Upcoming Deadlines

- **Project Team Formation**
  - If you have not yet formed a team, please email us
  - We will randomly assign teams if they are not formed by tonight
- **HW 2 due tonight at 8pm**
- Quiz 2 due tomorrow night (9/28) at 8pm

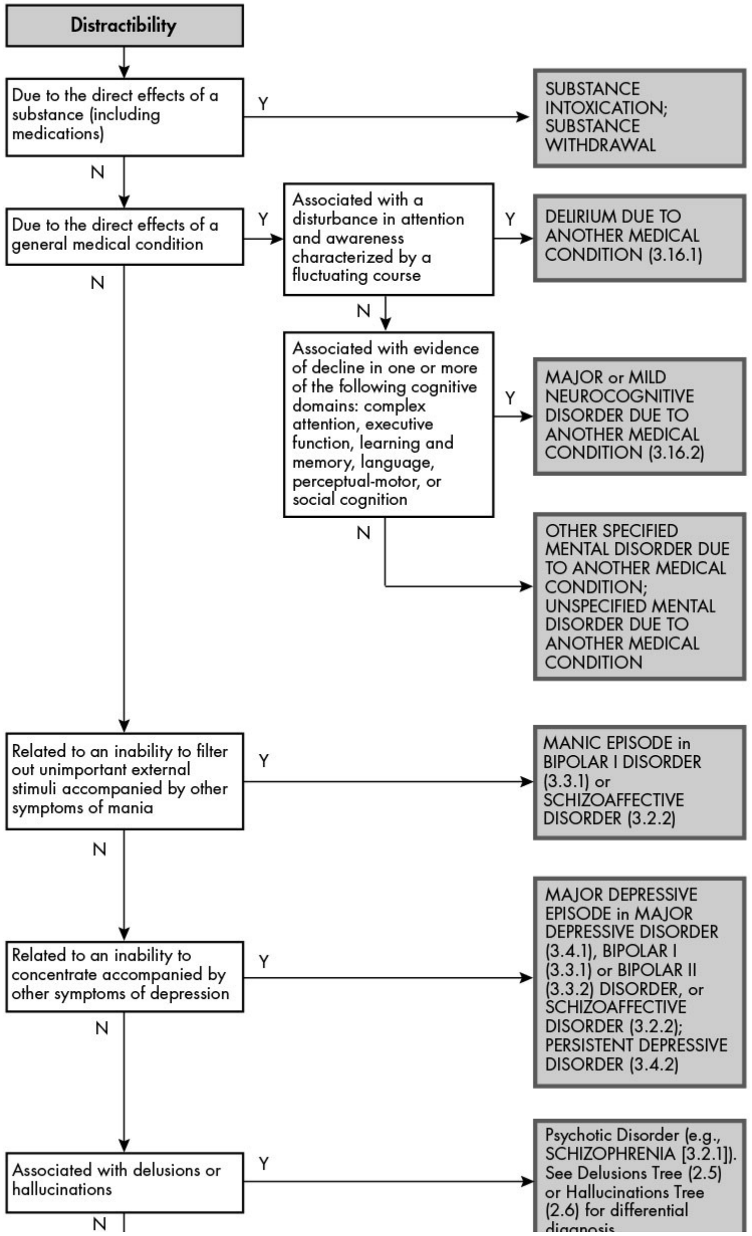
Lecture 8: Decision Trees

CIS 4190/5190

Fall 2023

Decision Trees

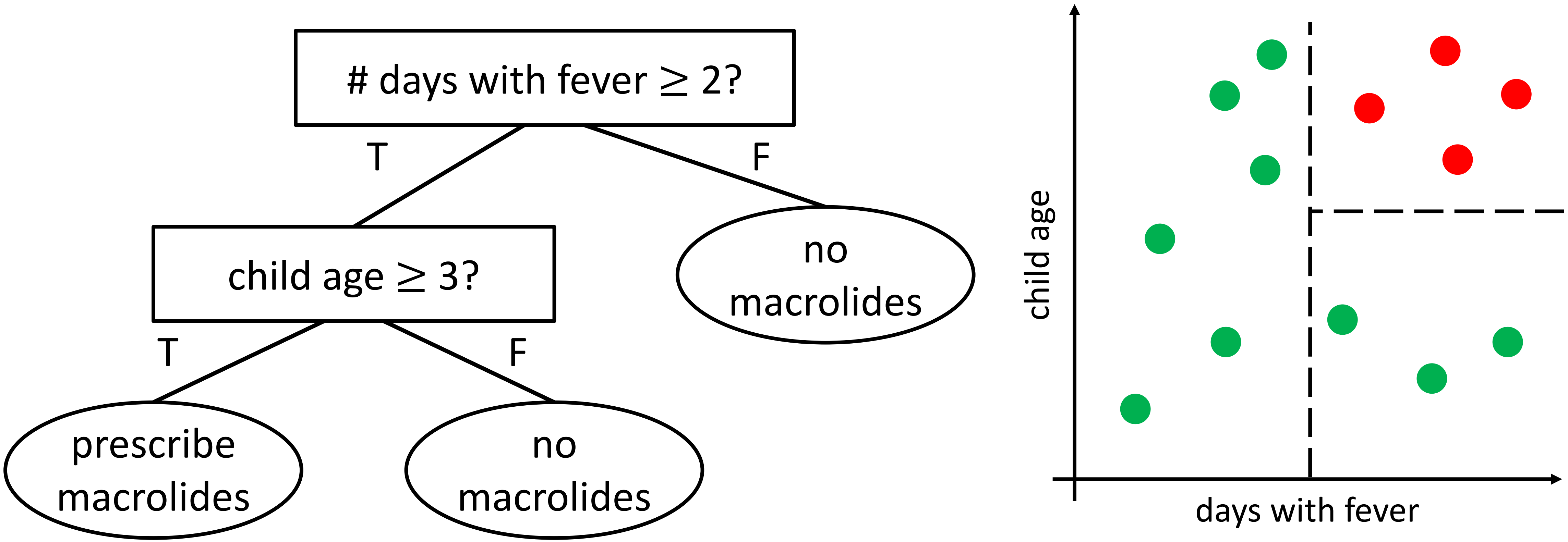
- A kind of flowchart based on tests
  - Commonly used in medicine
- “Explainable”, easy to mentally evaluate



APA DSM Library

Visualizing the Model Family

- Axis-aligned decision boundaries



Learning Algorithm

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

def LearnTree(Z):

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$\left(Z\left[x_j \geq t\right]\right)$

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

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

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:

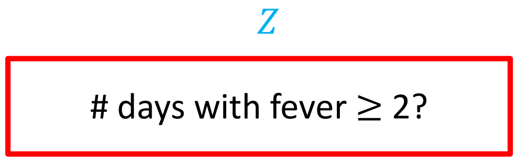
        return LeafNode(y)

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

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

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

    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:

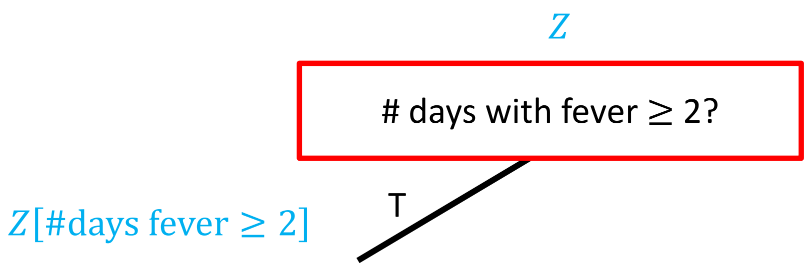
    return LeafNode(y)

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

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

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

  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:

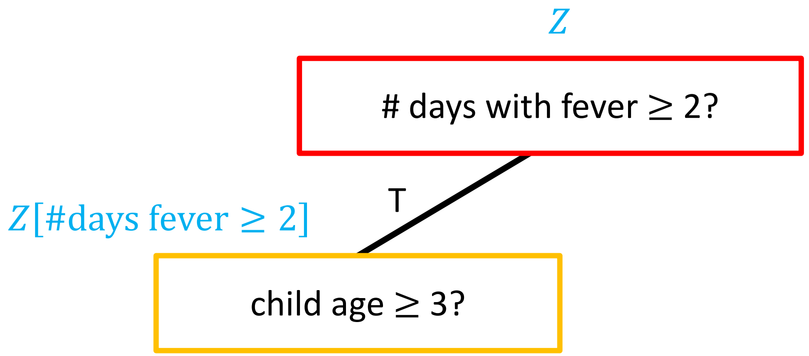
        return LeafNode(y)

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

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

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

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



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:

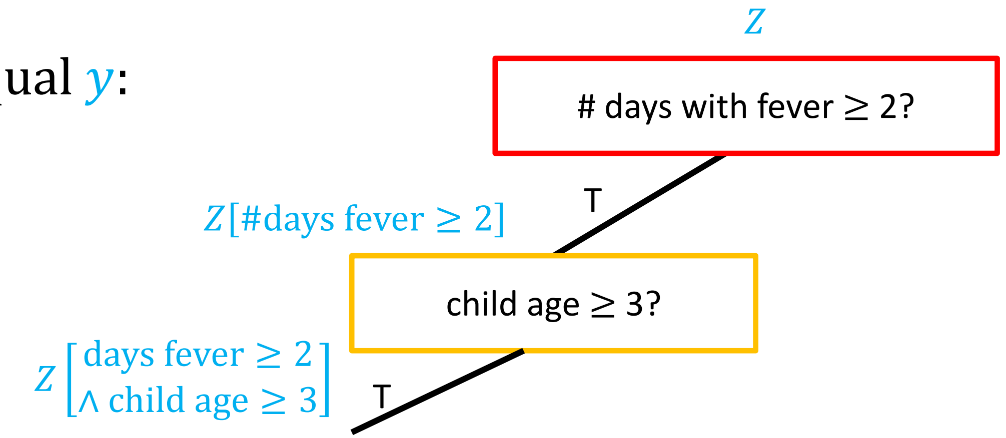
        return LeafNode(y)

    $(j, t) \leftarrow$ 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:

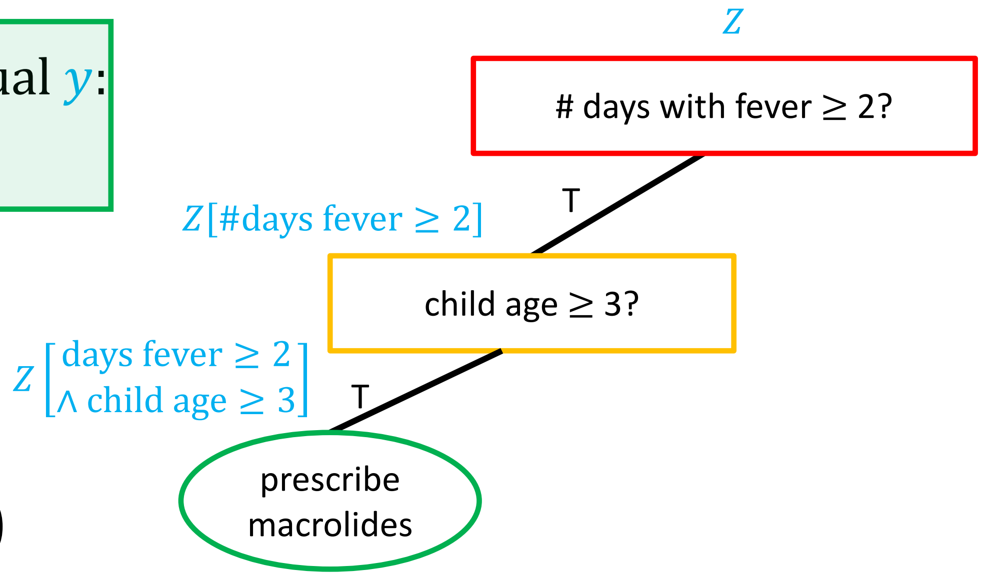
    return LeafNode(y)

  $(j, t) \leftarrow$ 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:

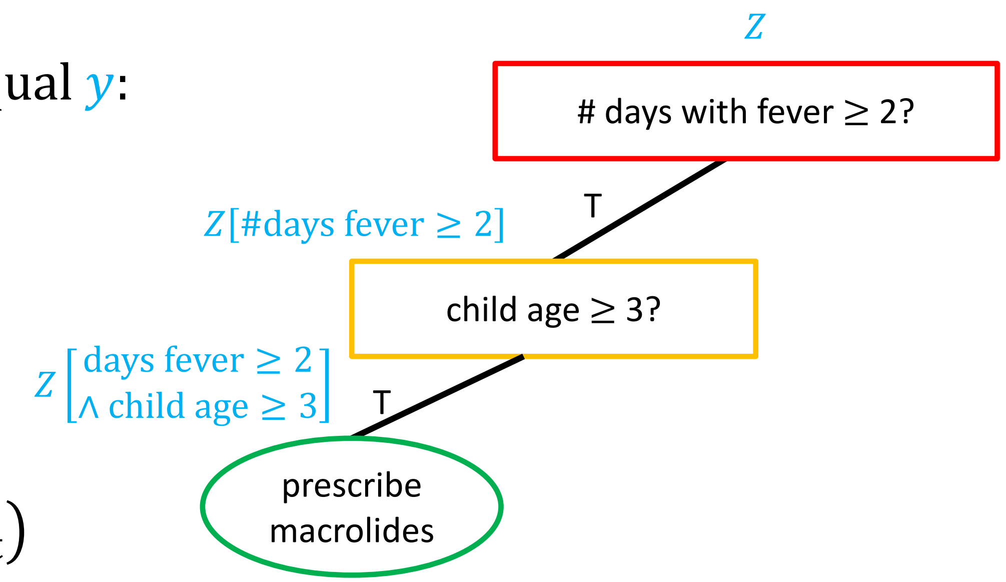
        return LeafNode(y)

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

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

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

    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:

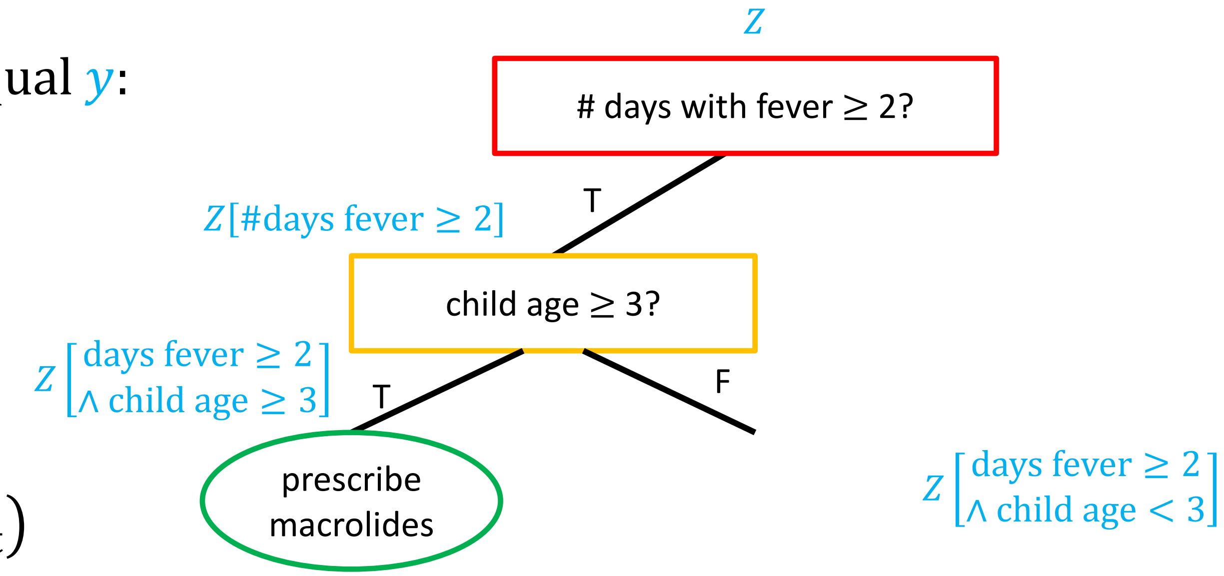
  return LeafNode(y)

 $(j, t) \leftarrow$ 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:

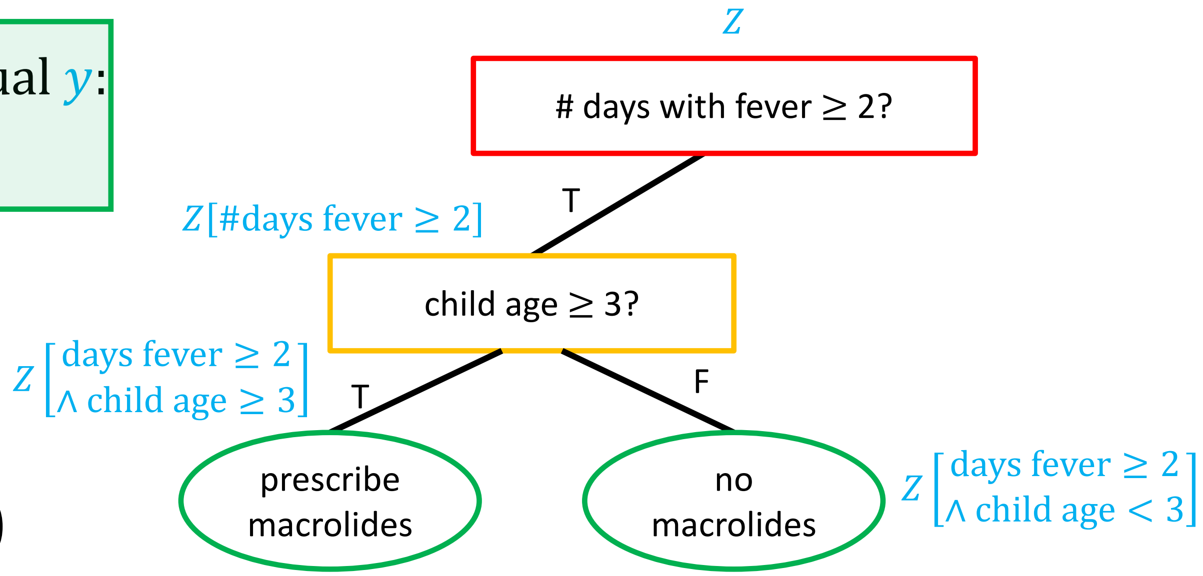
        return LeafNode(y)

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

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

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

    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:

