

Upcoming Deadlines

- HW 2 due Monday (3/3) at 8pm
- Midterm 1 will be on Wednesday (3/5)

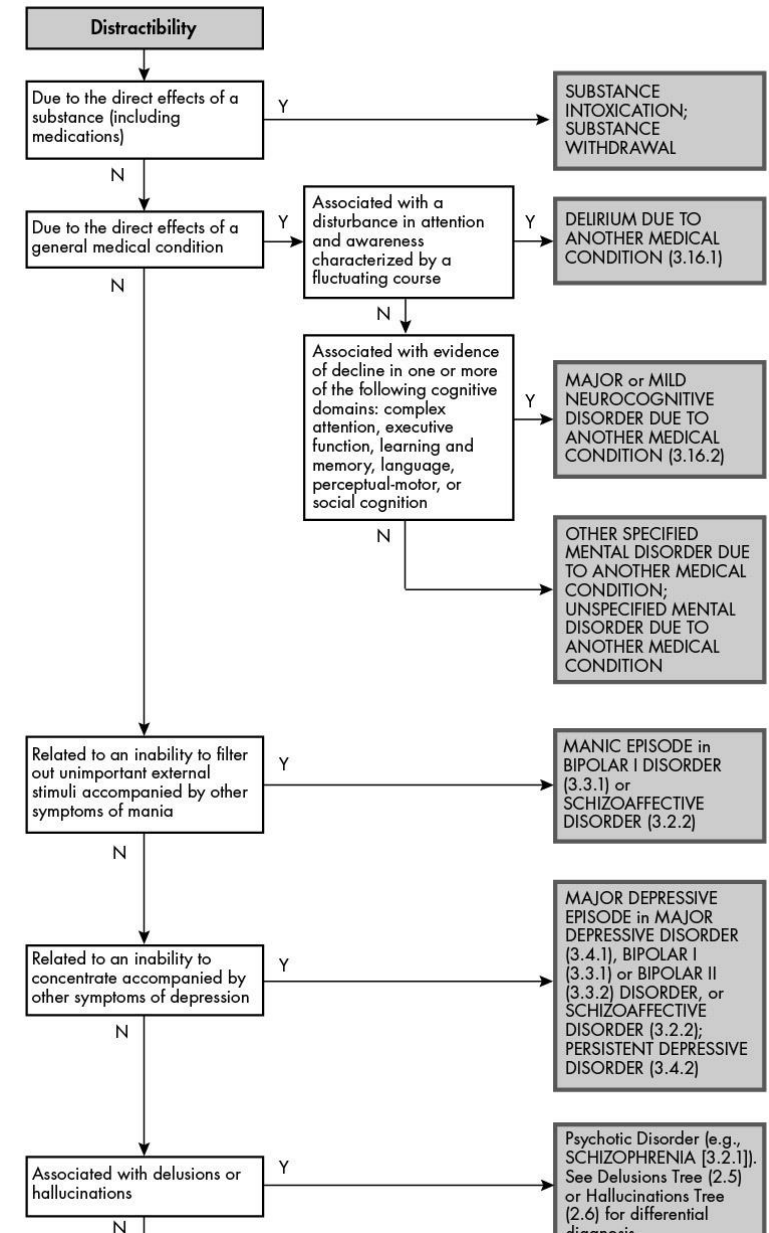
Lecture 10: Decision Trees

CIS 4190/5190

Spring 2025

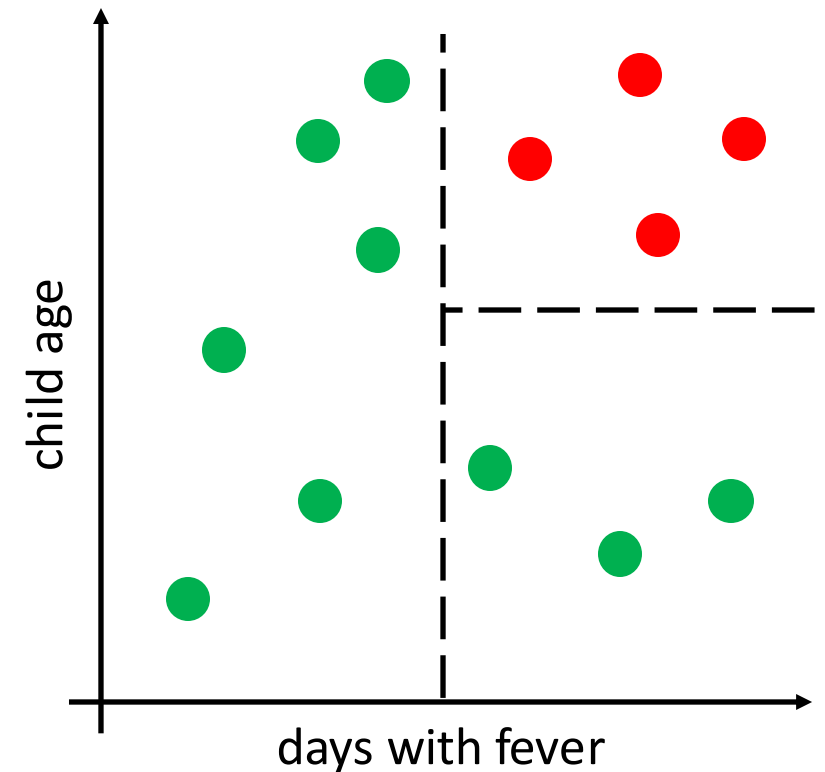
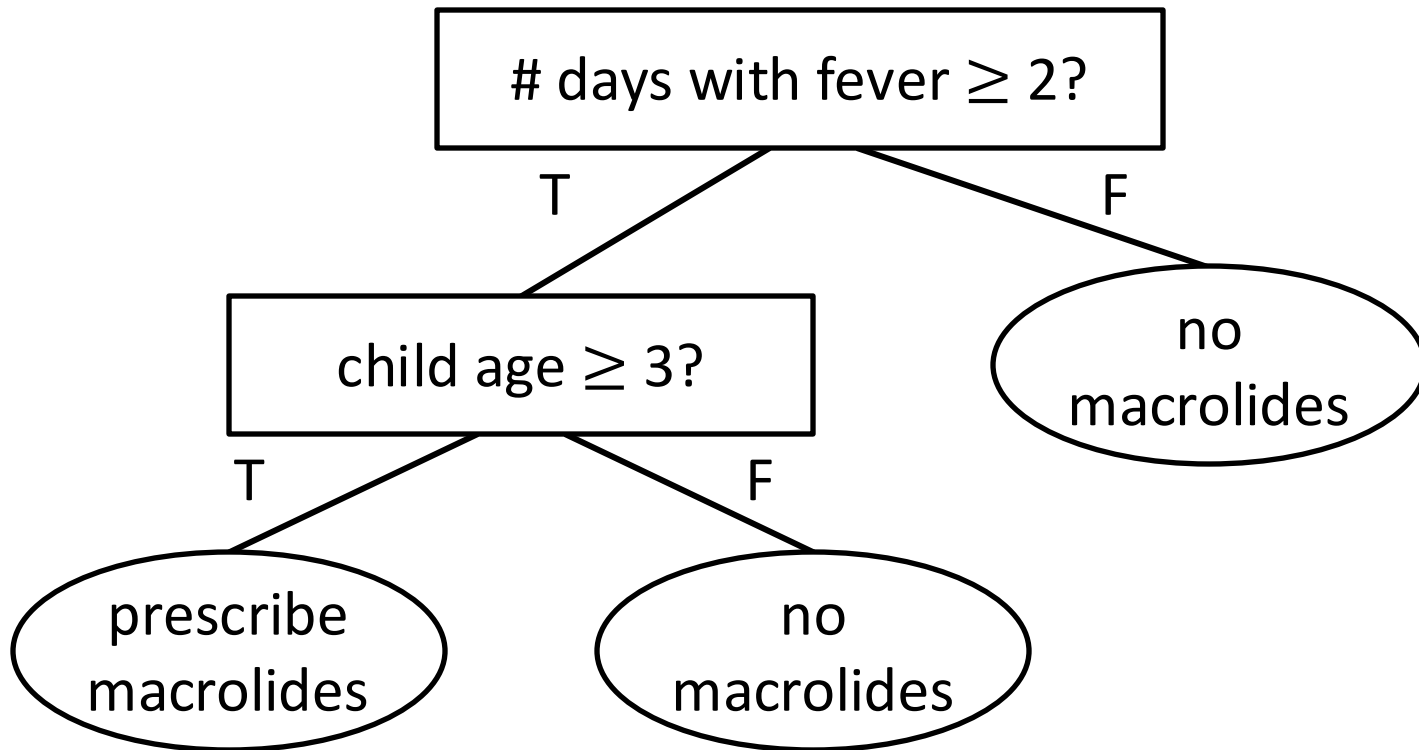
Decision Trees

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



Visualizing the Model Family

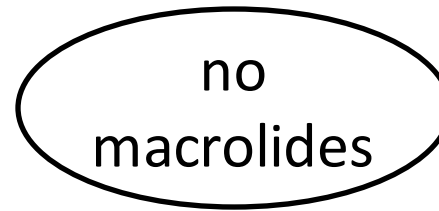
- Axis-aligned decision boundaries



Learning Algorithm

- 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

Learning Algorithm

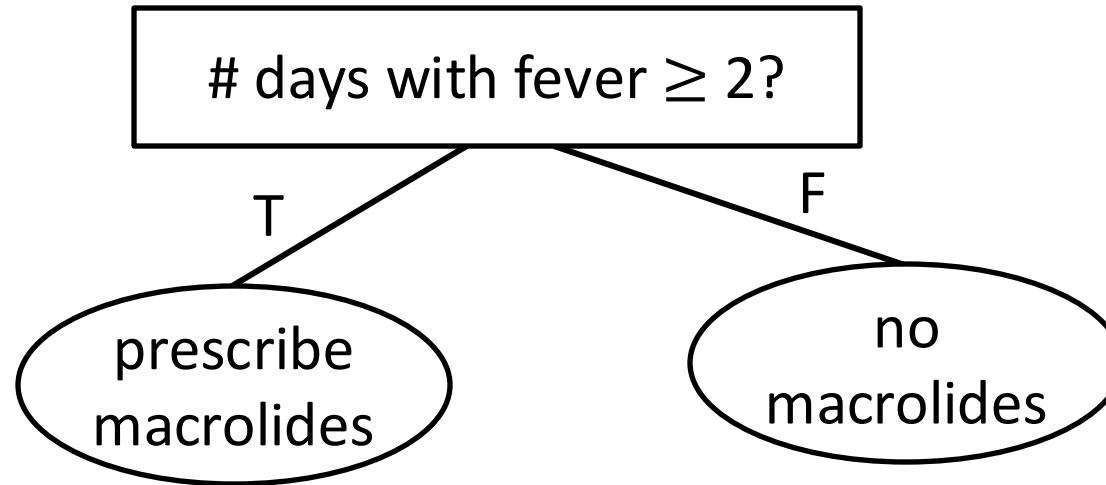


no
macrolides

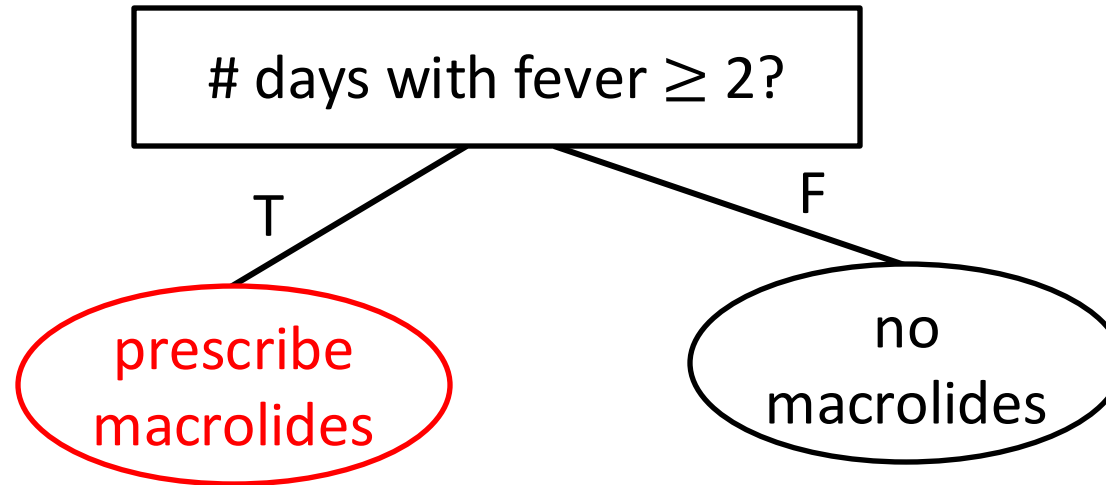
Learning Algorithm

no
macrolides

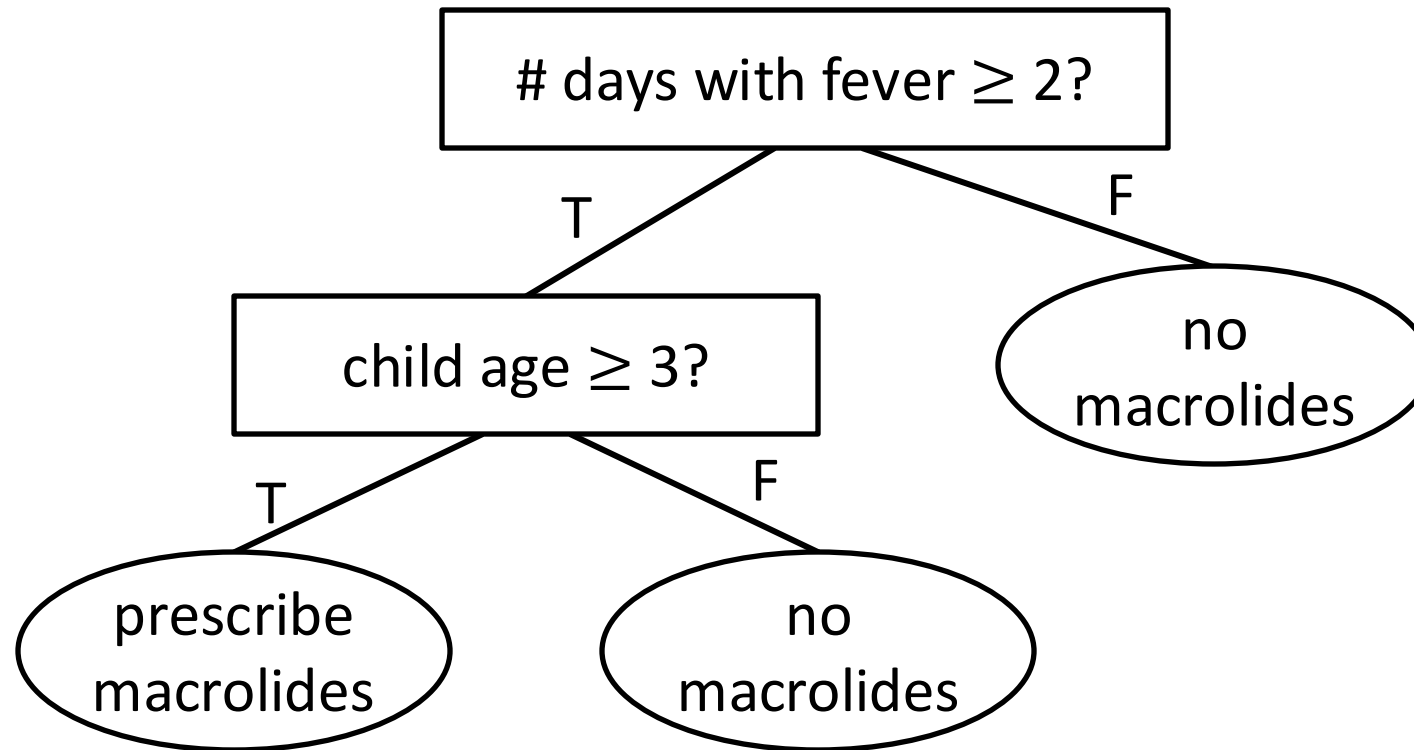
Learning Algorithm



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  $y$ :
```

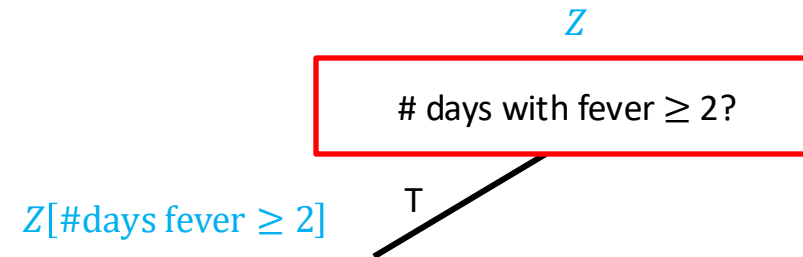
```
        return LeafNode( $y$ )
```

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

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

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

```
    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 :

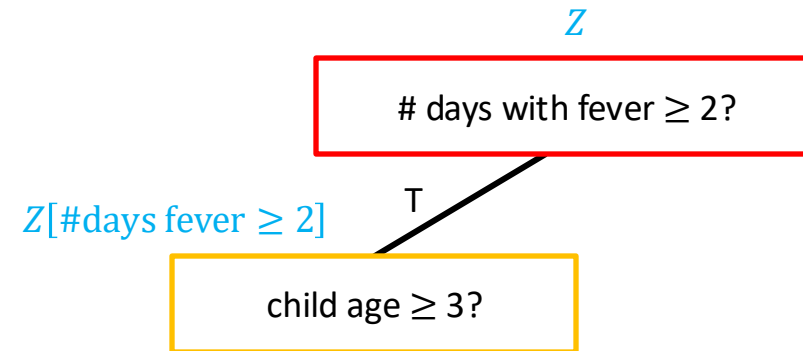
return LeafNode(y)

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

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

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

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  $y$ :
```

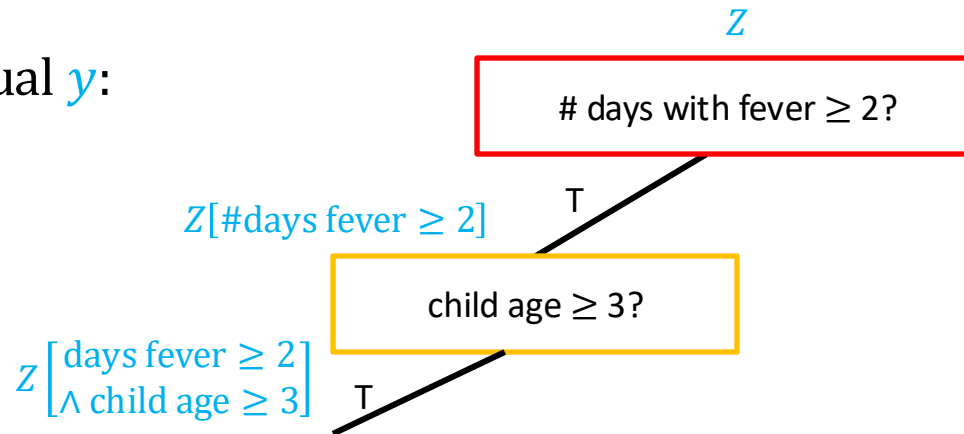
```
        return LeafNode( $y$ )
```

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

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

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

```
    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):

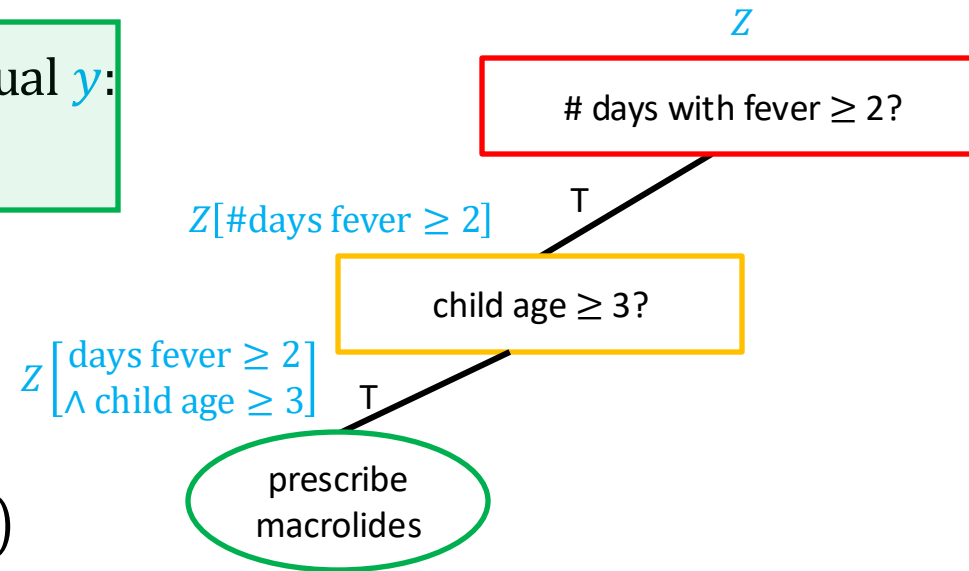
if all labels in Z are the same and equal y :
return LeafNode(y)

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

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

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

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 :

