



# Decision Trees, Bagging and Boosting

CSE 446: Machine Learning  
created by Emily Fox

# Predicting potential loan defaults

# What makes a loan risky?



Credit History



Income



Term



Personal Info



# Credit history explained

Did I pay previous  
loans on time?



**Example:** excellent,  
good, or fair

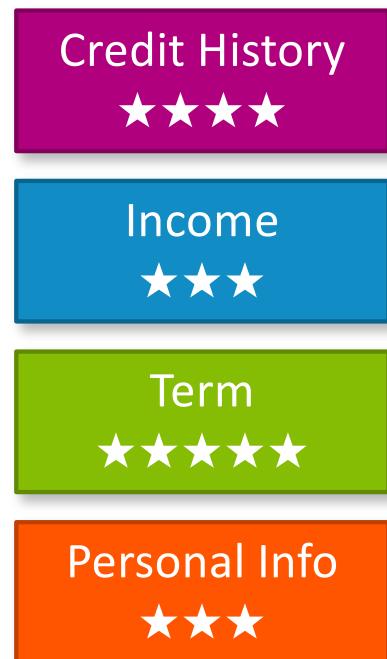


# Income

What's my income?

**Example:**

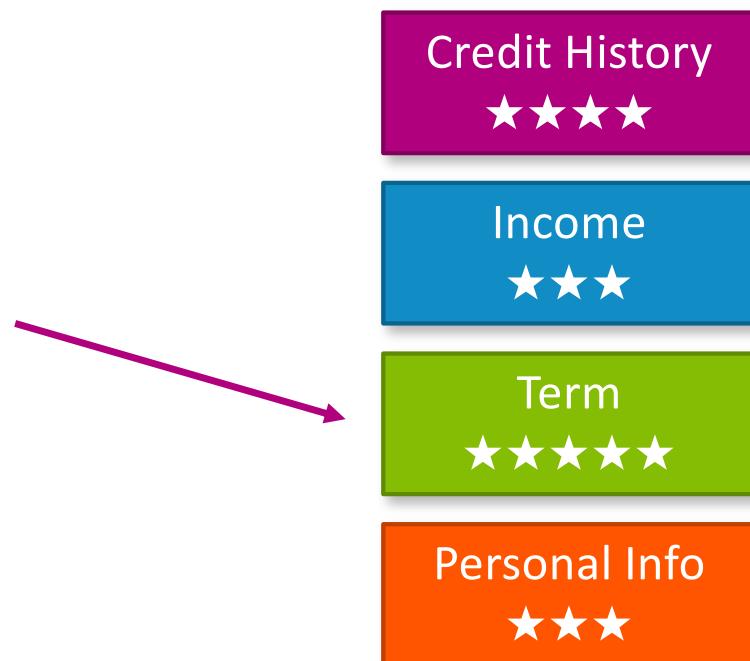
\$80K per year



# Loan terms

How soon do I need to pay the loan?

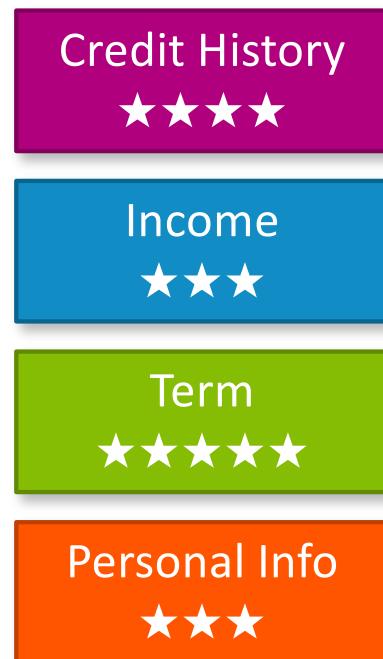
**Example:** 3 years,  
5 years,...



# Personal information

Age, reason for the loan,  
marital status,...

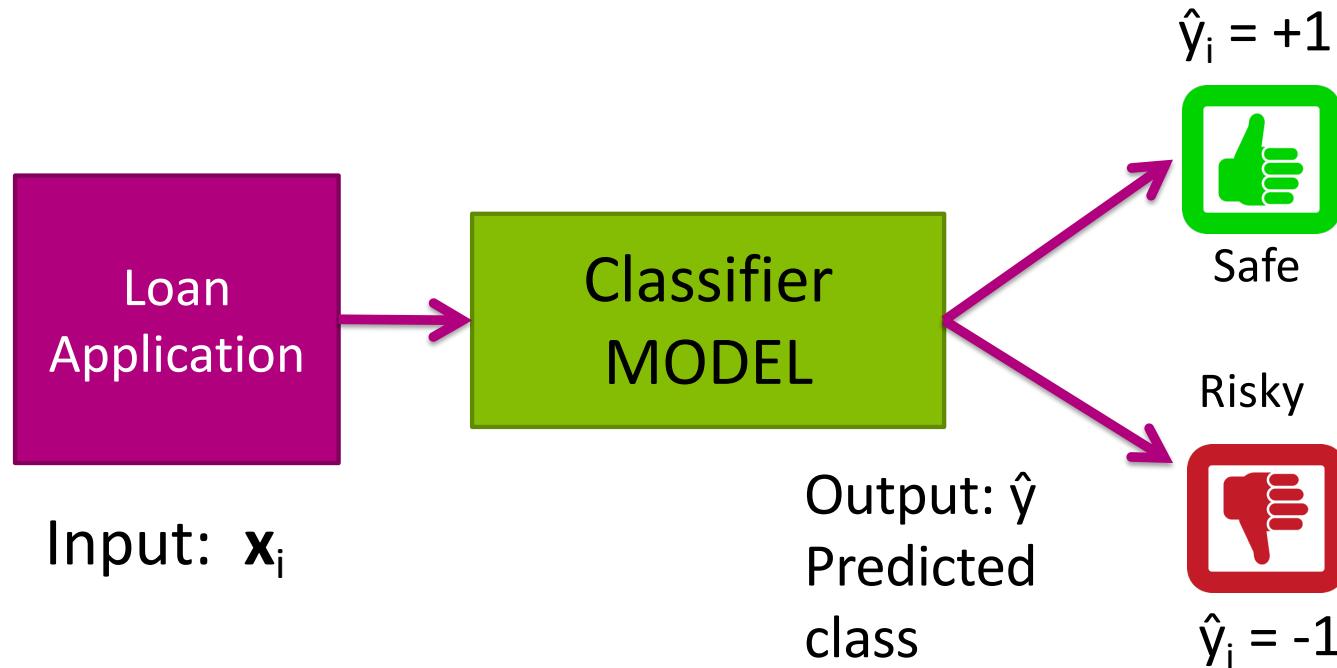
**Example:** Home loan for a  
married couple



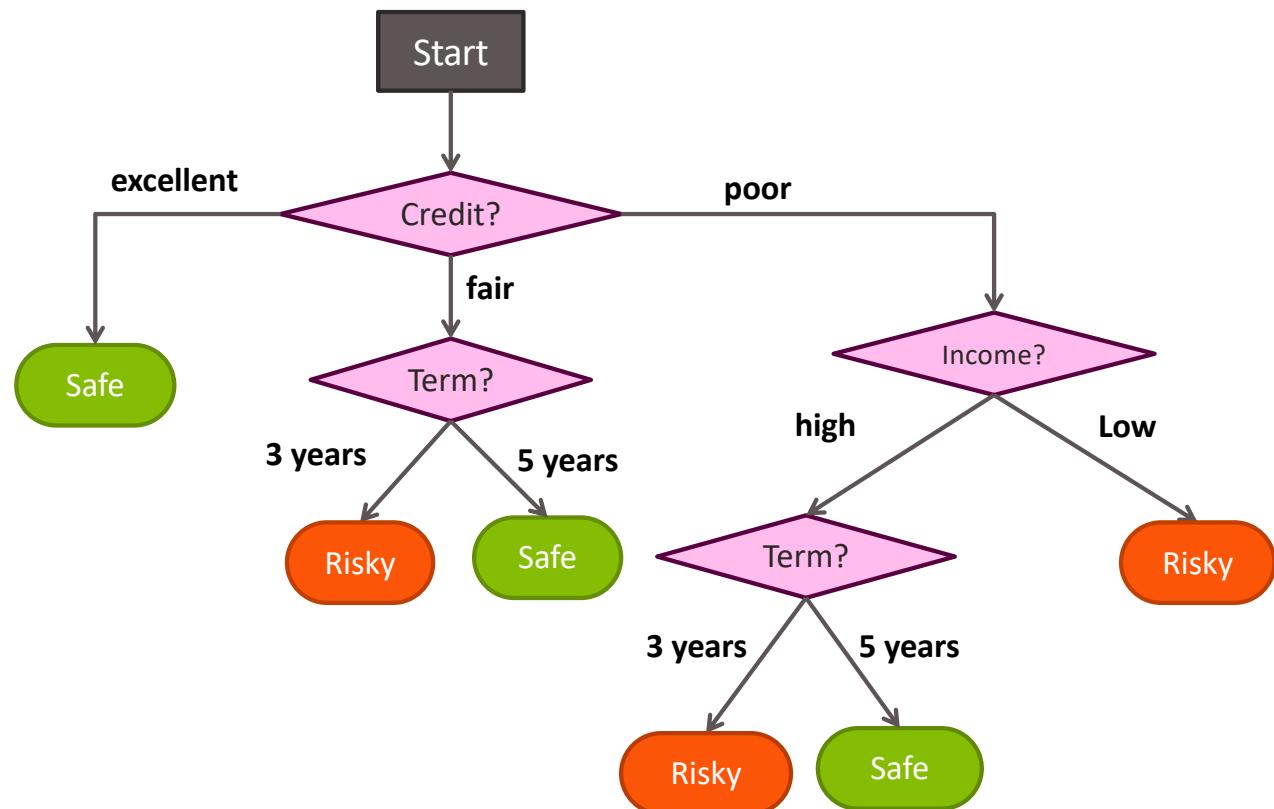
# Intelligent application



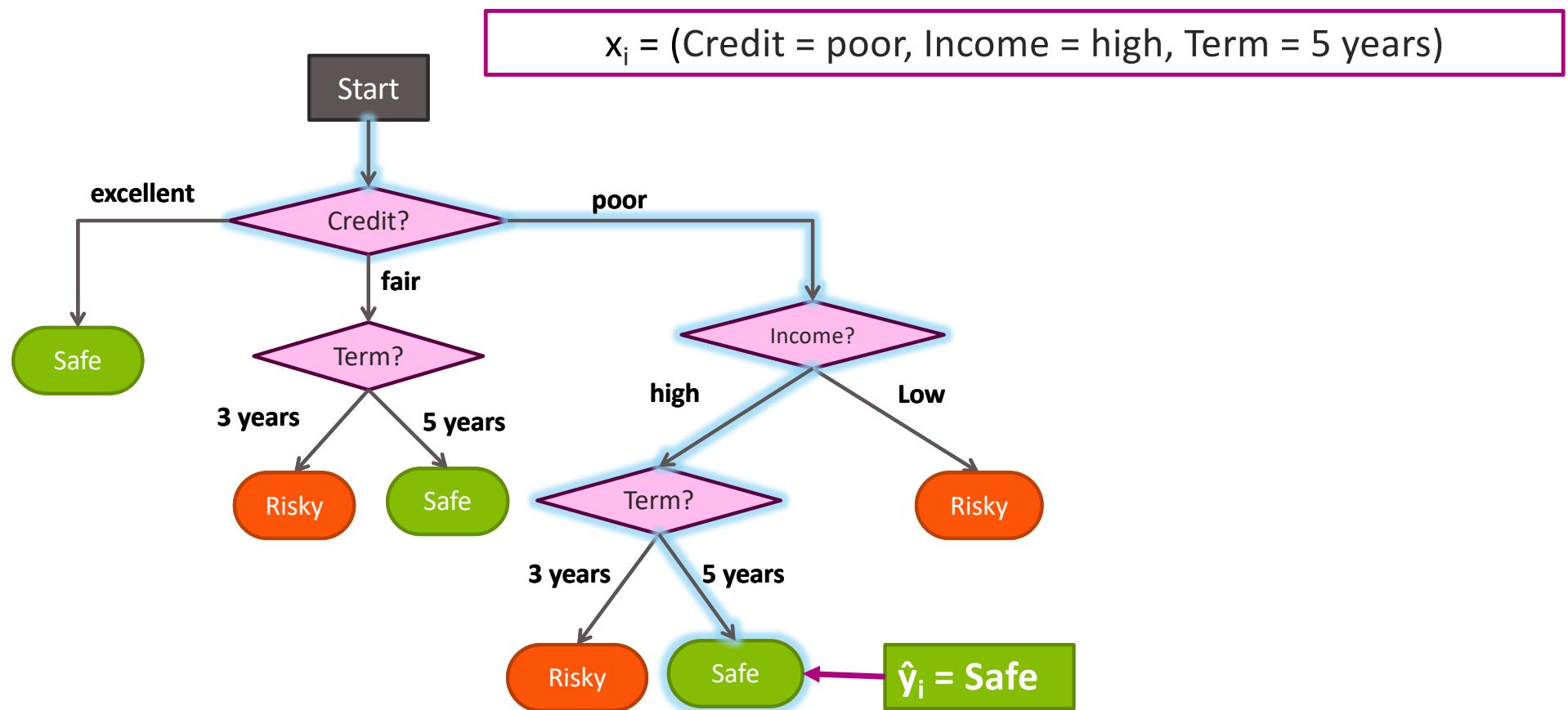
# Classifier review



# This module ... decision trees



# Scoring a loan application

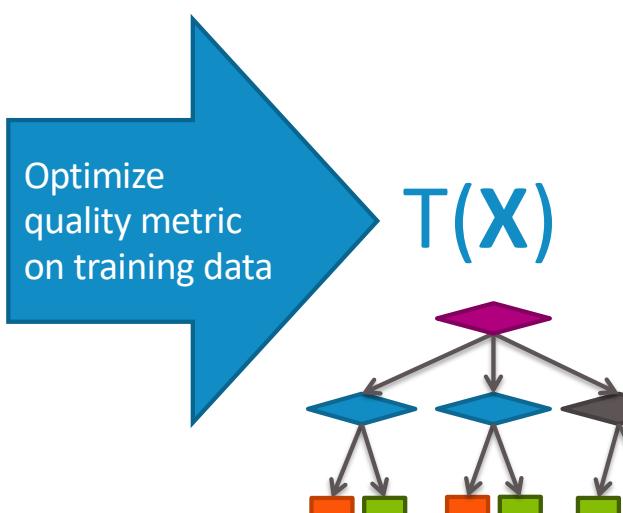


# Decision tree learning task

# Decision tree learning problem

Training data:  $N$  observations  $(x_i, y_i)$

Credit	Term	Income	y
excellent	3 yrs	high	safe
fair	5 yrs	low	risky
fair	3 yrs	high	safe
poor	5 yrs	high	risky
excellent	3 yrs	low	risky
fair	5 yrs	low	safe
poor	3 yrs	high	risky
poor	5 yrs	low	safe
fair	3 yrs	high	safe