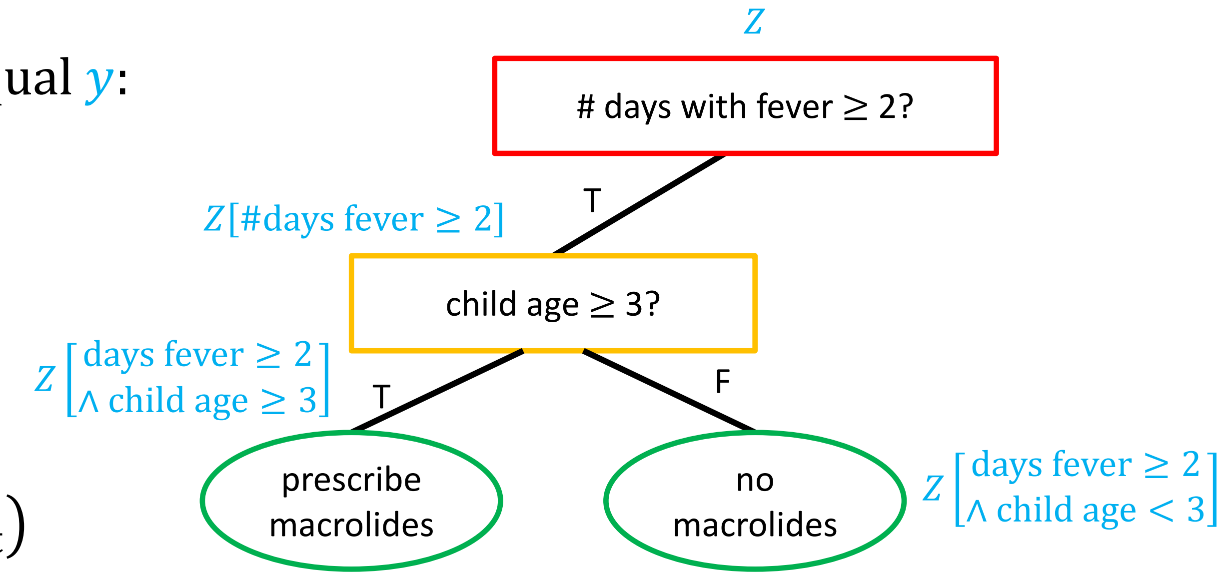
    return LeafNode(y)

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

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

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

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



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:

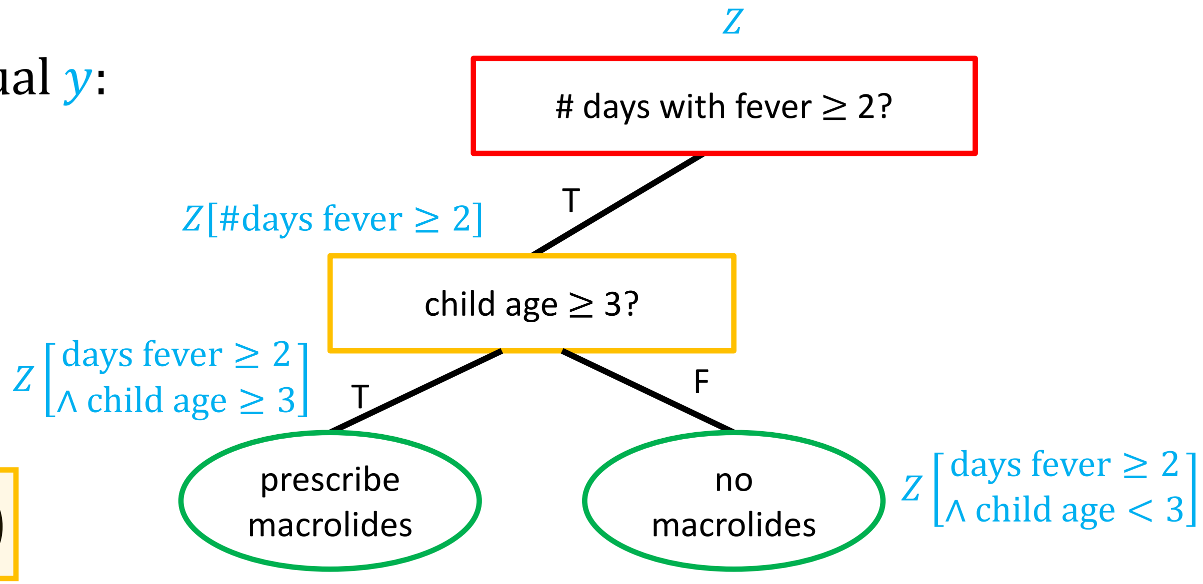
    return LeafNode(y)

  $(j,t) \leftarrow$ 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:

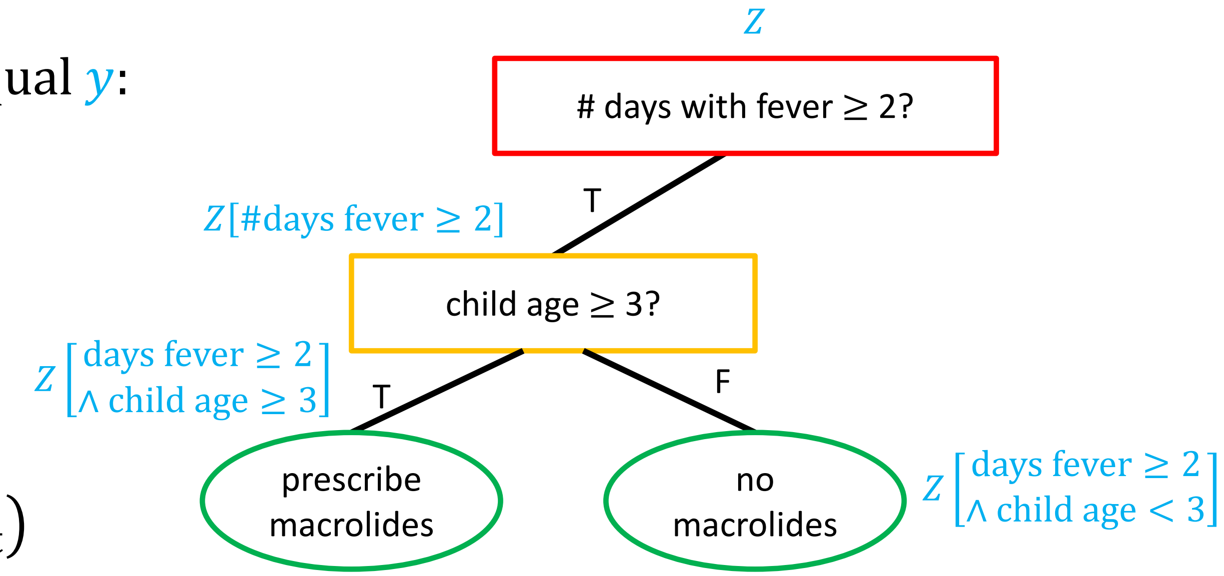
    return LeafNode(y)

  $(j, t) \leftarrow$ 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:

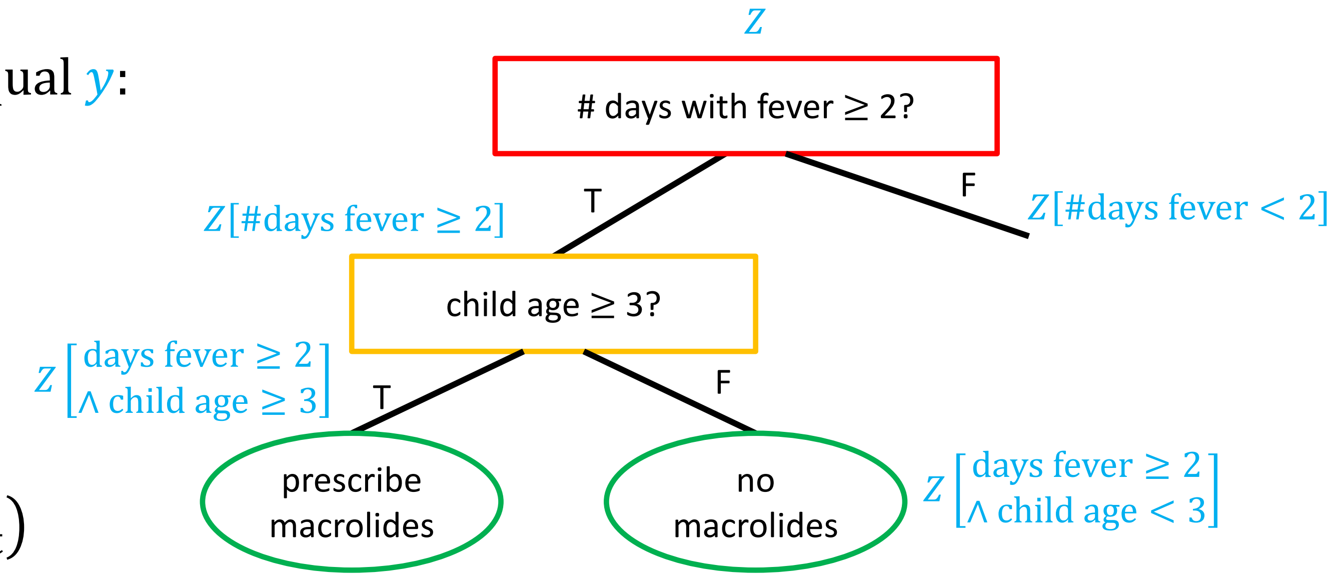
    return LeafNode(y)

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

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

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

  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:

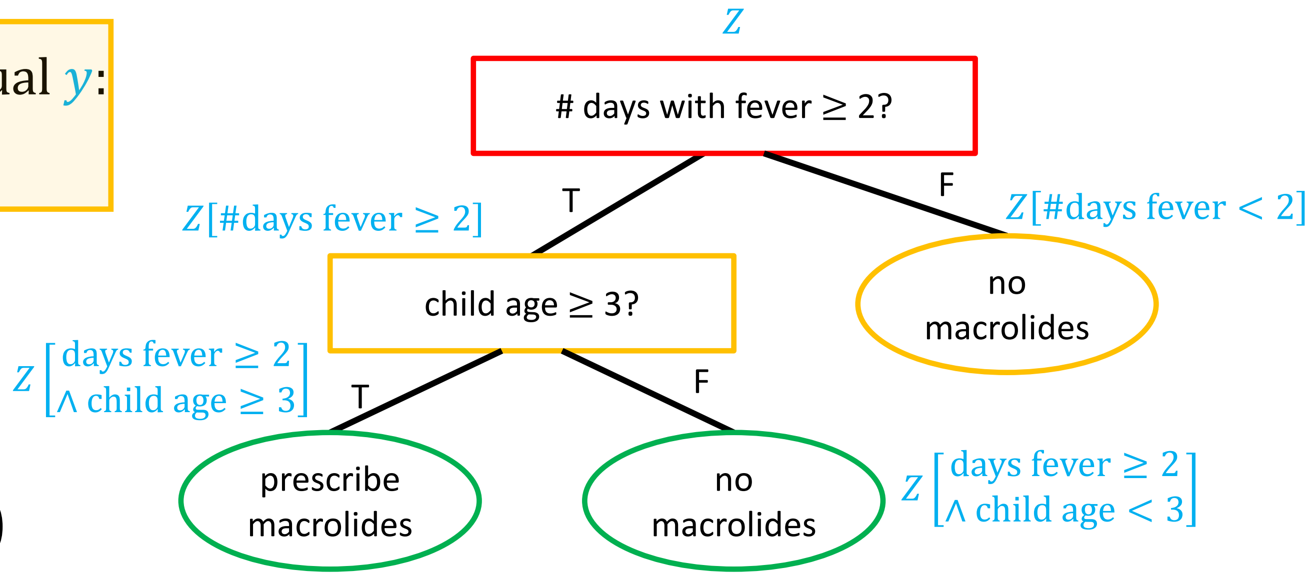
        return LeafNode(y)

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

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

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

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



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:

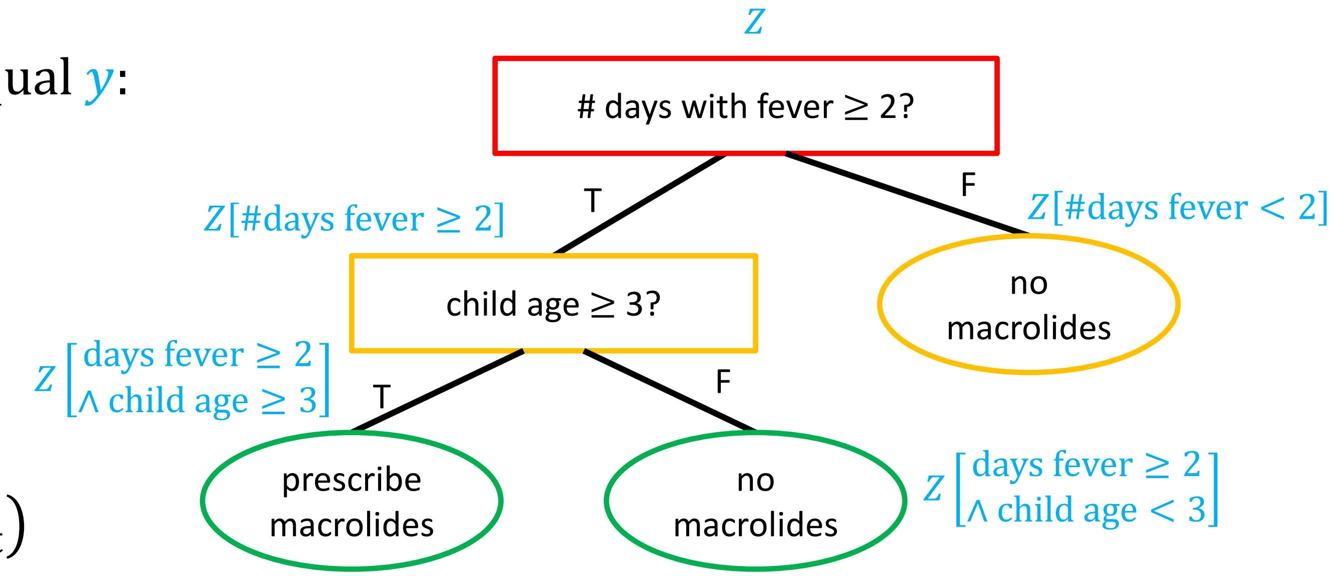
        return LeafNode(y)

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

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

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

    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:

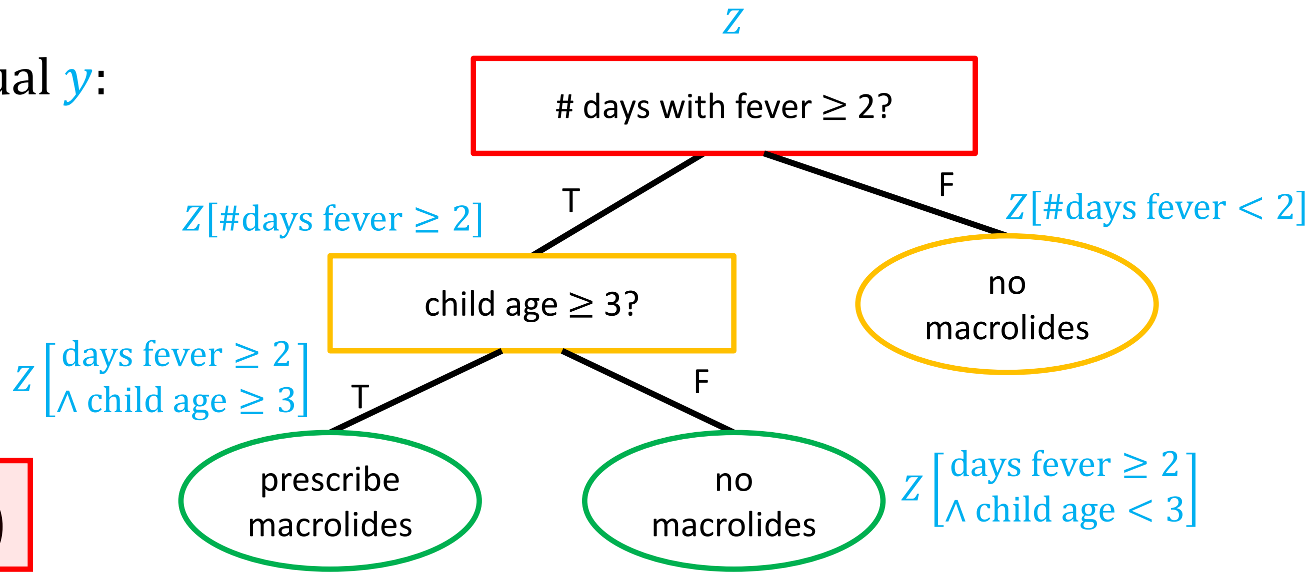
    return LeafNode(y)

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

  $T_{\text{left}} \leftarrow$ 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:

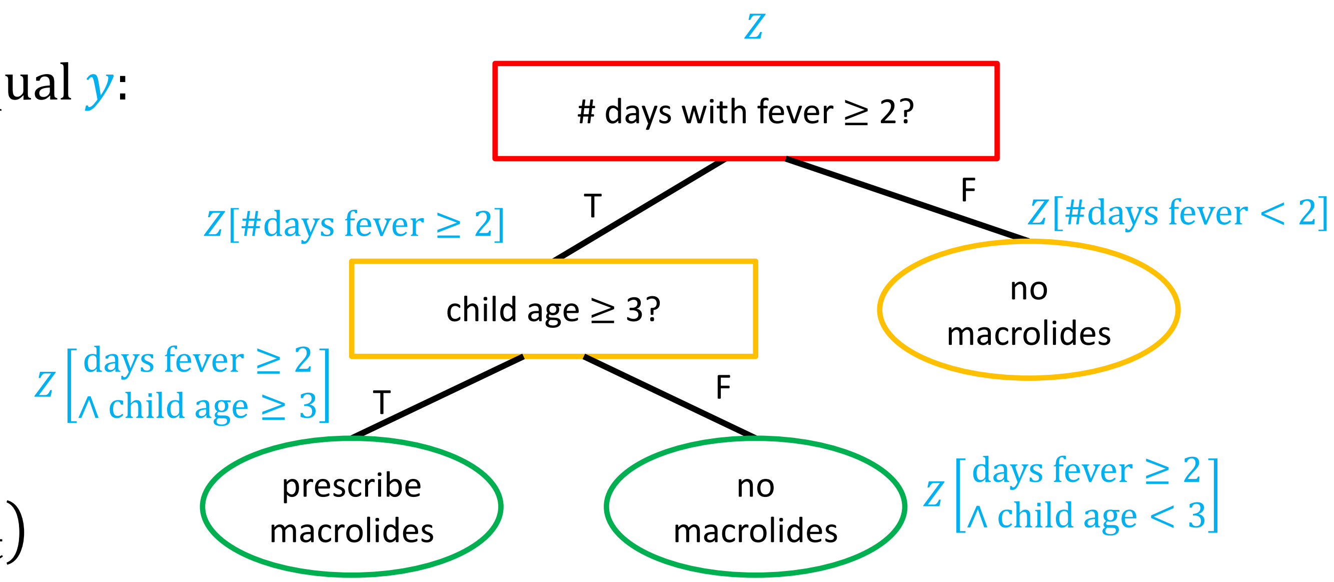
        return LeafNode(y)

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

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

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

    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)