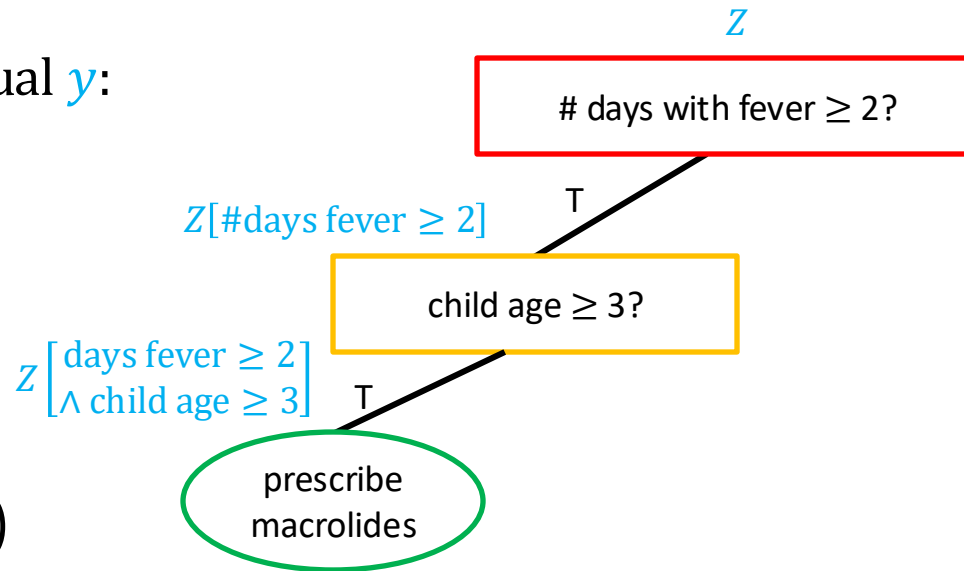
return LeafNode(y)

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

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

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

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  $y$ :
```

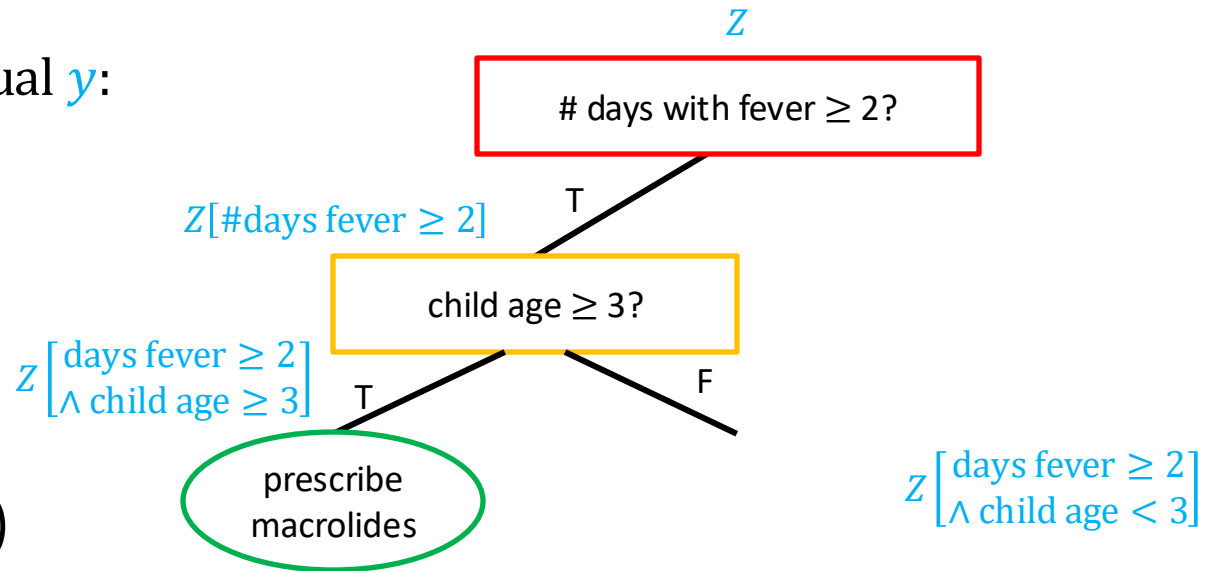
```
        return LeafNode( $y$ )
```

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

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

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

```
    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):

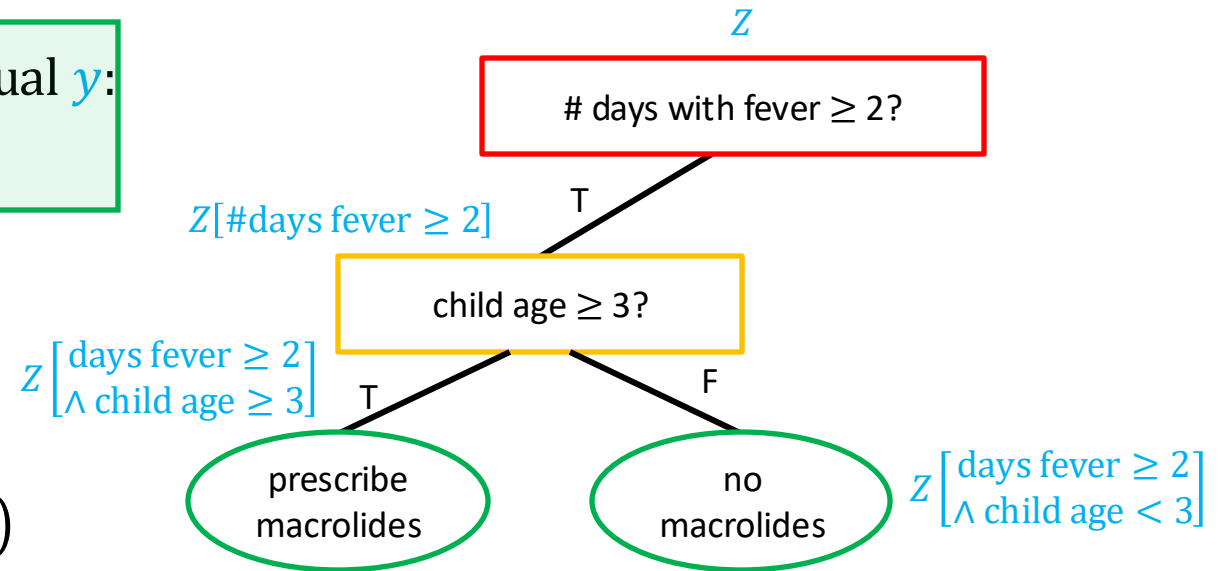
if all labels in Z are the same and equal y :
return LeafNode(y)

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

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

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

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 :

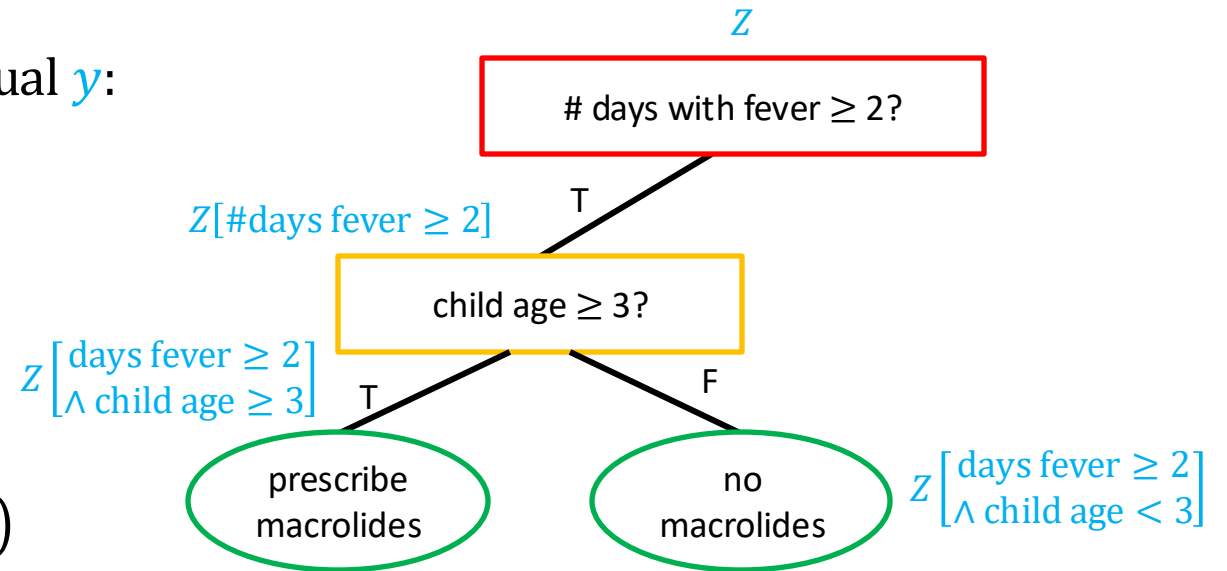
return LeafNode(y)

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

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

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

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 :

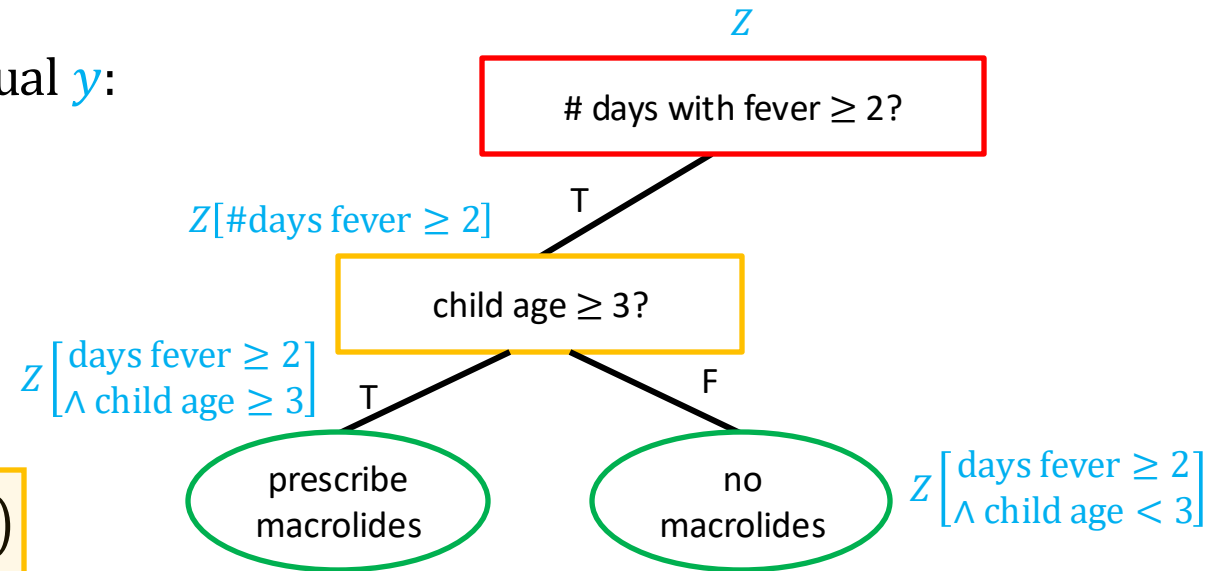
return LeafNode(y)

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

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

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

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 :

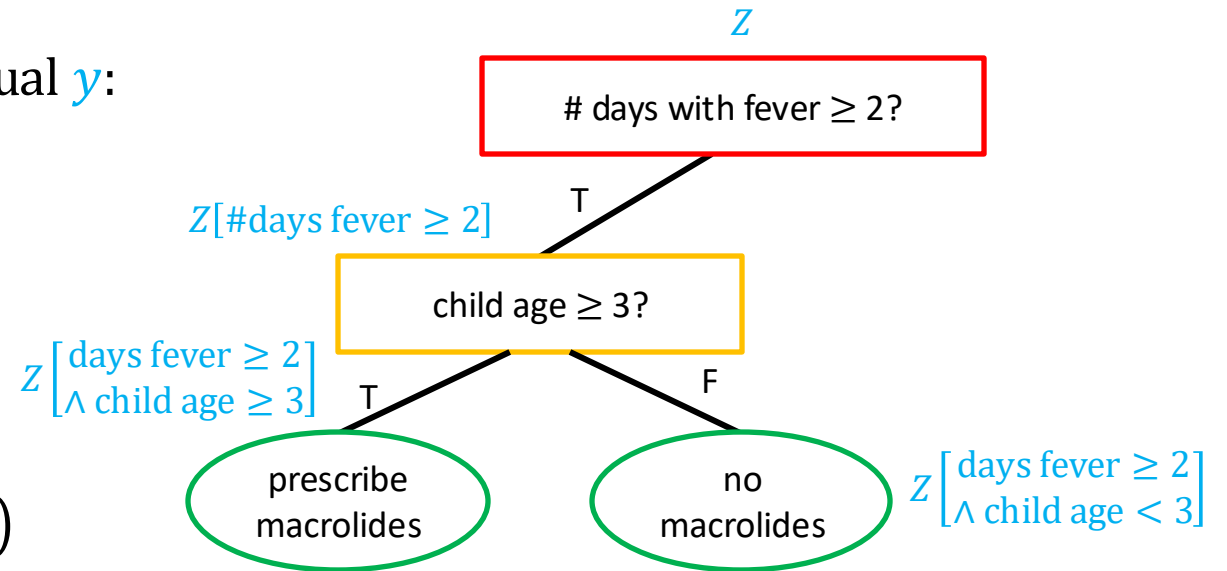
return LeafNode(y)

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

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

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

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  $y$ :
```

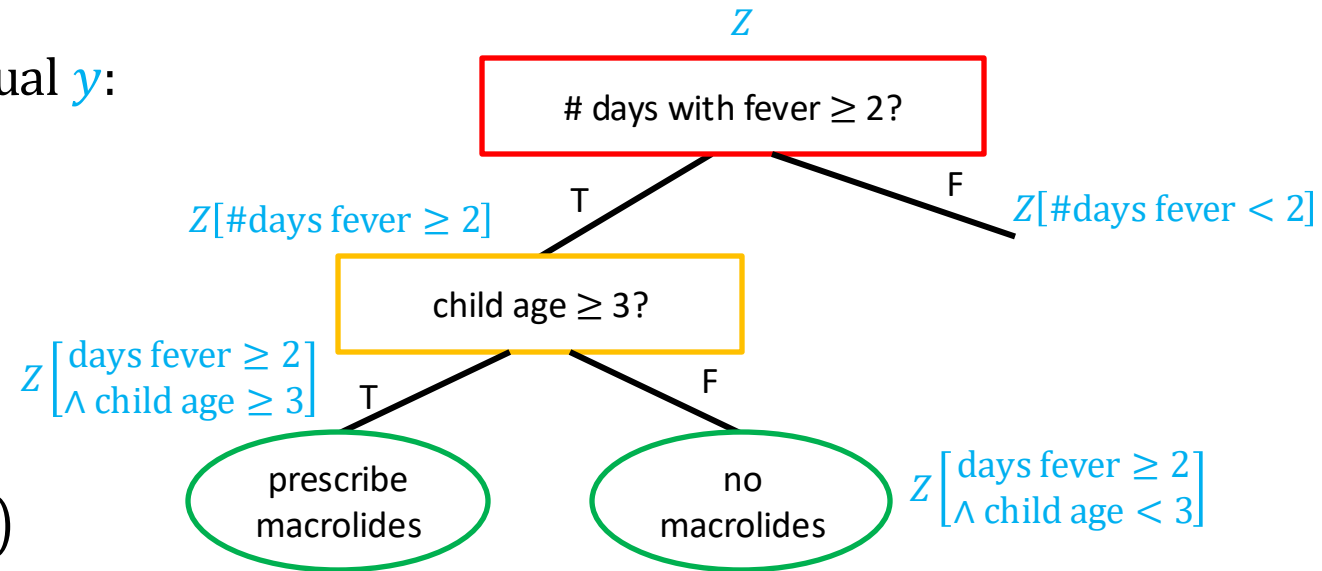
```
        return LeafNode( $y$ )
```

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

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

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

```
    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):

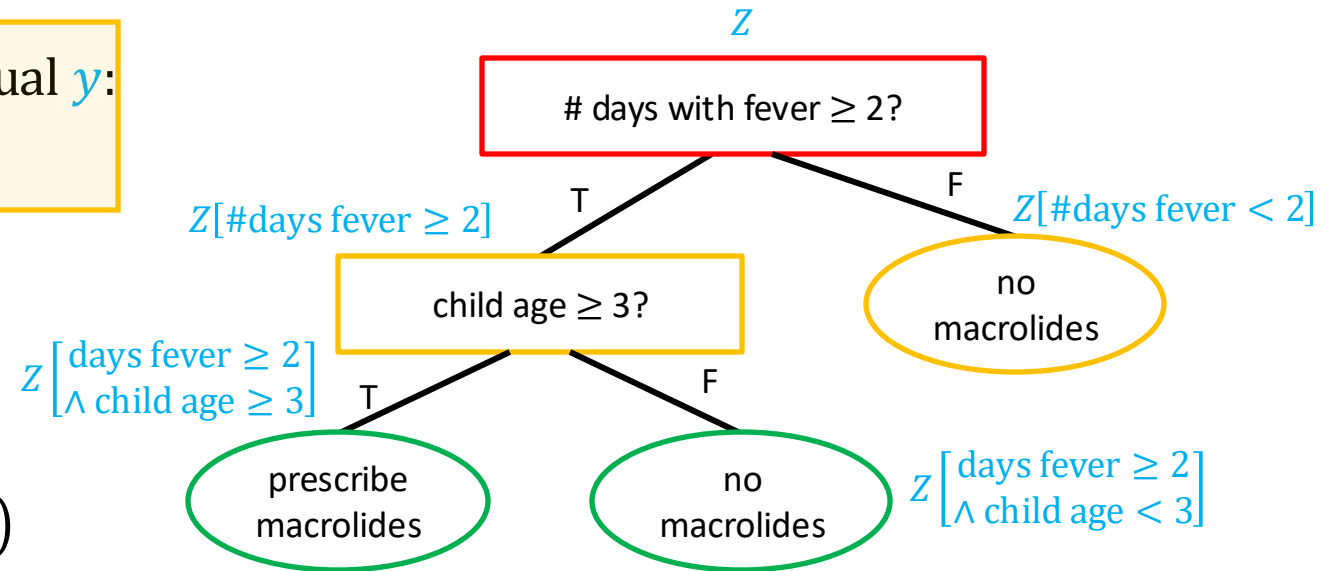
if all labels in Z are the same and equal y :
return LeafNode(y)

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

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

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

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 :

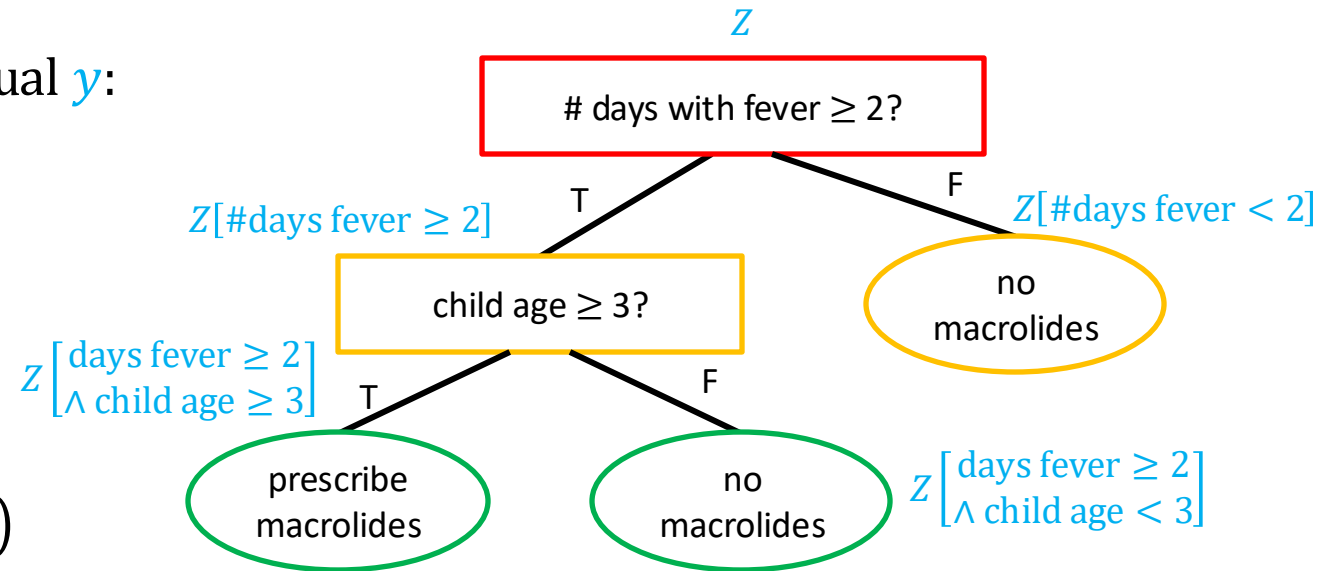
return LeafNode(y)

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

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

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

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 :

