Tree: Pruning

```

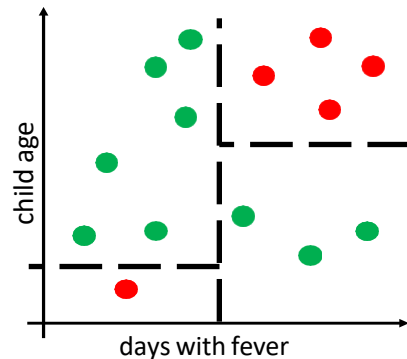
def PruneTree ( $T, Z_{\text{train}}, Z_{\text{val}}$ ):
    for each internal node  $N$  of  $T$ :
         $T_N \leftarrow \text{Replace} (T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])))$ 
         $g_N \leftarrow \text{Loss} (T, Z_{\text{val}}) - \text{Loss} (T_N, Z_{\text{val}})$ 
     $N_0 \leftarrow \arg \max_N g_N$ 
    if  $g_{N_0} > 0$ :
        return PruneTree ( $T_{N_0}, Z_{\text{train}}, Z_{\text{val}}$ )
    else:
        return  $T$ 
    
```

Validation



no macrolides $g = \frac{2}{14} - \frac{4}{14}$

Train

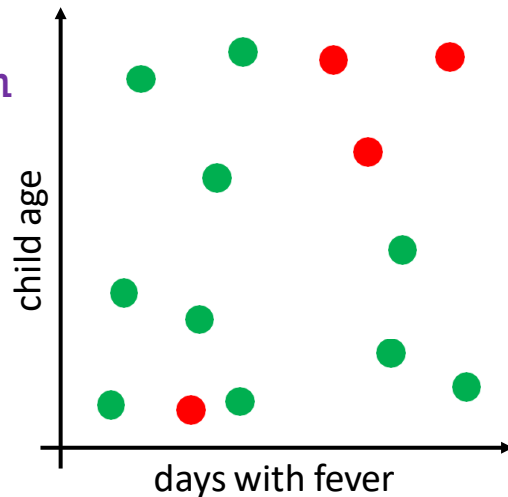


Decision Tree: Pruning

```

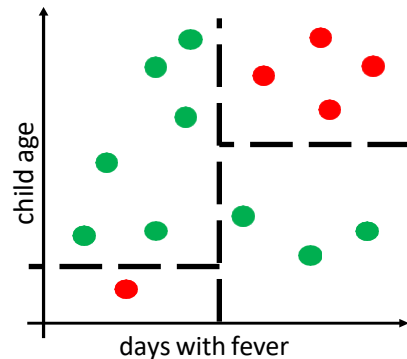
def PruneTree ( $T, Z_{\text{train}}, Z_{\text{val}}$ ):
    for each internal node  $N$  of  $T$ :
         $T_N \leftarrow \text{Replace} (T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])))$ 
         $g_N \leftarrow \text{Loss} (T, Z_{\text{val}}) - \text{Loss} (T_N, Z_{\text{val}})$ 
     $N_0 \leftarrow \arg \max_N g_N$ 
    if  $g_{N_0} > 0$ :
        return PruneTree ( $T_{N_0}, Z_{\text{train}}, Z_{\text{val}}$ )
    else:
        return  $T$ 
    
```