# Quality metric: Classification error

- Error measures fraction of mistakes

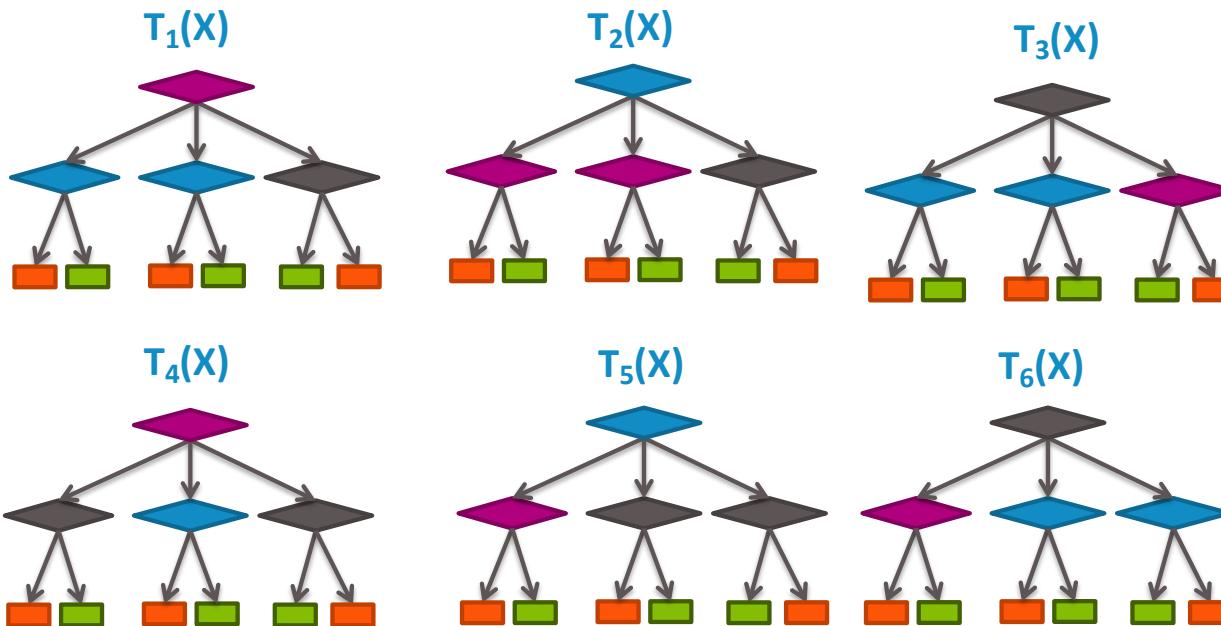
$$\text{Error} = \frac{\text{\# incorrect predictions}}{\text{\# examples}}$$

- Best possible value : 0.0
- Worst possible value: 1.0

# How do we find the best tree?

Exponentially large number of possible trees makes decision tree learning hard!

Learning the smallest decision tree is an *NP-hard problem* [Hyafil & Rivest '76]



# Greedy decision tree learning

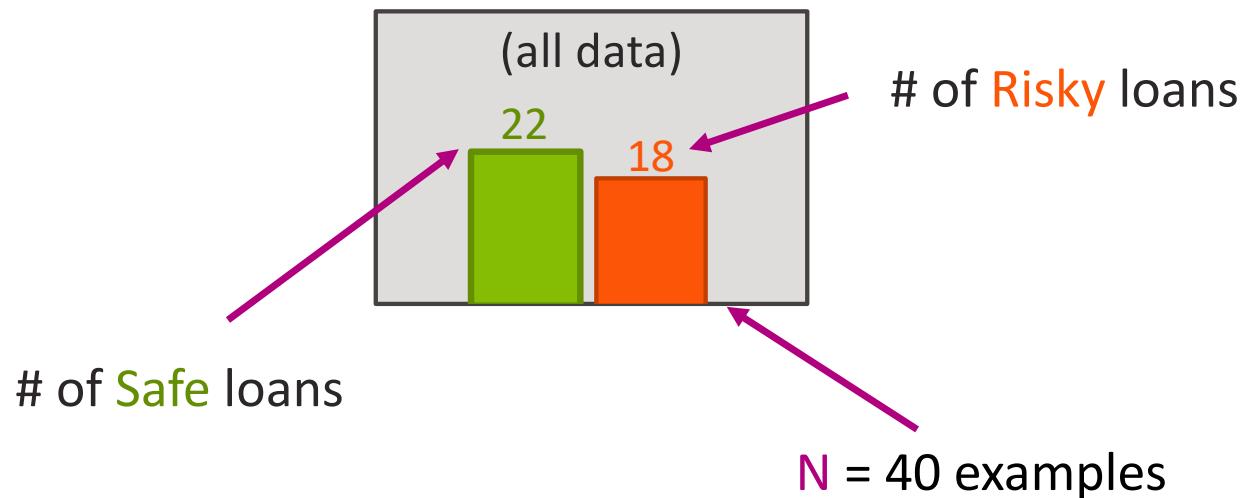
# Our training data table

Assume  $N = 40$ , 3 features

Credit	Term	Income	y
excellent	3 yrs	high	safe
fair	5 yrs	low	risky
fair	3 yrs	high	safe
poor	5 yrs	high	risky
excellent	3 yrs	low	risky
fair	5 yrs	low	safe
poor	3 yrs	high	risky
poor	5 yrs	low	safe
fair	3 yrs	high	safe