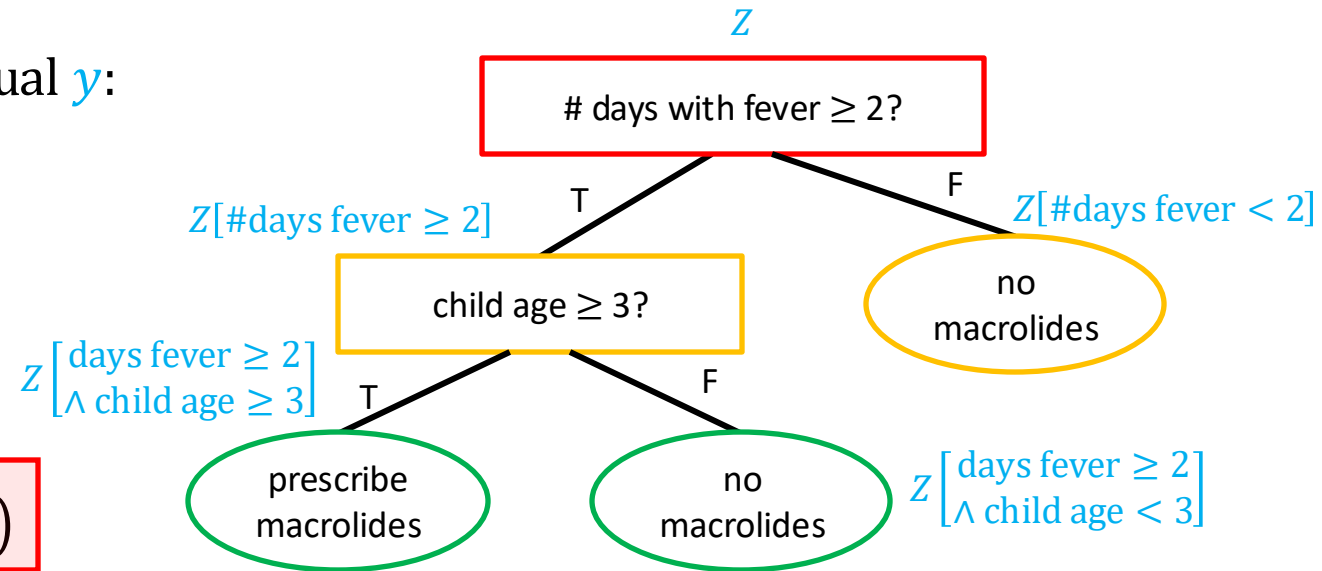
return LeafNode(y)

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

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

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

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 :

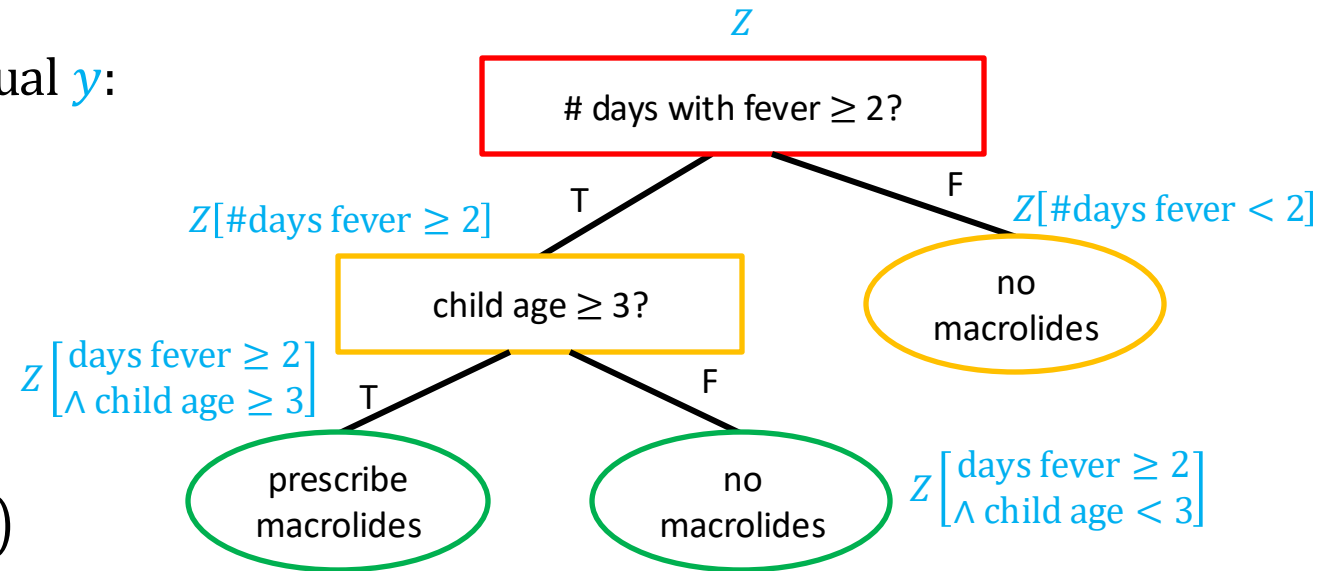
return LeafNode(y)

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

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

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

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)

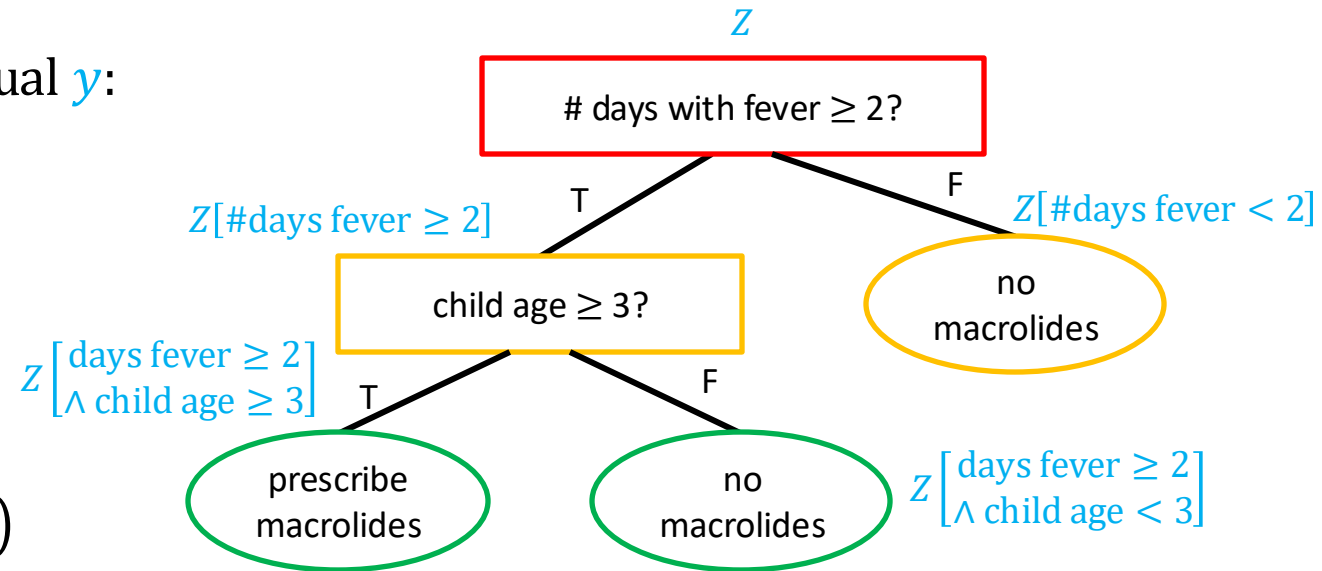
$(j, t) \leftarrow \text{BestSplit}(Z)$

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

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

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

How to
choose the
best split?



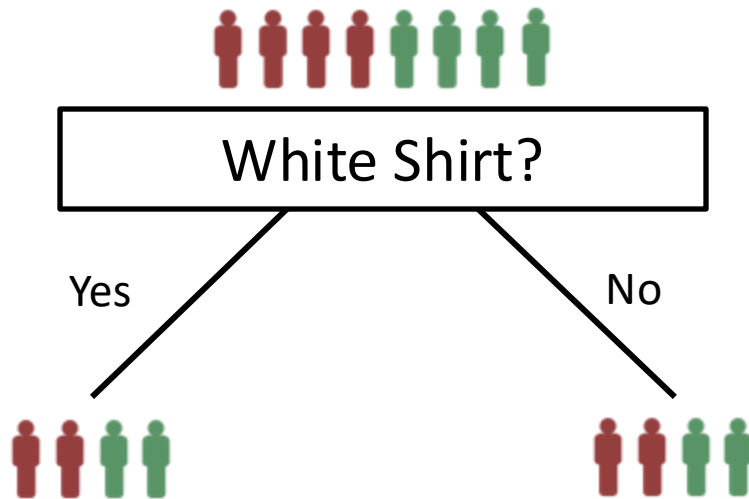
Choosing Splits

- 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**

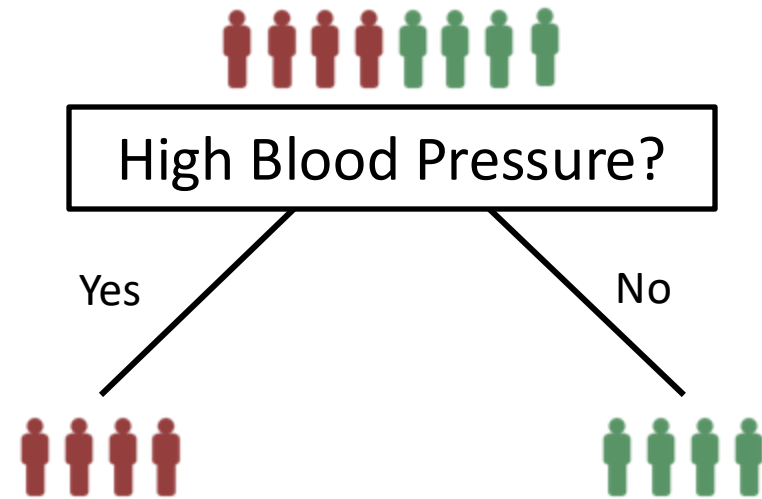
Search Space of Candidate Splits

- 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

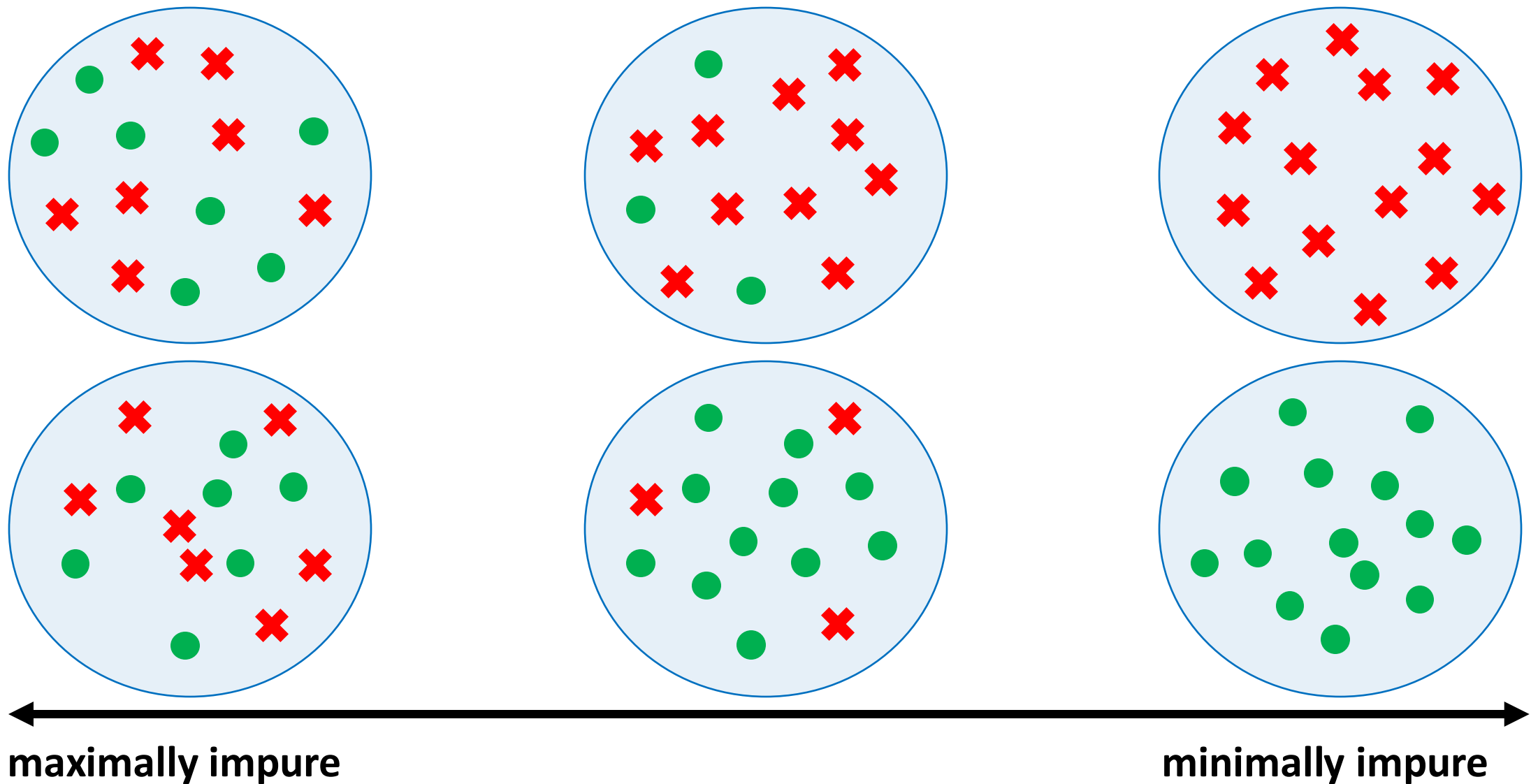


more impure



less impure

Choosing Splits



Quantifying Impurity

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

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

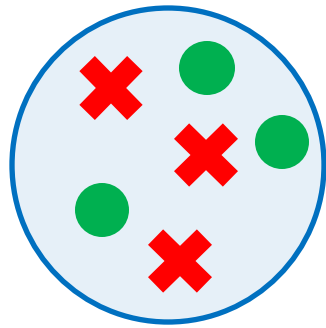
- Given a dataset Z , we define the random variable Y by

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

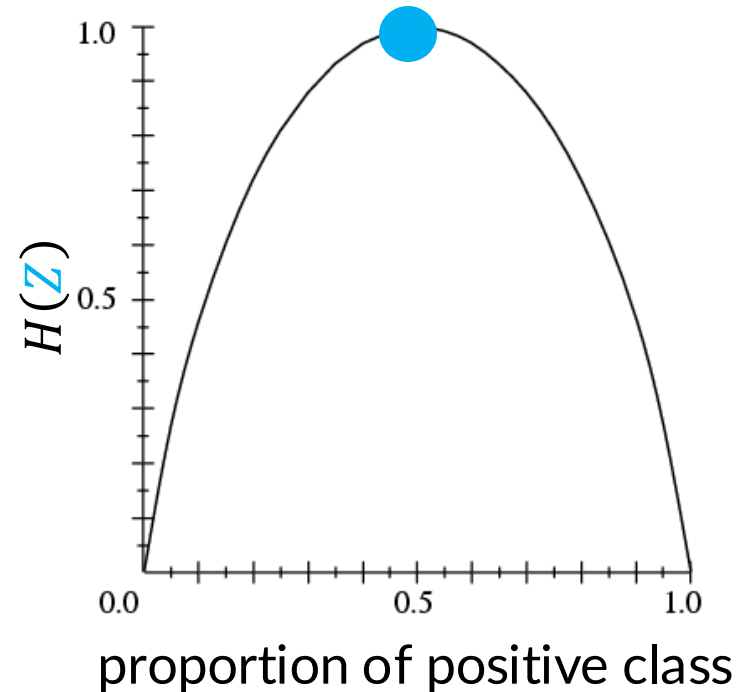
- Now, we define the entropy of Z to be $H(Z) = H(Y)$

Quantifying Impurity

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

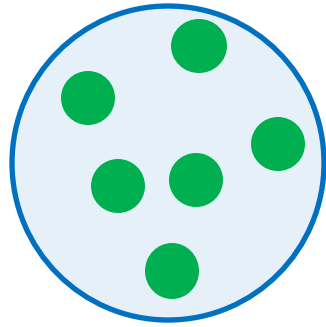


$$H(\textcolor{teal}{Z}) = -\frac{1}{2} \log \frac{1}{2} - \frac{1}{2} \log \frac{1}{2} = 1$$

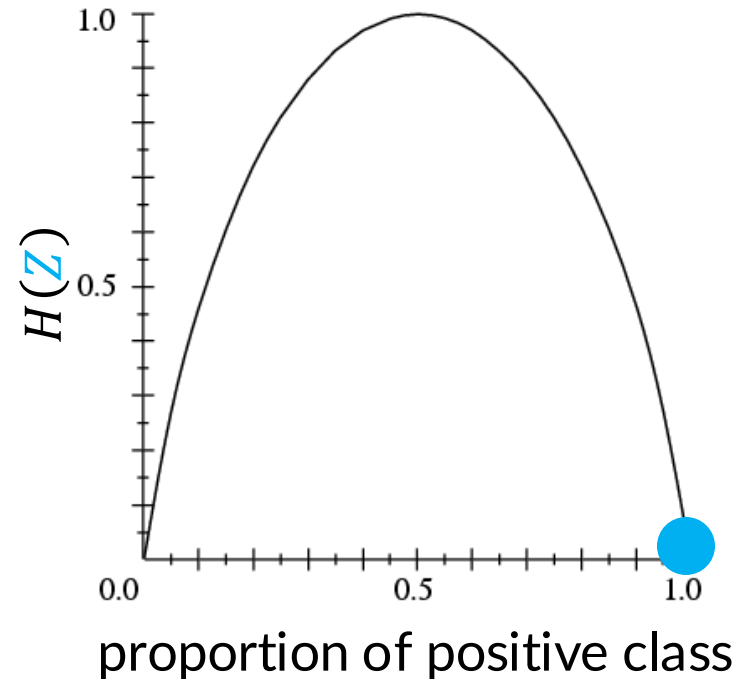


Quantifying Impurity

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



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



Choosing Splits

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

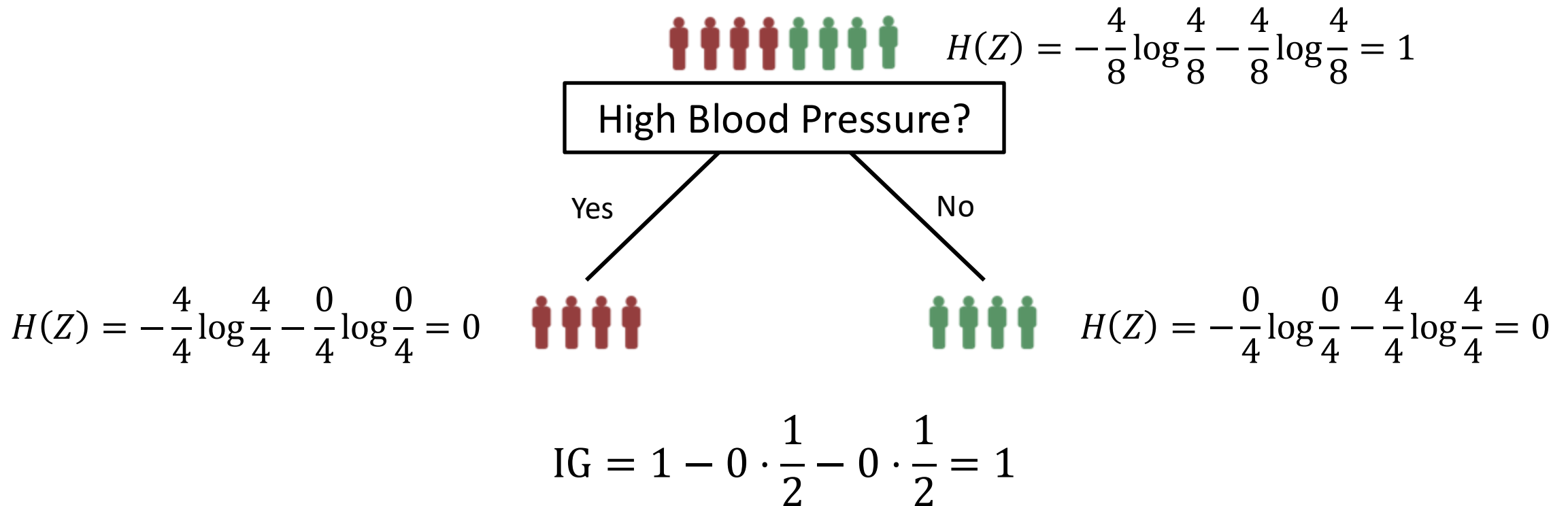
$$\text{IG}(Z, j, t) = H(Z) - H(Z[x_j \geq t])P(x_j \geq t) - H(Z[x_j < t])P(x_j < t)$$