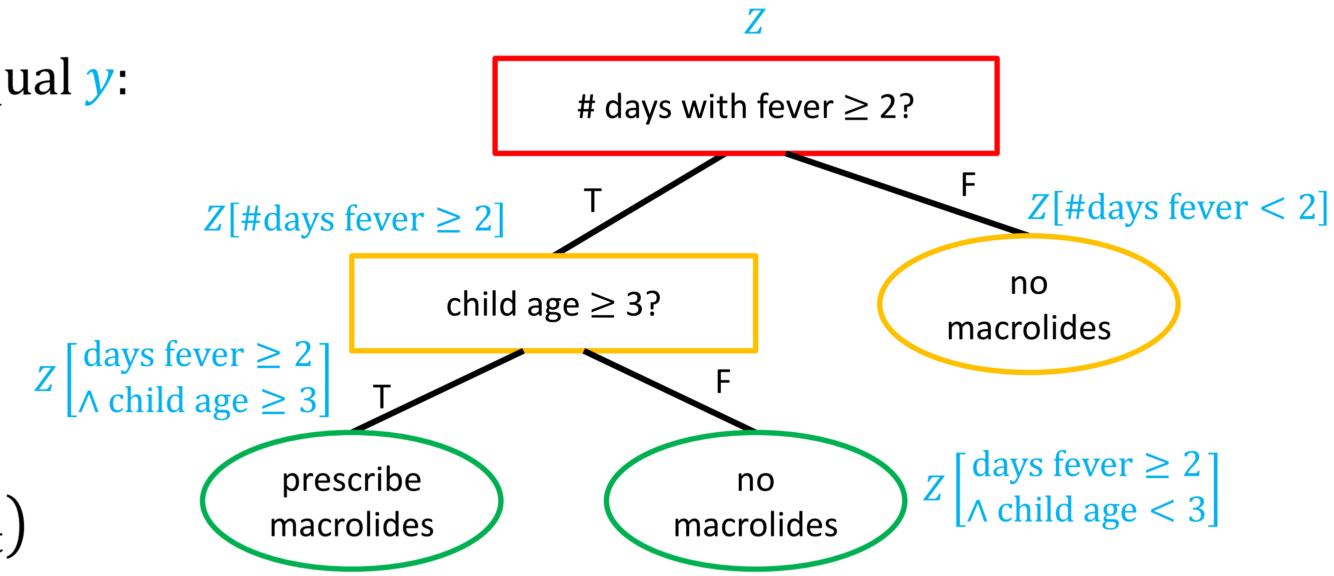
  $(j,t) \leftarrow$ BestSplit(Z)

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

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

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

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 → more parameters → 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**

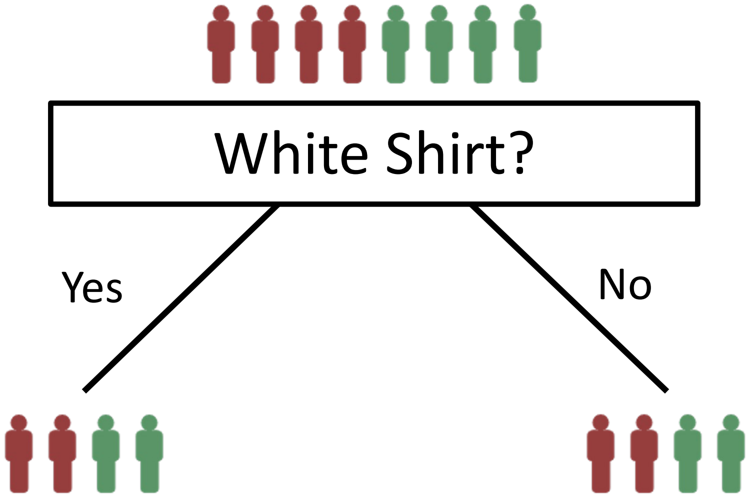
Search Space of Candidate Splits

- 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)

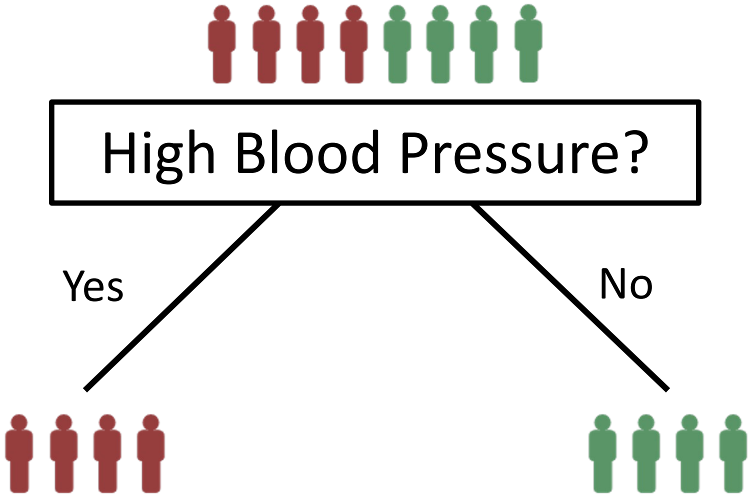
- **Search space**
  - All features $j \in \{1, \dots, d\}$
  - All values $t \in \{ x_{ij} \mid x_{ij} \in Z \}$

- The choices of t induce all possible splits of Z

Choosing Splits

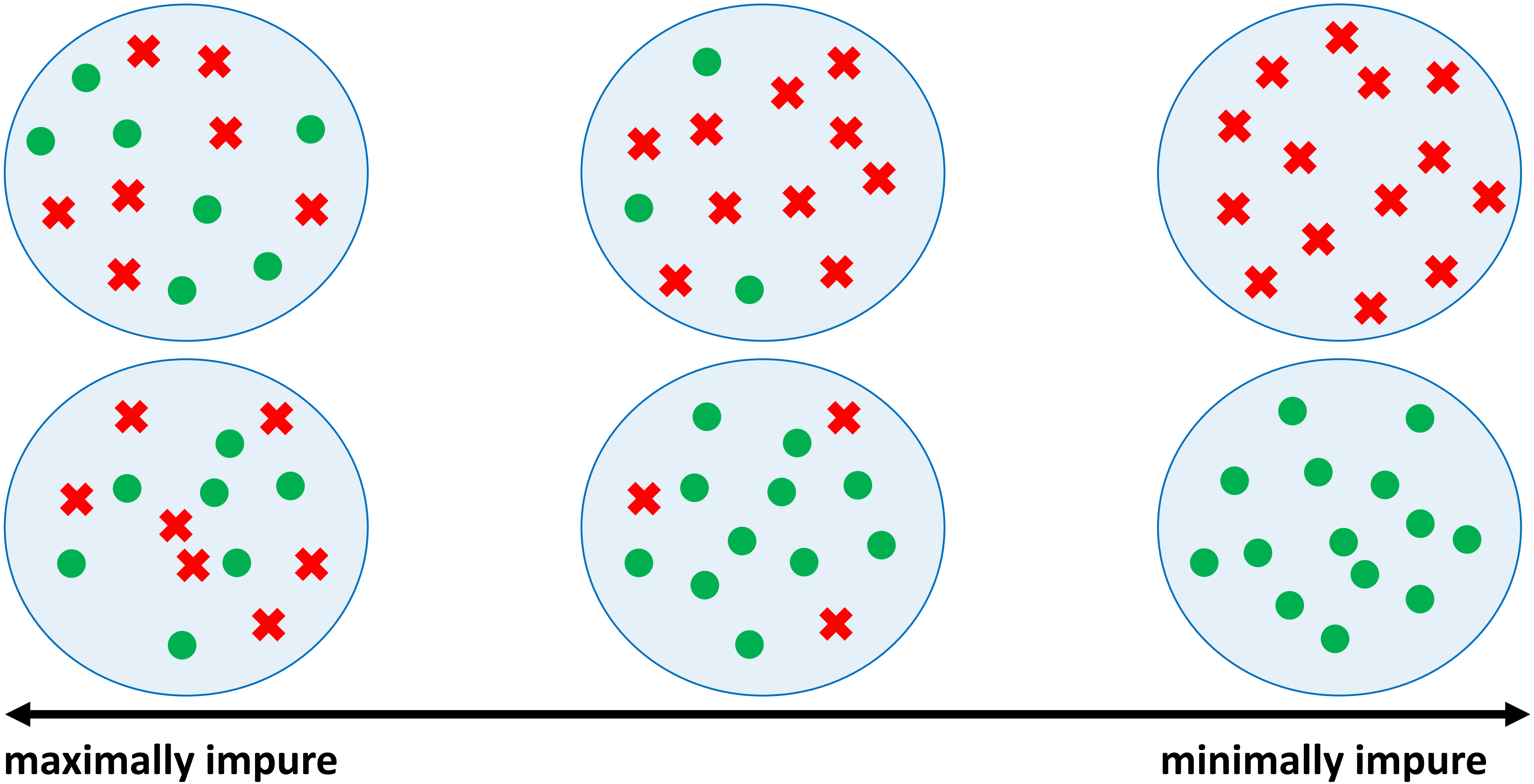


more impure



less impure

Choosing Splits



Quantifying Impurity

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

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

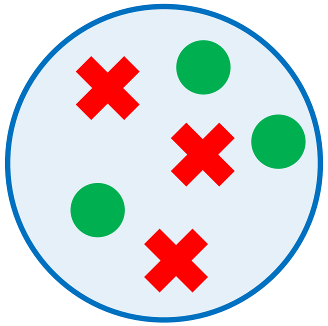
- Given a dataset Z, we define the random variable Y by

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

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

Quantifying Impurity

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



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

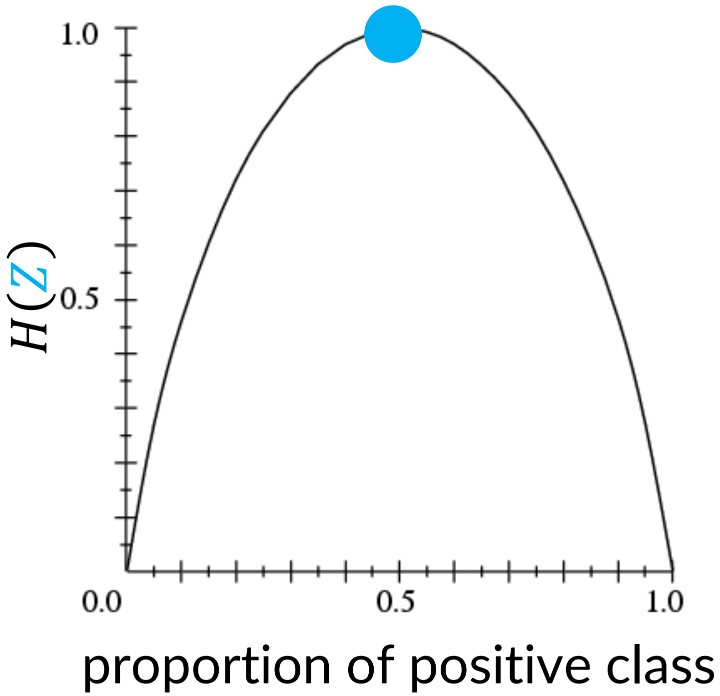
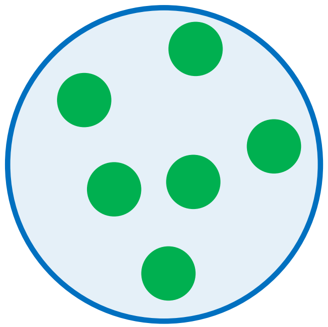


Image: Tom Mitchell

Quantifying Impurity

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



$$H(Z) = -1 \log 1 = 0$$

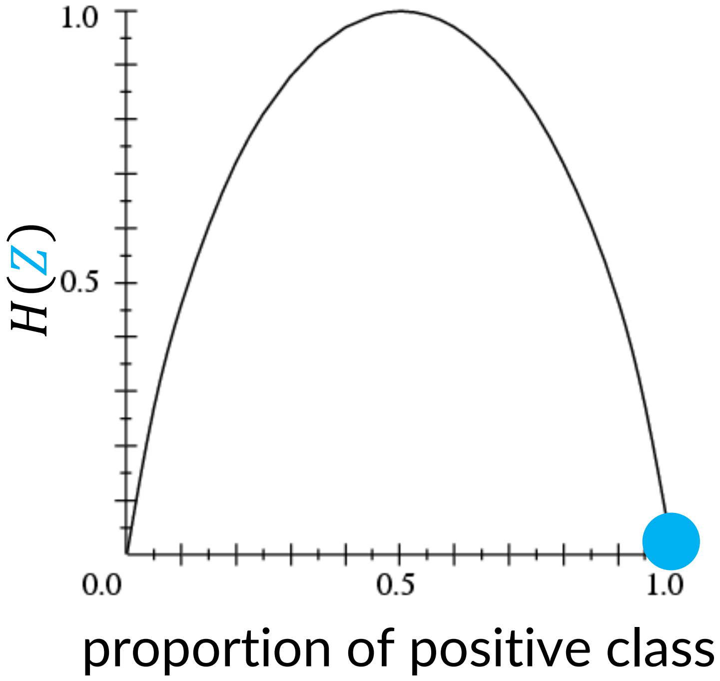


Image: Tom Mitchell

Choosing Splits

- Choose splits that most **reduce** impurity (i.e., entropy)

- Quantified by the **information gain:**

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

- 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

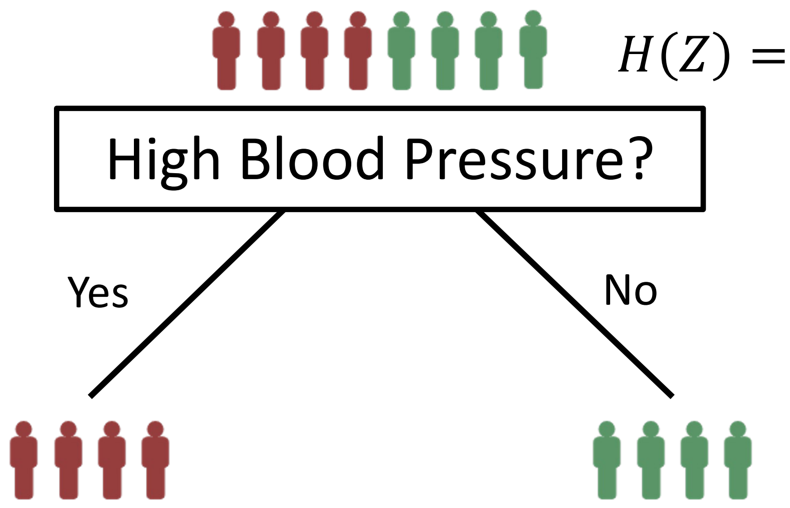
- Information gain is higher if split reduces impurity by more

Choosing Splits

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

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

$$H(Z) = -\frac{4}{8}\log\frac{4}{8} - \frac{4}{8}\log\frac{4}{8} = 1$$



$$H(Z) = -\frac{4}{4}\log\frac{4}{4} - \frac{0}{4}\log\frac{0}{4} = 0$$

$$H(Z) = -\frac{0}{4}\log\frac{0}{4} - \frac{4}{4}\log\frac{4}{4} = 0$$

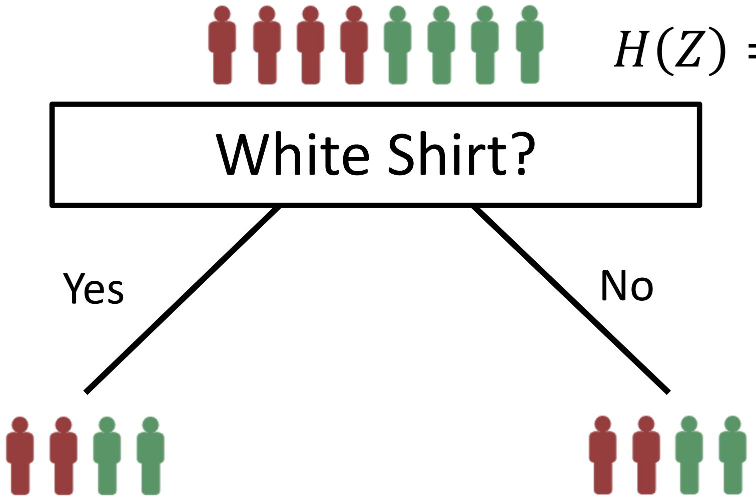
$$\mathrm{IG} = 1 - 0\cdot\frac{1}{2} - 0\cdot\frac{1}{2} = 1$$

Choosing Splits

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

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

$$H(Z) = -\frac{4}{8}\log\frac{4}{8} - \frac{4}{8}\log\frac{4}{8} = 1$$

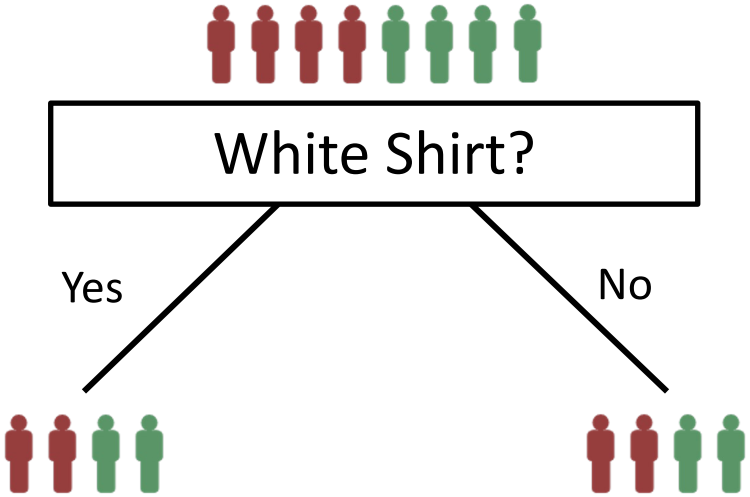


$$H(Z) = -\frac{2}{4}\log\frac{2}{4} - \frac{2}{4}\log\frac{2}{4} = 1$$

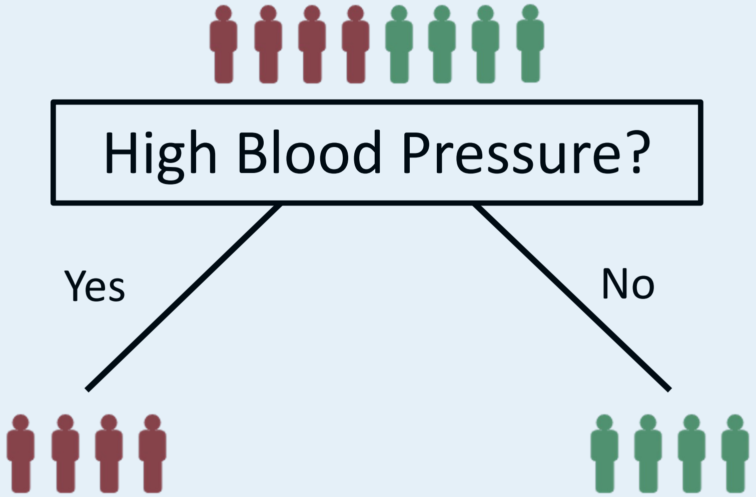
$$H(Z) = -\frac{2}{4}\log\frac{2}{4} - \frac{2}{4}\log\frac{2}{4} = 1$$