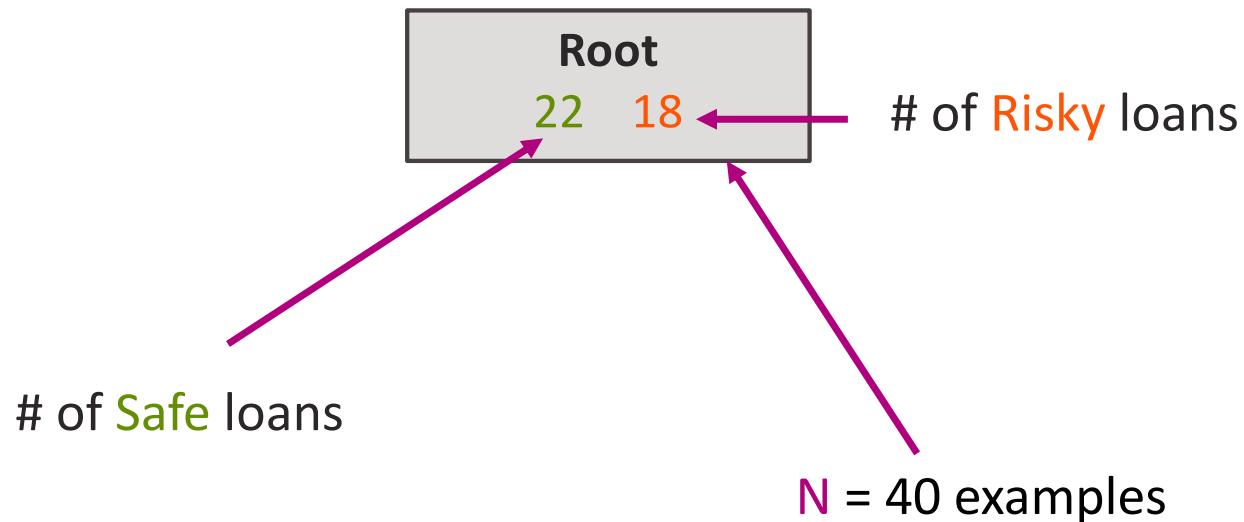
# Start with all the data

Loan status: **Safe** **Risky**



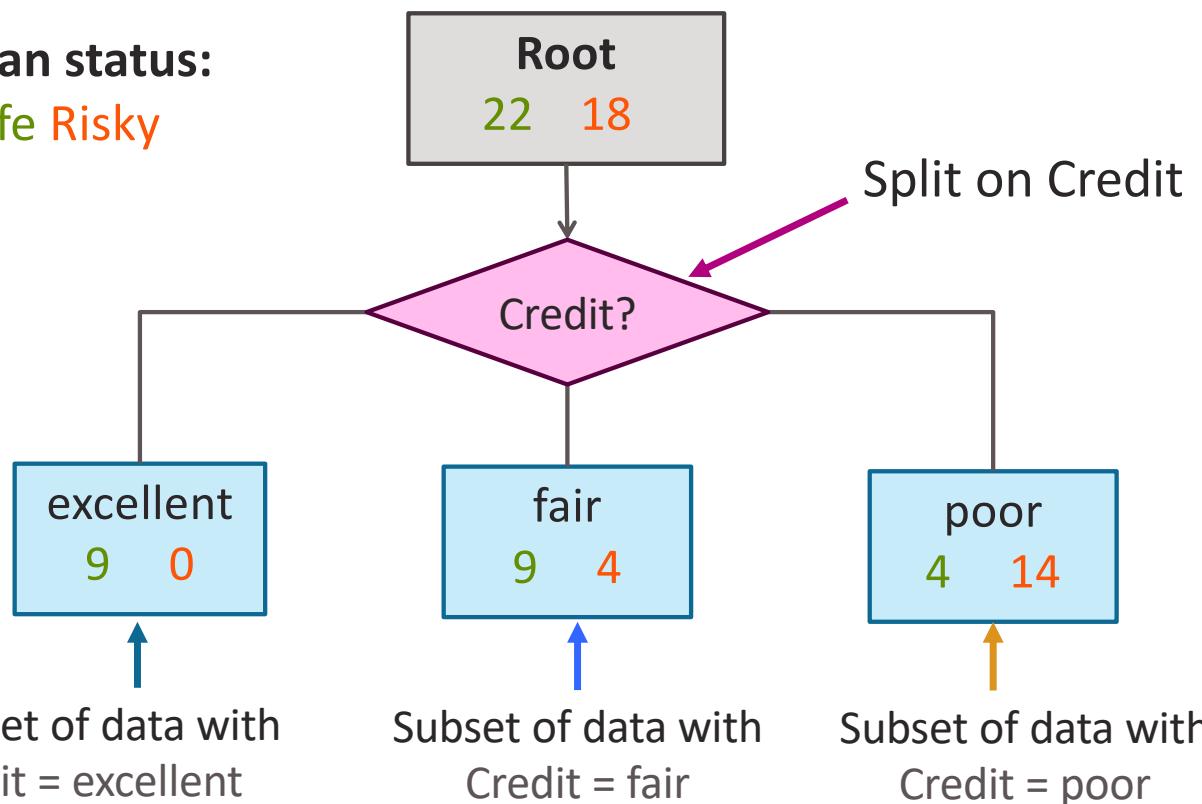
# Compact visual notation: Root node

Loan status: **Safe** **Risky**



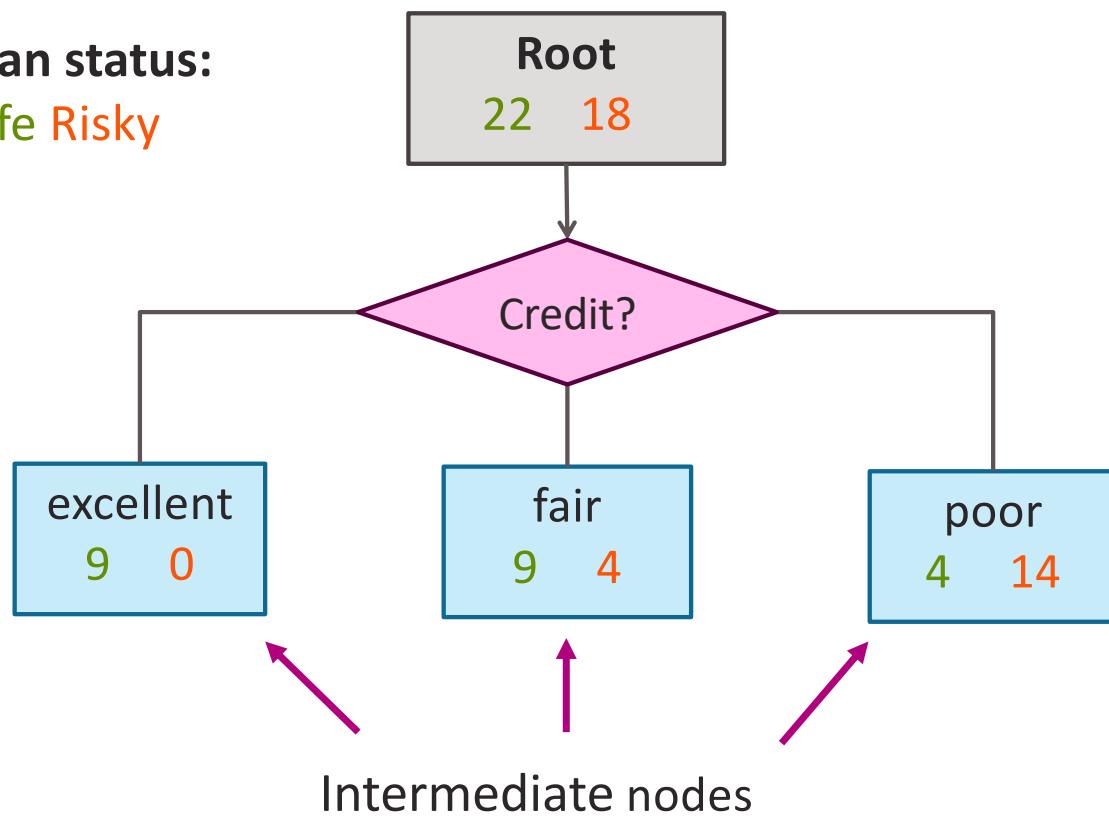
# Decision stump: Single level tree

Loan status:  
Safe Risky

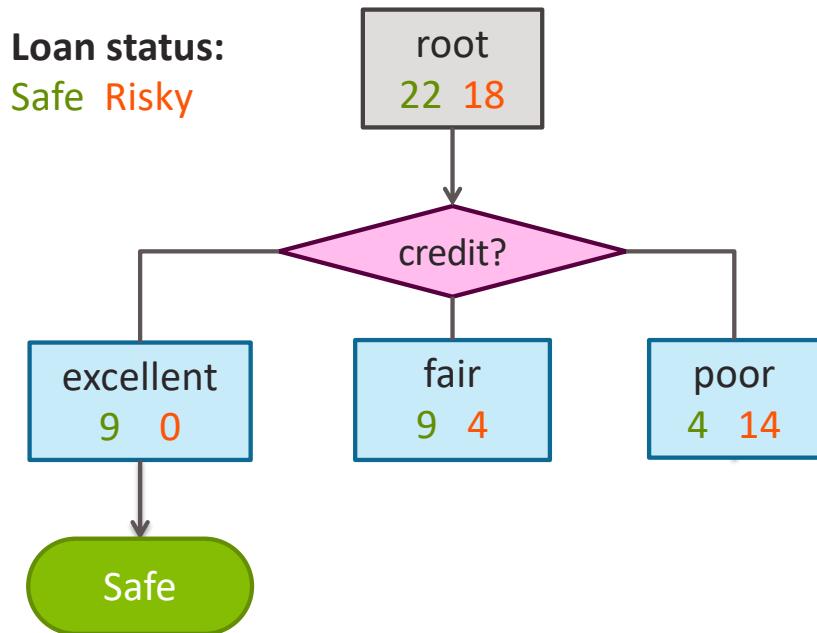


# Visual notation: Intermediate nodes

Loan status:  
Safe Risky



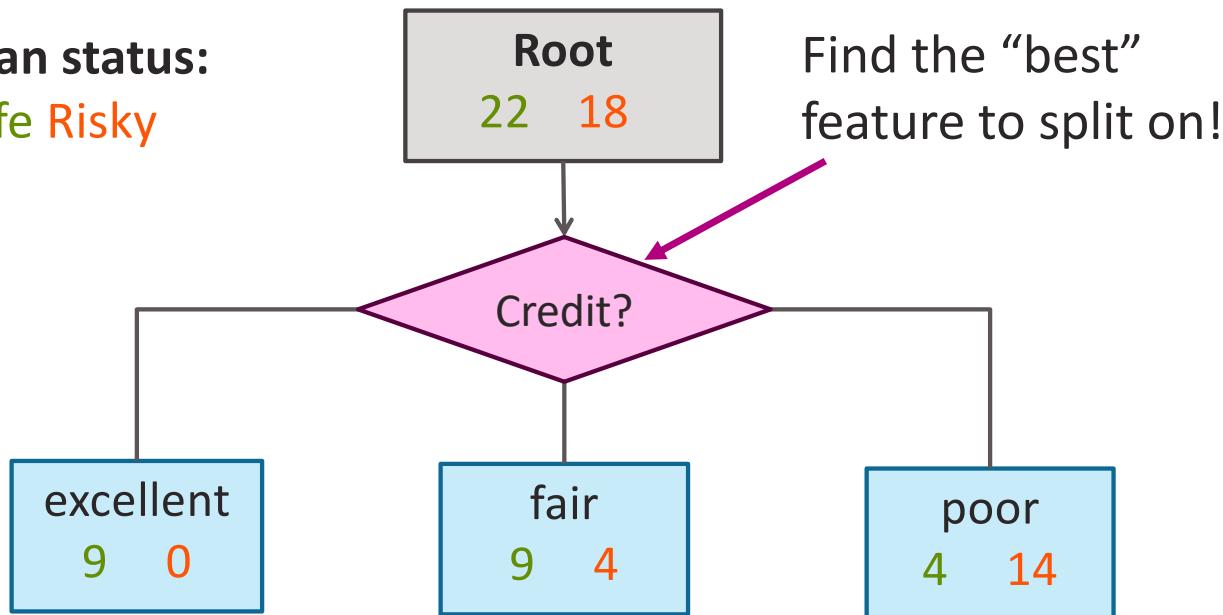
# Proceed to build a tree on each of children



# Selecting best feature to split on

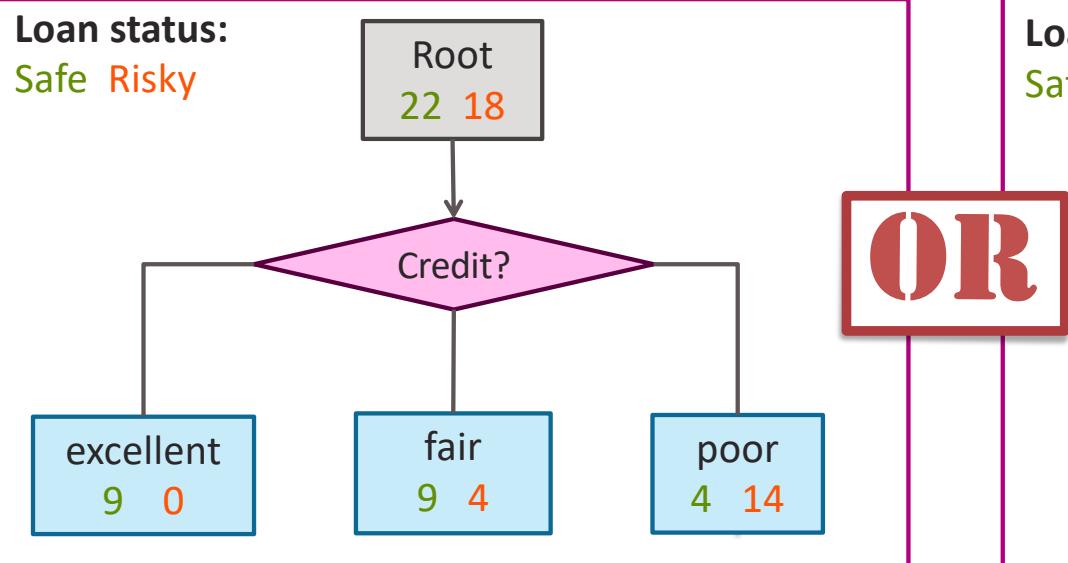
# How do we pick our first split?

Loan status:  
Safe Risky

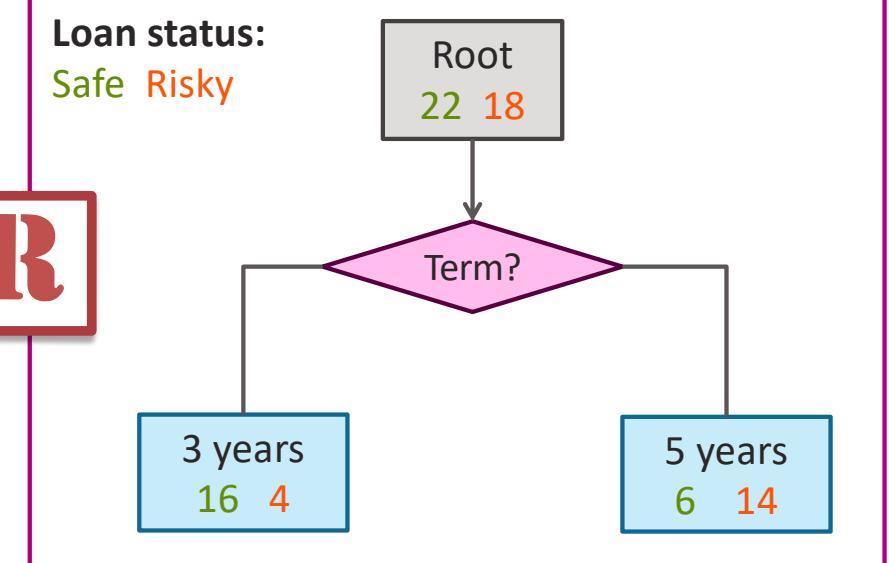


# How do we select the best feature?

**Choice 1: Split on Credit**

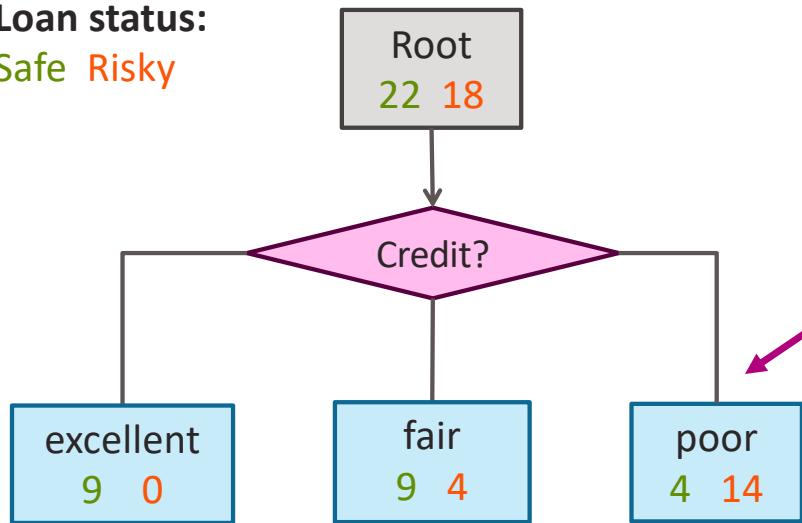


**Choice 2: Split on Term**



# How do we measure effectiveness of a split?

Loan status:  
Safe Risky

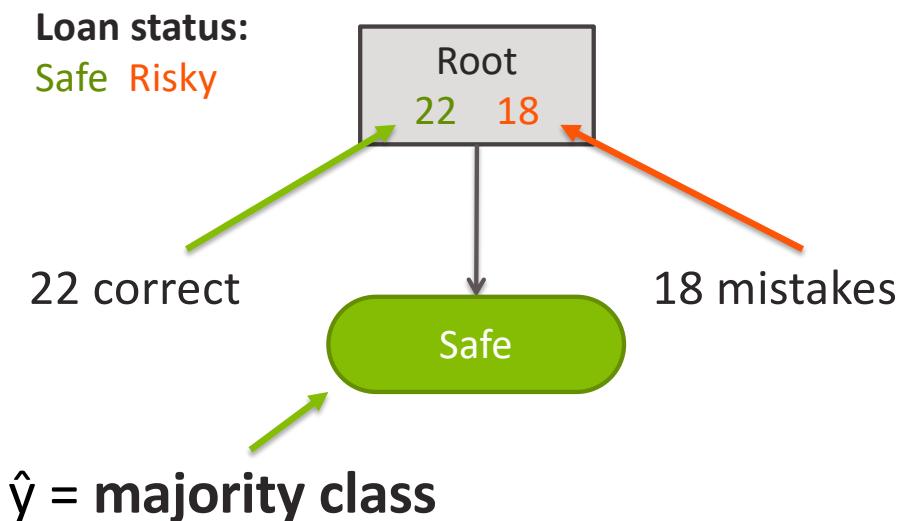


Idea: Calculate classification error  
of this decision stump

$$\text{Error} = \frac{\# \text{ mistakes}}{\# \text{ data points}}$$

# Calculating classification error

- **Step 1:**  $\hat{y}$  = class of majority of data in node
- **Step 2:** Calculate classification error of predicting  $\hat{y}$  for this data