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

Choosing Splits

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

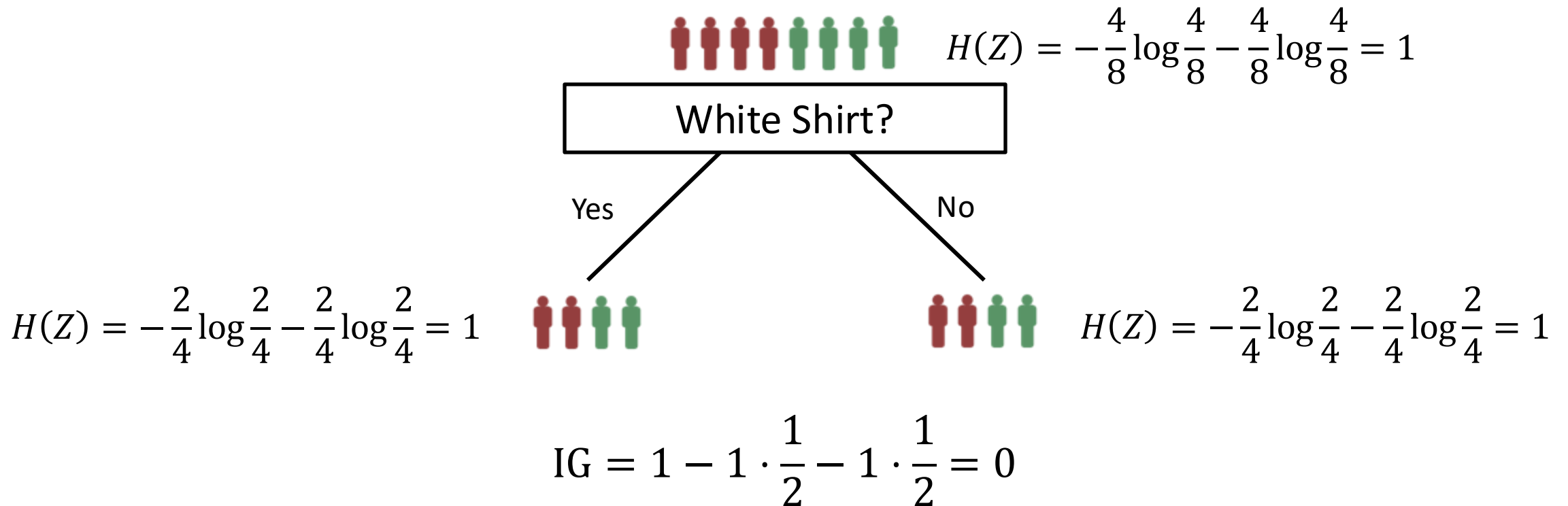
$$\text{IG}(Z, j, t) = H(Z) - H(Z[x_j \geq t])P(x_j \geq t) - H(Z[x_j < t])P(x_j < t)$$



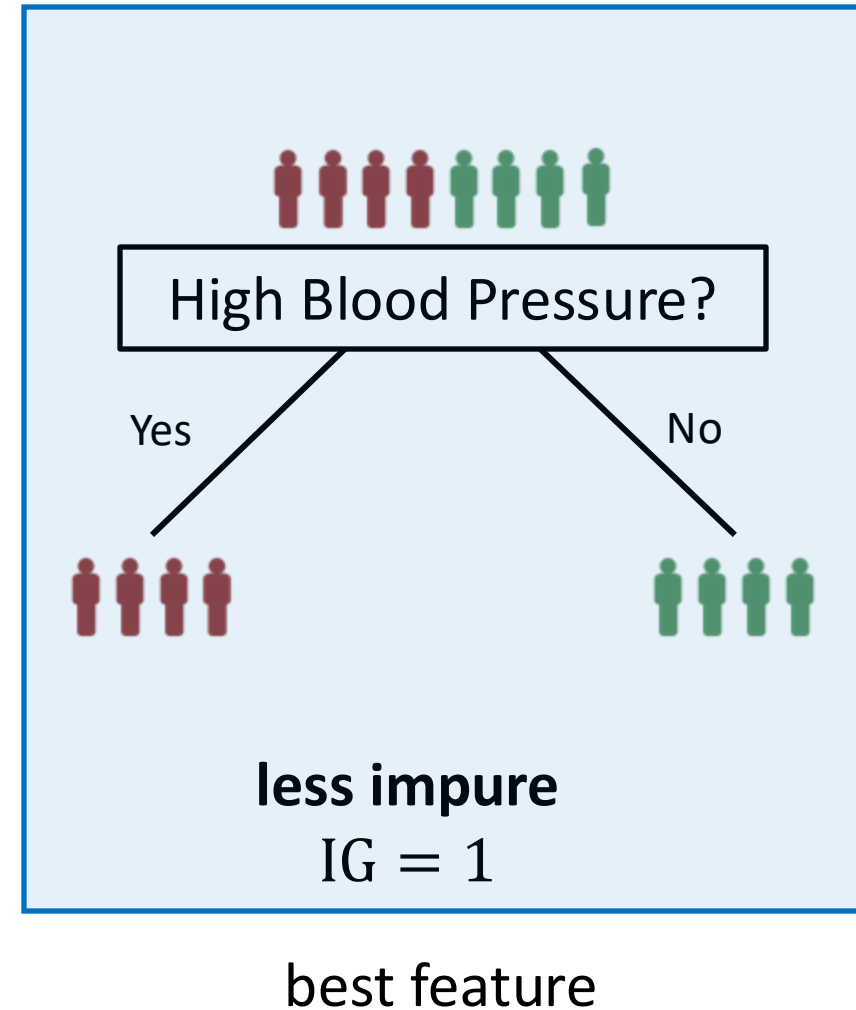
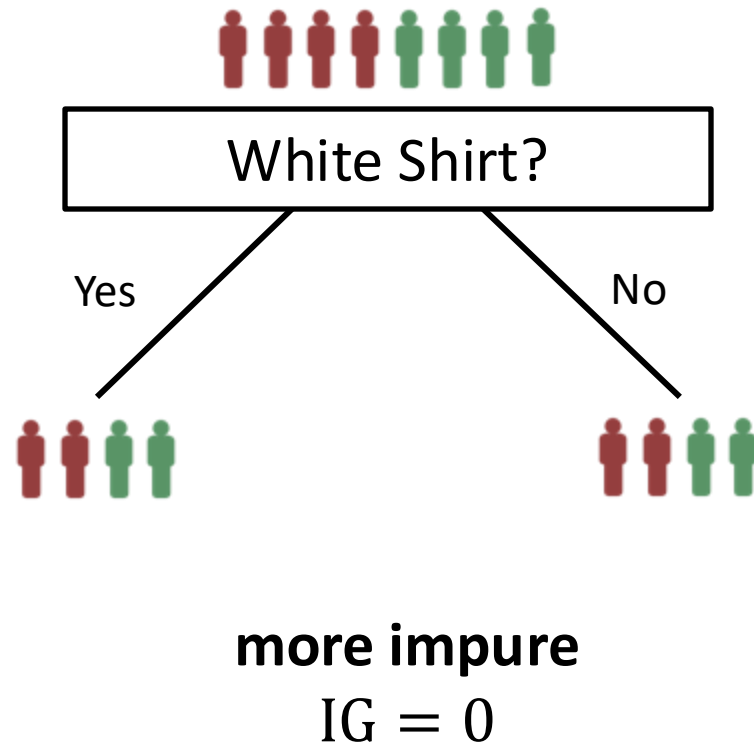
Choosing Splits

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

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



Choosing Splits



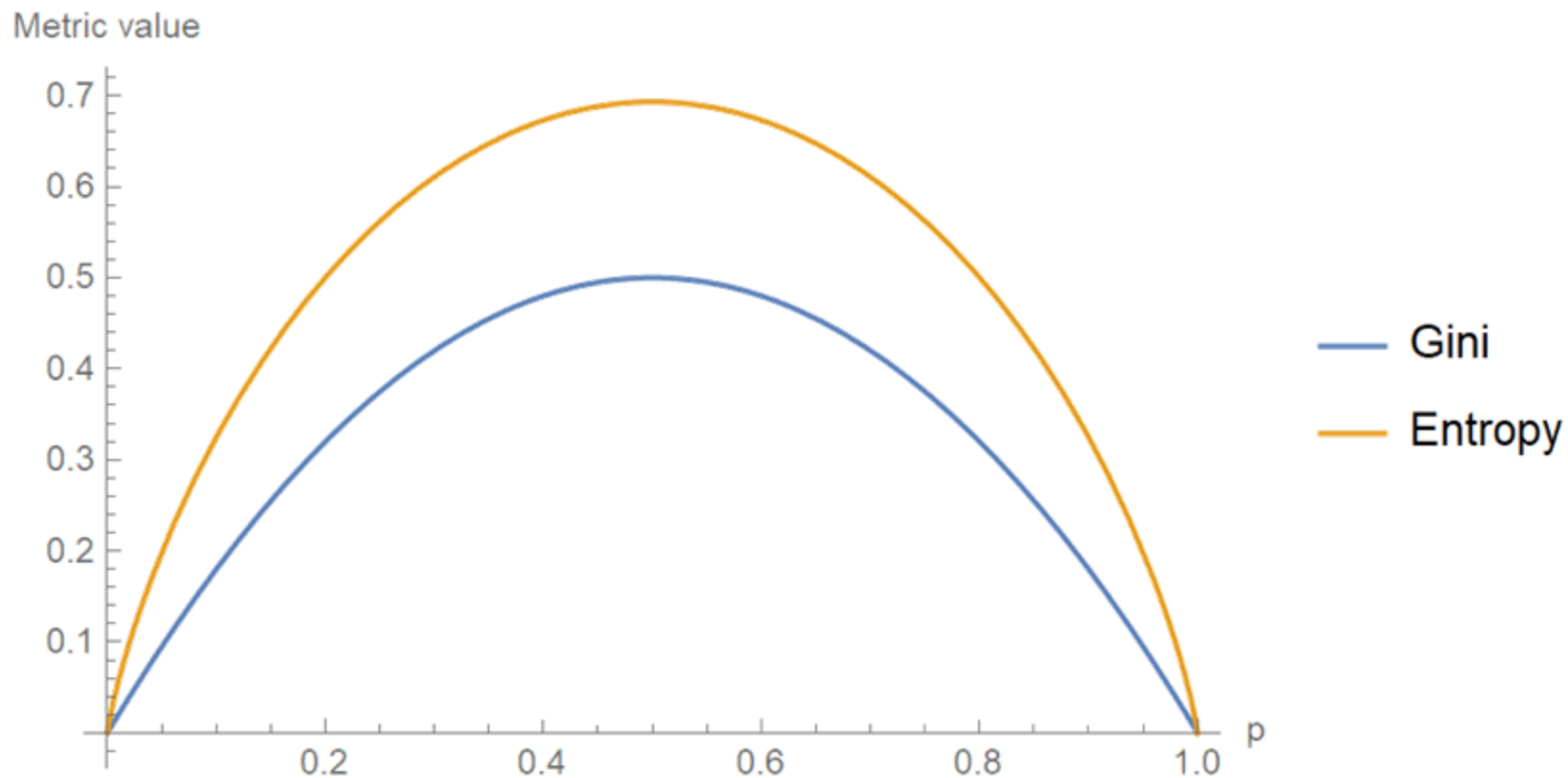
Alternative to Entropy: Gini Index

- 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 $\text{Gini}(Z)$ 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



Feature Standardization

- Like vanilla linear regression, decision trees scale with the data
- If $x_j \leftarrow 2 \cdot x_j$, then $x_j \geq t \Rightarrow x_j \geq t$

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

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

Decision Trees for Regression

- Use MSE instead of entropy (or Gini index):

$$\text{MSE}(Y) = \sum_{y=1}^k P(Y = y) (y - \mu(Y))^2$$

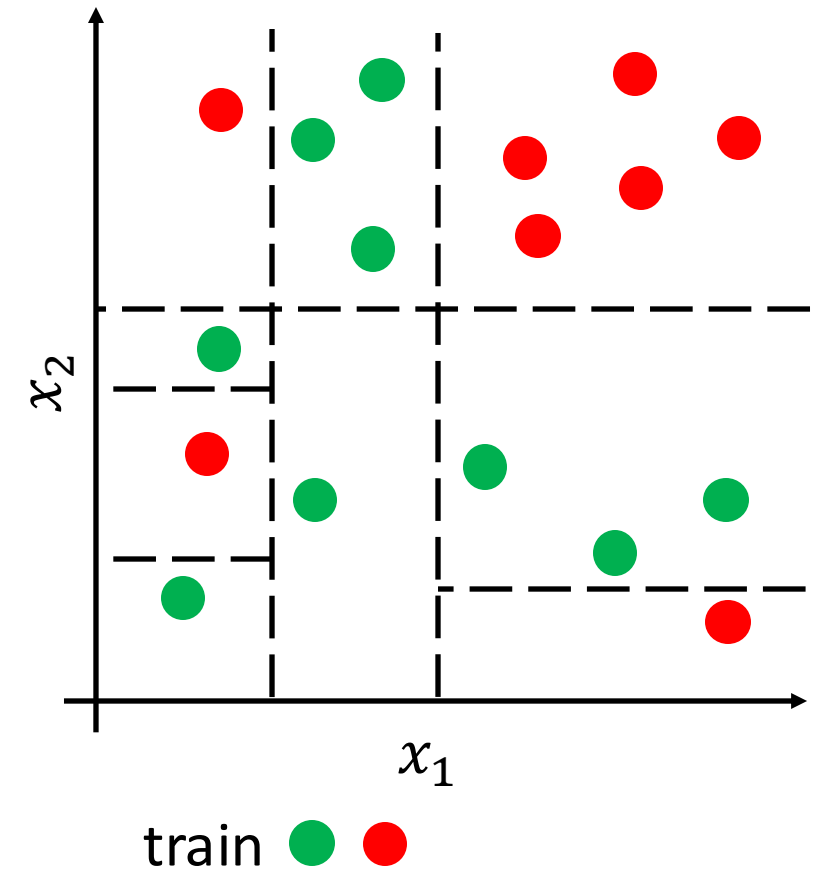
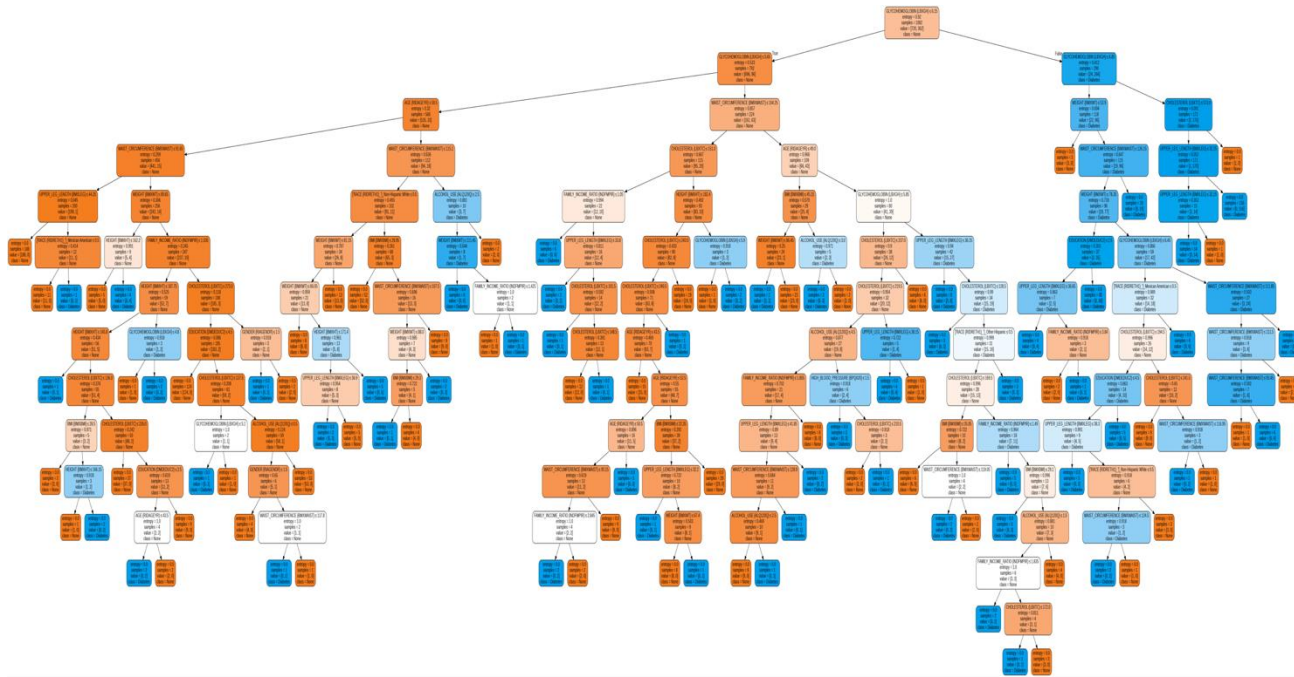
- Here, $\mu(Y) = \sum_{y=1}^k P(Y = y) \cdot y$ is the mean of Y
- Define $\text{MSE}(\textcolor{blue}{Z}) = \text{MSE}(Y)$ as before

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

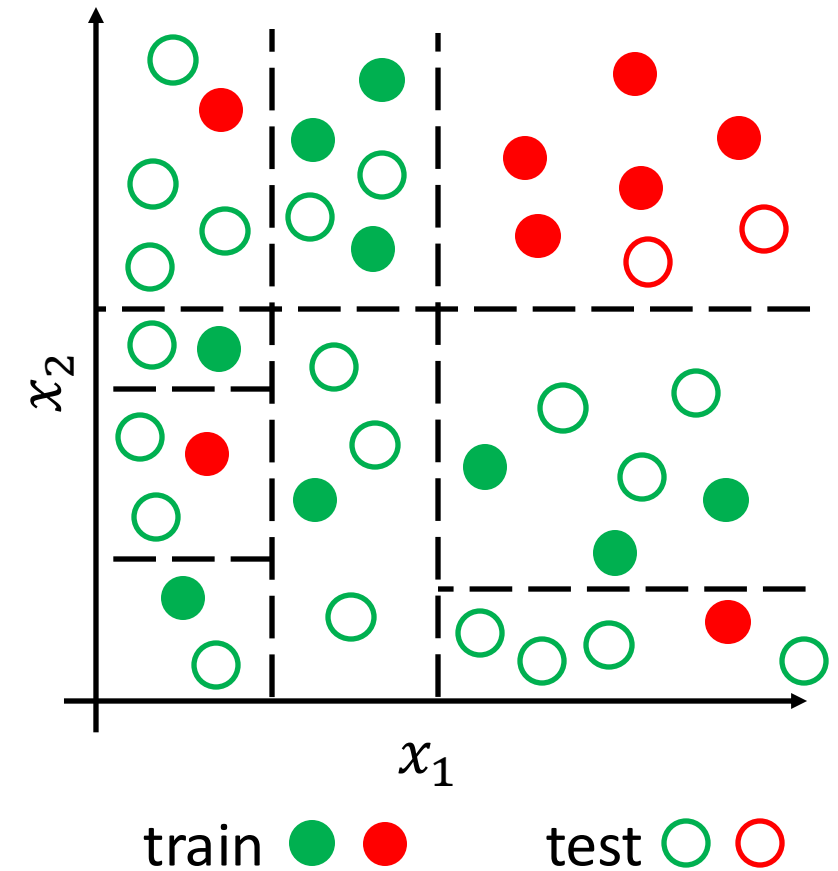
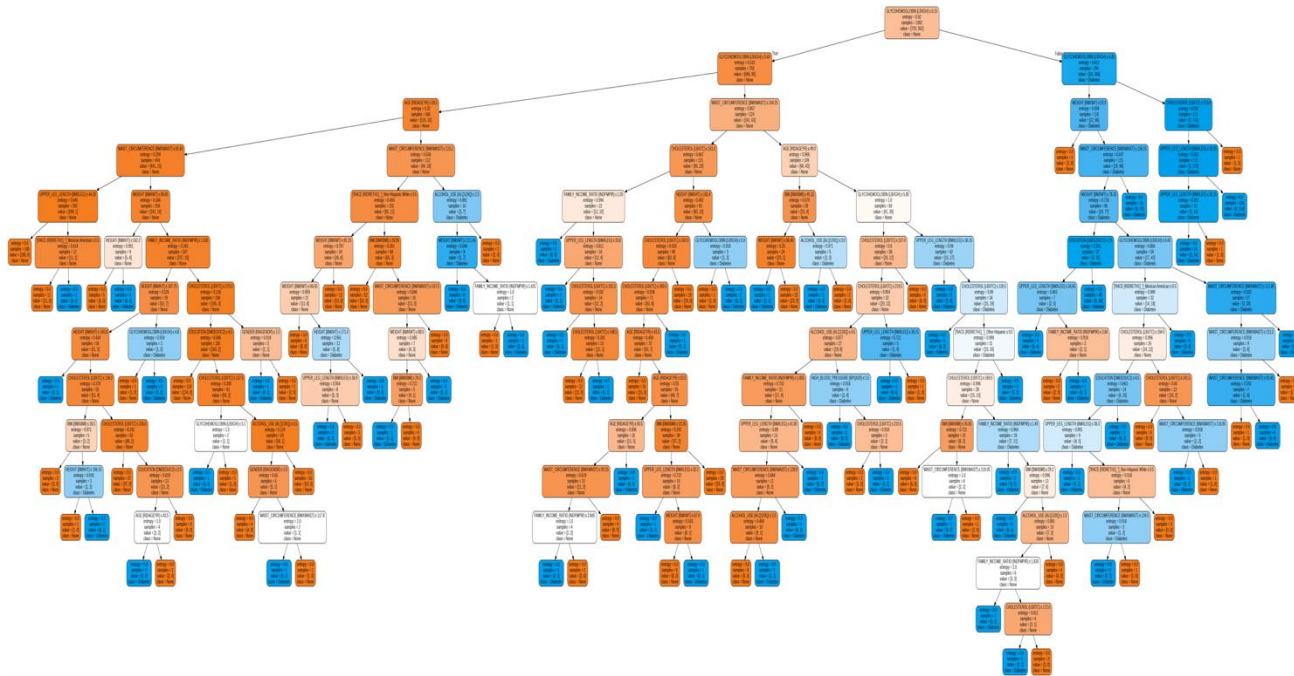
Bias-Variance Tradeoff