$$\mathrm{IG} = 1 - 1 \cdot \frac{1}{2} - 1 \cdot \frac{1}{2} = 0$$

Choosing Splits



more impure
$\mathrm{IG} = 0$



less impure
$\mathrm{IG} = 1$

best feature

Alternative to Entropy: Gini Index

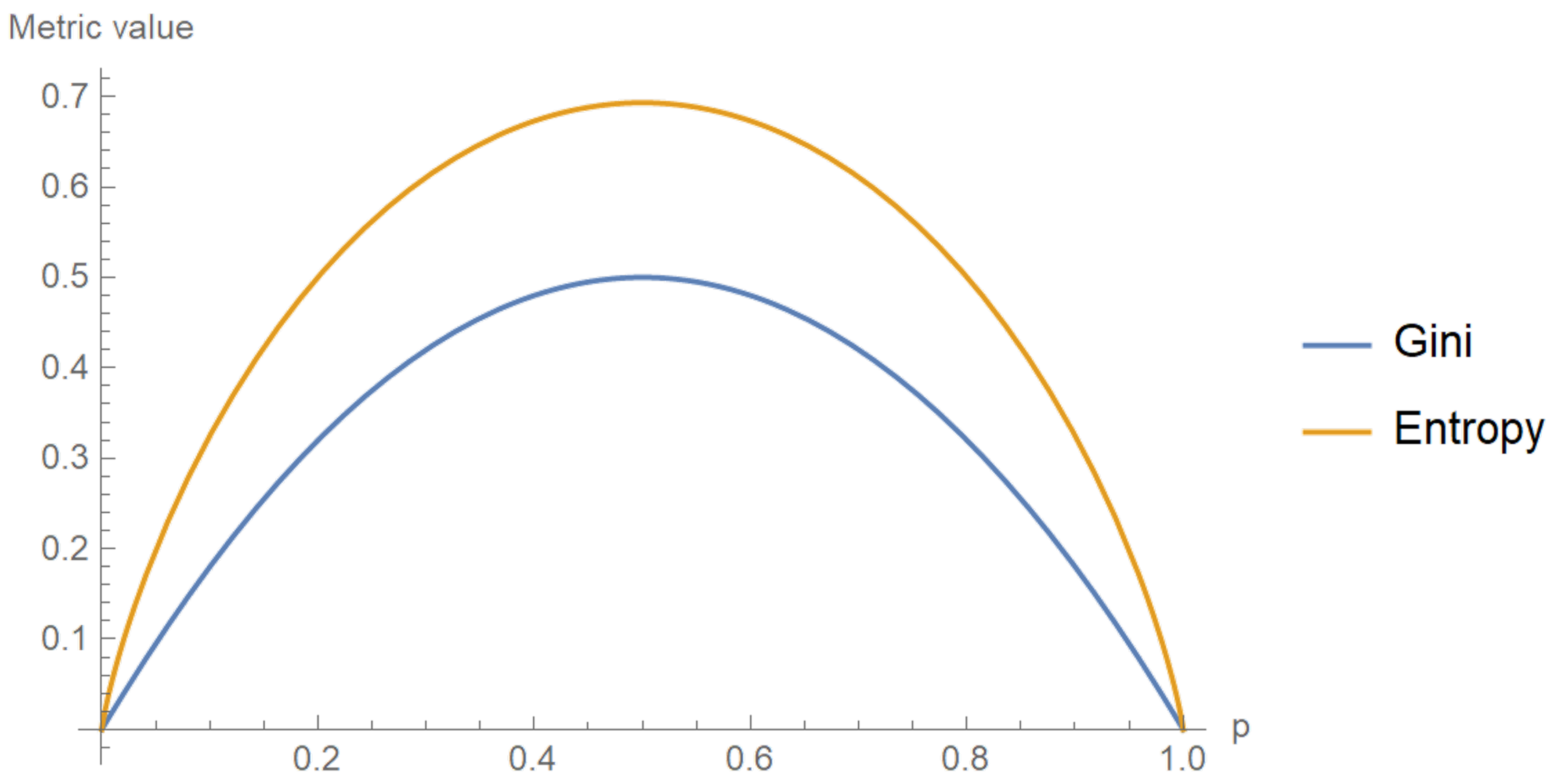
- Given a random variable Y with domain $\{1, \ldots, k\}$, its **Gini index** is

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

- 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

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

Decision Trees for Regression

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

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

- Here, $\mu(Y) = \sum_{y=1}^{k} P(Y = y) \cdot y$ is the mean of Y

- Define $\mathrm{MSE}(Z) = \mathrm{MSE}(Y)$ as before

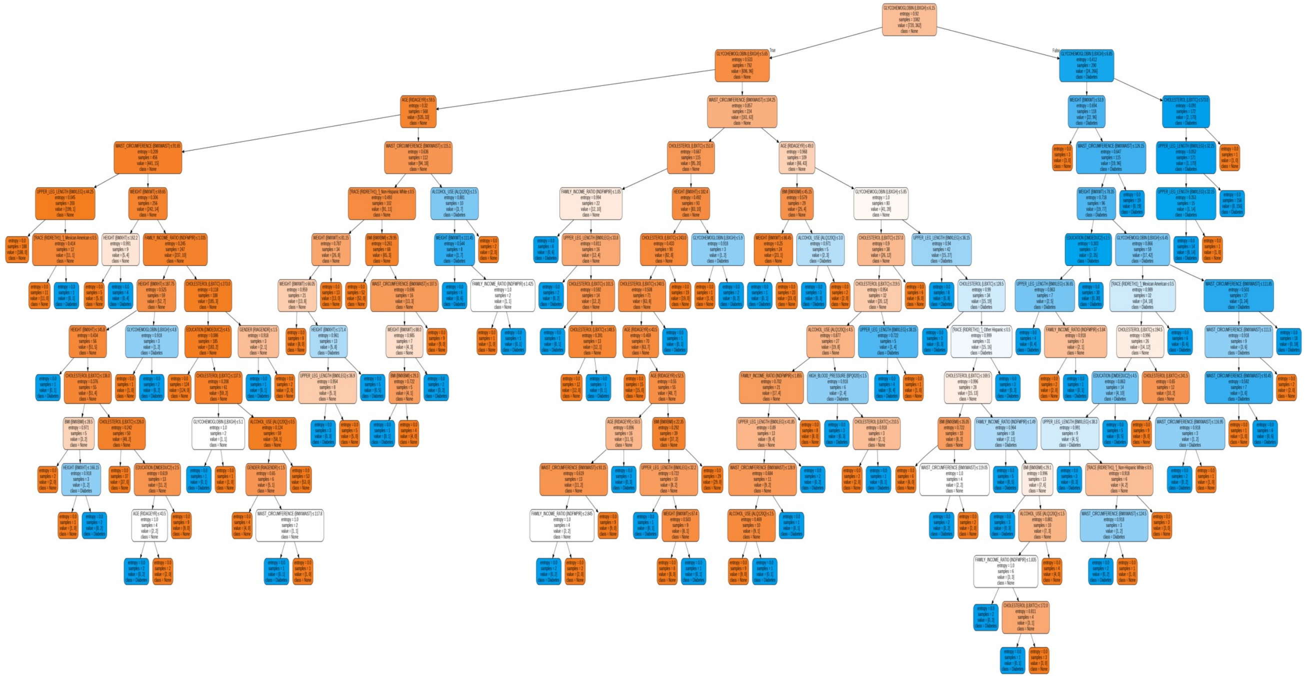
Bias-Variance Tradeoff

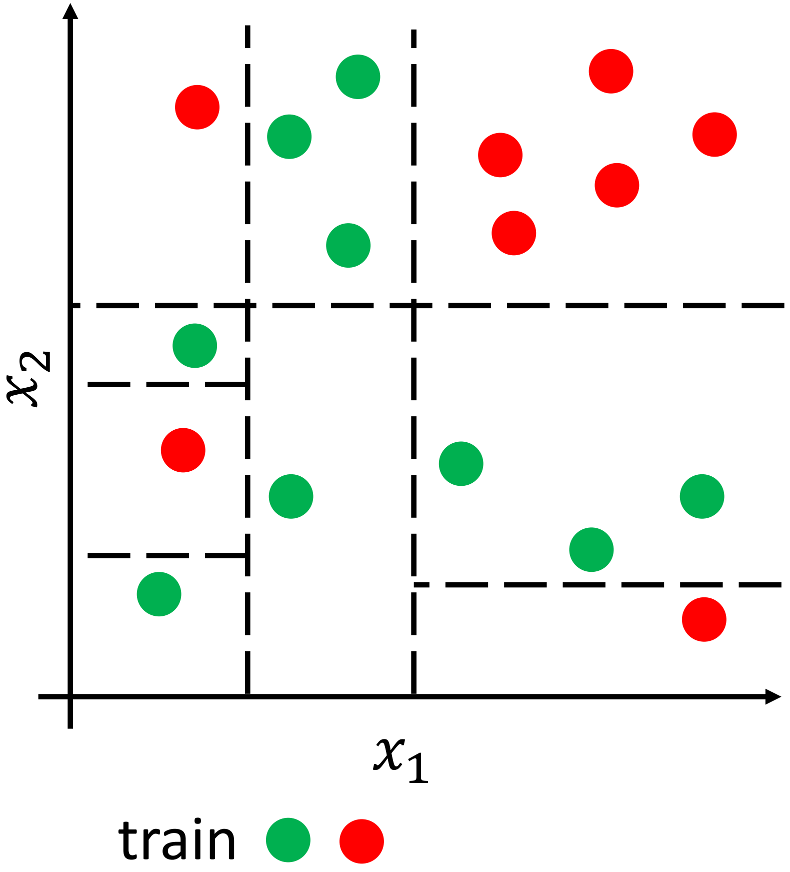
- On the diabetes dataset:
  - **Training accuracy:** 100%
  - **Test accuracy:** 82.8%

- The training accuracy is **always** 100%!
  - Because we grow trees until each leaf node is pure

Bias-Variance Tradeoff

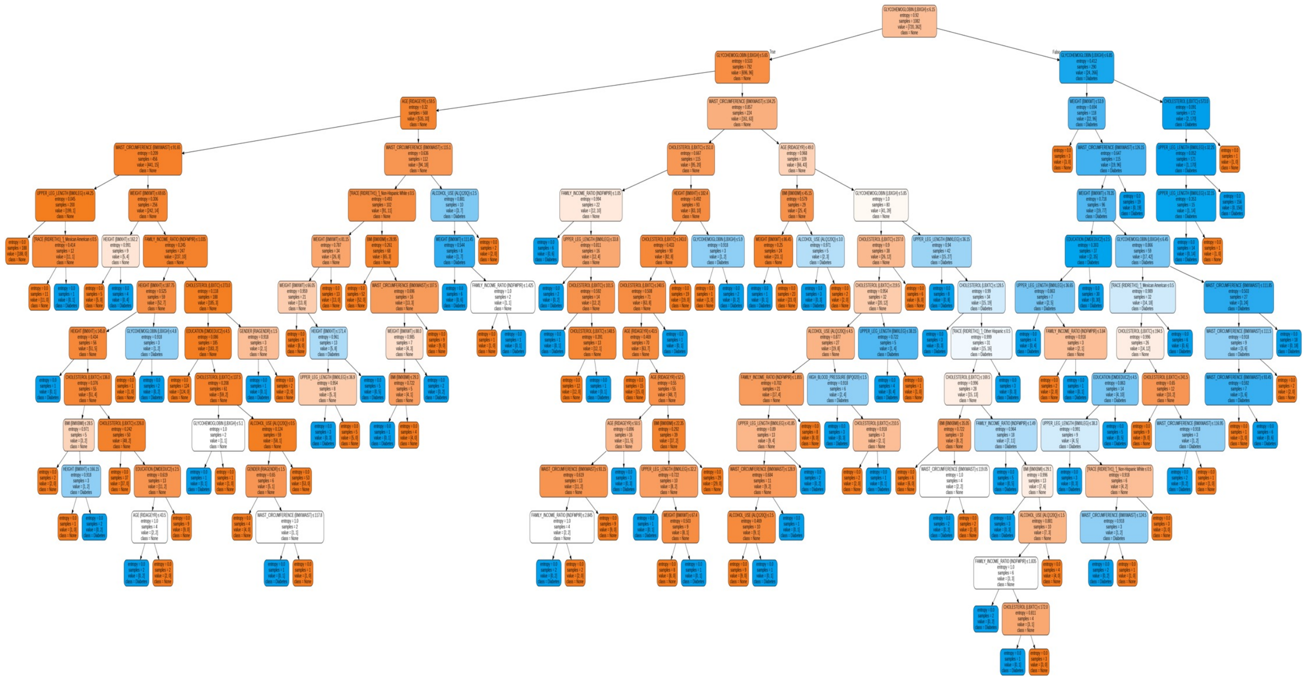
- Grow trees until each leaf node is pure

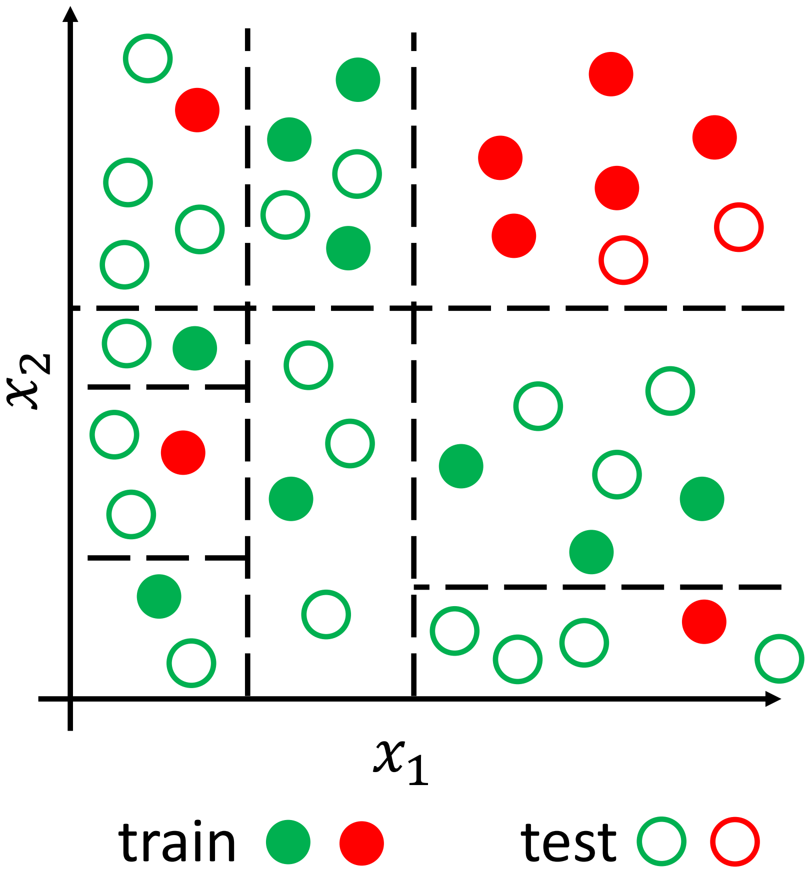




Bias-Variance Tradeoff

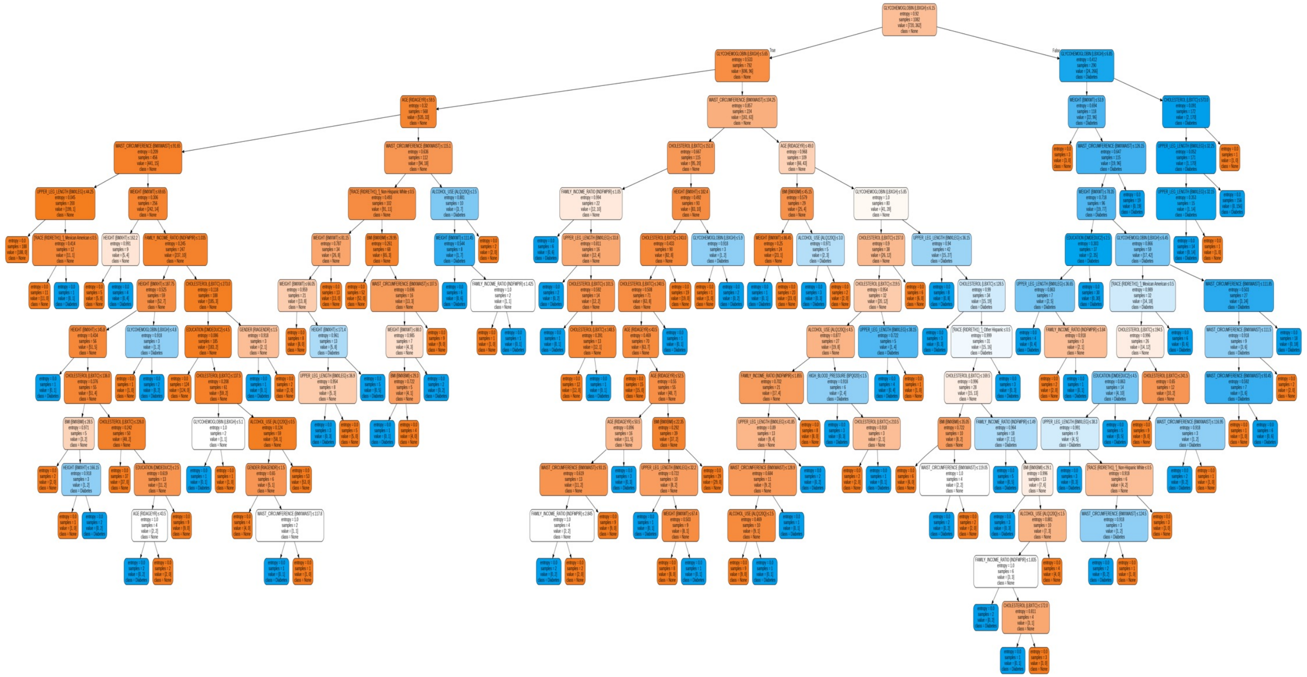
- Grow trees until each leaf node is pure