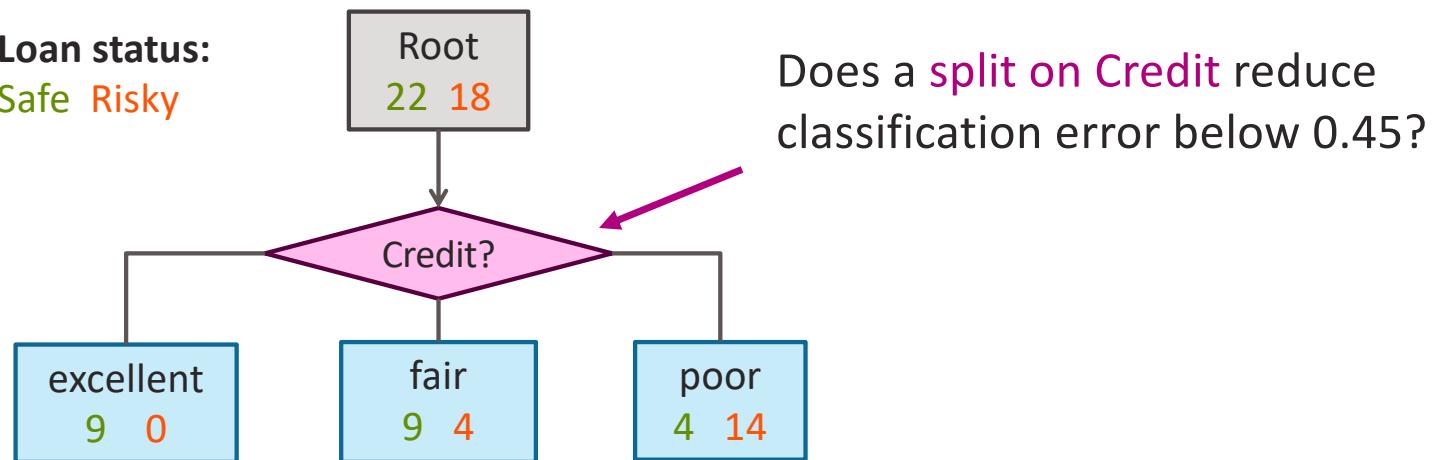


Tree	Classification error
(root)	0.45

# Choice 1: Split on Credit history?

## Choice 1: Split on Credit

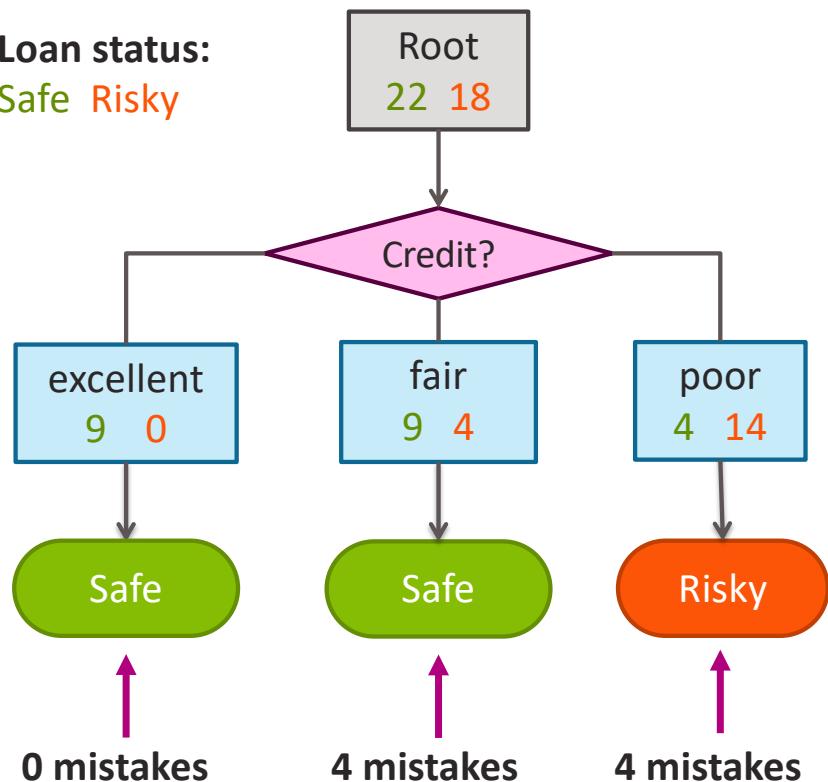
Loan status:  
Safe Risky



# Split on Credit: Classification error

## Choice 1: Split on Credit

Loan status:  
Safe Risky

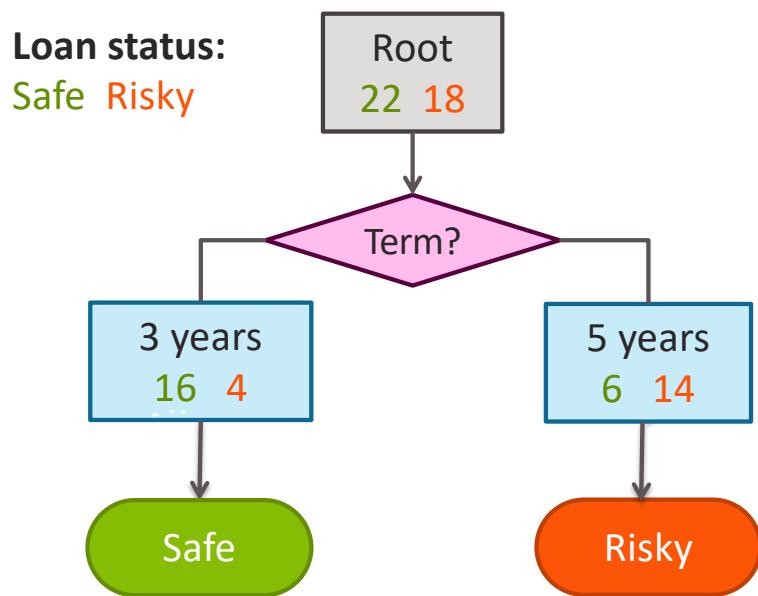


$$\text{Error} = \frac{4+4}{22 + 18} = 0.2$$

Tree	Classification error
(root)	0.45
Split on credit	0.2

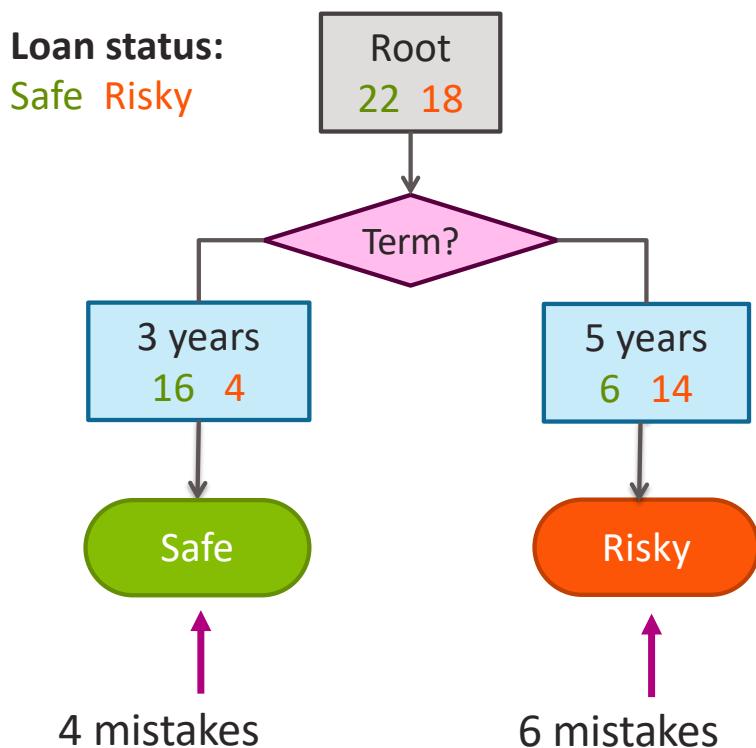
# Choice 2: Split on Term?

Choice 2: Split on Term



# Evaluating the split on Term

## Choice 2: Split on Term



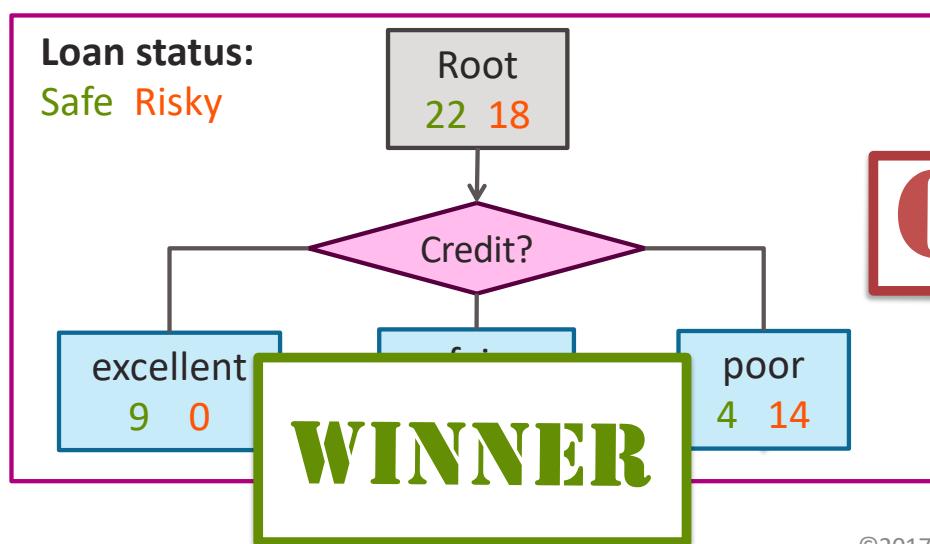
$$\text{Error} = \frac{4+6}{40} = 0.25$$

Tree	Classification error
(root)	0.45
Split on credit	0.2
Split on term	0.25

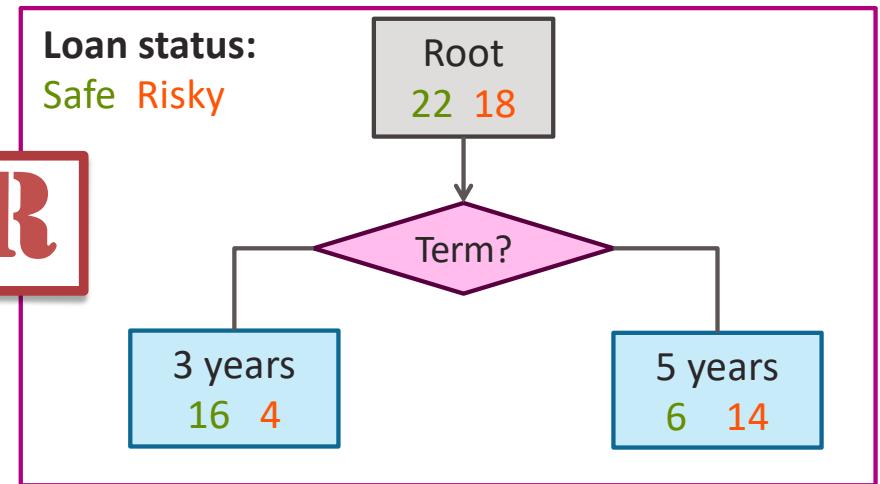
# Choice 1 vs Choice 2: Comparing split on **Credit** vs **Term**

Tree	Classification error
(root)	0.45
split on credit	0.2
split on loan term	0.25