- Grow trees until each leaf node is pure



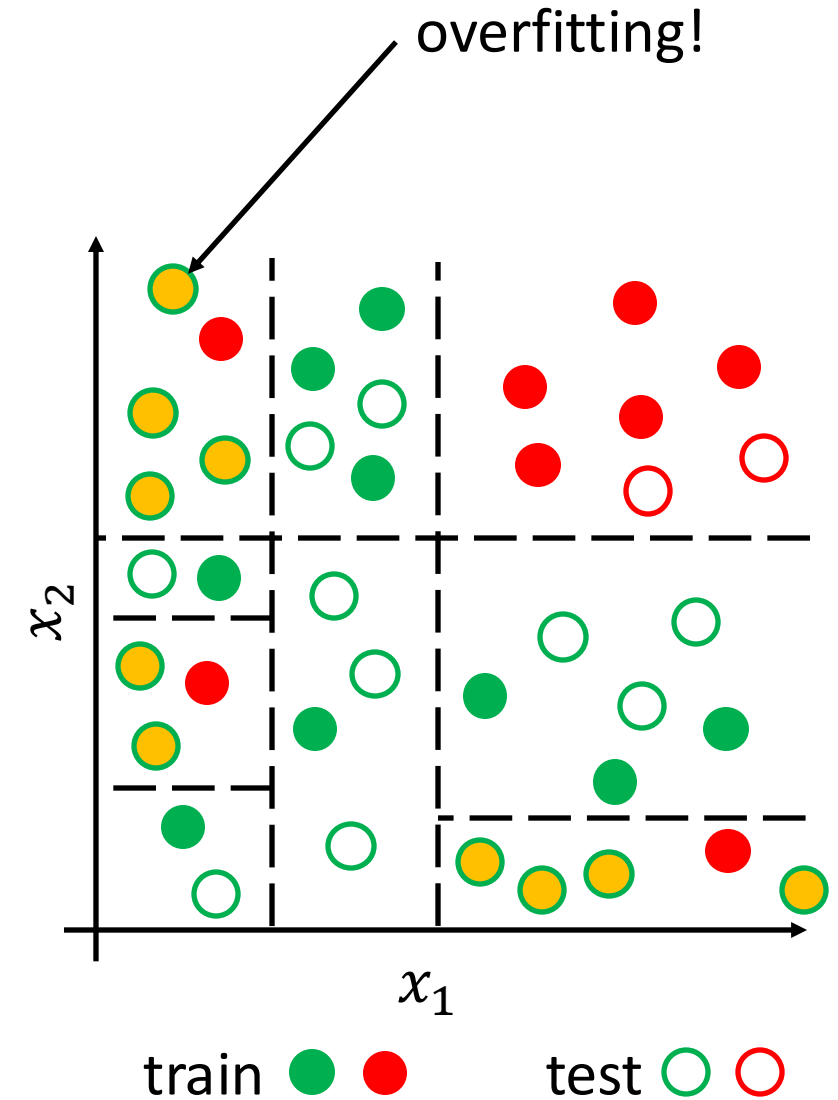
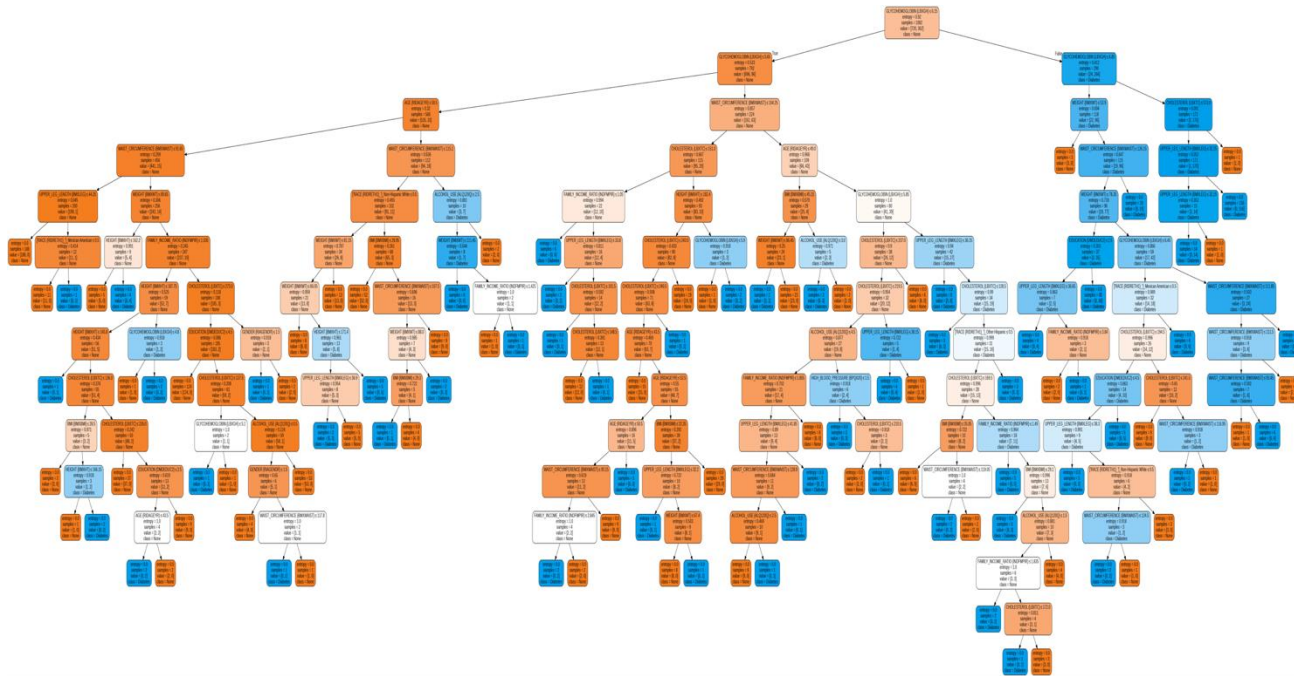
Bias-Variance Tradeoff

- Grow trees until each leaf node is pure



Bias-Variance Tradeoff

- Grow trees until each leaf node is pure

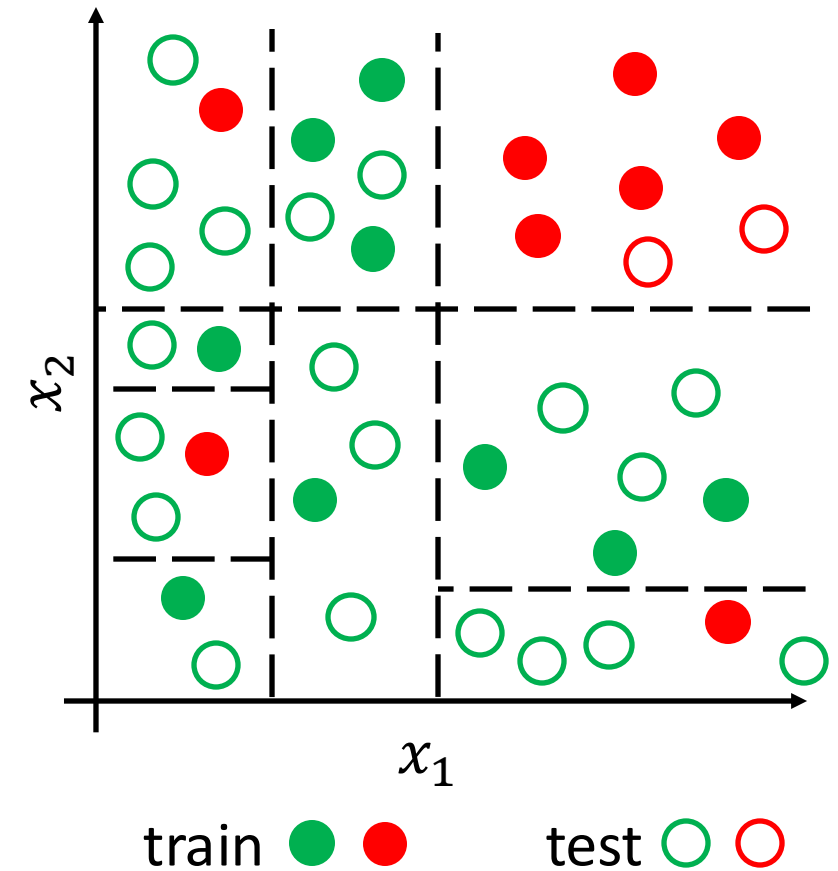
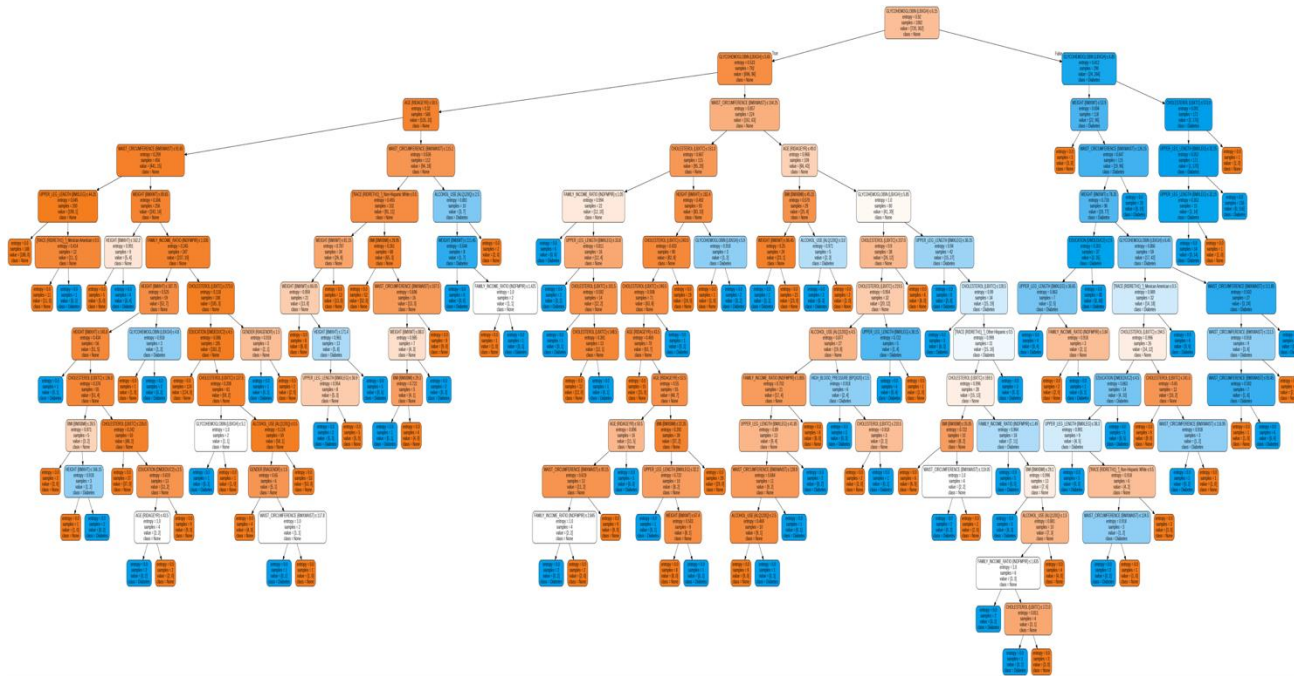


Bias-Variance Tradeoff

- **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

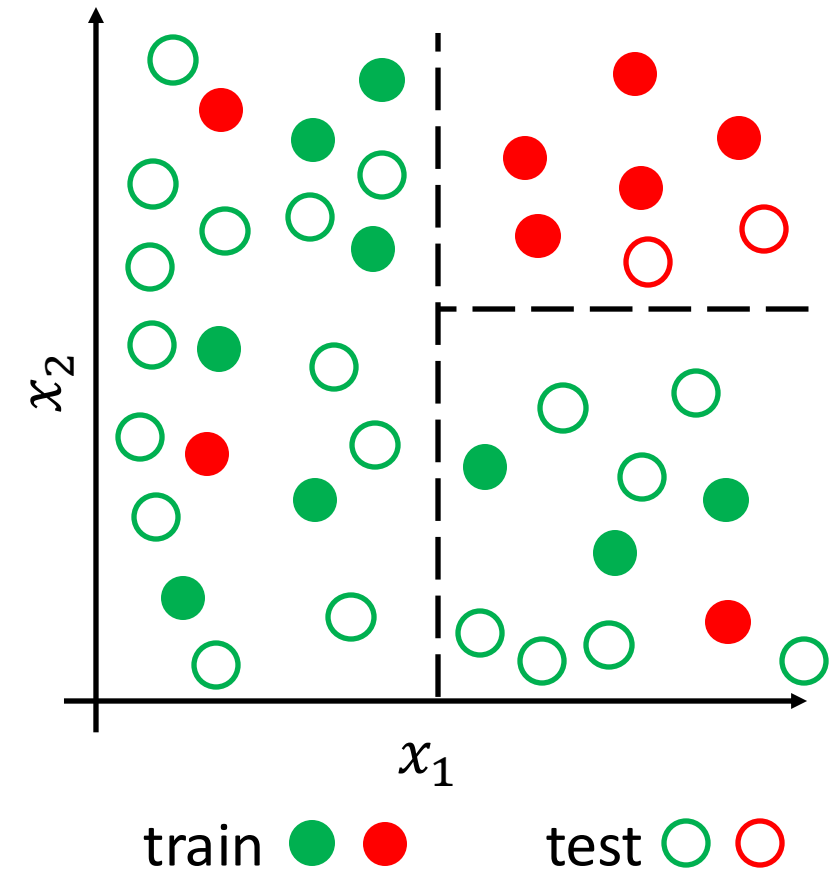
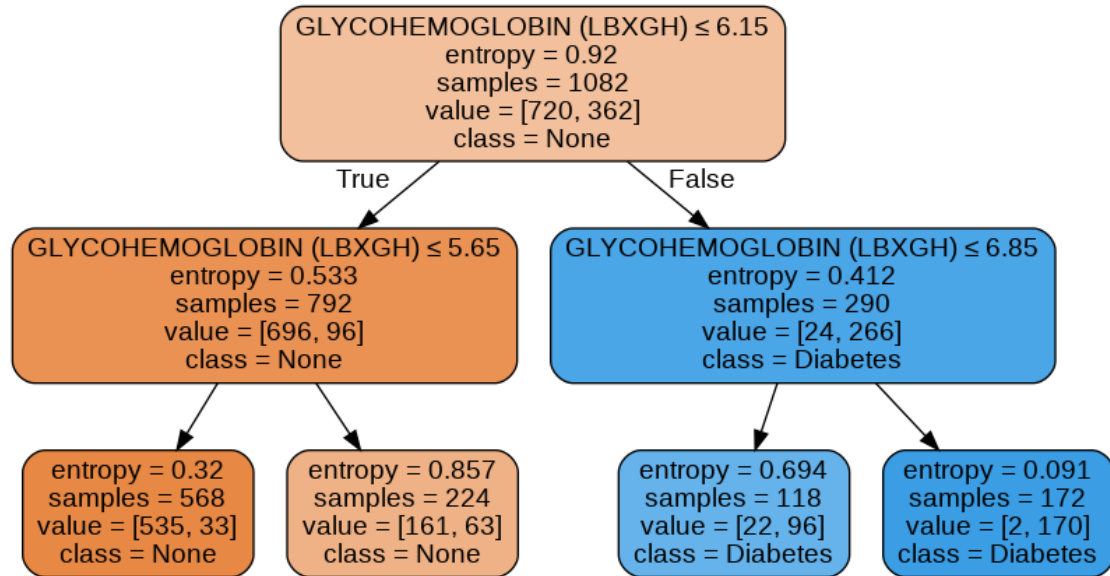
Bias-Variance Tradeoff

- 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**

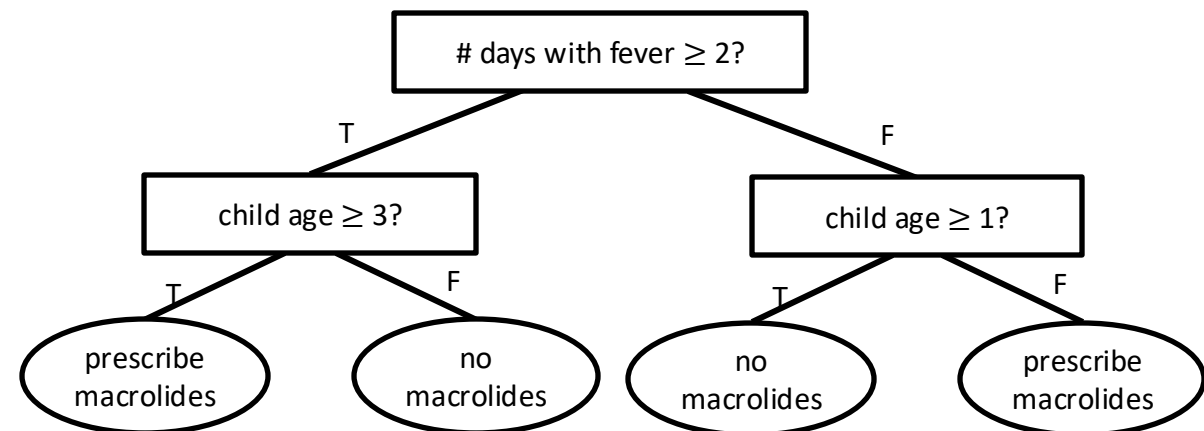
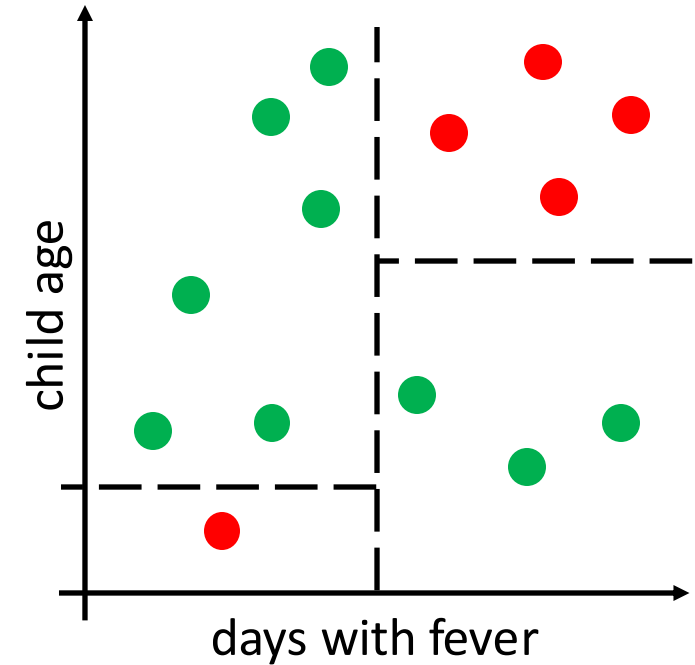
- **Step 1:** Grow full decision tree
- **Step 2:** Prune unhelpful nodes based on **validation dataset**

Post Pruning

```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):  
    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 PostPruneTree( $T_N$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ )  
    else:  
        return  $T$ 
```