Validation



no macrolides $g = -\frac{2}{14}$

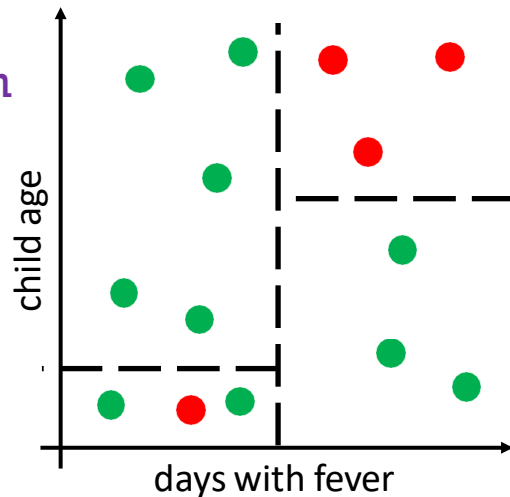
Train



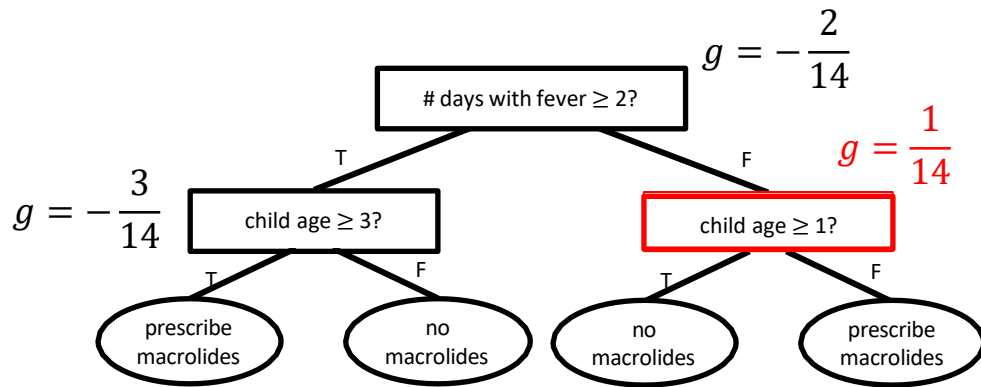
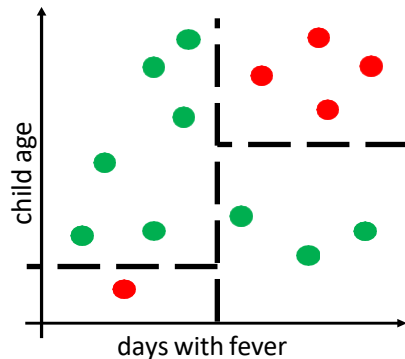
Decision Tree: Pruning

```
def PruneTree ( $T, Z_{\text{train}}, Z_{\text{val}}$ ):
    for each internal node  $N$  of  $T$ :
         $T_N \leftarrow \text{Replace} \left( T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])) \right)$ 
         $g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$ 
         $N_0 \leftarrow \arg \max_N g_N$ 
        if  $g_{N_0} > 0$ :
            return PruneTree ( $T_N, Z_{\text{train}}, Z_{\text{val}}$ )
        else:
            return  $T$ 
```

Validation



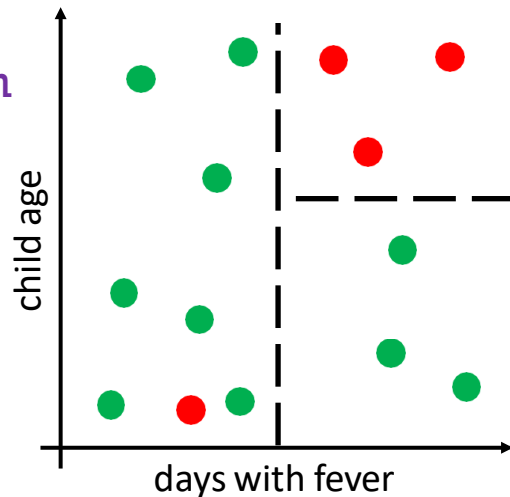
Train



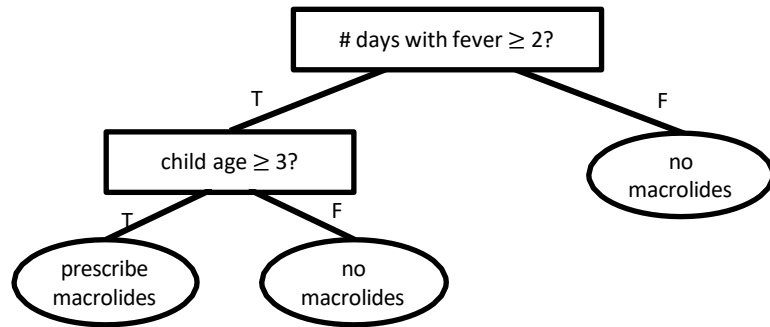
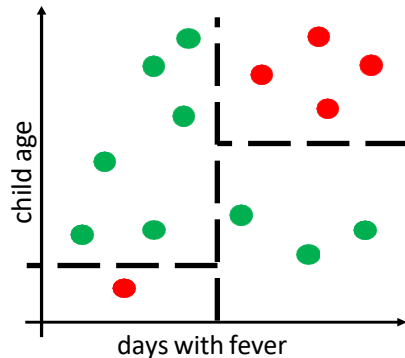
Decision Tree: Pruning

```
def PruneTree ( $T, Z_{\text{train}}, Z_{\text{val}}$ ):
    for each internal node  $N$  of  $T$ :
         $T_N \leftarrow \text{Replace} \left( T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])) \right)$ 
         $g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$ 
         $N_0 \leftarrow \arg \max_N g_N$ 
        if  $g_{N_0} > 0$ :
            return PruneTree ( $T_{N_0}, Z_{\text{train}}, Z_{\text{val}}$ )
        else:
            return  $T$ 
```

Validation



Train



Decision Tree: Pruning

```
def PruneTree ( $T, Z_{\text{train}}, Z_{\text{val}}$ ):
```

for each internal node N of T :

$T_N \leftarrow \text{Replace} \left(T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])) \right)$