Choice 1: Split on Credit



Choice 2: Split on Term



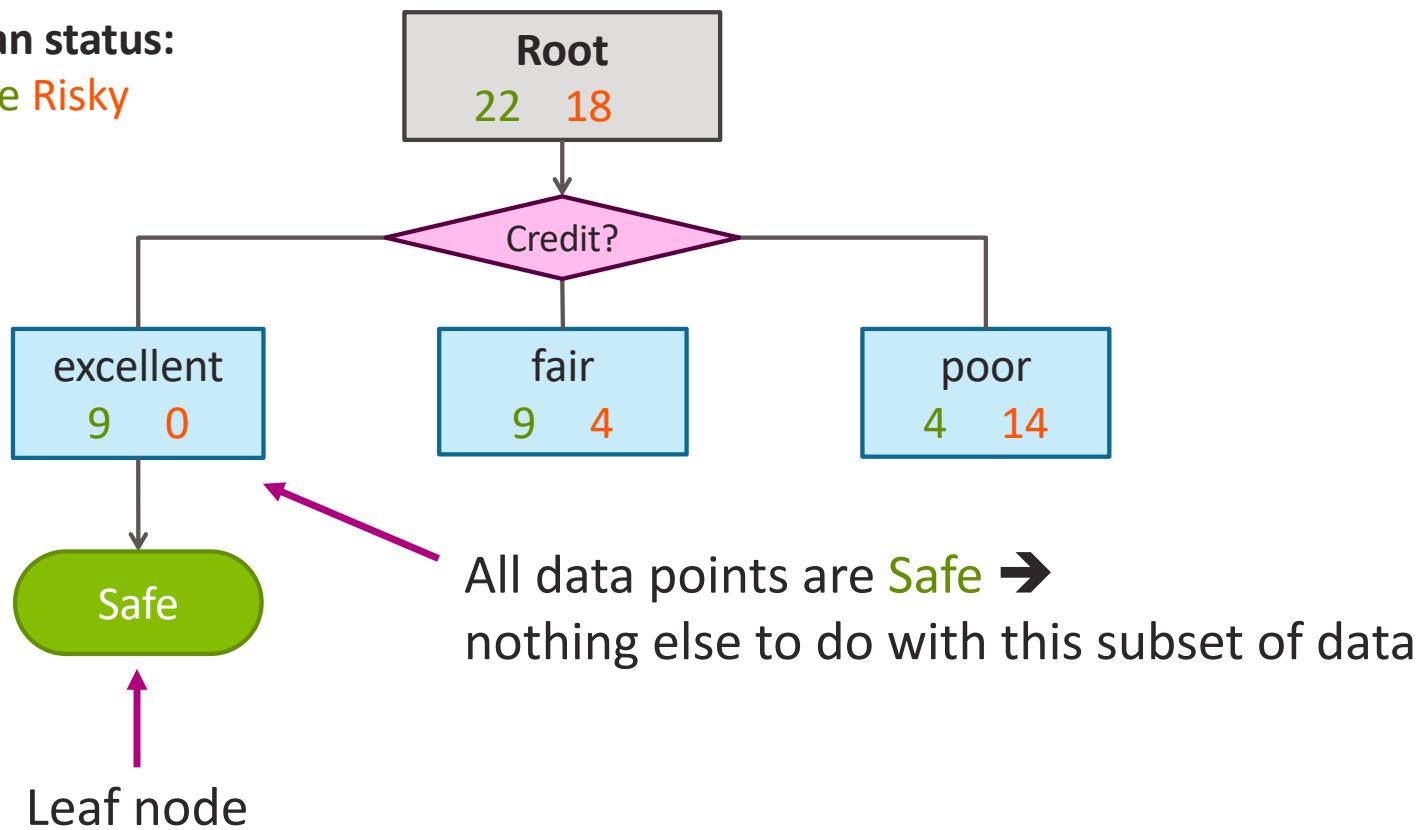
# Feature split selection algorithm

- Given a subset of data  $M$  (a node in a tree)
- For each feature  $\phi_i(x)$ :
  1. Split data of  $M$  according to feature  $\phi_i(x)$
  2. Compute classification error split
- Choose feature  $\phi^*(x)$  with lowest classification error

# Recursion & Stopping conditions

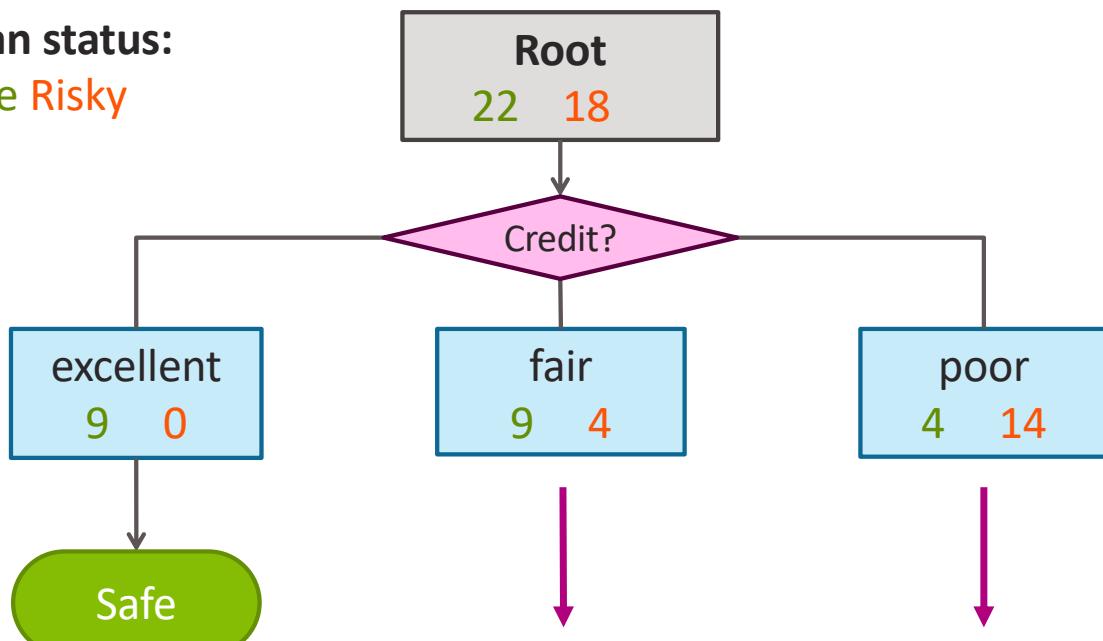
# We've learned a decision stump, what next?