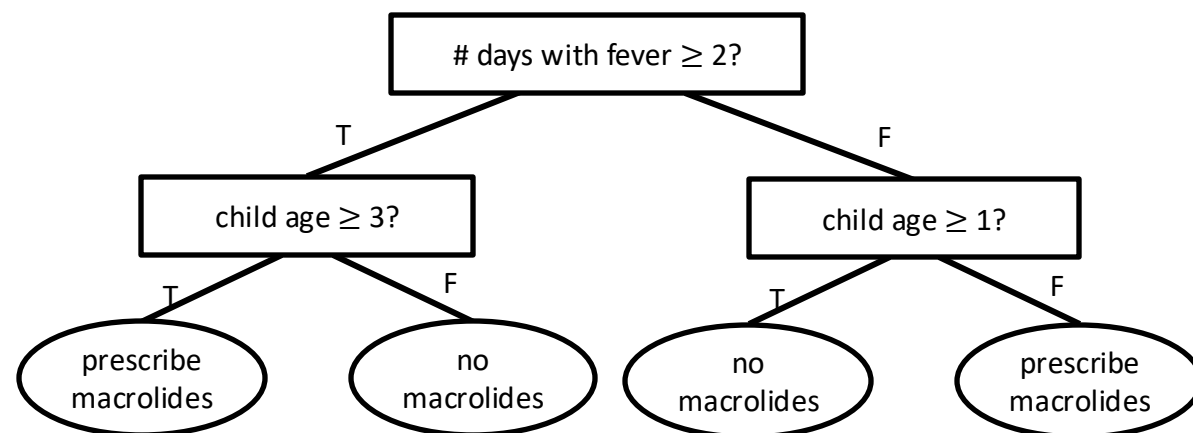
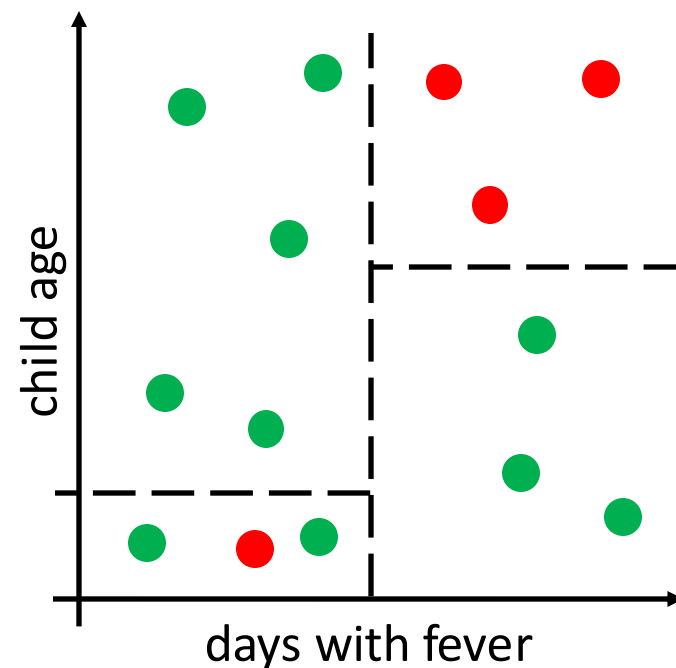
Post Pruning

```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):  
    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  $\text{PostPruneTree}(T_{N_0}, Z_{\text{train}}, Z_{\text{val}})$   
    else:  
        return  $T$ 
```



Post Pruning

```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):  
    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  $\text{PostPruneTree}(T_{N_0}, Z_{\text{train}}, Z_{\text{val}})$   
    else:  
        return  $T$ 
```



Post Pruning

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

```
    for each internal node  $N$  of  $T$ :
```

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

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

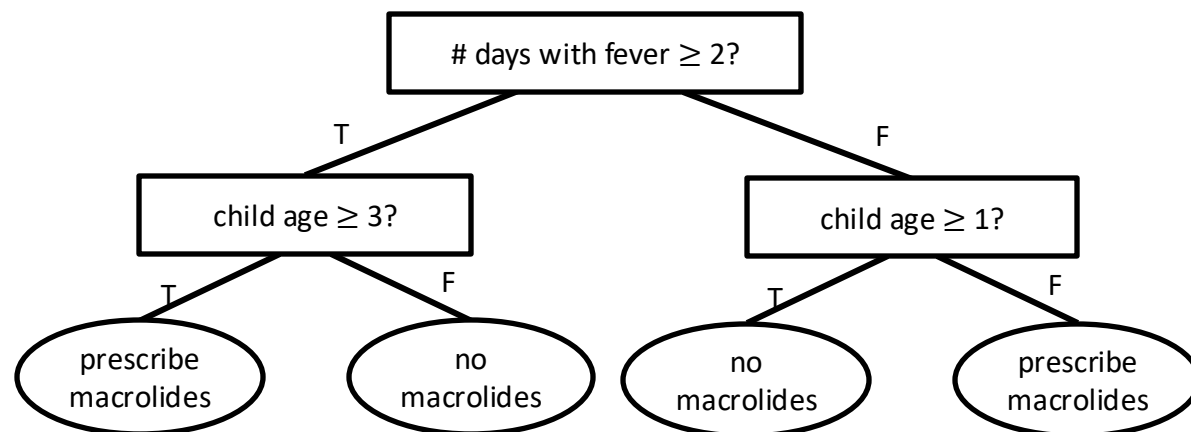
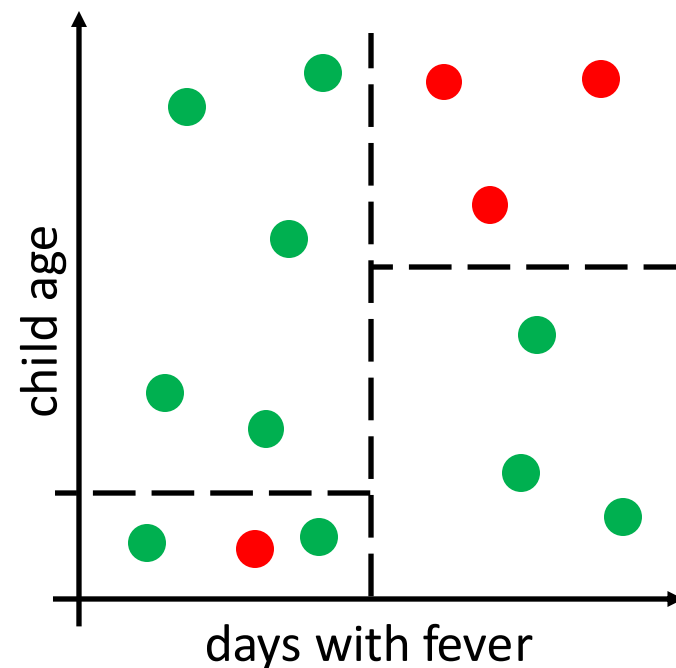
```
     $N_0 \leftarrow \arg \max_N g_N$ 
```

```
    if  $g_{N_0} > 0$ :
```

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

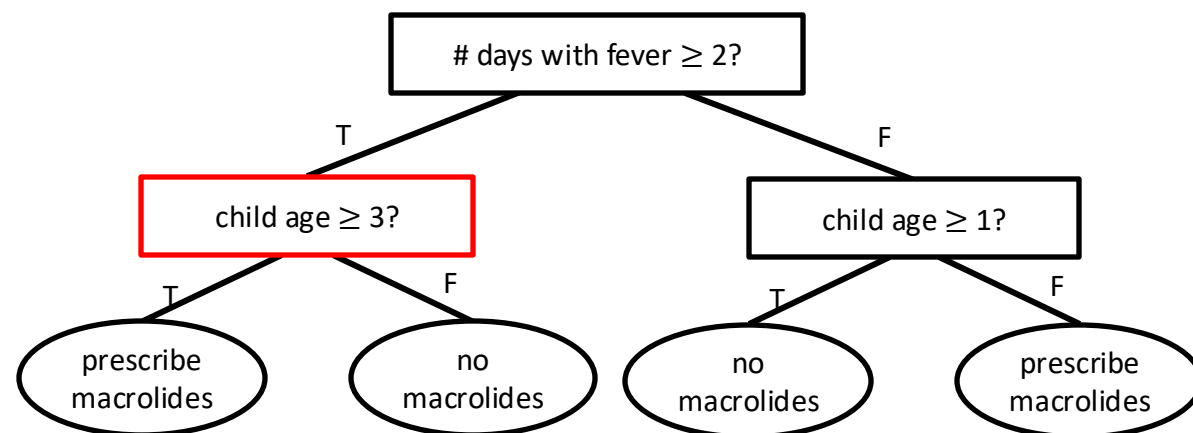
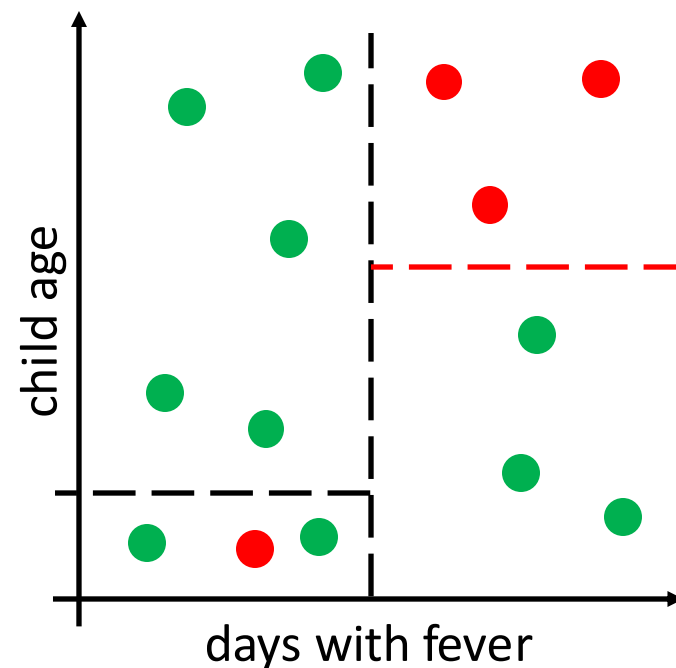
```
    else:
```

```
        return  $T$ 
```



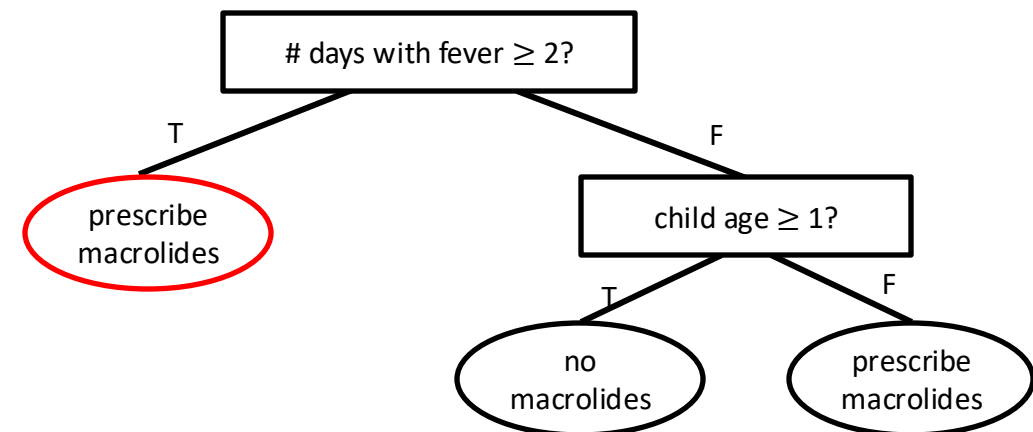
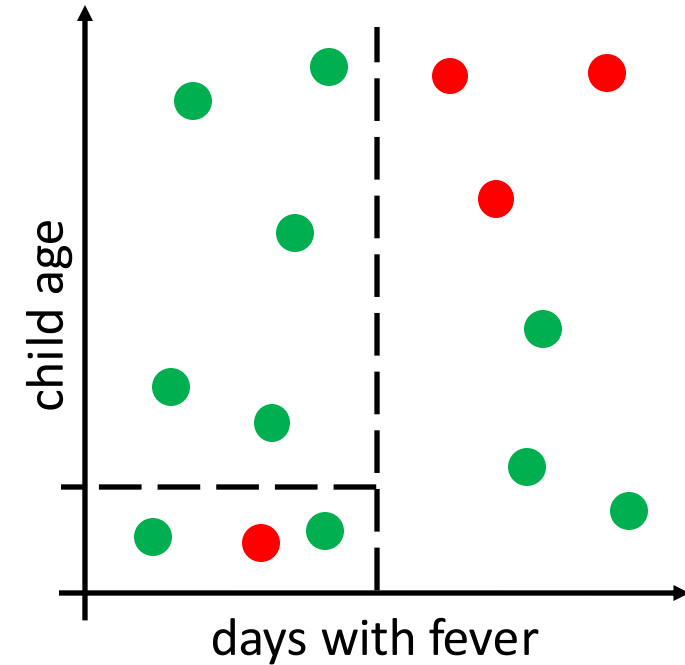
Post Pruning

```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):  
    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  $\text{PostPruneTree}(T_{N_0}, Z_{\text{train}}, Z_{\text{val}})$   
    else:  
        return  $T$ 
```



Post Pruning

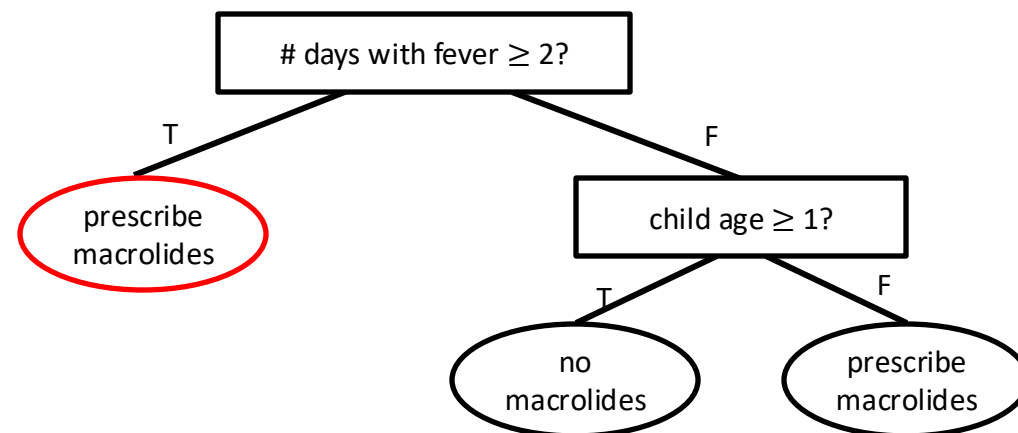
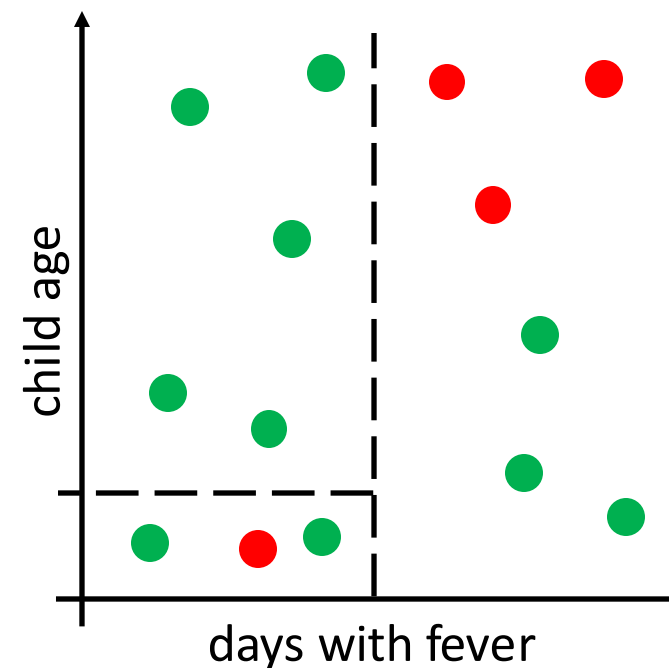
```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):  
    for each internal node  $N$  of  $T$ :  
         $T_N \leftarrow \text{Replace}(T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])))$   
         $g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$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



Post Pruning

```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):
    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 PostPruneTree( $T_{N_0}$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ )
    else:
        return  $T$ 
```

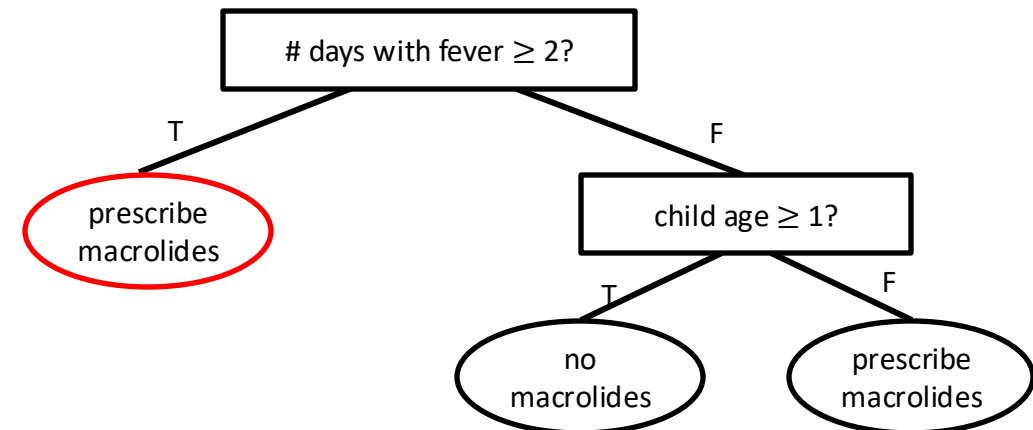
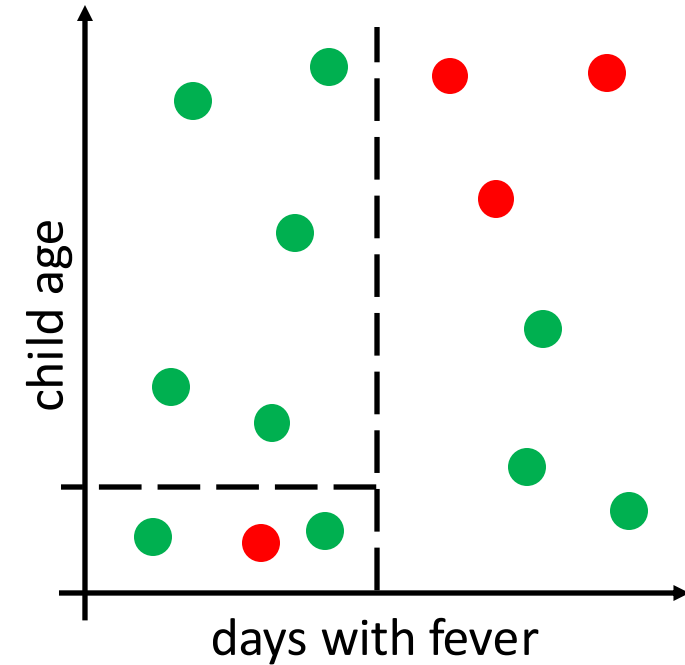
$$g = \frac{2}{14} - \frac{5}{14}$$



Post Pruning

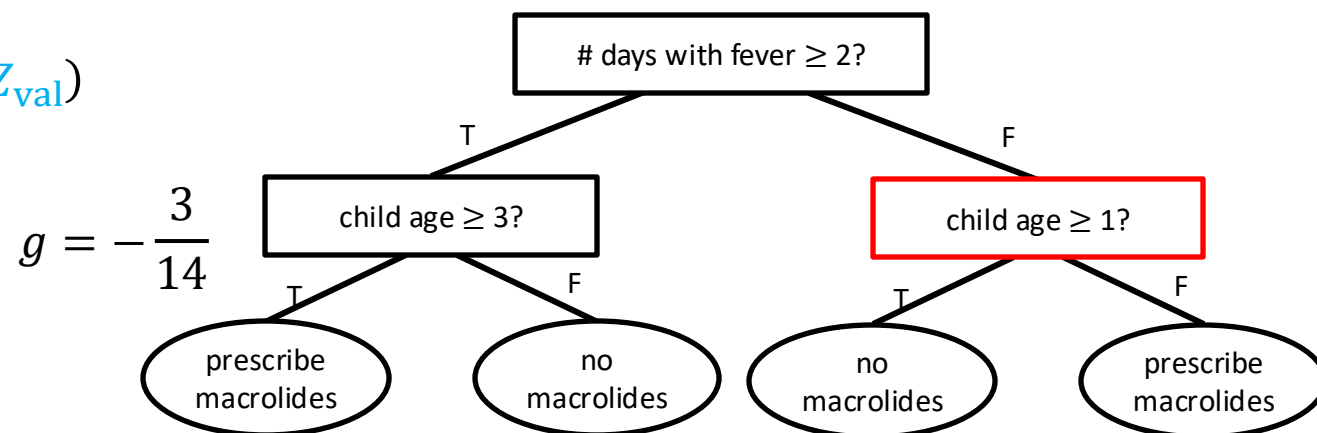
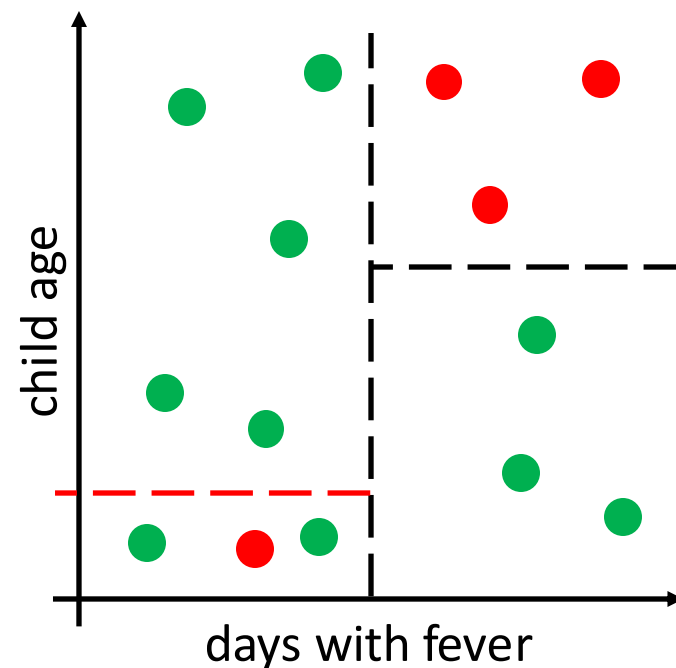
```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):
    for each internal node  $N$  of  $T$ :
         $T_N \leftarrow \text{Replace}(T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])))$ 
         $g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$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