$g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$

$N_0 \leftarrow \arg \max_N g_N$

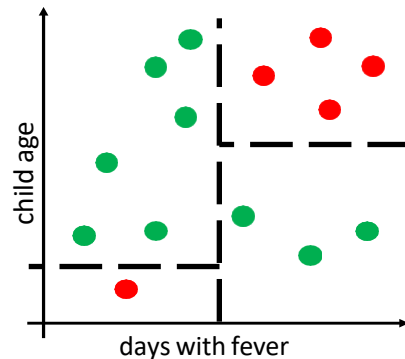
if $g_{N_0} > 0$:

return $\text{PruneTree}(T_{N_0}, Z_{\text{train}}, Z_{\text{val}})$

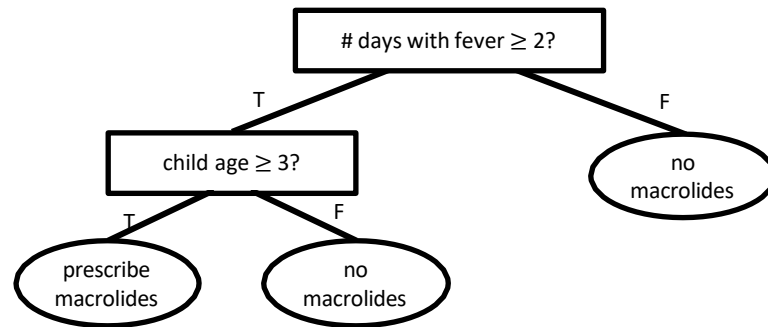
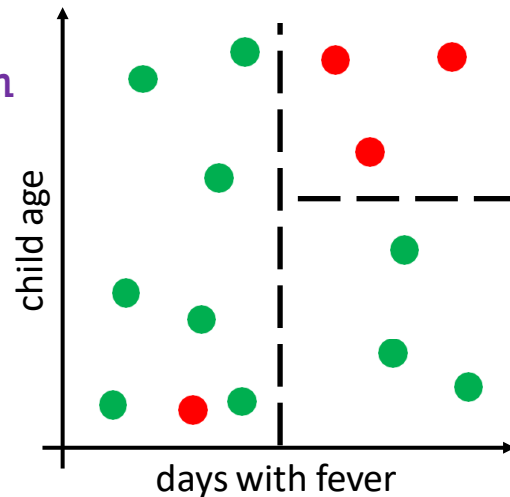
else:

return T

Train



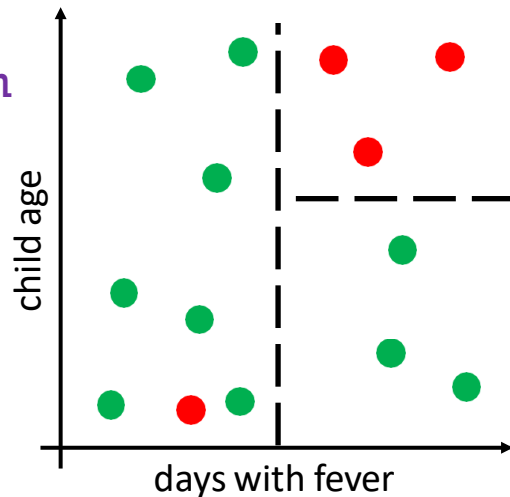
Validation



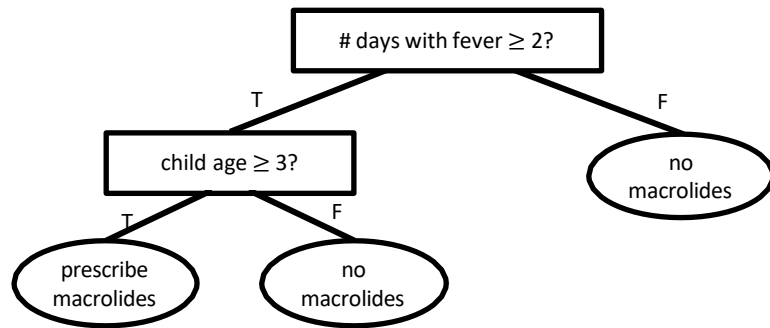
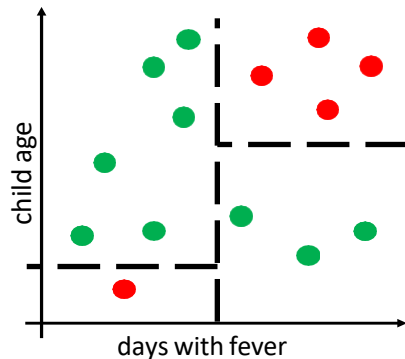
Decision Tree: Pruning

```
def PruneTree ( $T, Z_{\text{train}}, Z_{\text{val}}$ ):
    for each internal node  $N$  of  $T$ :
         $T_N \leftarrow \text{Replace} \left( T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])) \right)$ 
         $g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$ 
     $N_0 \leftarrow \arg \max_N g_N$ 
    if  $g_{N_0} > 0$ :
        return PruneTree ( $T_{N_0}, Z_{\text{train}}, Z_{\text{val}}$ )
    else:
        return  $T$ 
```