Loan status:  
Safe Risky



# Tree learning = Recursive stump learning

Loan status:  
Safe Risky

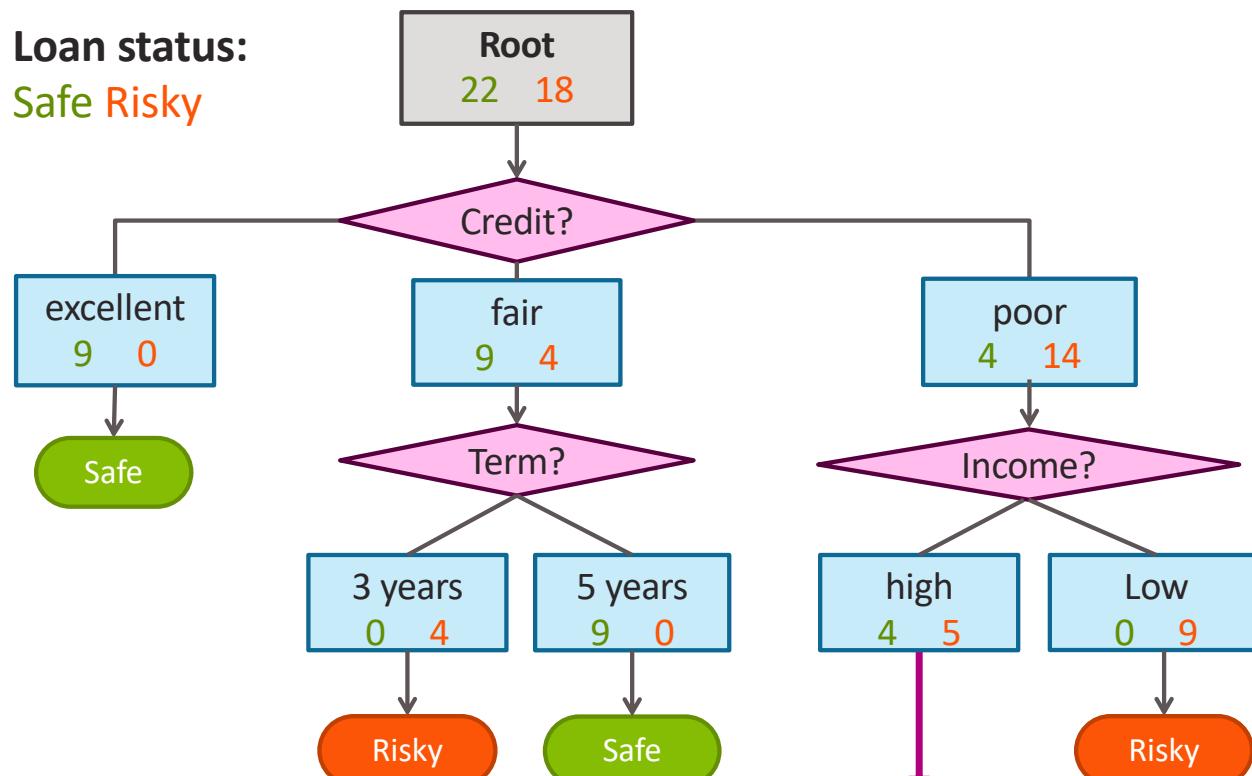


Build decision stump with  
subset of data where  
Credit = fair

Build decision stump with  
subset of data where  
Credit = poor

# Second level

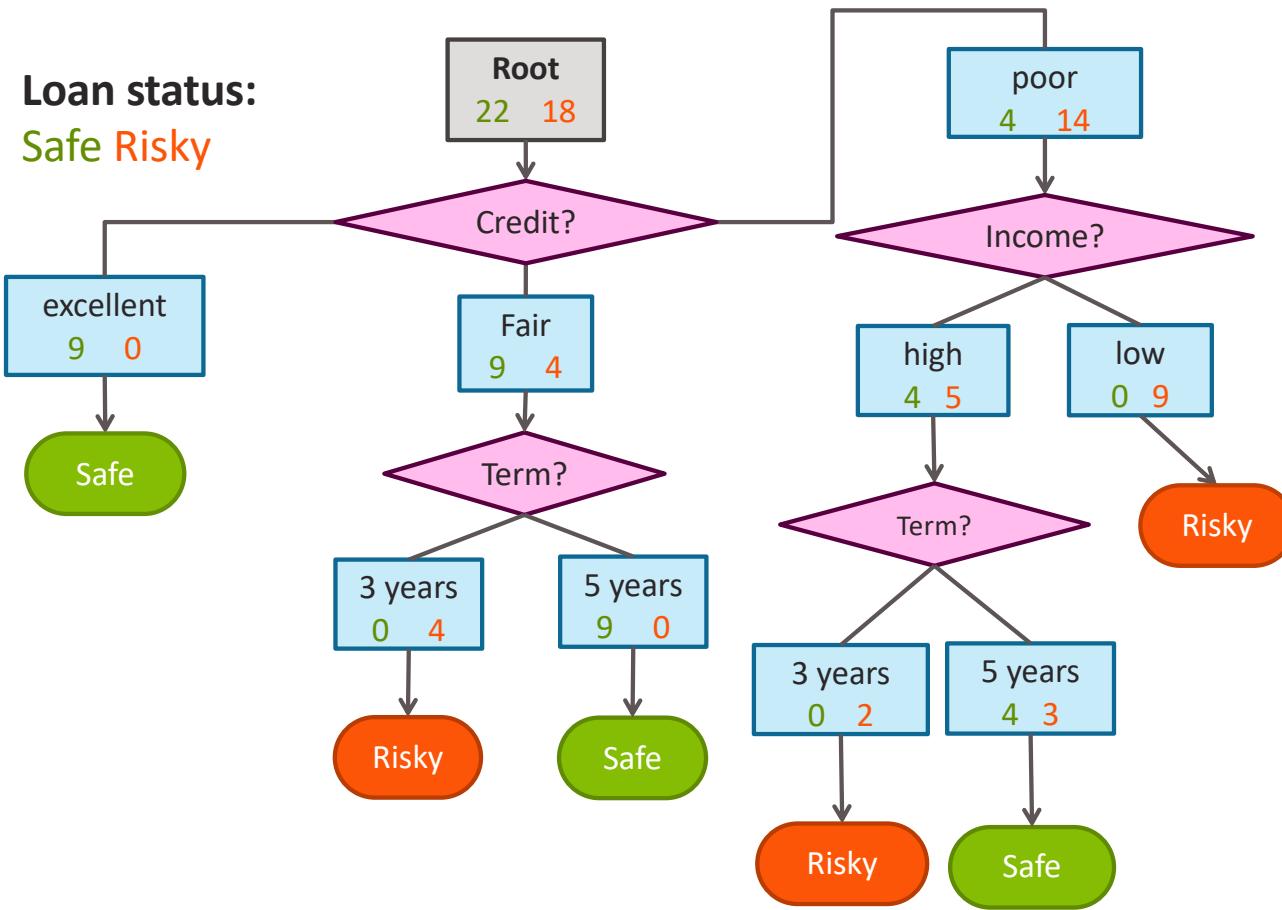
Loan status:  
Safe Risky



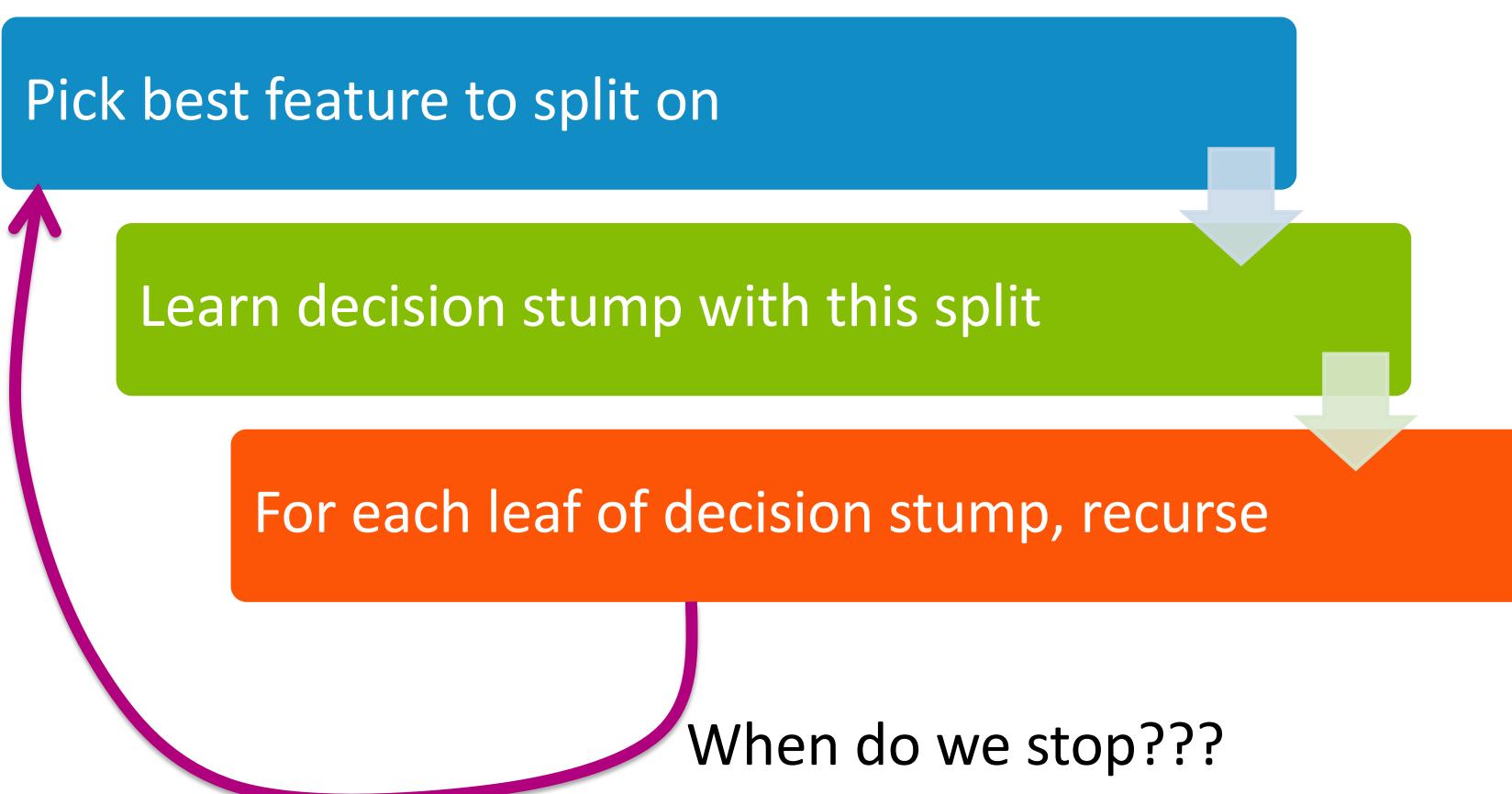
Build another stump  
these data points

©2017 Emily Fox

# Final decision tree

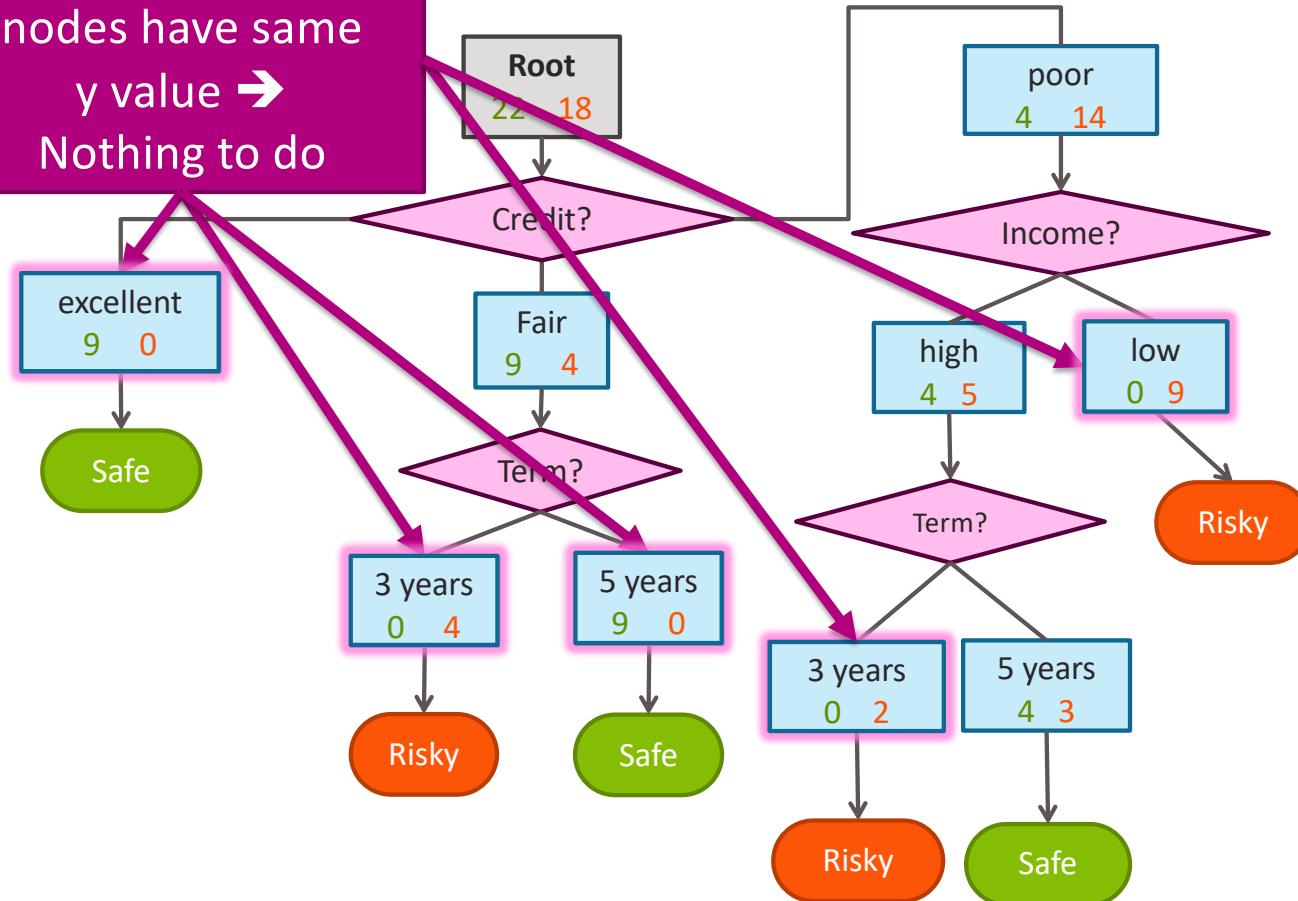


# Simple greedy decision tree learning



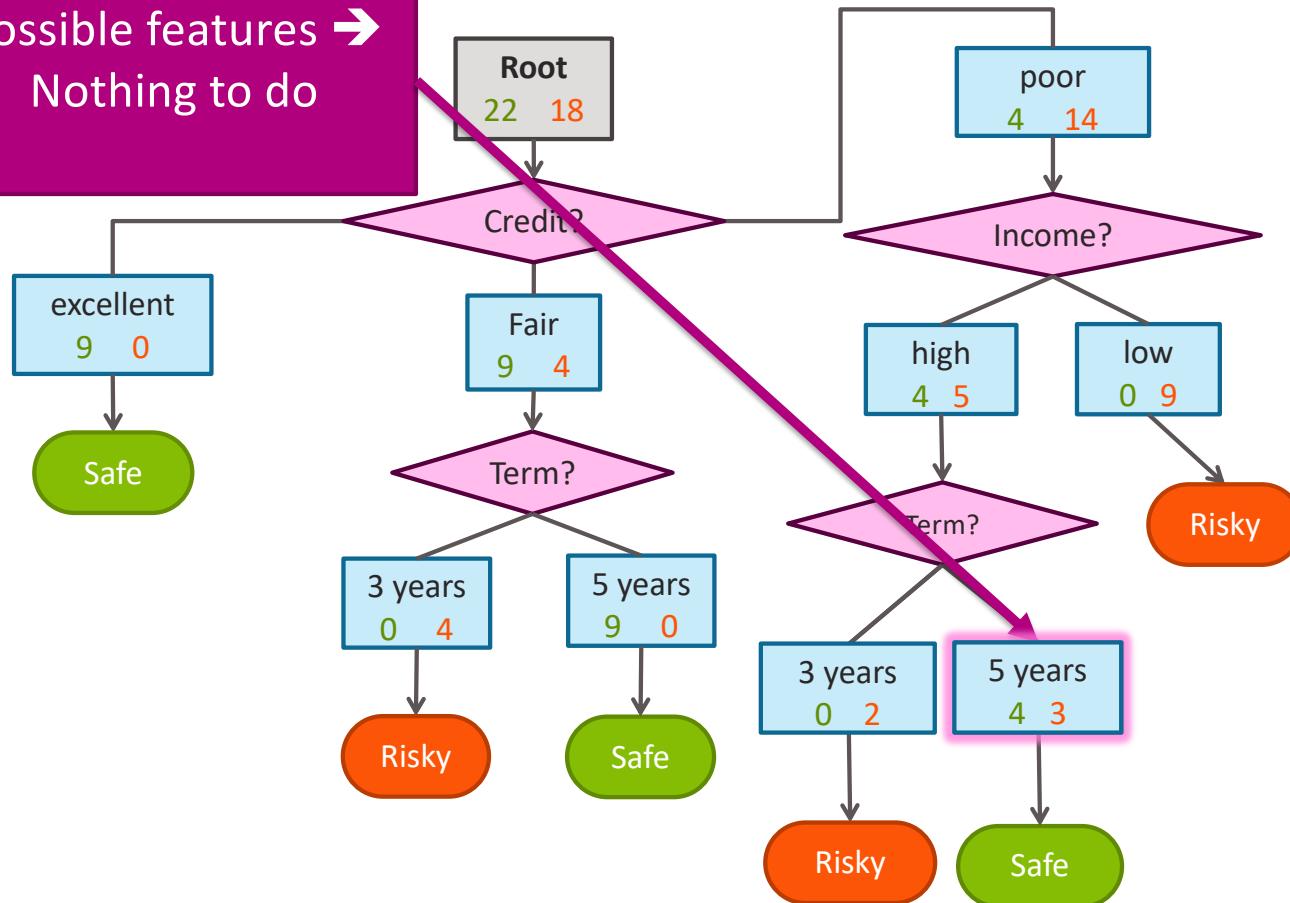
# Stopping condition 1: All data agrees on y

All data in these nodes have same y value → Nothing to do



# Stopping condition 2: Already split on all features

Already split on all possible features →  
Nothing to do



# Greedy decision tree learning

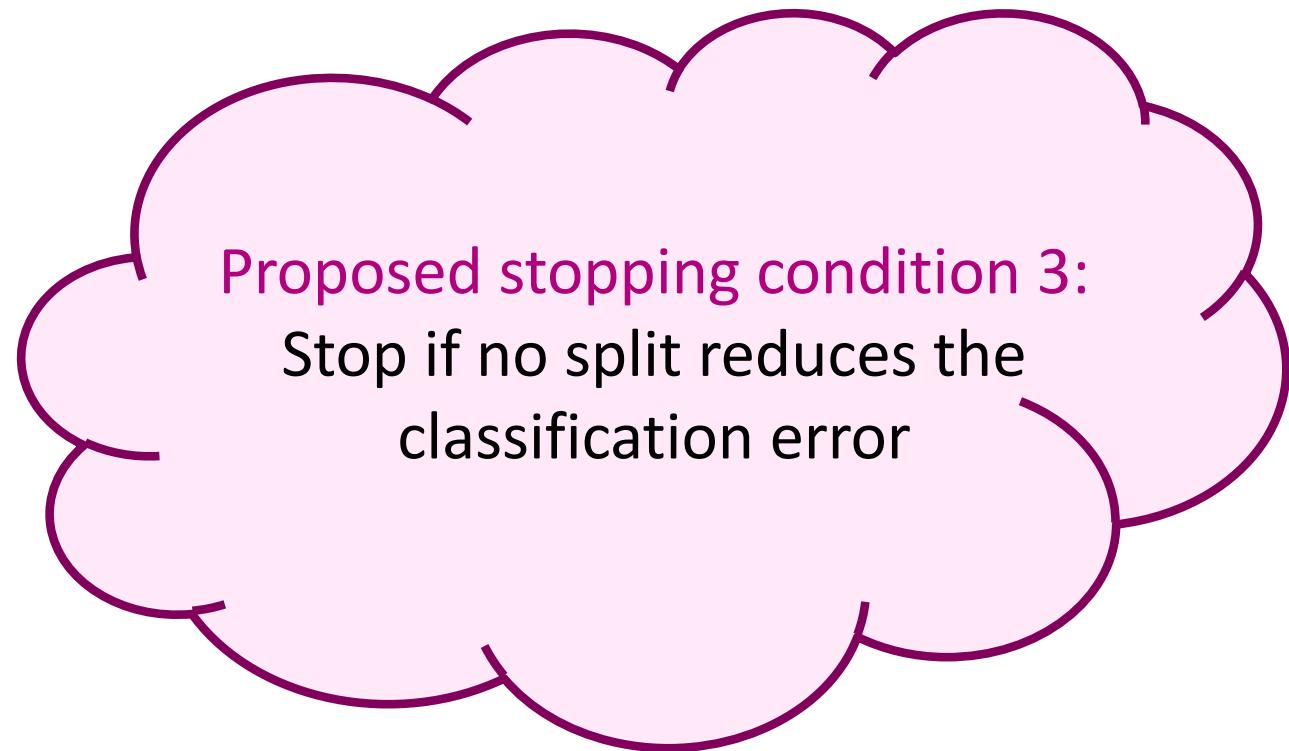
- Step 1: Start with an empty tree
- Step 2: Select a feature to split data
- For each split of the tree:
  - Step 3: If nothing more to do, make predictions
  - Step 4: Otherwise, go to Step 2 & continue (recurse) on this split