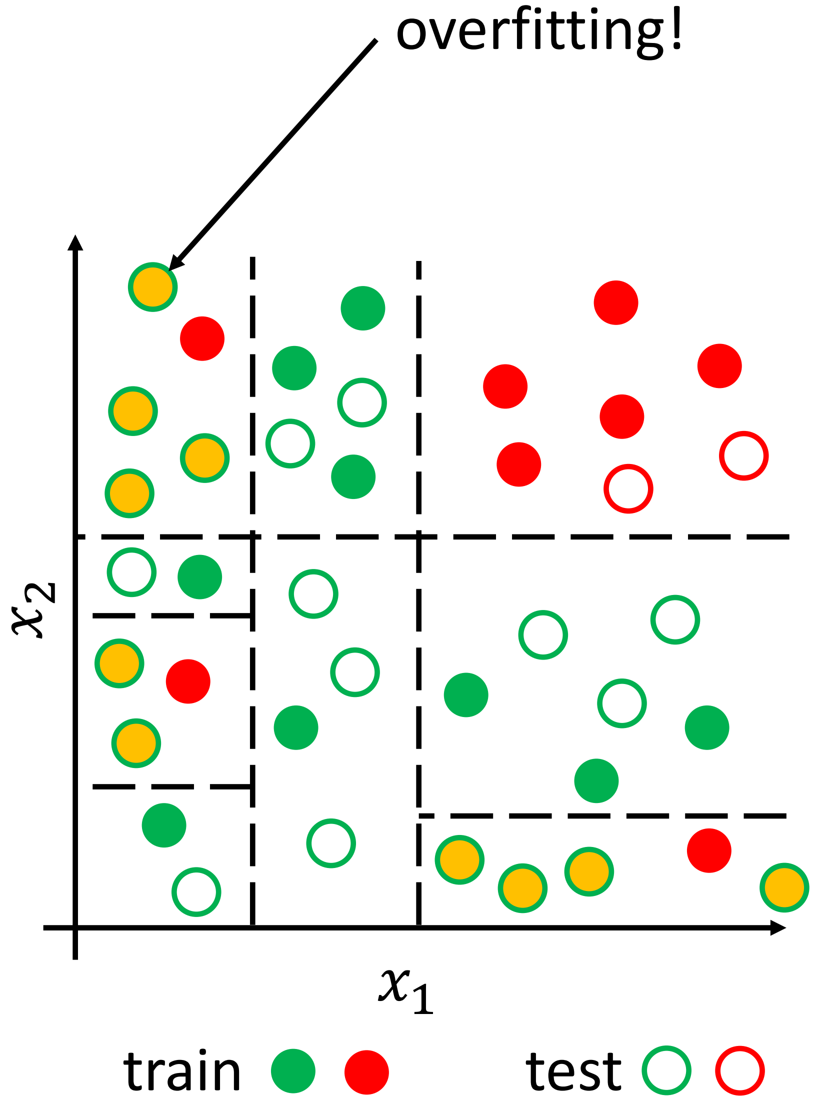




Bias-Variance Tradeoff

- Grow trees until each leaf node is pure





Bias-Variance Tradeoff

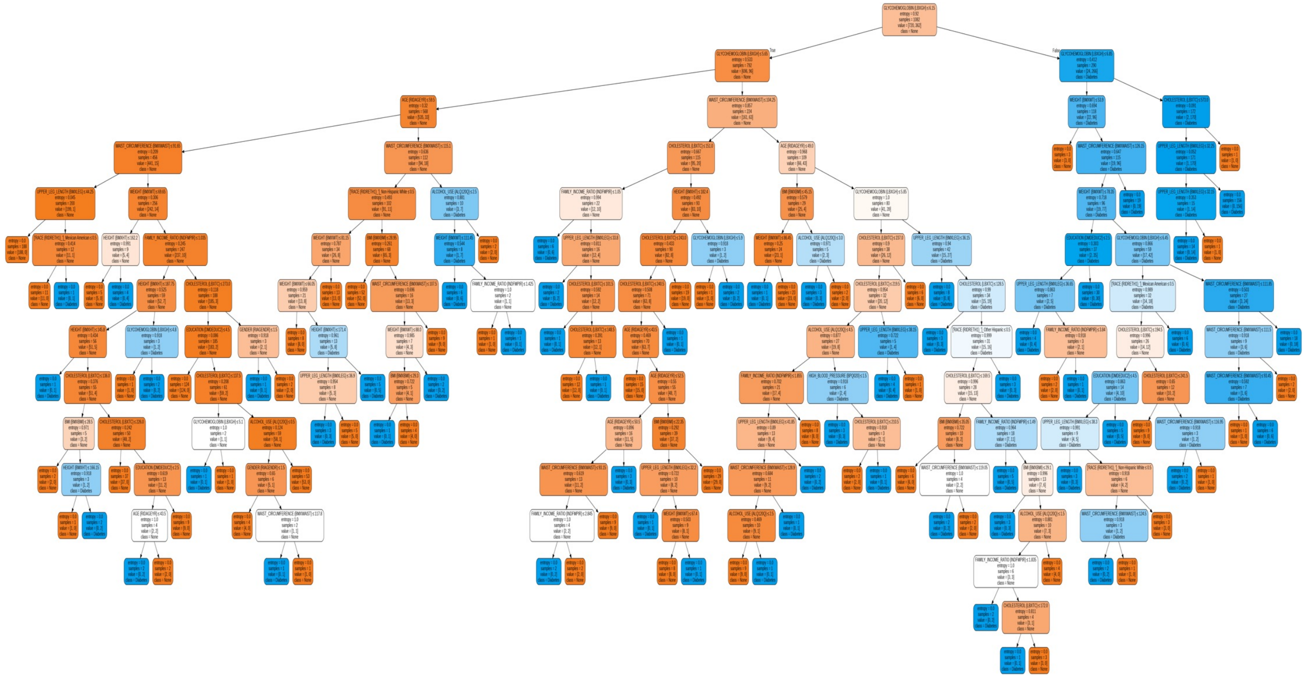
- **Recall:** Overfitting happens because we fit “noise” in the data

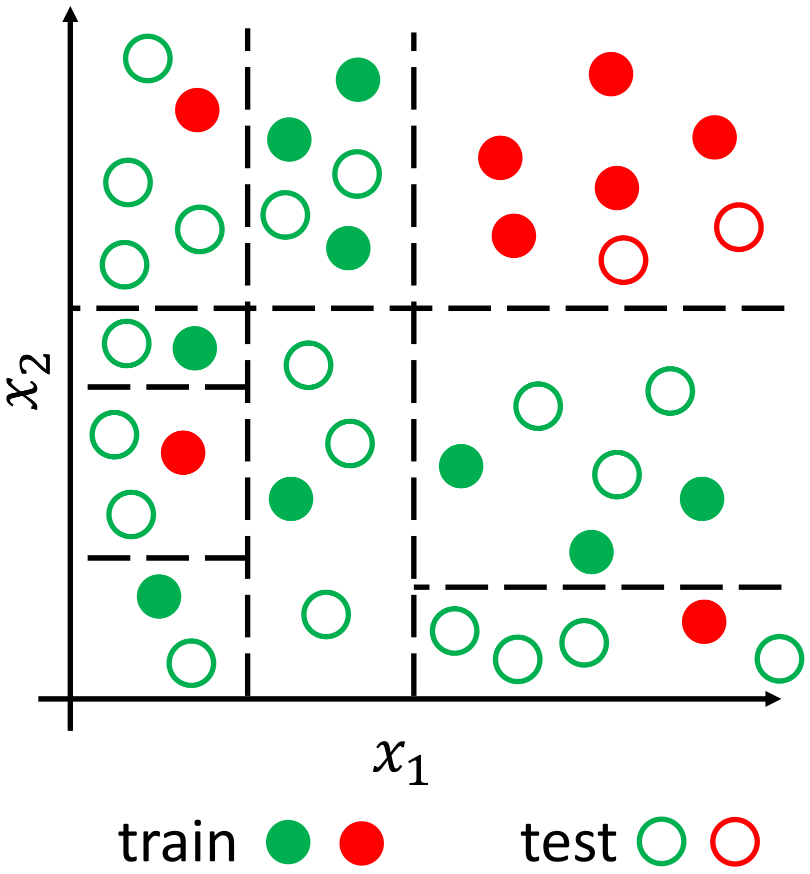
- **Avoiding overfitting**
  - Gather more data
  - Remove features
  - **Regularization:** Bias towards simpler models

- **Idea:** “Prune” decision tree to reduce its size
  - Smaller decision tree → fewer parameters → lower model capacity

Bias-Variance Tradeoff

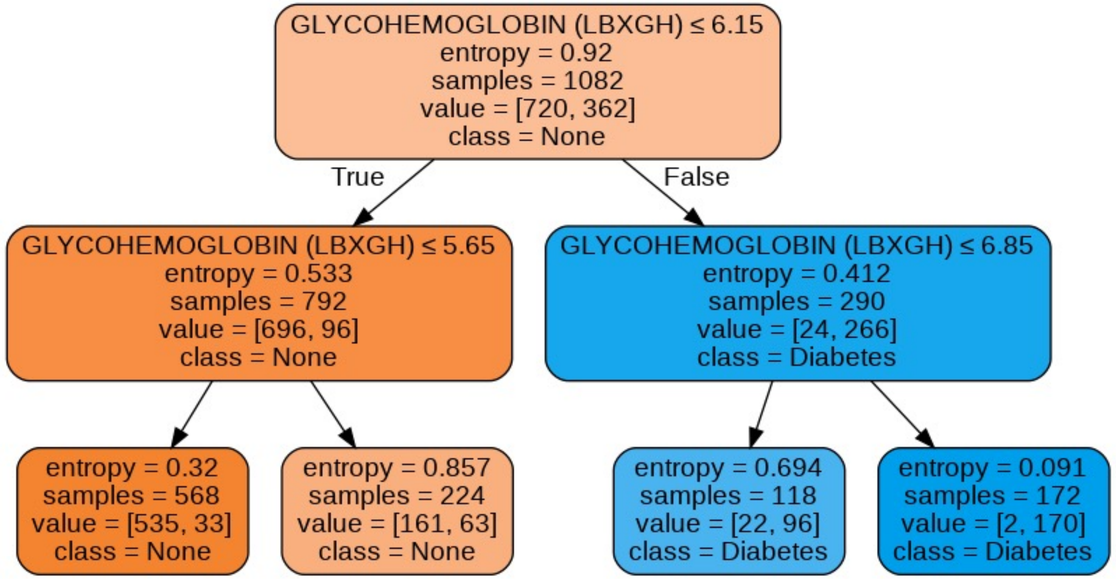
- Prune decision tree to simplify

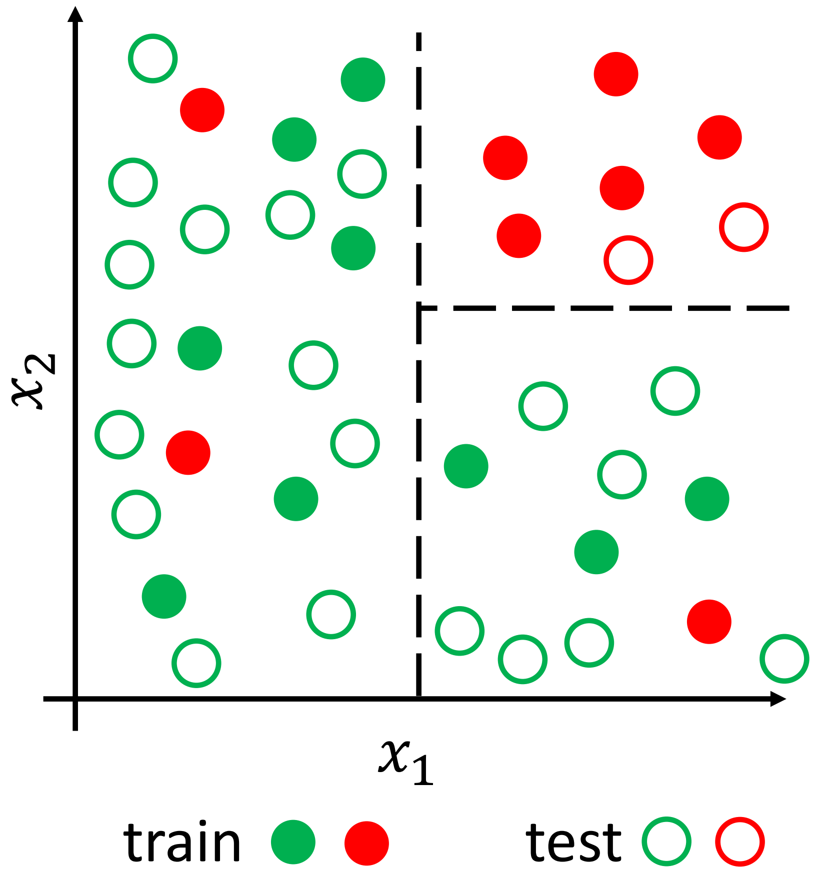




Bias-Variance Tradeoff

- 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:

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

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

$N_0 \leftarrow \arg\max_N g_N$

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:

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

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

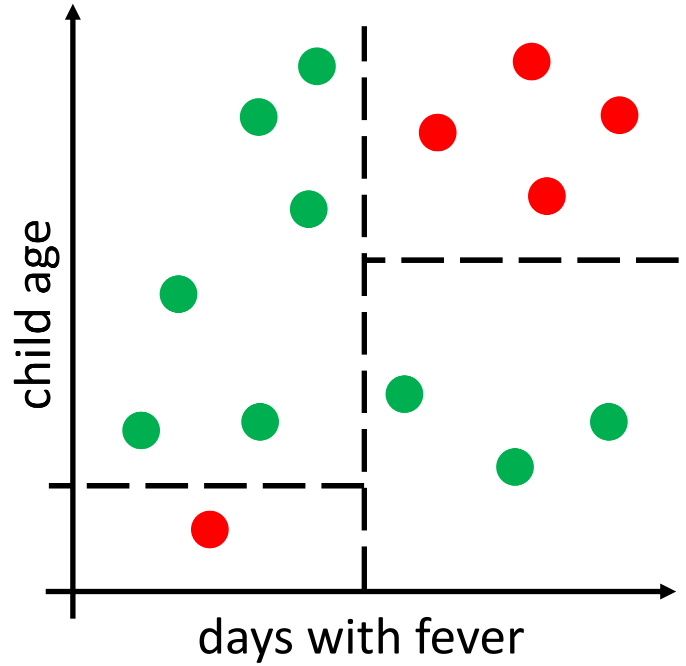
  $N_0 \leftarrow \arg\max_N g_N$

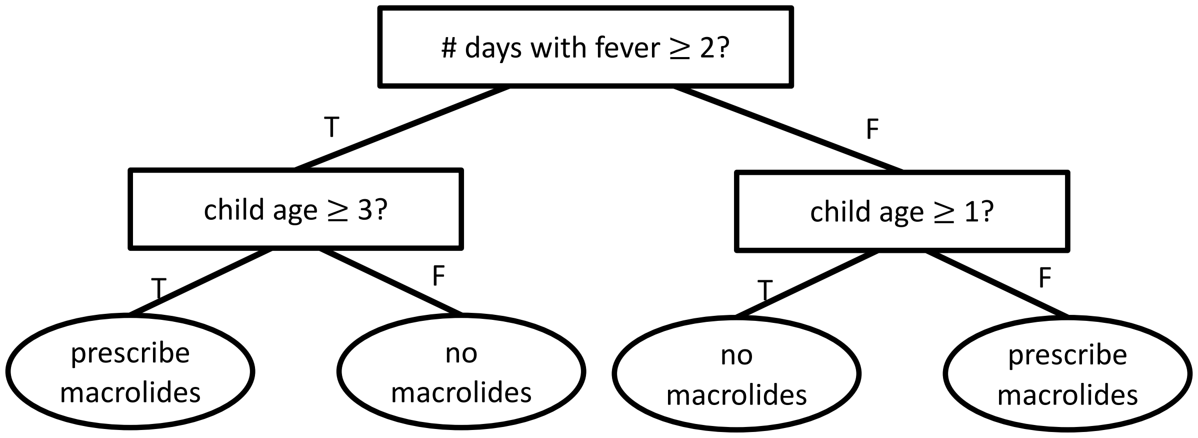
  if $g_{N_0} > 0$:

    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:

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

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

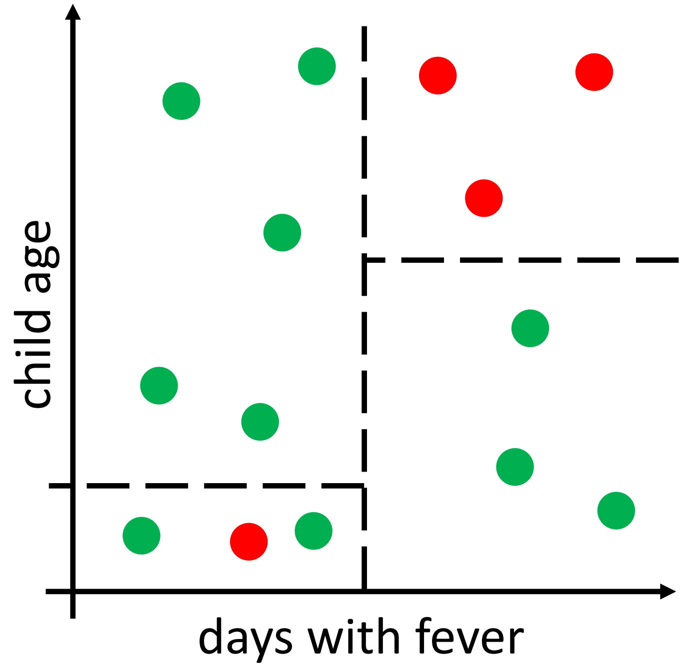
  $N_0 \leftarrow \arg\max_N g_N$

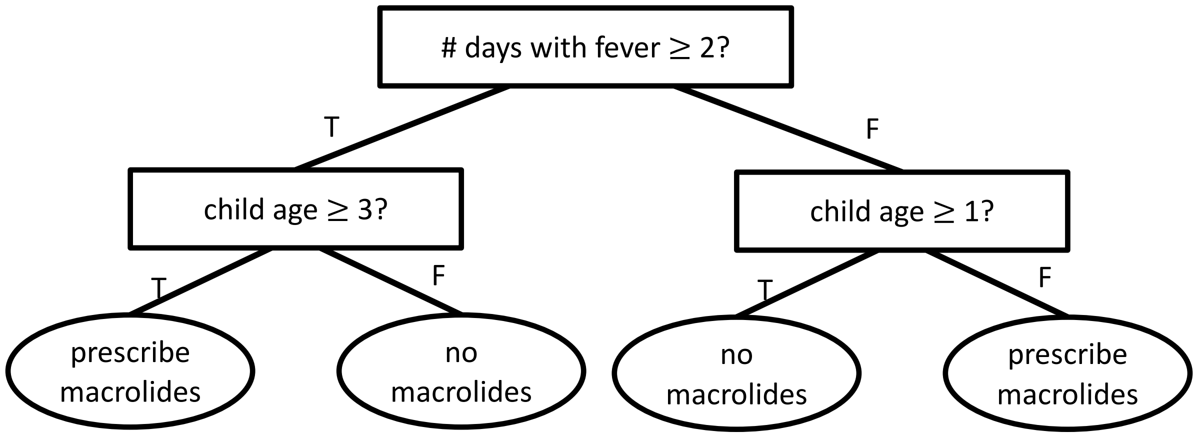
  if $g_{N_0} > 0$:

    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:

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

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

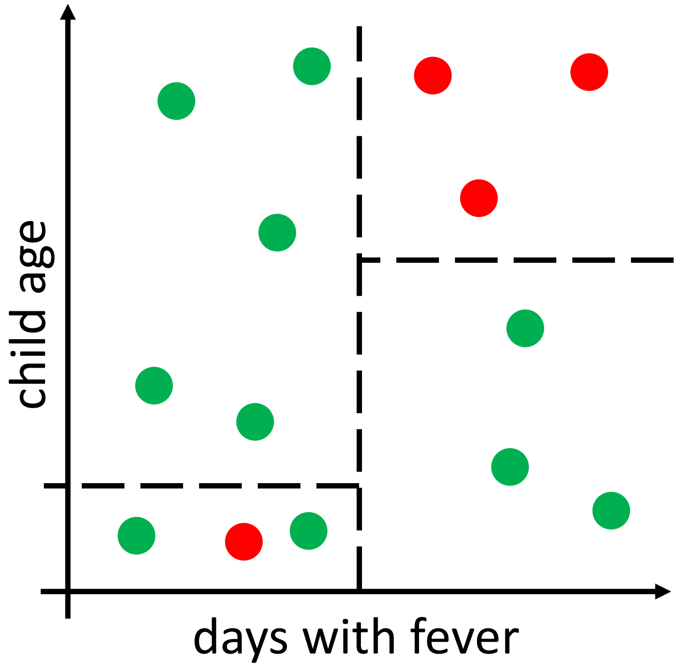
$N_0 \leftarrow \arg\max_N g_N$

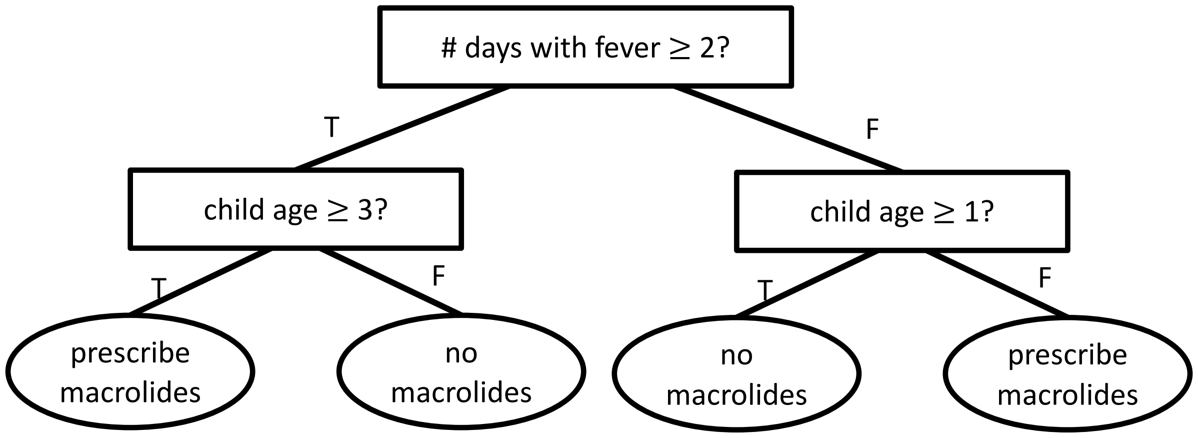
if $g_{N_0} > 0$:

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:

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

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

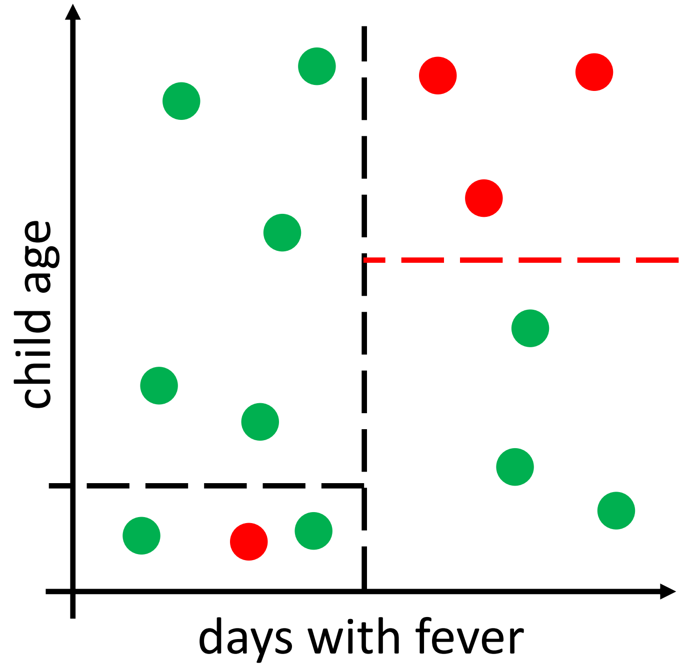
$N_0 \leftarrow \arg\max_N g_N$

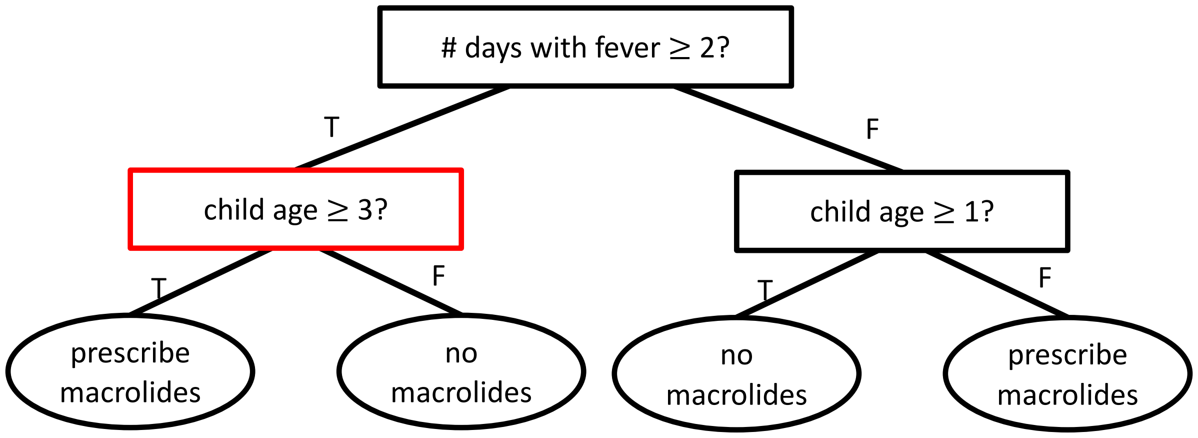
if $g_{N_0} > 0$:

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:

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

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

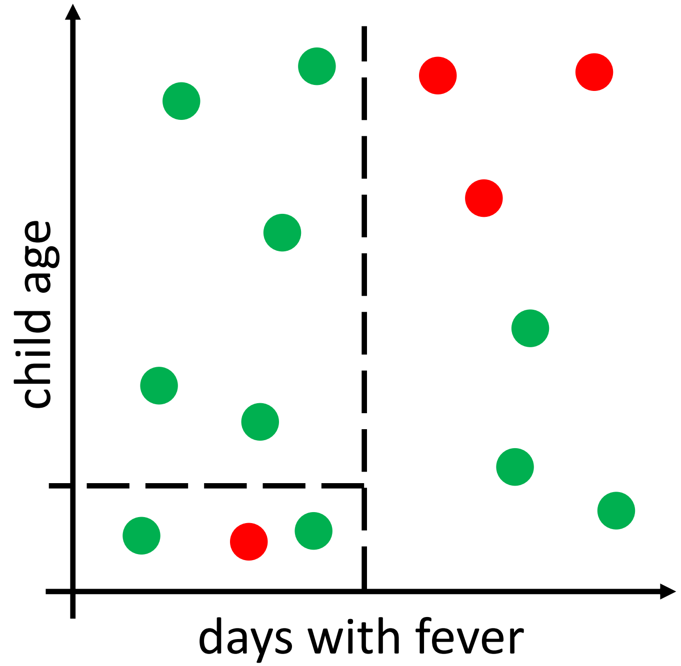
  $N_0 \leftarrow \arg\max_N g_N$

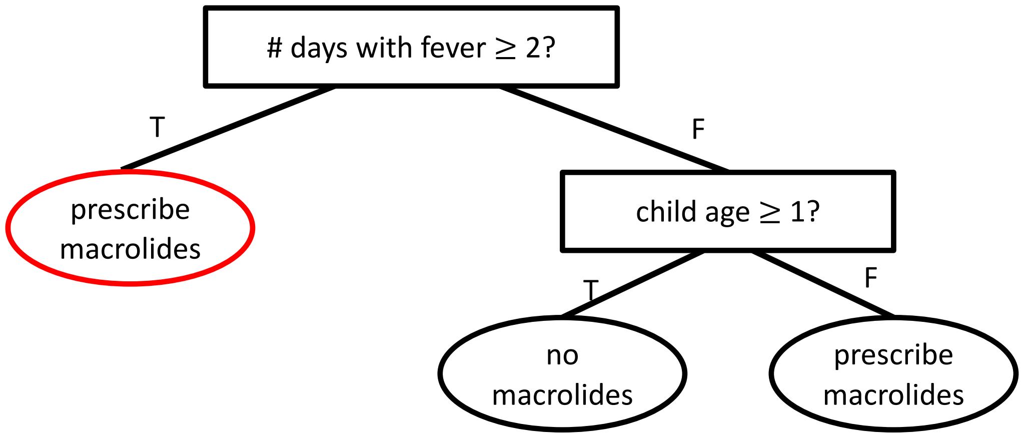
  if $g_{N_0} > 0$:

    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:

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

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

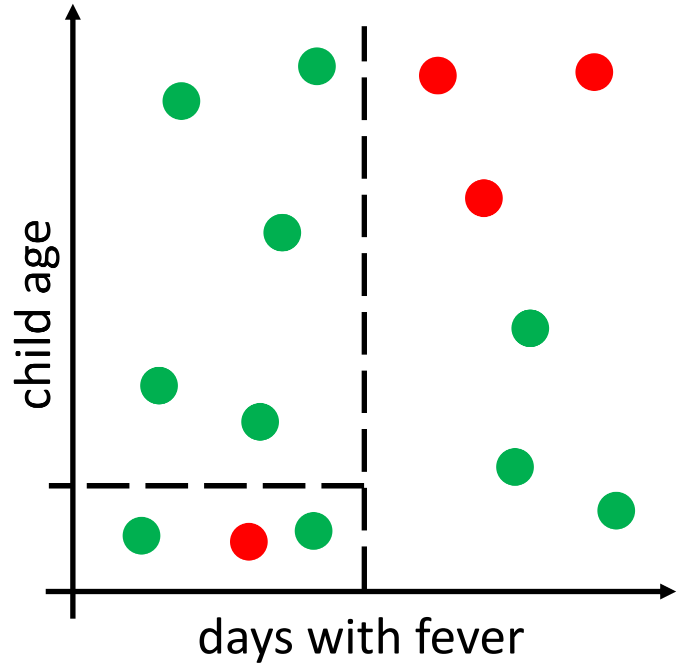
  $N_0 \leftarrow \arg\max_N g_N$

  if $g_{N_0} > 0$:

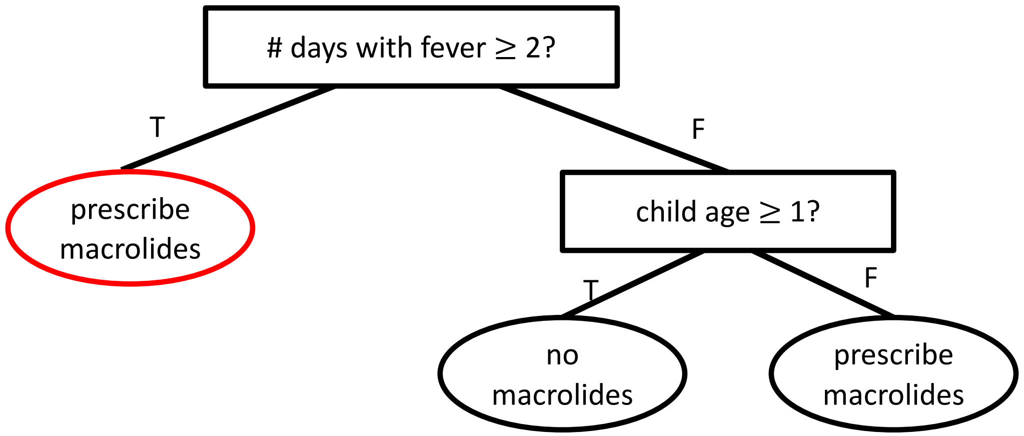
    return PostPruneTree($T_N, Z_{\text{train}}, Z_{\text{val}}$)

  else:

    return T



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



Post Pruning

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

  for each internal node N of T:

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

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

  $N_0 \leftarrow \arg\max_N g_N$

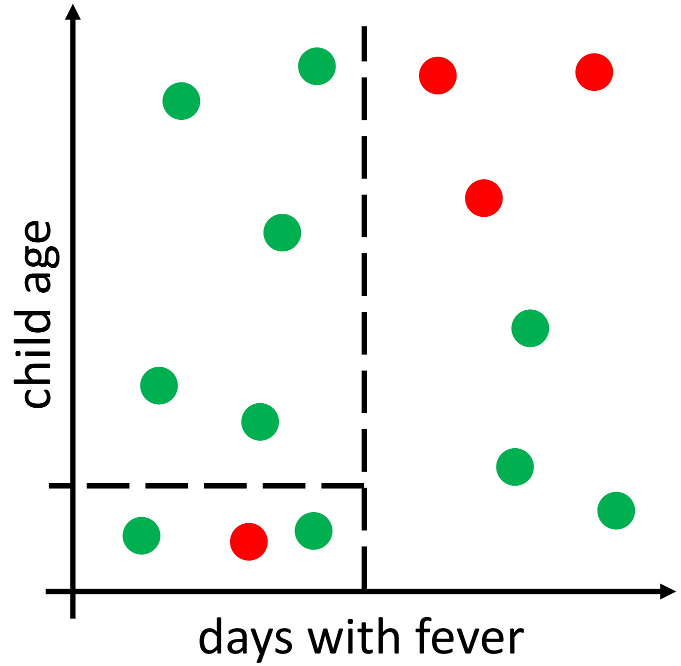
  if $g_{N_0} > 0$:

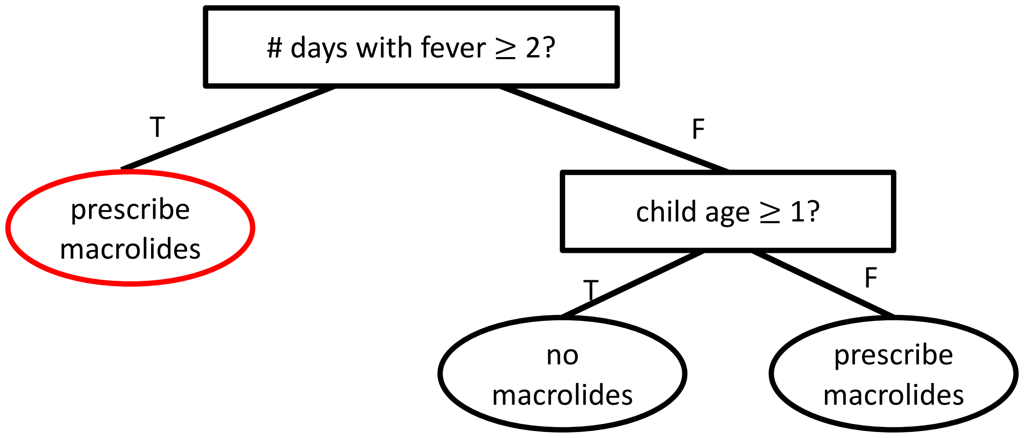
    return PostPruneTree($T_N, Z_{\text{train}}, Z_{\text{val}}$)

  else:

    return T

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





Post Pruning

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

  for each internal node N of T:

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

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

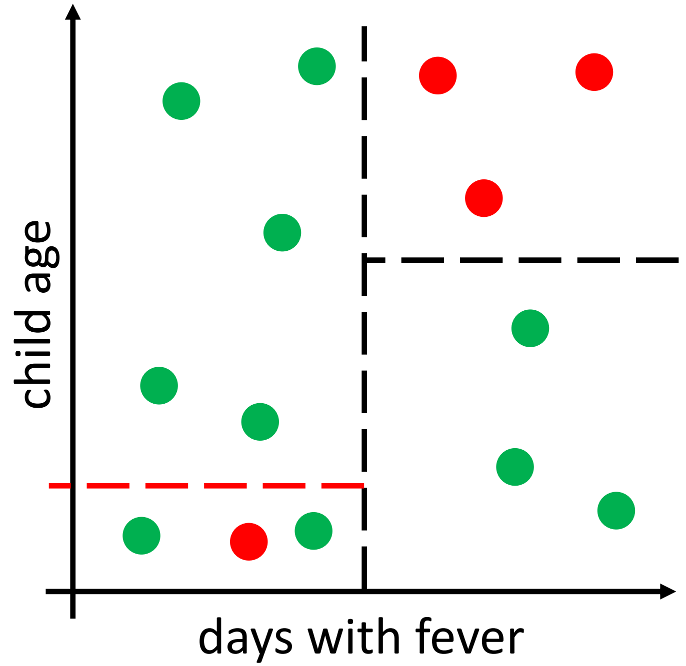
  $N_0 \leftarrow \arg\max_N g_N$

  if $g_{N_0} > 0$:

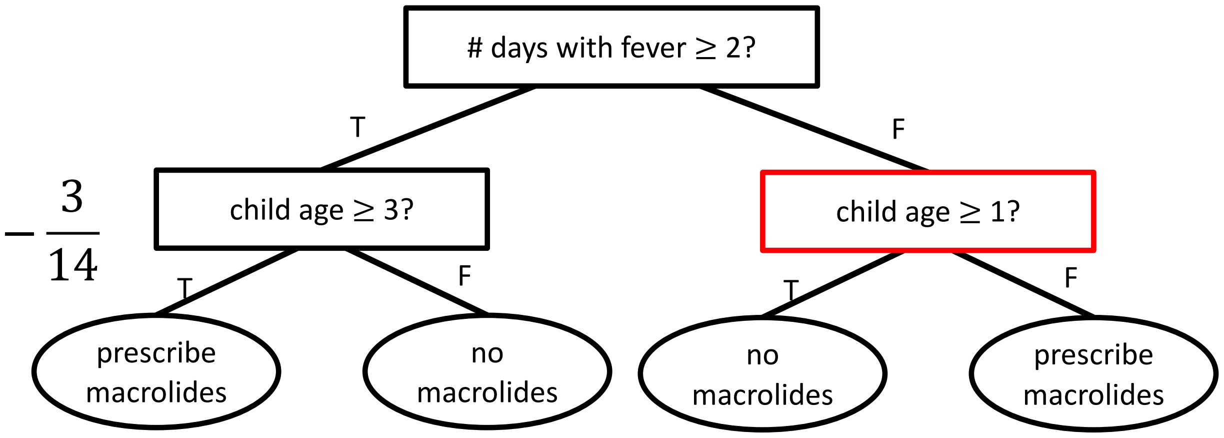
    return PostPruneTree($T_N, Z_{\text{train}}, Z_{\text{val}}$)

  else:

    return T



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



Post Pruning

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

  for each internal node N of T:

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

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

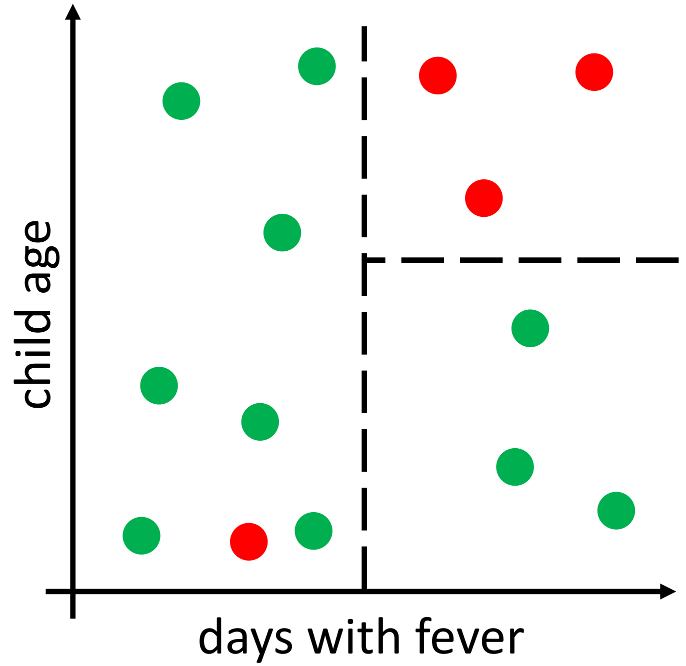
  $N_0 \leftarrow \arg\max_N g_N$

  if $g_{N_0} > 0$:

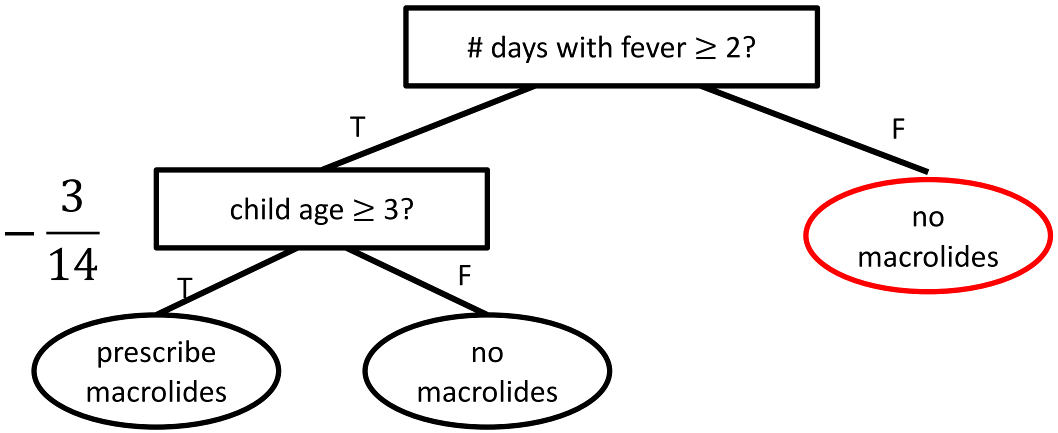
    return PostPruneTree($T_N, Z_{\text{train}}, Z_{\text{val}}$)

  else:

    return T



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



Post Pruning

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

  for each internal node N of T:

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

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

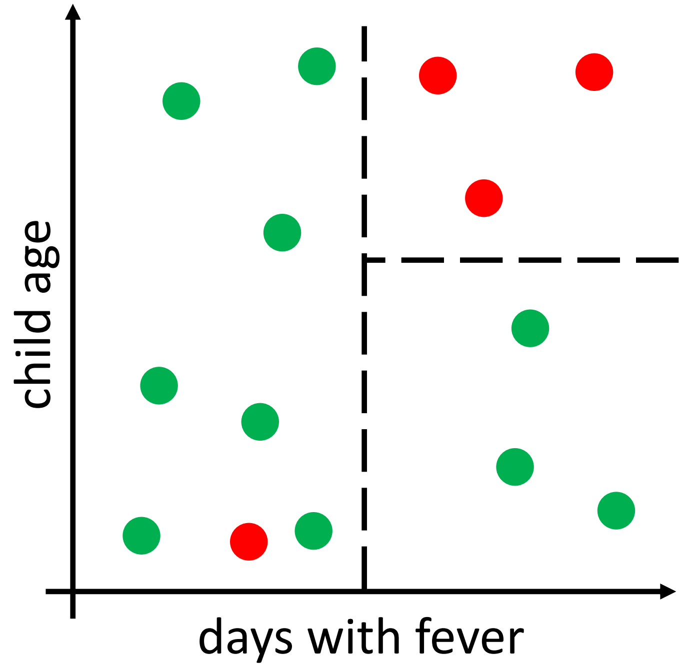
  $N_0 \leftarrow \arg\max_N g_N$

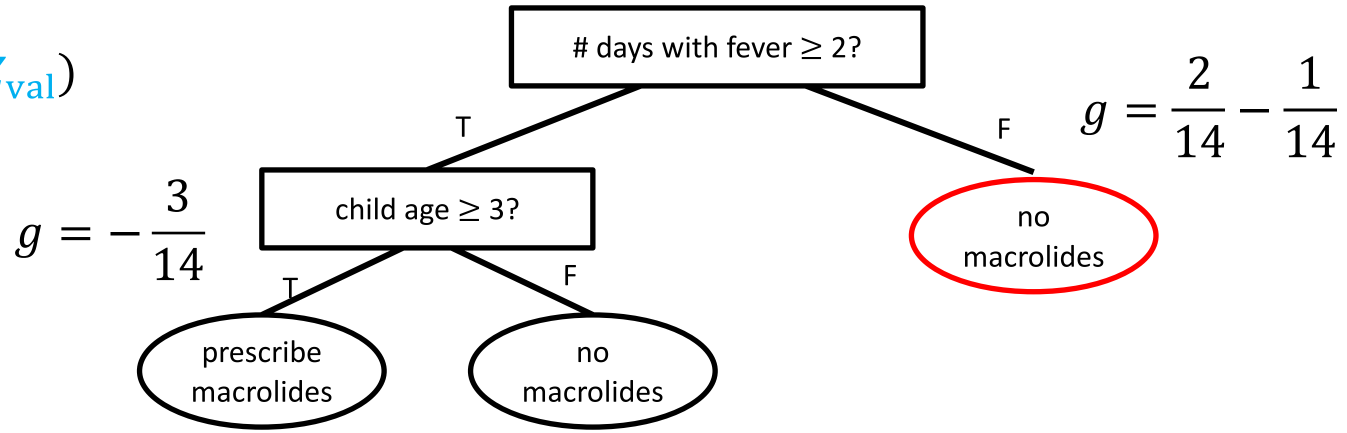
  if $g_{N_0} > 0$:

    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:

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

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

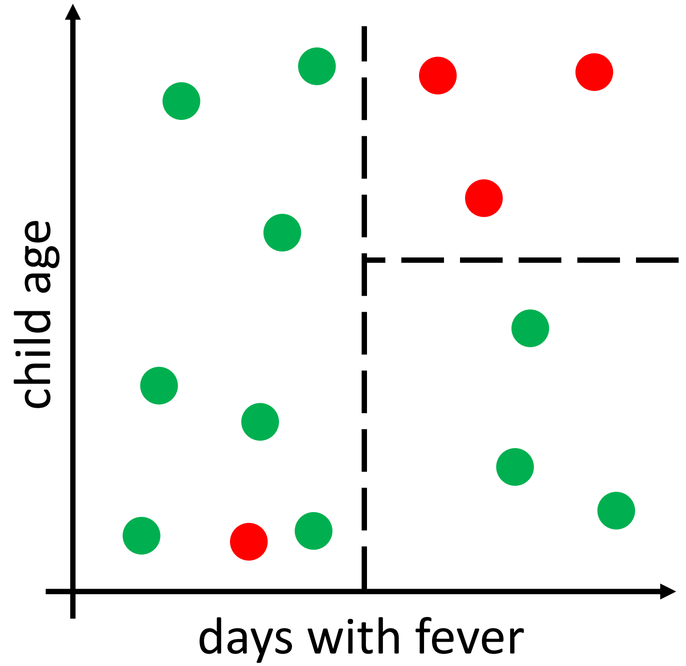
  $N_0 \leftarrow \arg\max_N g_N$

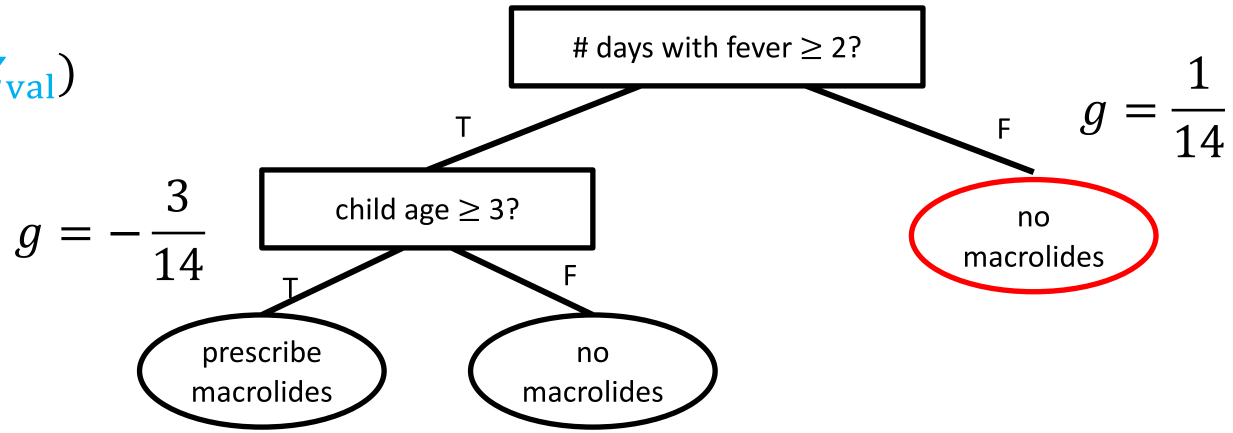
  if $g_{N_0} > 0$:

    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:

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

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

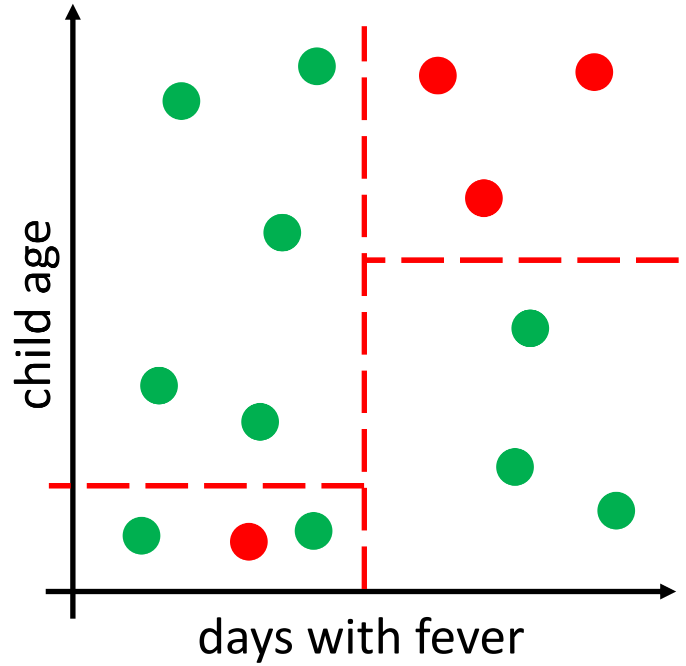
  $N_0 \leftarrow \arg\max_N g_N$

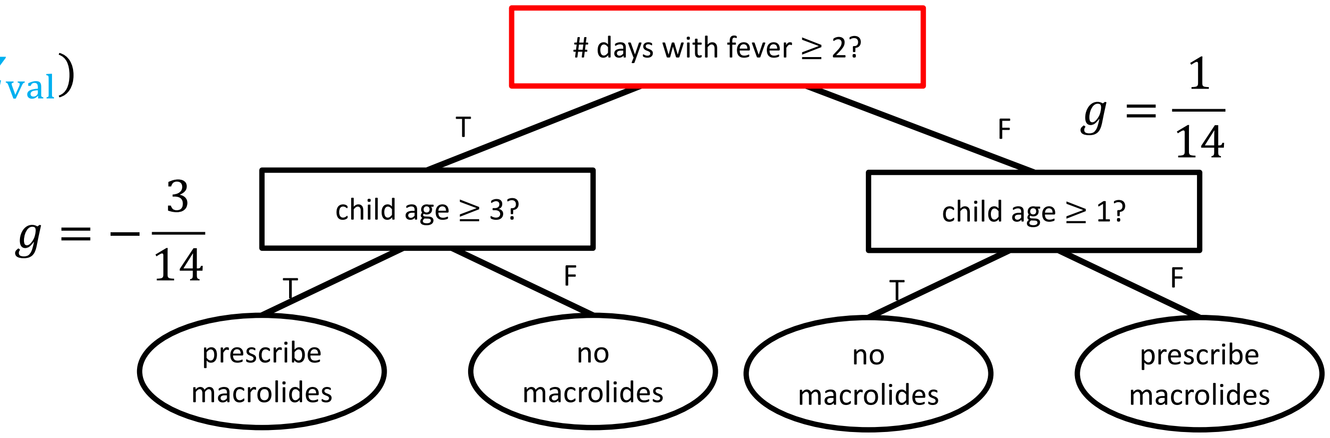
  if $g_{N_0} > 0$:

    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:

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

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

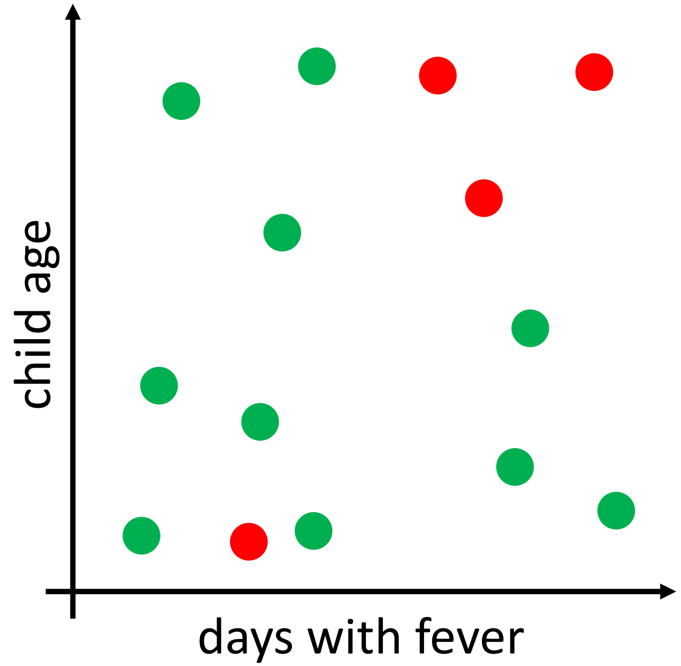
  $N_0 \leftarrow \arg\max_N g_N$

  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:

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

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

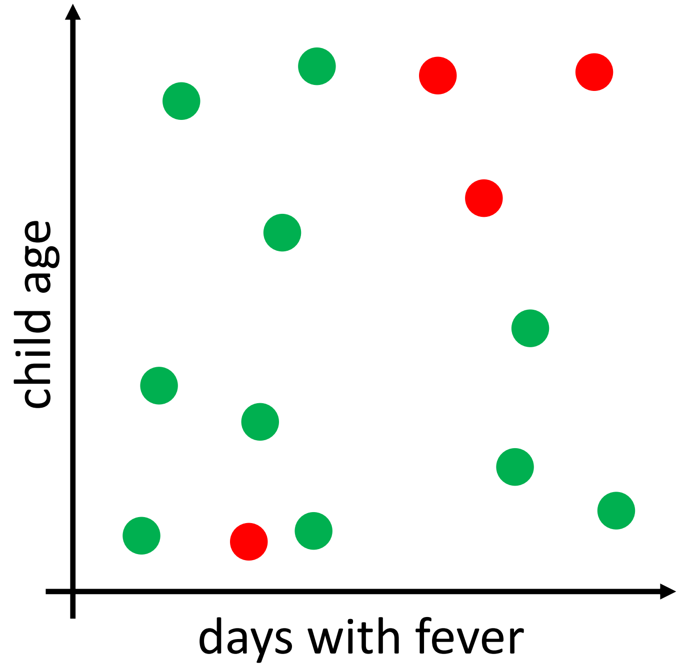
  $N_0 \leftarrow \arg\max_N g_N$

  if $g_{N_0} > 0$:

    return PostPruneTree($T_N, Z_{\text{train}}, Z_{\text{val}}$)

  else:

    return T



$$g = \frac{2}{14} - \frac{4}{14}$$

no macrolides

Post Pruning

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

    for each internal node N of T:

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

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

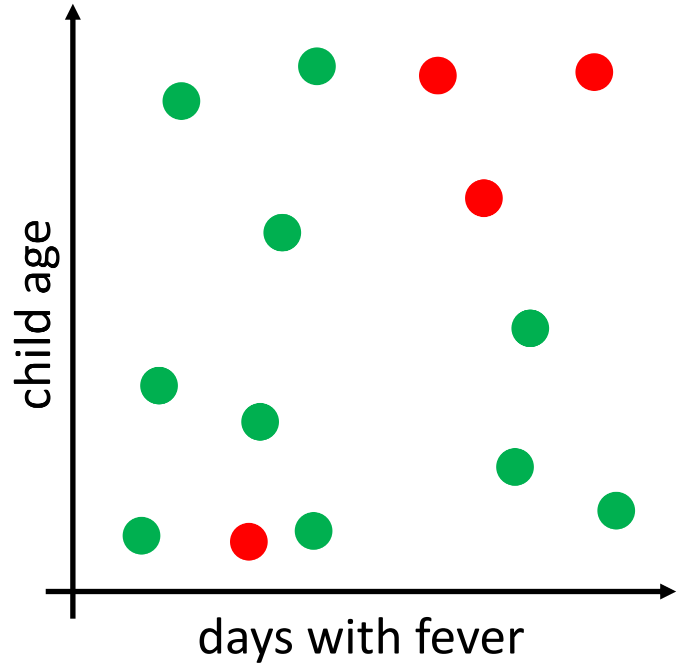
    $N_0 \leftarrow \arg\max_N g_N$

    if $g_{N_0} > 0$:

        return PostPruneTree($T_N, Z_{\text{train}}, Z_{\text{val}}$)

    else:

        return T



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

Post Pruning

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

  for each internal node N of T:

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

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

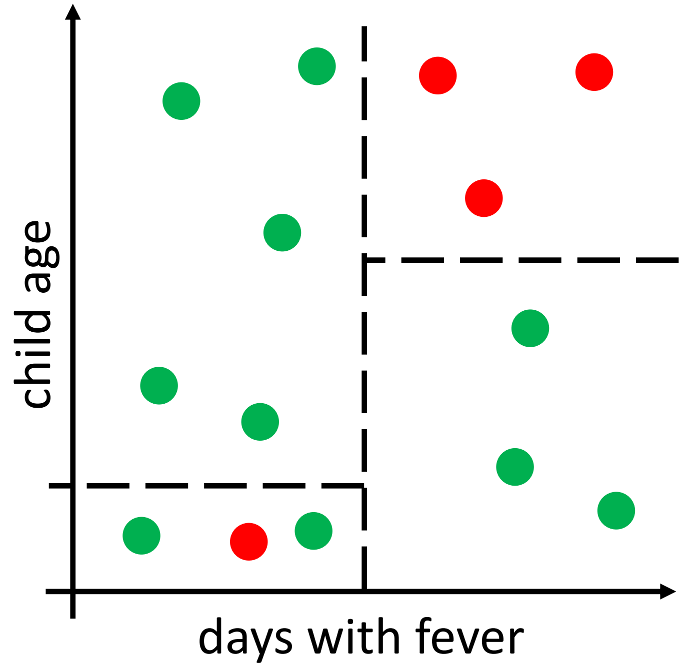
  $N_0 \leftarrow \arg\max_N g_N$

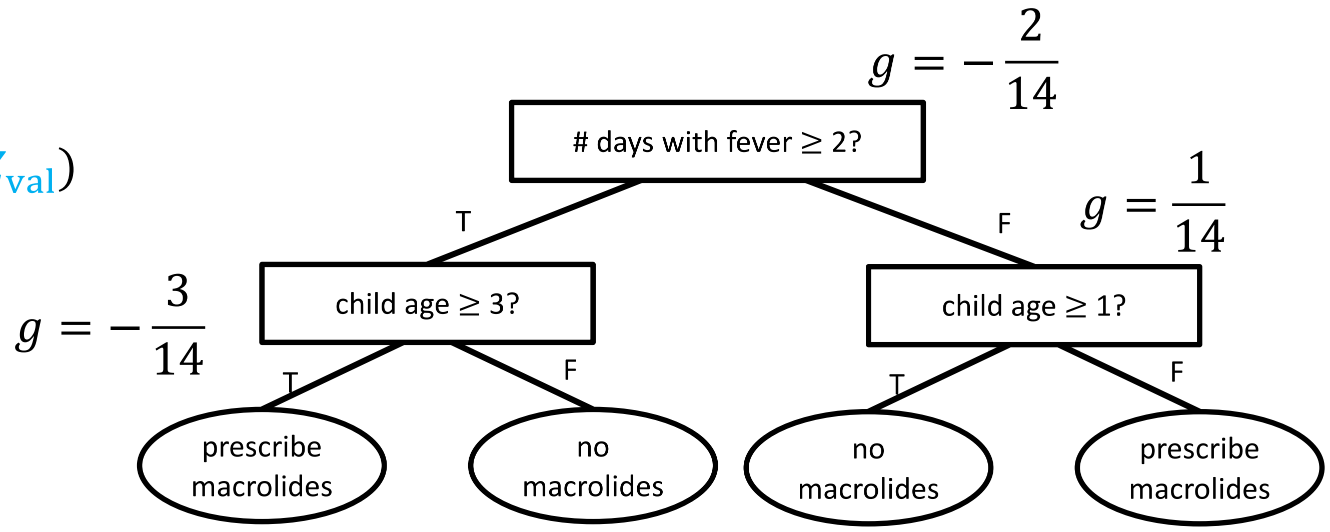
  if $g_{N_0} > 0$:

    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:

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

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

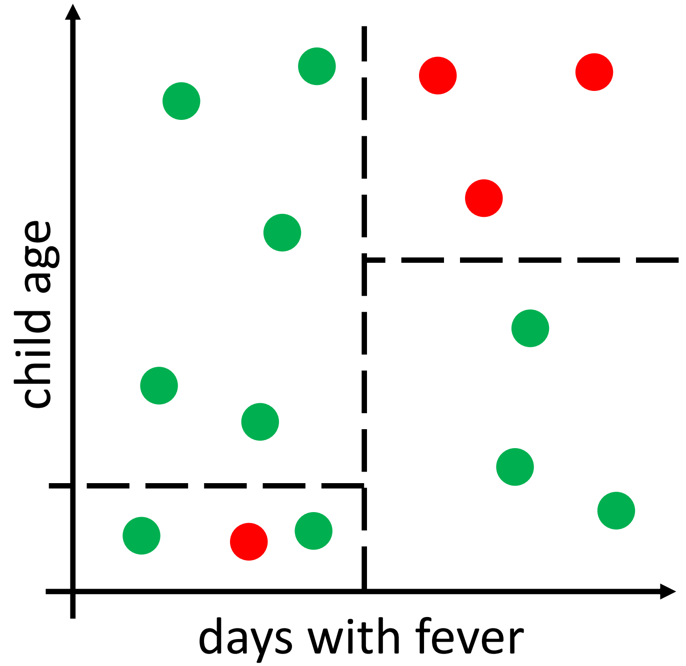
$$N_0 \leftarrow \arg\max_N g_N$$

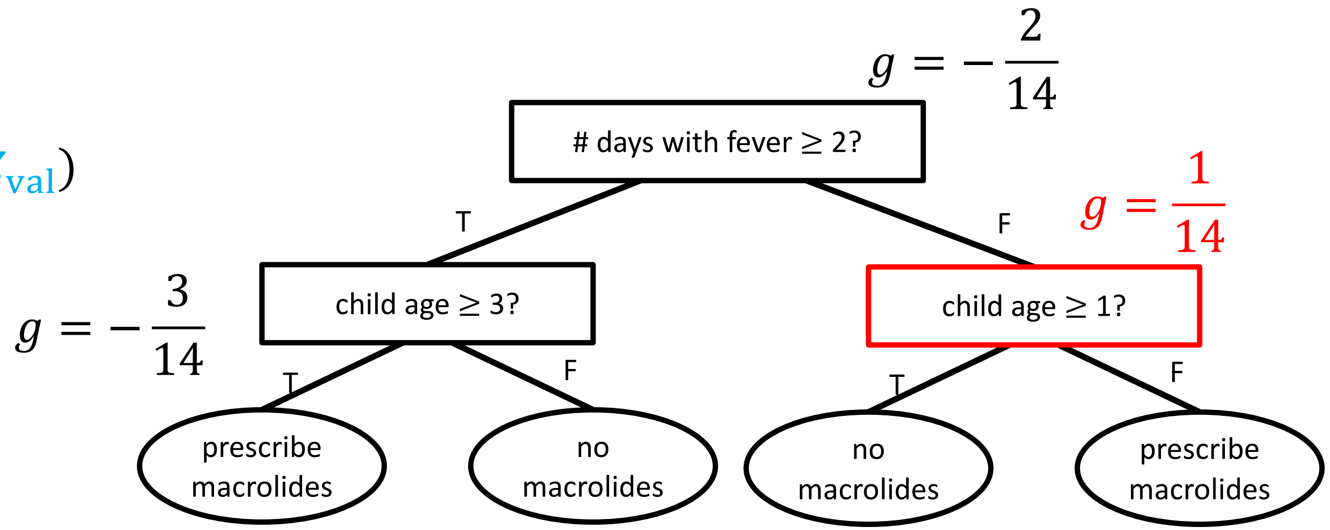
if $g_{N_0} > 0$:

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:

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

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

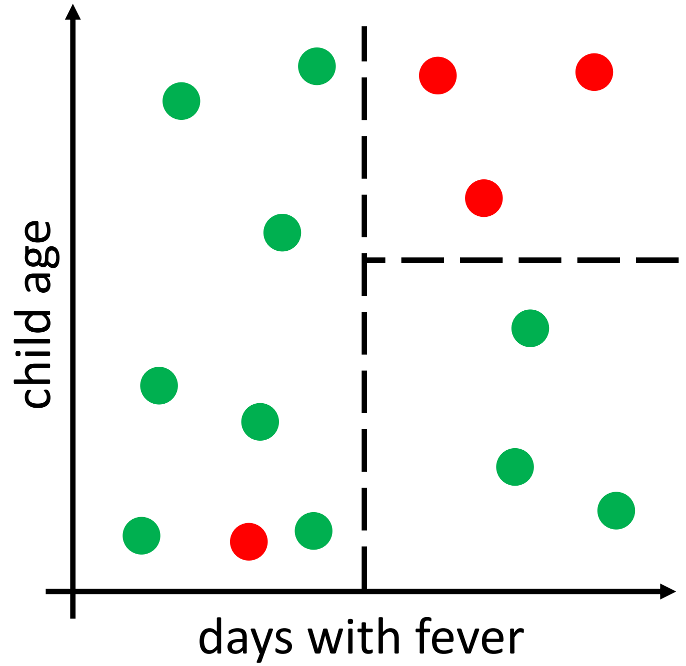
    $N_0 \leftarrow \arg\max_N g_N$

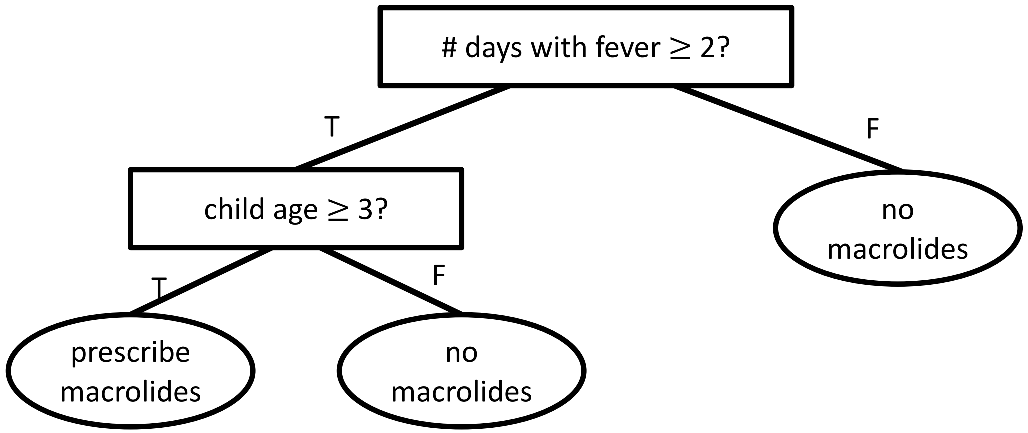
    if $g_{N_0} > 0$:

        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:

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

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

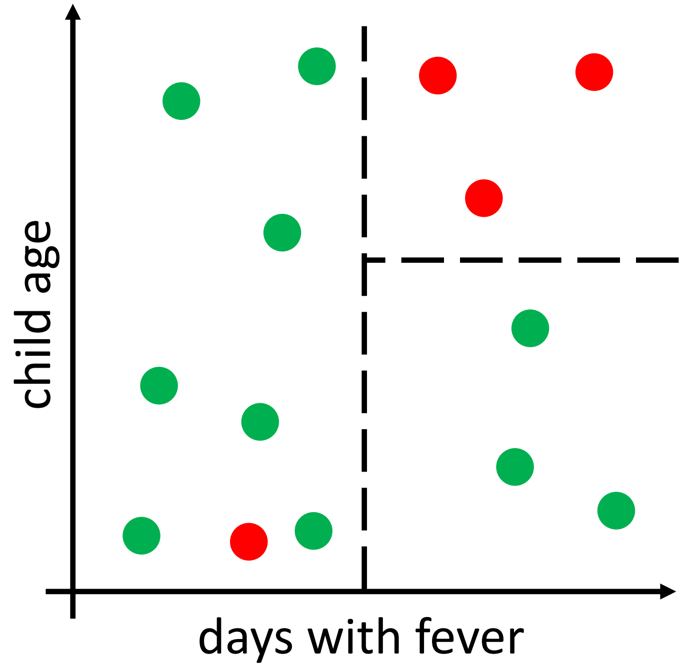
$N_0 \leftarrow \arg\max_N g_N$

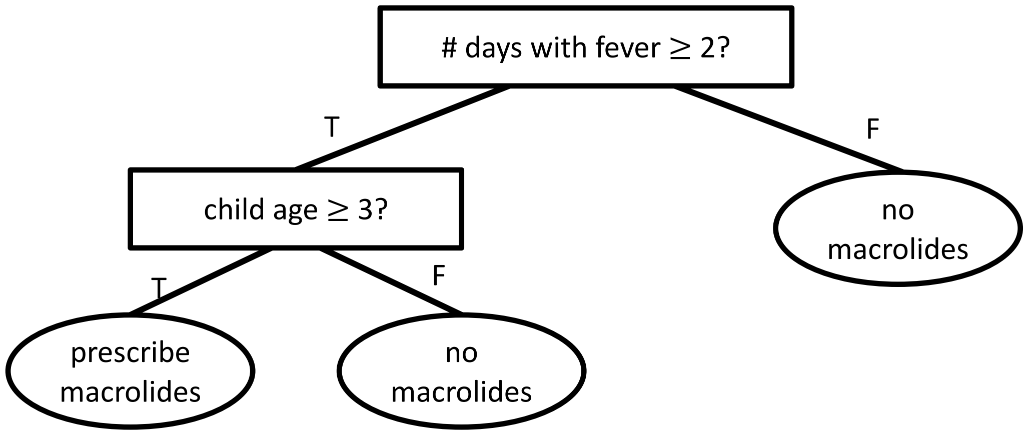
if $g_{N_0} > 0$:

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:

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

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

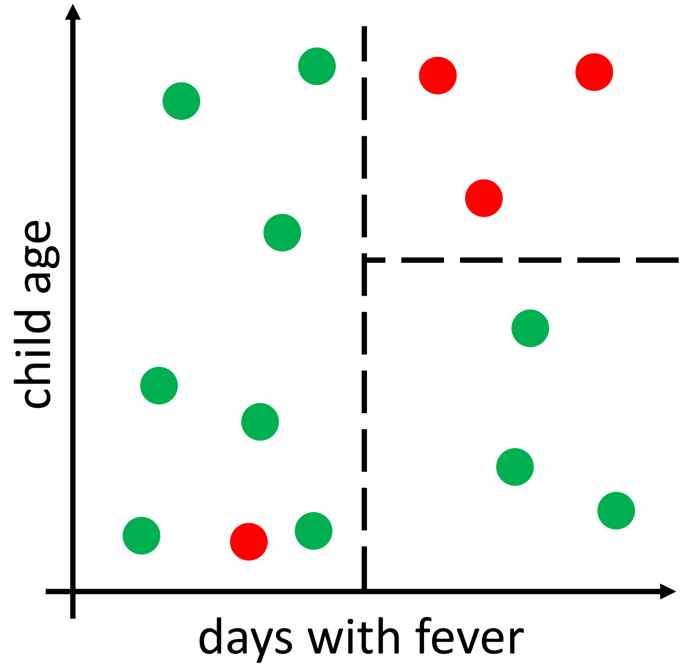
  $N_0 \leftarrow \arg\max_N g_N$

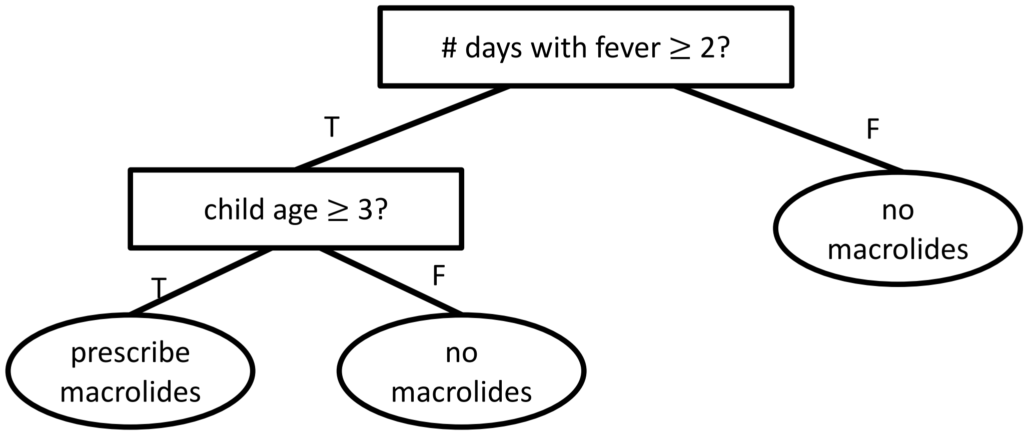
  if $g_{N_0} > 0$:

    return PostPruneTree($T_N, Z_{\text{train}}, Z_{\text{val}}$)

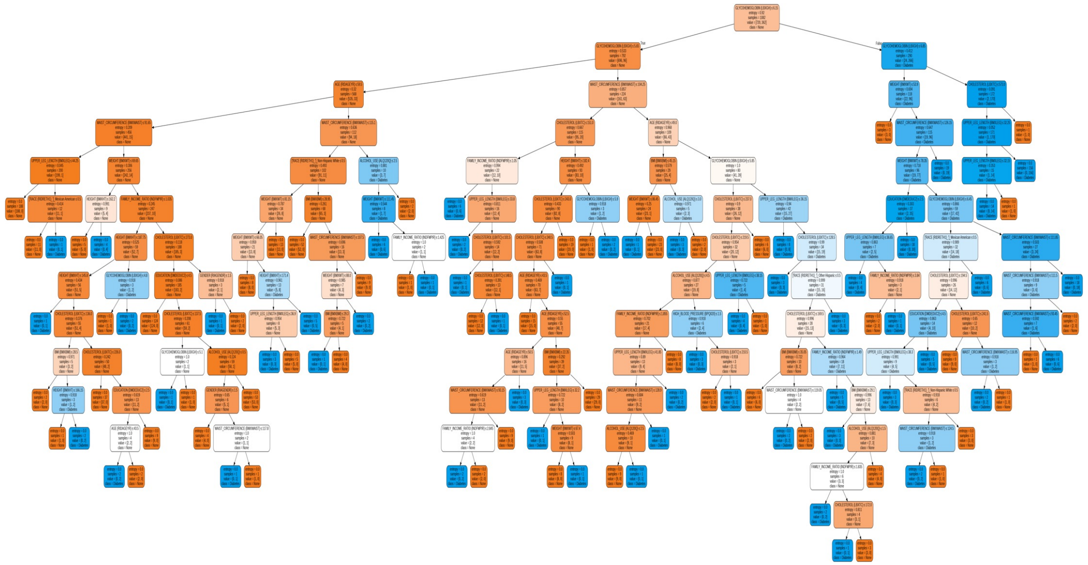
  else:

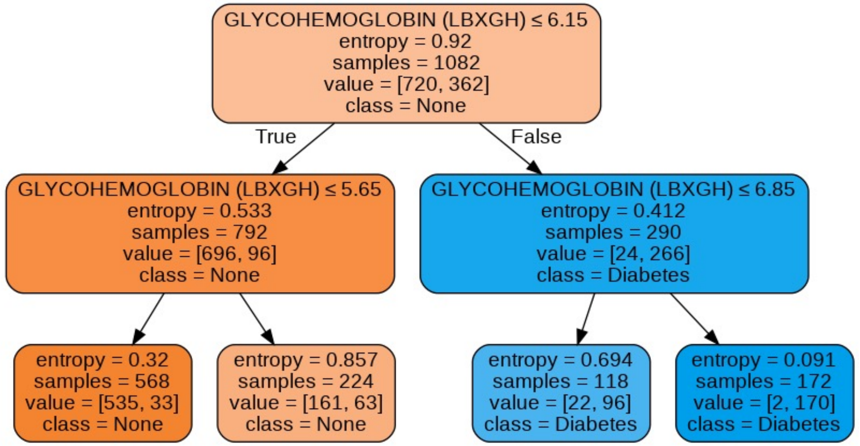
    return T





Diabetes Prediction

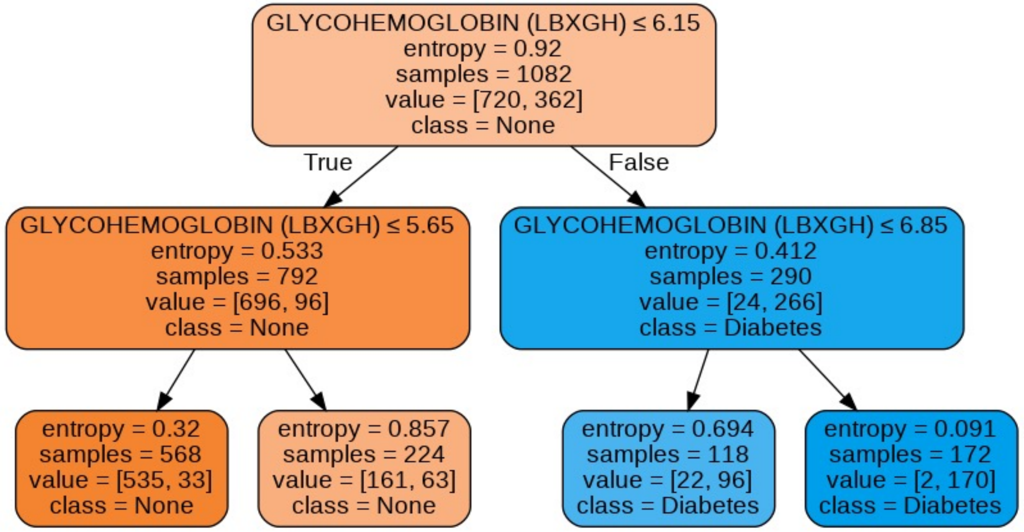




Training accuracy: 100%
Test accuracy: 82.8%

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