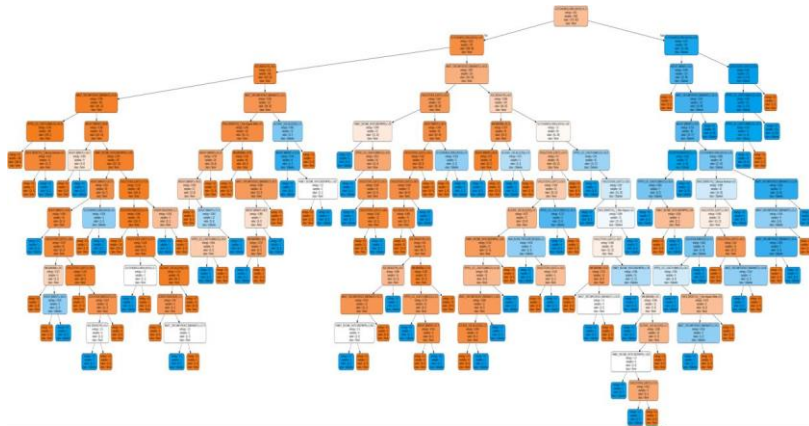
Validation



Train

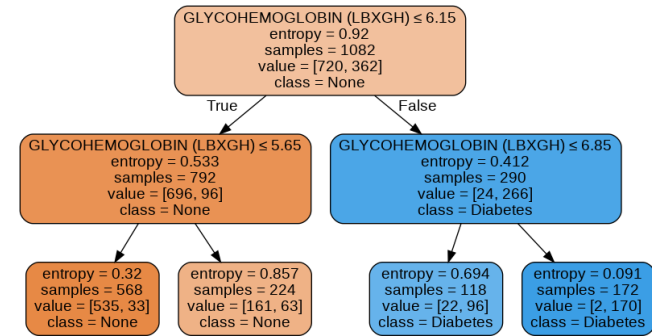


Diabetes Prediction



Training accuracy: 100%

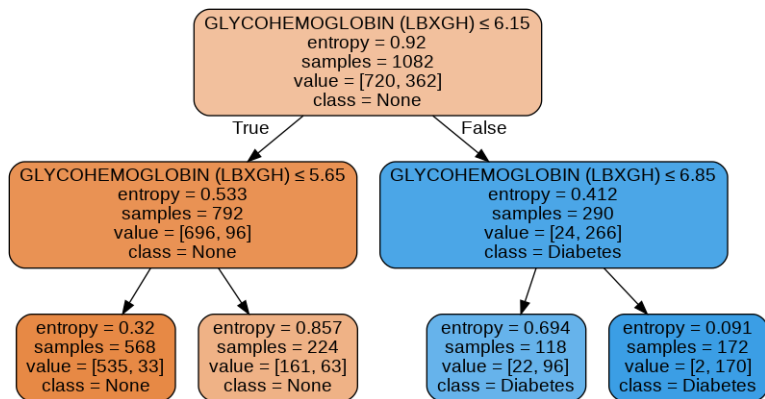
Test accuracy: 82.8%



Training accuracy: 88.9%

Test accuracy: 85.6%

Diabetes Prediction



- If your A1C level is between 5.7 and less than 6.5%, your levels have been in the prediabetes range.
- If you have an A1C level of 6.5% or higher, your levels were in the diabetes range.

(from diabetes.org)

Decision tree is similar to actual diagnosis strategy

Decision Trees

- **Strengths**

- Interpretability
- Flexible (useful for both regression and classification)
- Fast and scalable
- Learns nonlinear decision boundaries

- **Weaknesses**

- Suboptimal due to heuristic strategy
- Requires enormous amounts of data to learn large, accurate models

Decision Trees

- **Solution 1:** Better decision tree learning algorithms
 - Combinatorial search algorithms to minimize loss
 - Lots of recent work
 - Maintains (or improves) interpretability
 - Can reduce scalability
- **Solution 2:** Ensembles → Random Forests
 - Learn many decision trees to reduce variance
 - Maintains ability to learn complex nonlinear decision boundaries
 - Uninterpretable

Acknowledgement

Most of the slides are based on the ML course of:

- Mingmin Zhao and Jiatao Gu, University of Pennsylvania

THANK YOU