Pick feature split leading to lowest classification error

Stopping conditions 1 & 2

Recursion

# Is this a good idea?



Proposed stopping condition 3:  
Stop if no split reduces the  
classification error

Stopping condition 3:  
Don't stop if error doesn't decrease???

$$y = x[1] \text{ xor } x[2]$$

x[1]	x[2]	y
False	False	False
False	True	True
True	False	True
True	True	False

y values  
True   False



$$\text{Error} = \frac{2}{4} = 0.5$$

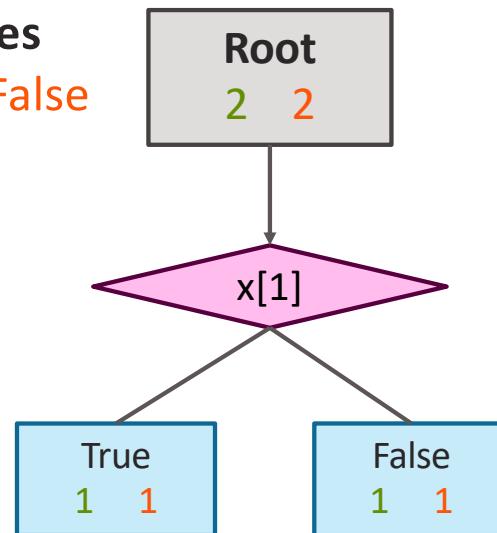
Tree	Classification error
(root)	0.5

# Consider split on $x[1]$

$$y = x[1] \text{ xor } x[2]$$

$x[1]$	$x[2]$	$y$
False	False	False
False	True	True
True	False	True
True	True	False

$y$  values  
True   False



$$\text{Error} = \frac{2}{4} = 0.5$$

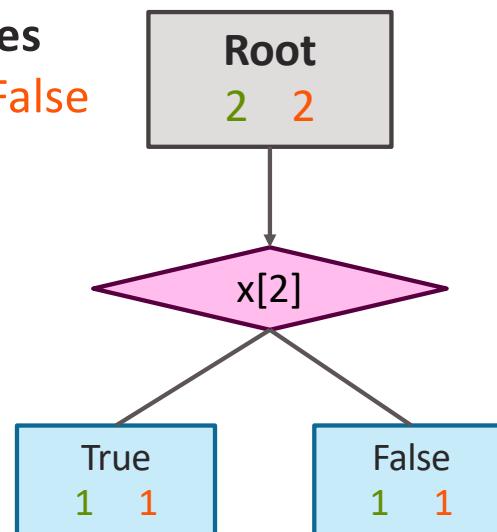
Tree	Classification error
(root)	0.5
Split on $x[1]$	0.5

# Consider split on $x[2]$

$$y = x[1] \text{ xor } x[2]$$

$x[1]$	$x[2]$	$y$
False	False	False
False	True	True
True	False	True
True	True	False

$y$  values  
True   False



$$\text{Error} = \frac{2}{4} = 0.5$$

Neither features improve training error...  
Stop now???

→

Tree	Classification error
(root)	0.5
Split on $x[1]$	0.5
Split on $x[2]$	0.5

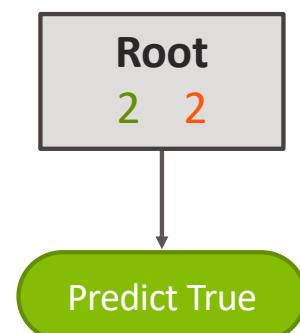
©2017 Emily Fox

# Final tree with stopping condition 3

$$y = x[1] \text{ xor } x[2]$$

x[1]	x[2]	y
False	False	False
False	True	True
True	False	True
True	True	False

y values  
True   False



Tree	Classification error
with stopping condition 3	0.5

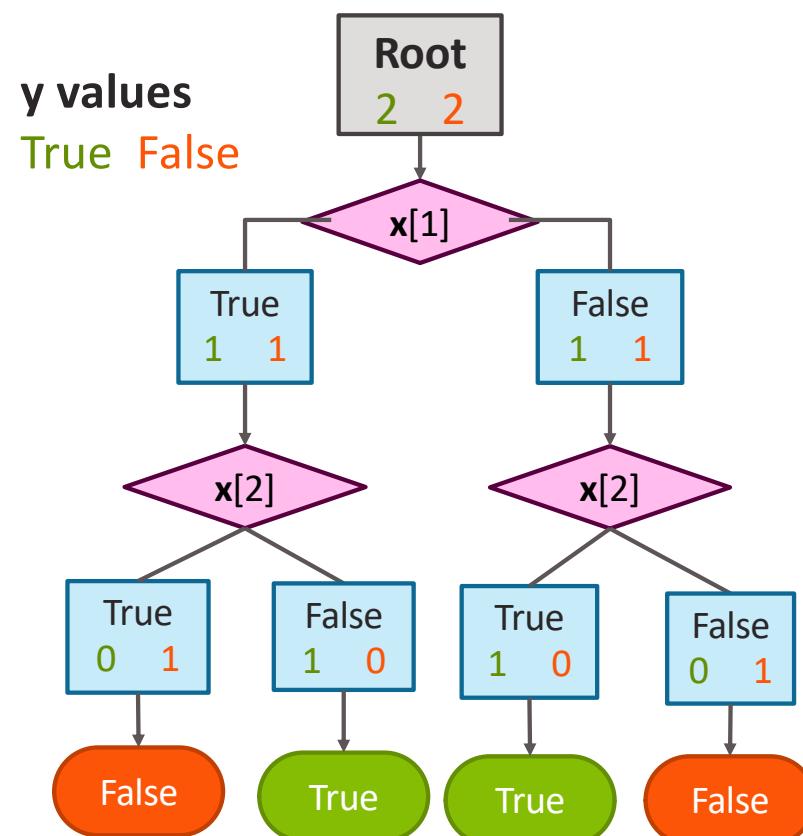
# Without stopping condition 3

Condition 3 (stopping when training error doesn't improve) not always recommended!

$$y = x[1] \text{ xor } x[2]$$

x[1]	x[2]	y
False	False	False
False	True	True
True	False	True
True	True	False

Tree	Classification error
with stopping condition 3	0.5
without stopping condition 3	0



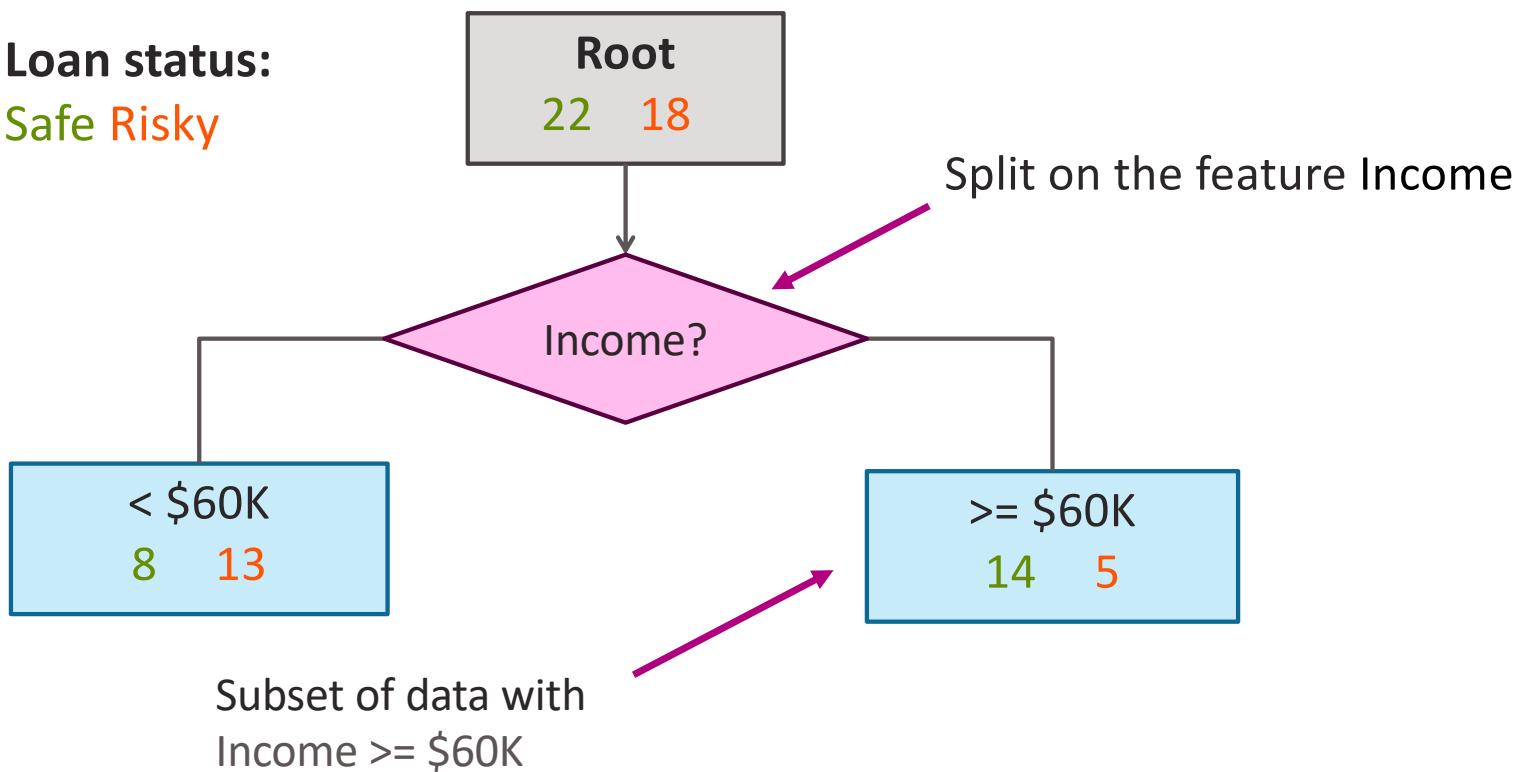
# Decision tree learning: *Real valued features*

# How do we use real values inputs?

Income	Credit	Term	y
\$105 K	excellent	3 yrs	Safe
\$112 K	good	5 yrs	Risky
\$73 K	fair	3 yrs	Safe
\$69 K	excellent	5 yrs	Safe
\$217 K	excellent	3 yrs	Risky
\$120 K	good	5 yrs	Safe
\$64 K	fair	3 yrs	Risky
\$340 K	excellent	5 yrs	Safe
\$60 K	good	3 yrs	Risky