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



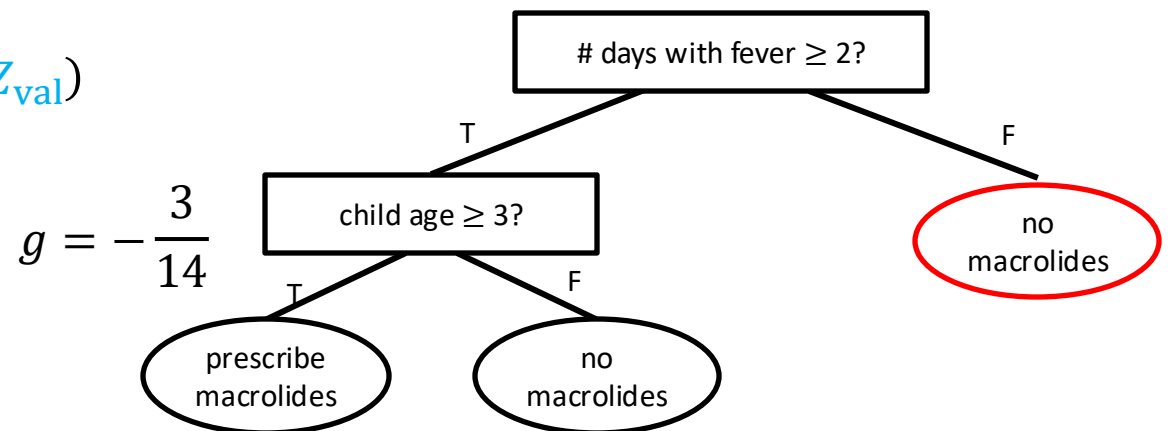
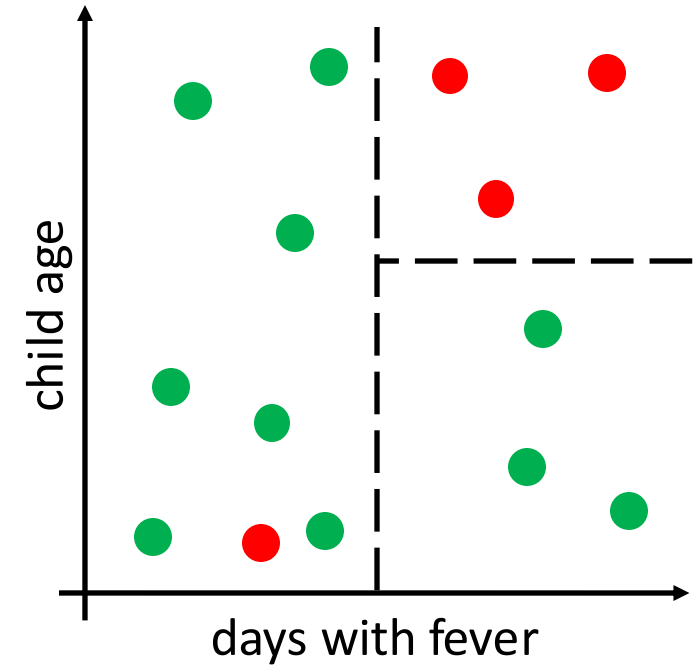
Post Pruning

```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):
    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 PostPruneTree( $T_N$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ )
        else:
            return  $T$ 
```



Post Pruning

```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):
    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 PostPruneTree( $T_N$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ )
        else:
            return  $T$ 
```



Post Pruning

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

```
    for each internal node  $N$  of  $T$ :
```

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

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

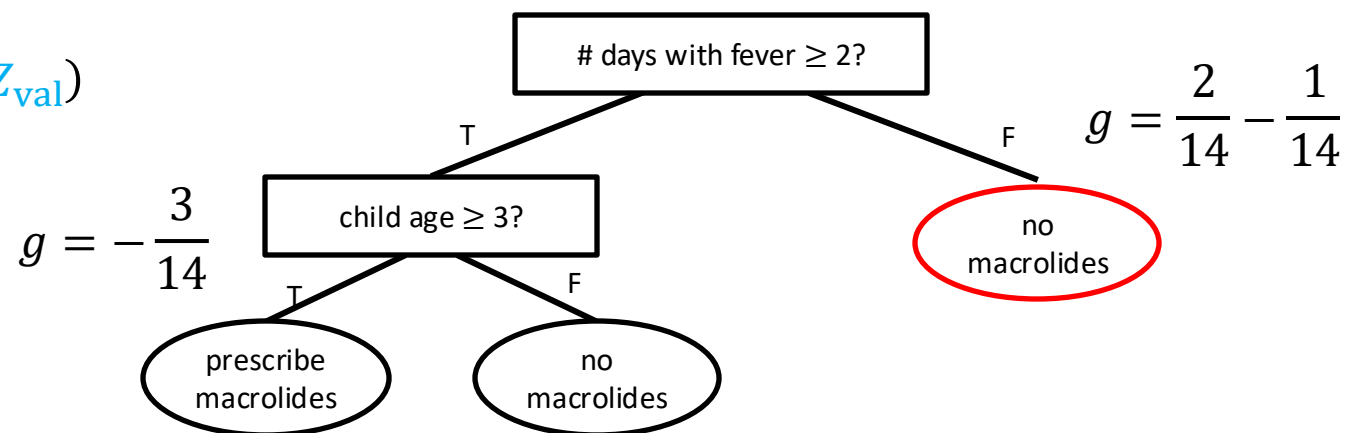
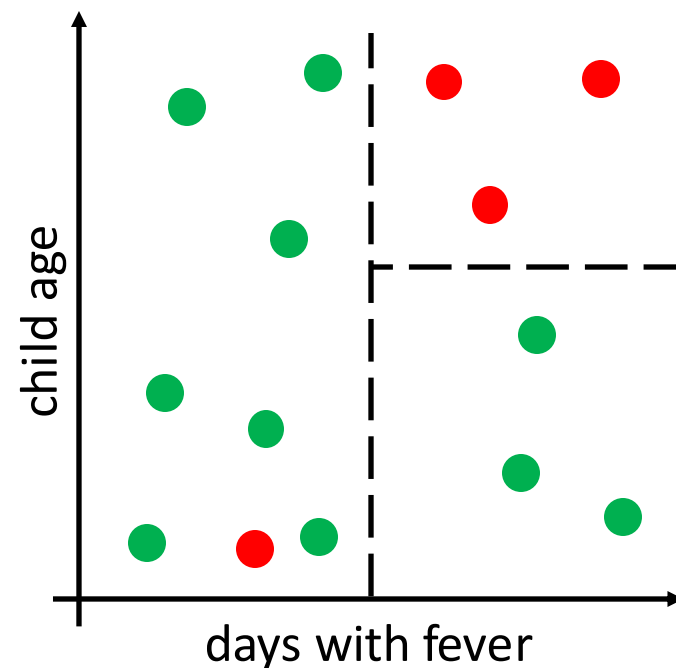
```
     $N_0 \leftarrow \arg \max_N g_N$ 
```

```
    if  $g_{N_0} > 0$ :
```

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

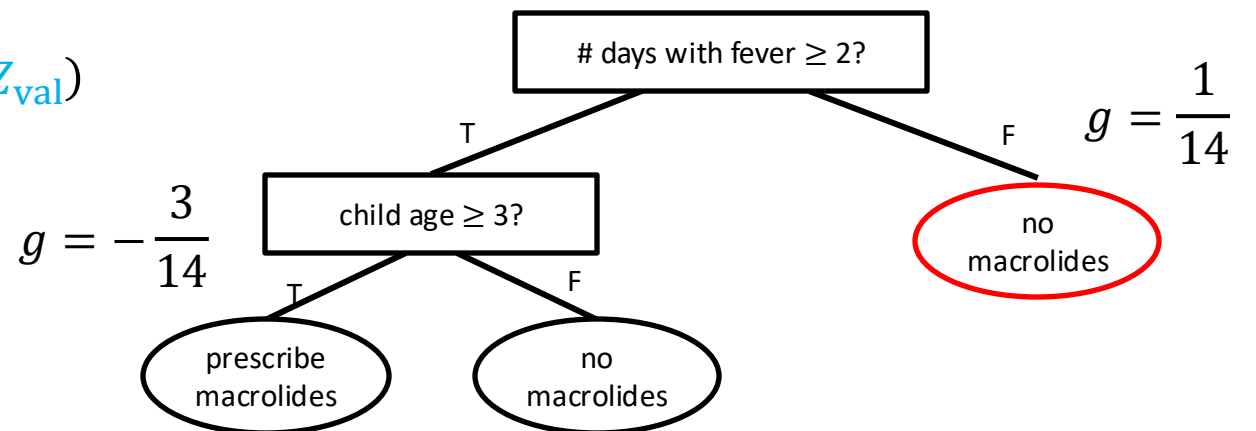
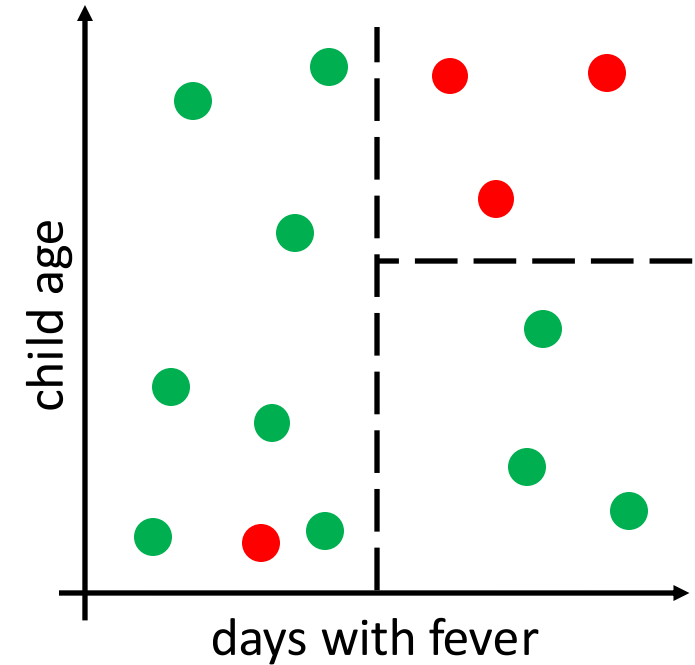
```
    else:
```

```
        return  $T$ 
```



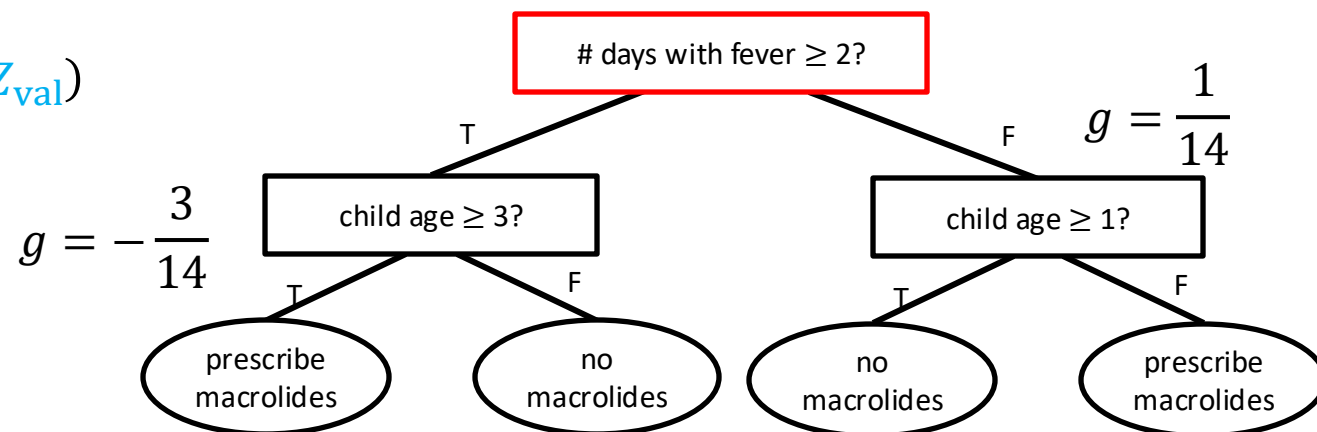
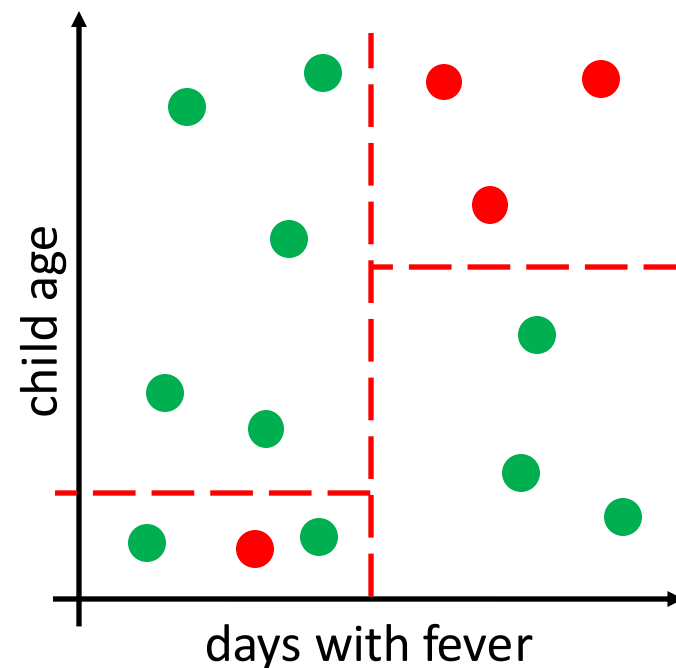
Post Pruning

```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):
    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 PostPruneTree( $T_{N_0}$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ )
    else:
        return  $T$ 
```



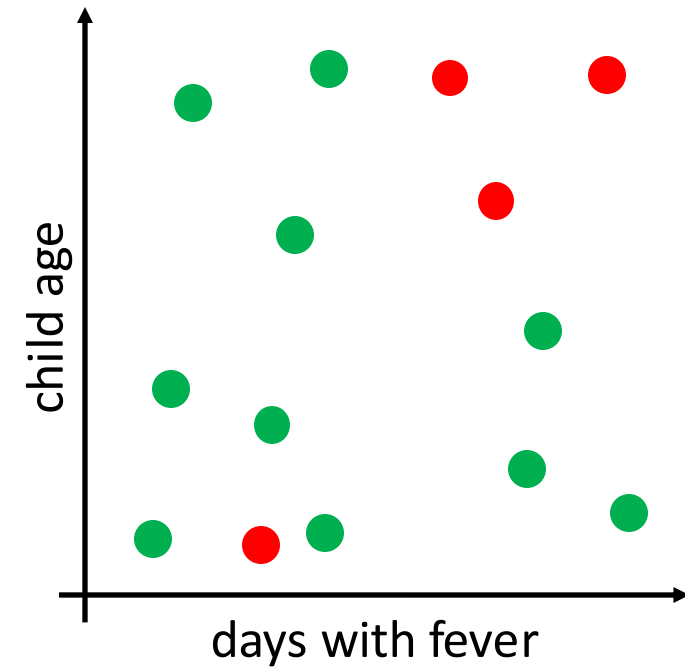
Post Pruning

```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):
    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 PostPruneTree( $T_{N_0}$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ )
        else:
            return  $T$ 
```



Post Pruning

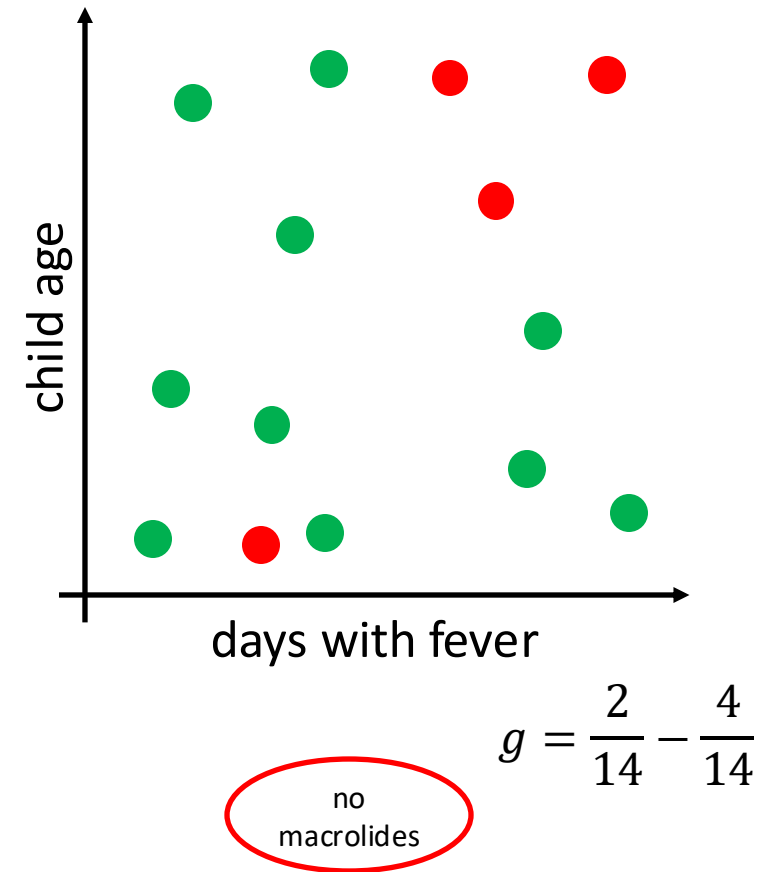
```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):  
    for each internal node  $N$  of  $T$ :  
         $T_N \leftarrow \text{Replace} \left( T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])) \right)$   
         $g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N,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



no
macrolides

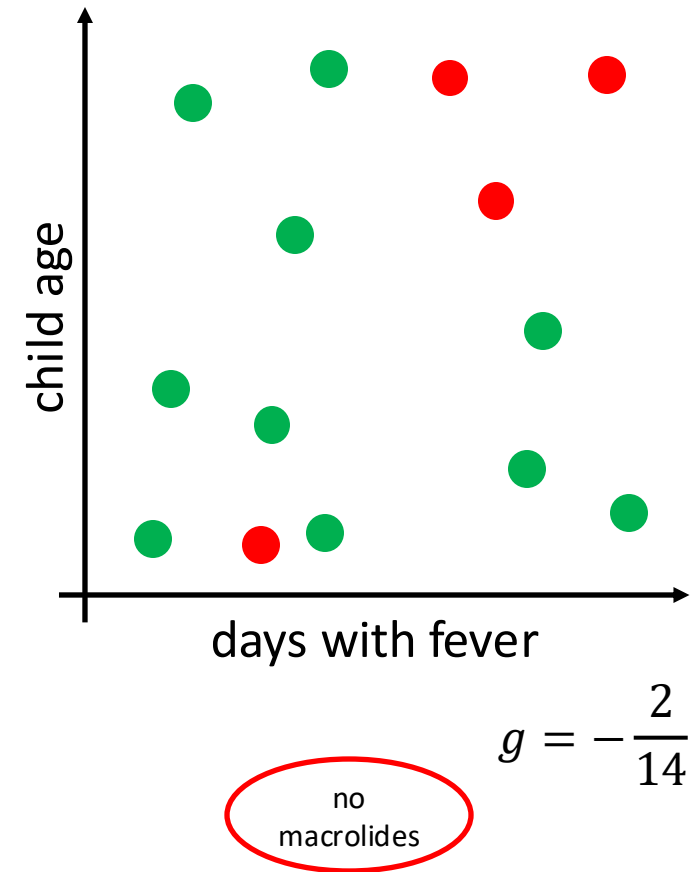
Post Pruning

```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):  
    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 PostPruneTree( $T_{N_0}$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ )  
    else:  
        return  $T$ 
```



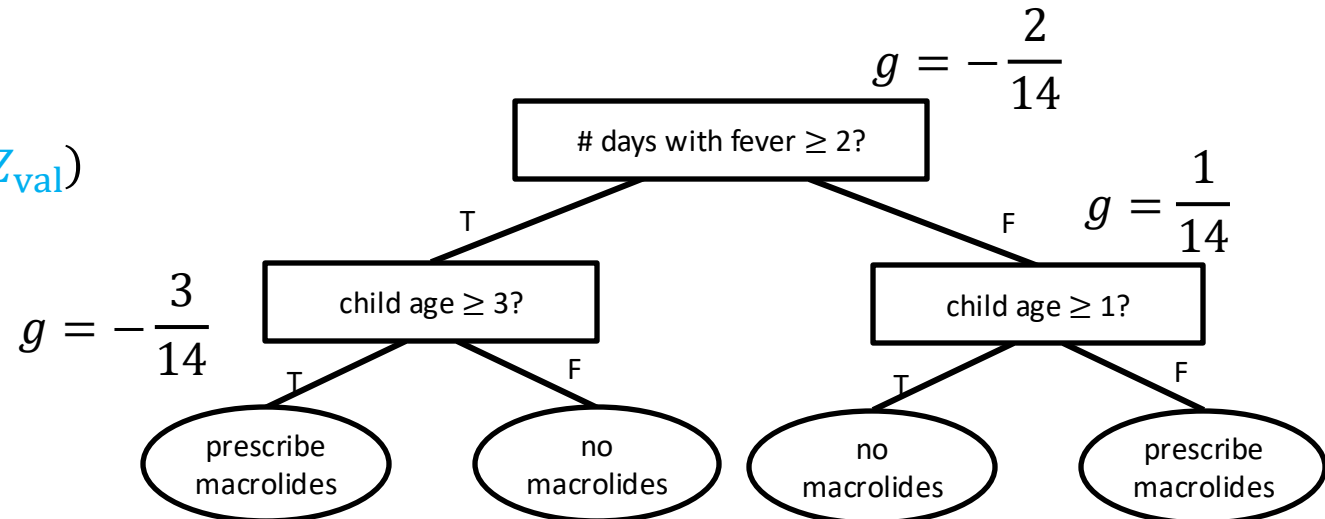
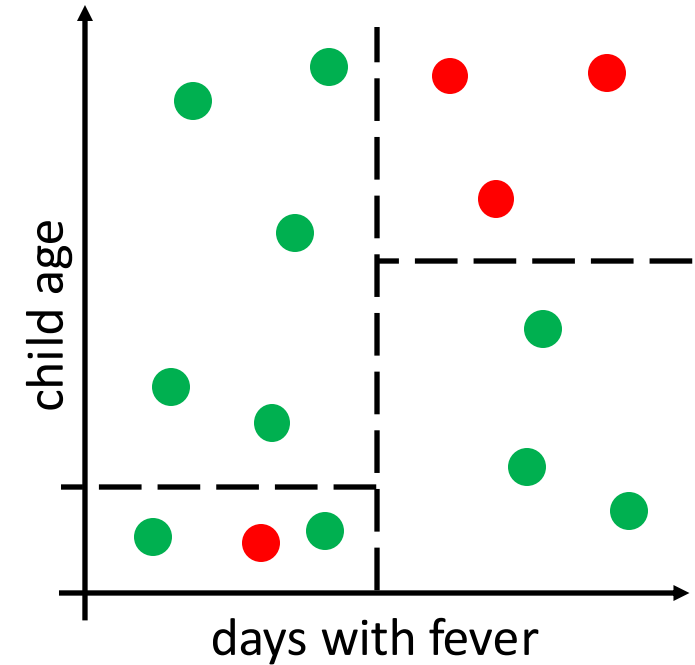
Post Pruning

```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):  
    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 PostPruneTree( $T_{N_0}$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ )  
    else:  
        return  $T$ 
```



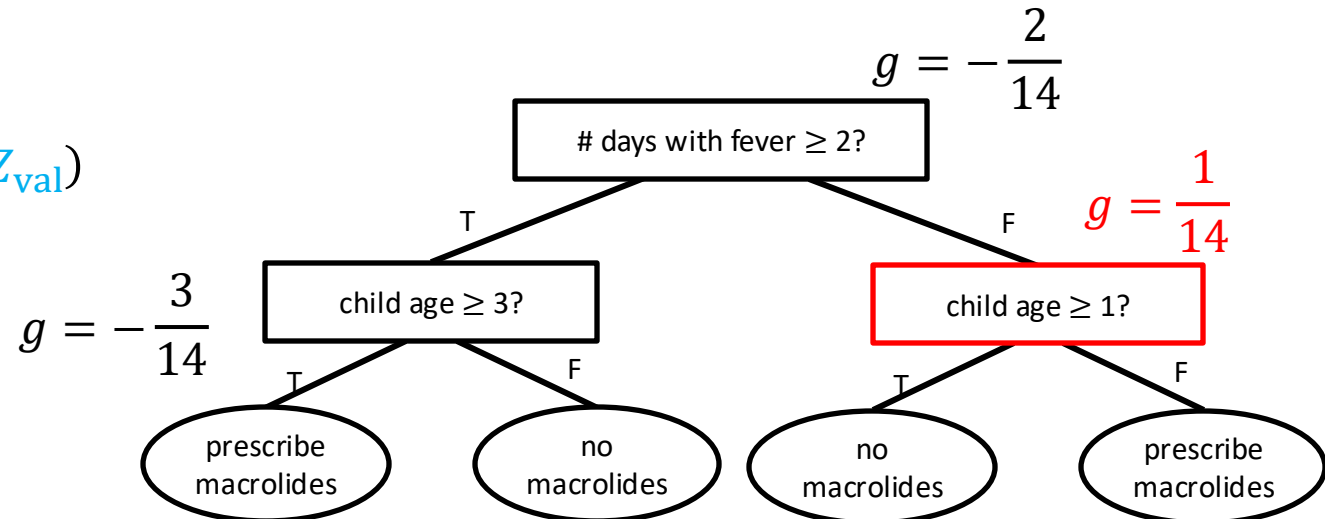
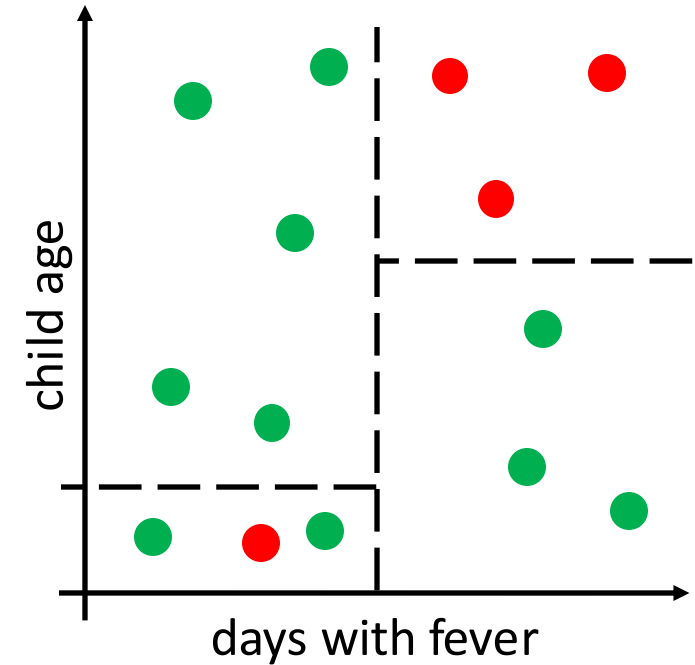
Post Pruning

```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):
    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 PostPruneTree( $T_N$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ )
        else:
            return  $T$ 
```



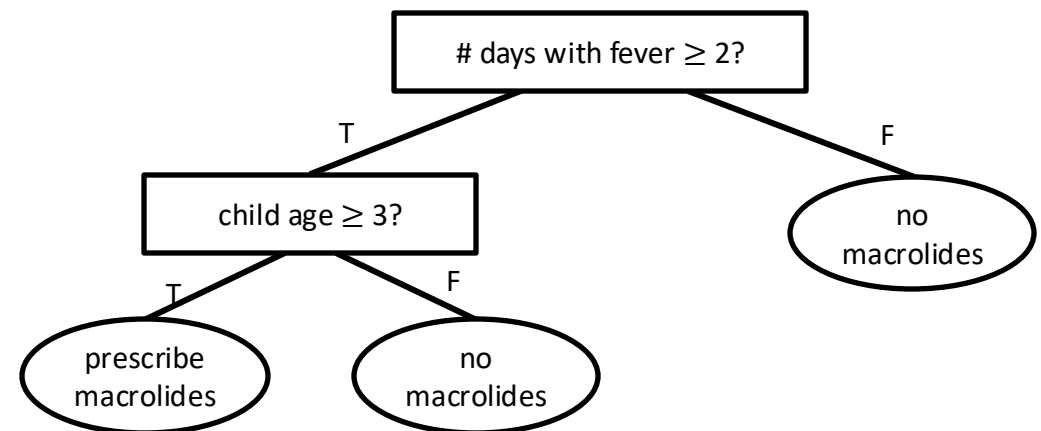
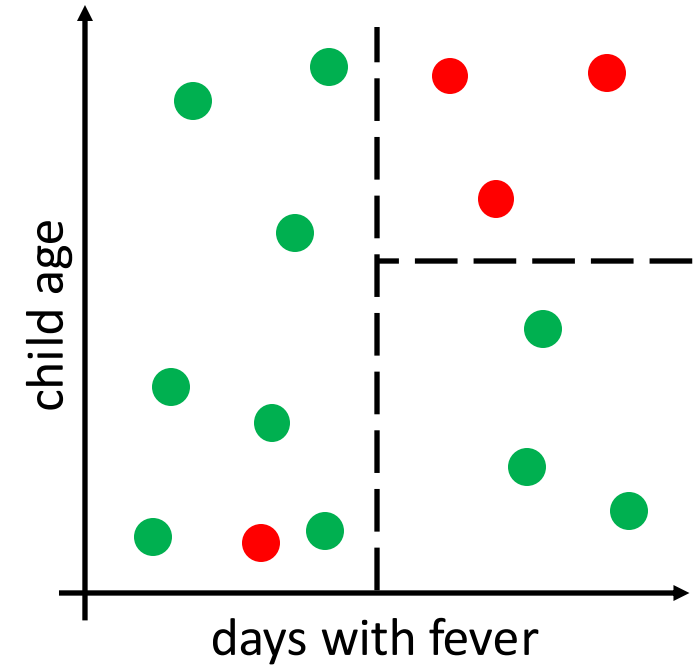
Post Pruning

```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):
    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 PostPruneTree( $T_N$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ )
        else:
            return  $T$ 
```



Post Pruning

```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):  
    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 PostPruneTree( $T_{N_0}$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ )  
    else:  
        return  $T$ 
```



Post Pruning

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

```
    for each internal node  $N$  of  $T$ :
```

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

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

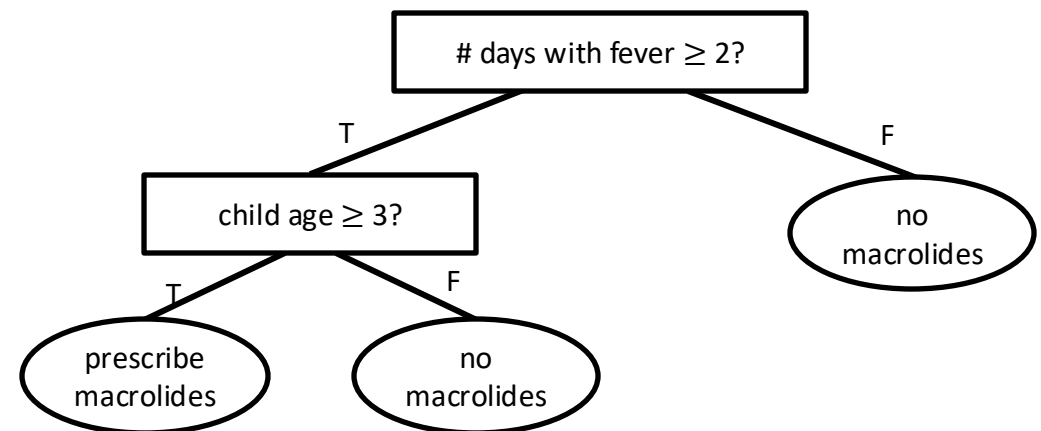
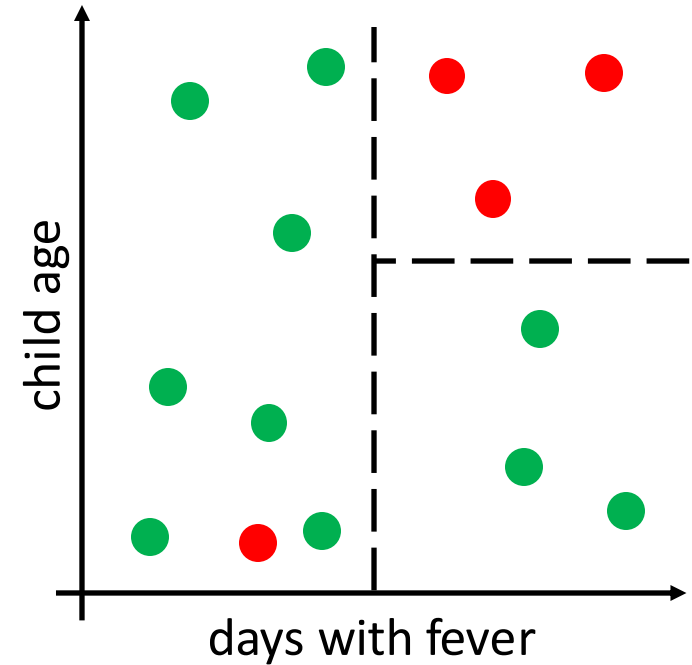
```
     $N_0 \leftarrow \arg \max_N g_N$ 
```

```
    if  $g_{N_0} > 0$ :
```

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

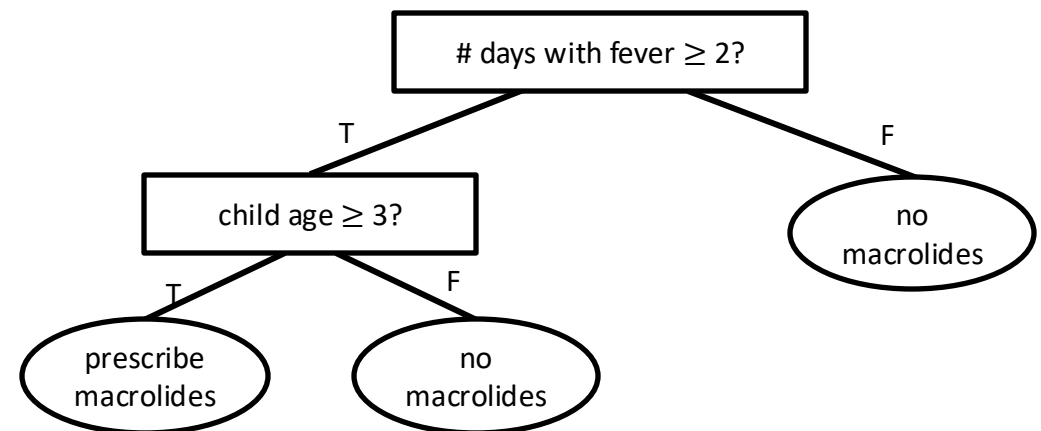
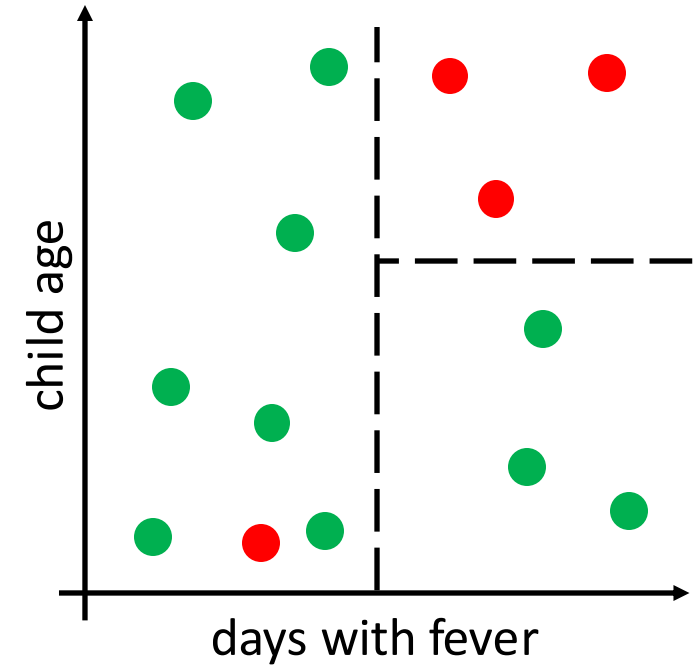
```
    else:
```

```
        return  $T$ 
```

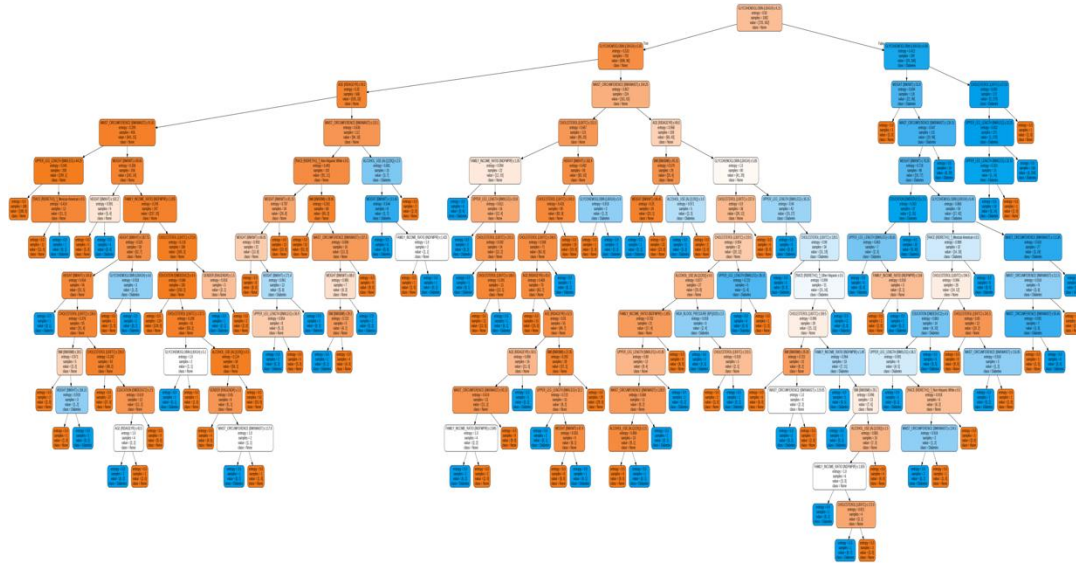


Post Pruning

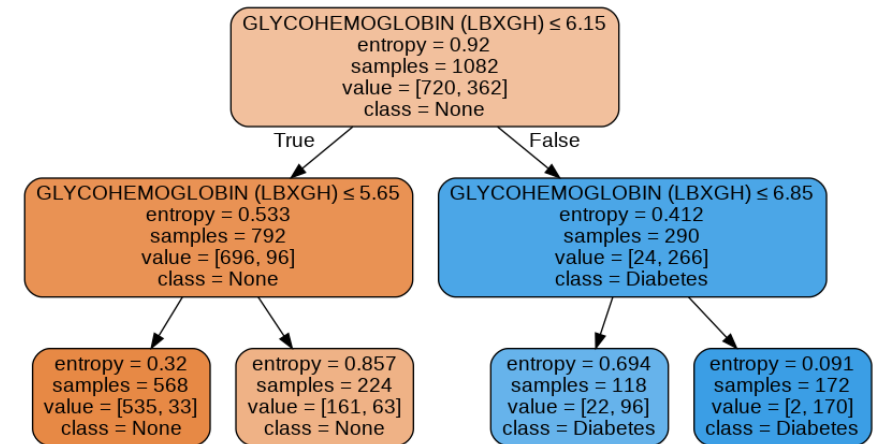
```
def PostPruneTree( $T$ ,  $Z_{\text{train}}$ ,  $Z_{\text{val}}$ ):  
    for each internal node  $N$  of  $T$ :  
         $T_N \leftarrow \text{Replace}(T, N, \text{LeafNode}(\text{Mode}(Z_{\text{train}}[N])))$   
         $g_N \leftarrow \text{Loss}(T, Z_{\text{val}}) - \text{Loss}(T_N, Z_{\text{val}})$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



Diabetes Prediction

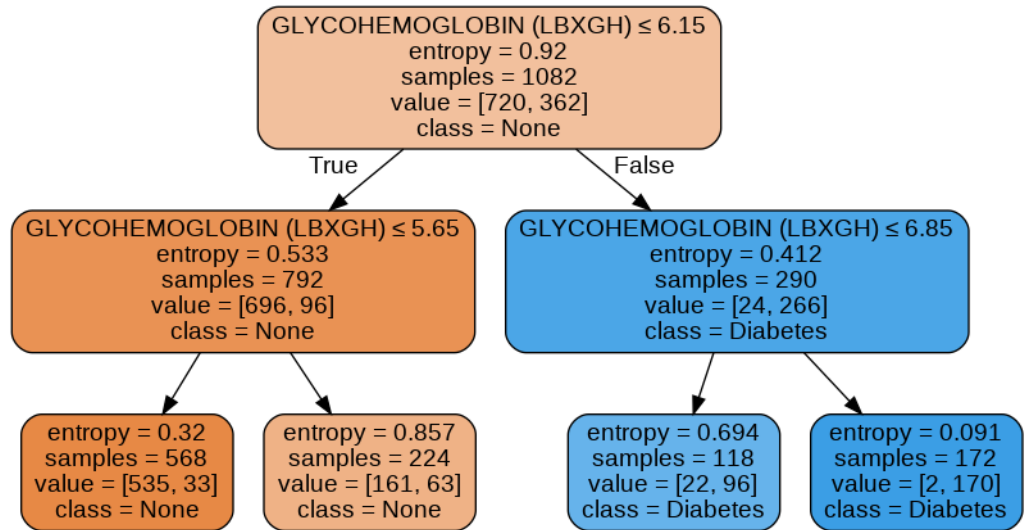


Training accuracy: 100%
Test accuracy: 82.8%



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 Tree Learning Algorithms

- **Older algorithms:** ID3, C4.5, ...
- Most popular current algorithm is CART
 - “Classification And Regression Trees”
 - (Mostly) implemented in SciKit Learn

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 (next topic)
 - Learn many decision trees to reduce variance
 - Maintains ability to learn complex nonlinear decision boundaries
 - Uninterpretable