# Threshold split

Loan status:  
Safe Risky



# Finding the best threshold split

Infinite possible  
values of  $t$



Income =  $t^*$

Income <  $t^*$

Income  $\geq t^*$

Income

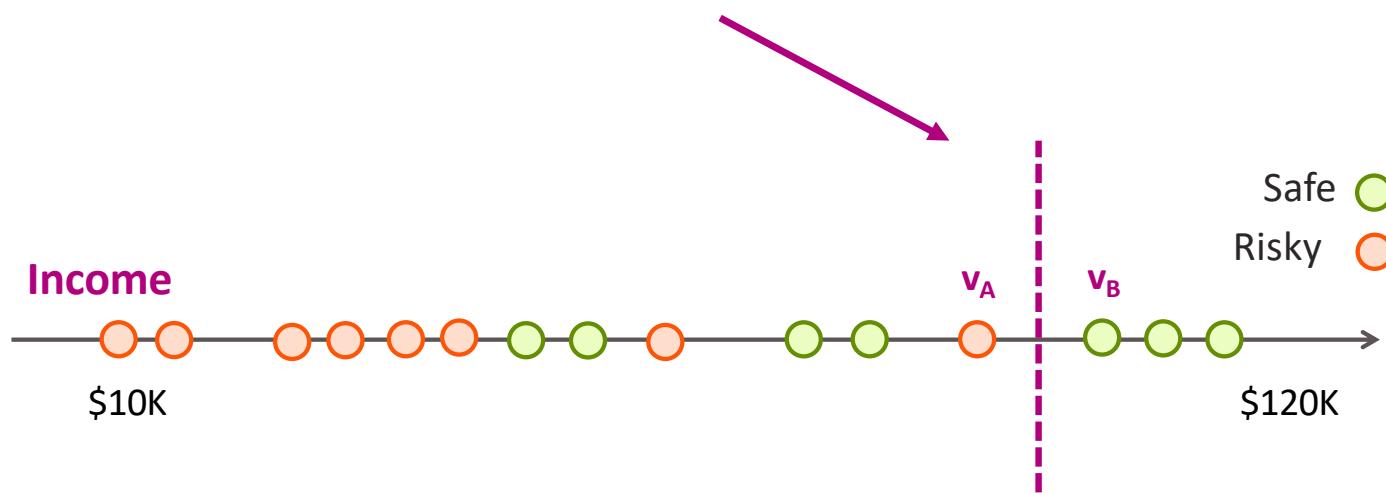
\$10K

Safe   
Risky

\$120K

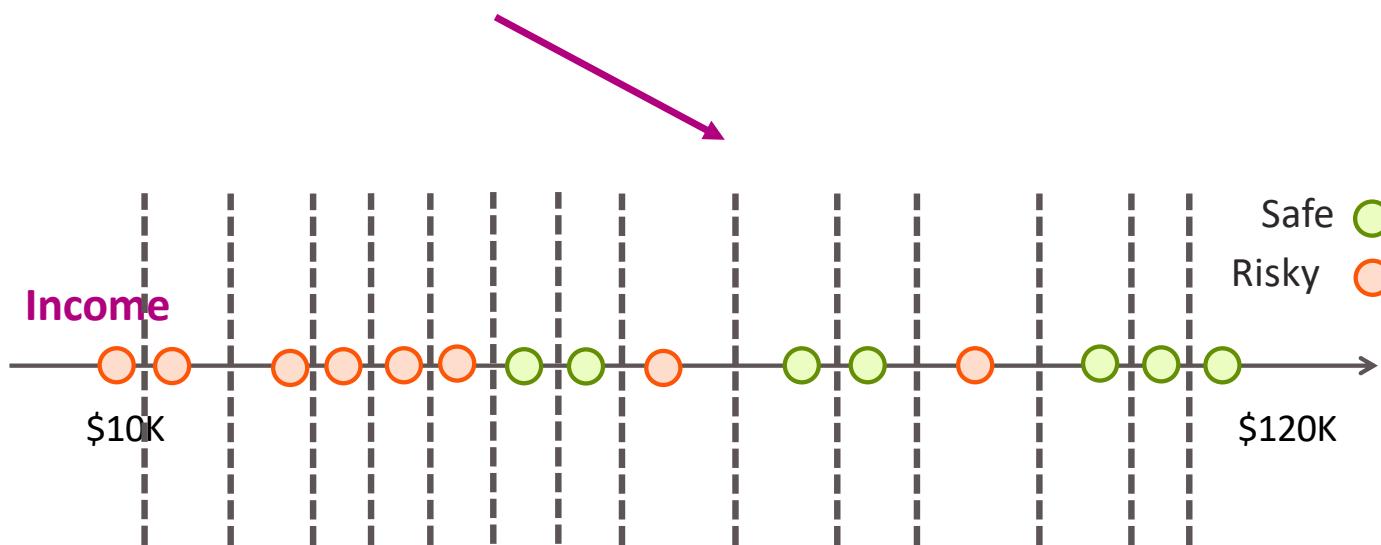
# Consider a threshold between points

Same **classification error** for any threshold split between  $v_A$  and  $v_B$



# Only need to consider mid-points

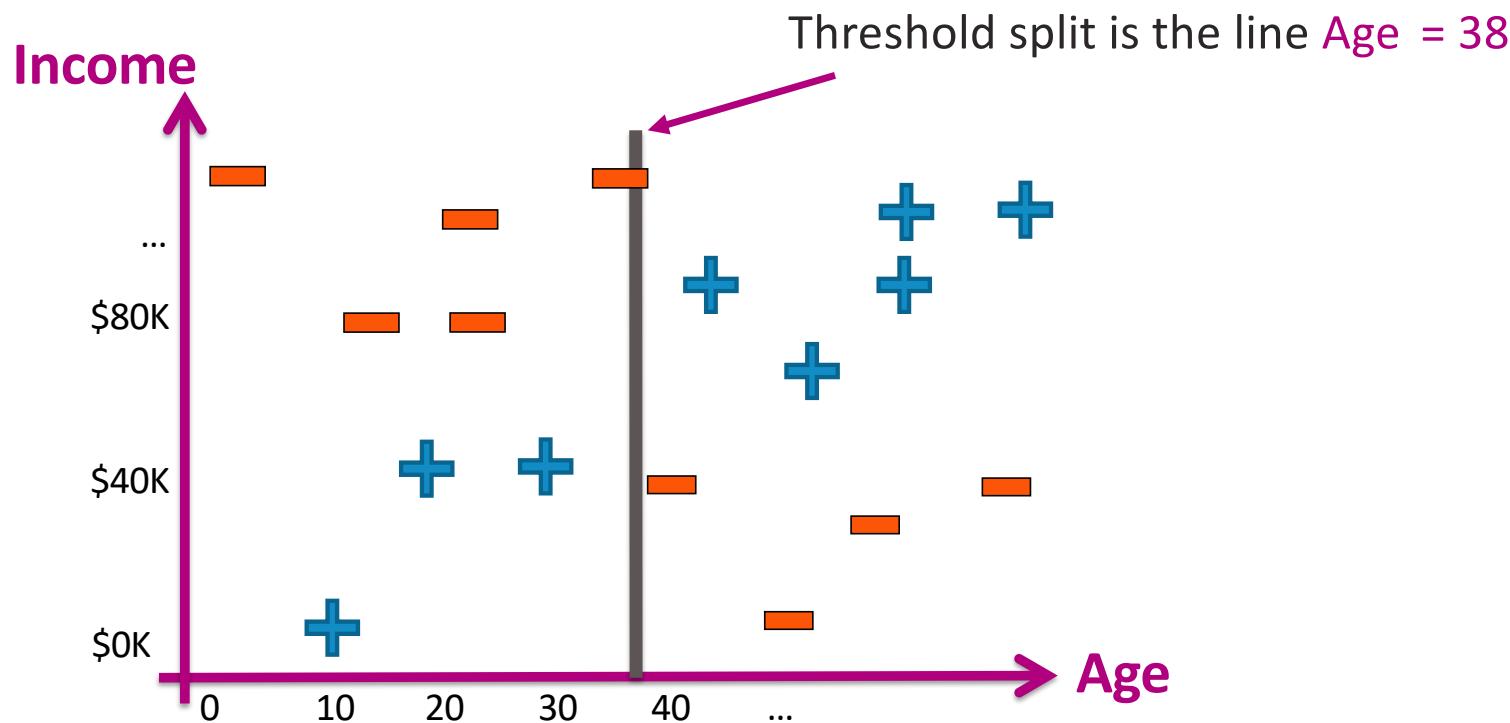
Finite number of splits  
to consider



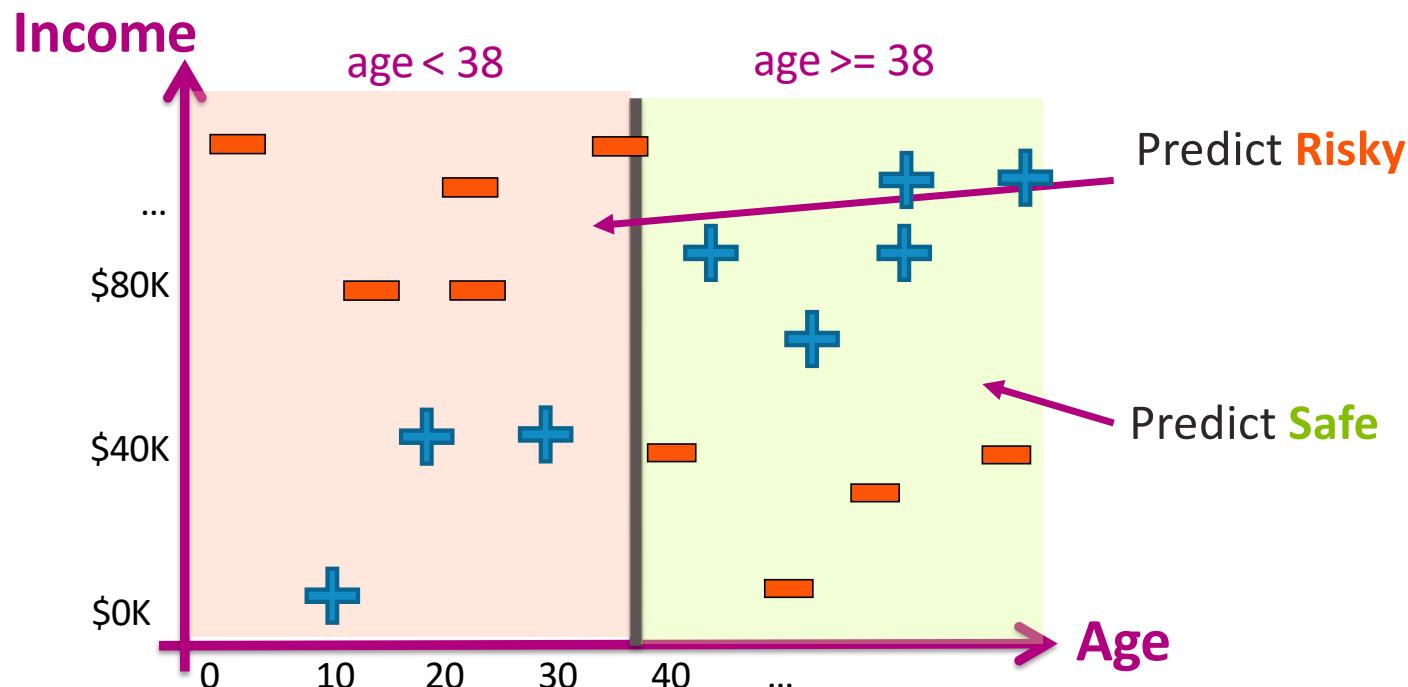
# Threshold split selection algorithm

- **Step 1:** Sort the values of a feature  $h_j(x)$  :  
Let  $\{v_1, v_2, v_3, \dots, v_N\}$  denote sorted values
- **Step 2:**
  - For  $i = 1 \dots N-1$ 
    - Consider split  $t_i = (v_i + v_{i+1}) / 2$
    - Compute classification error for threshold split  $h_j(x) \geq t_i$
  - Choose the  $t^*$  with the lowest classification error

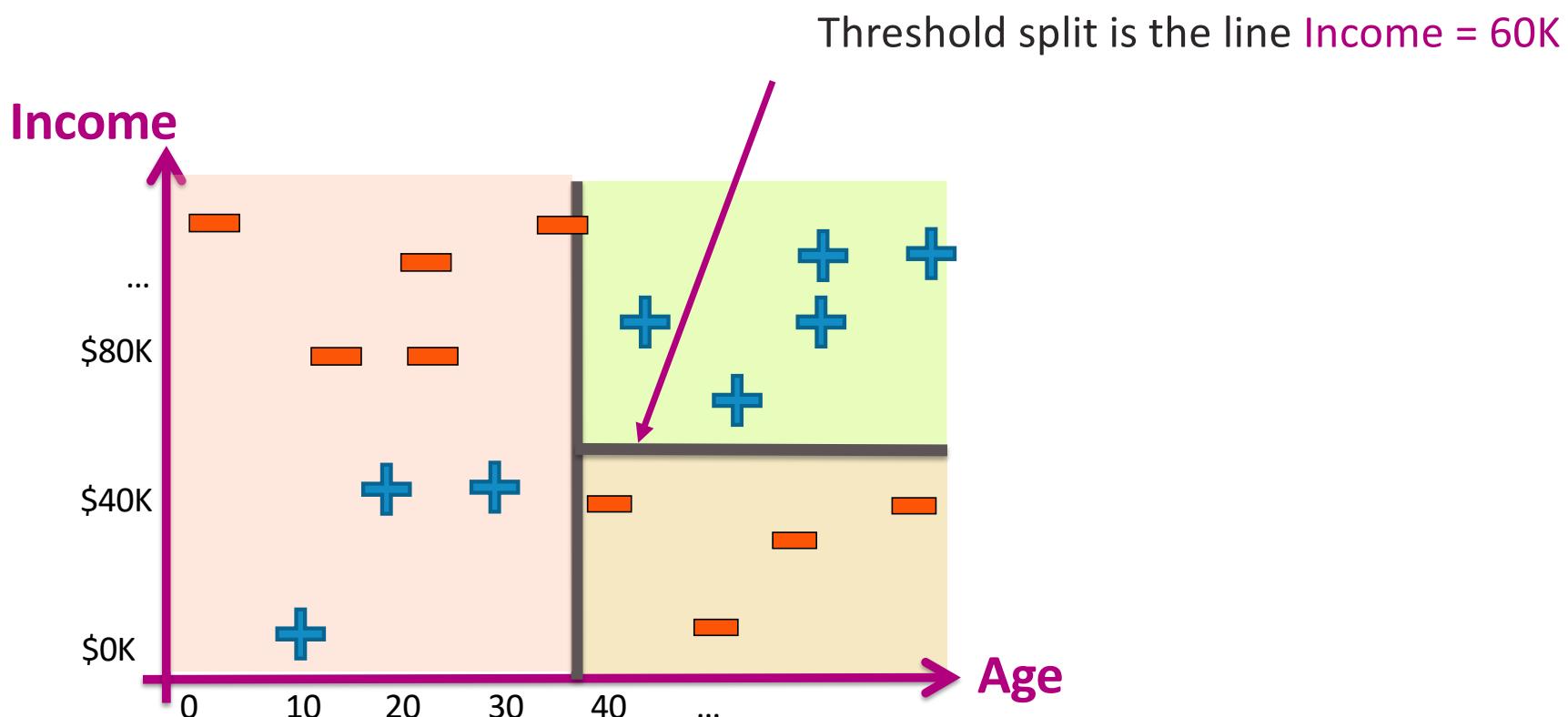
# Visualizing the threshold split



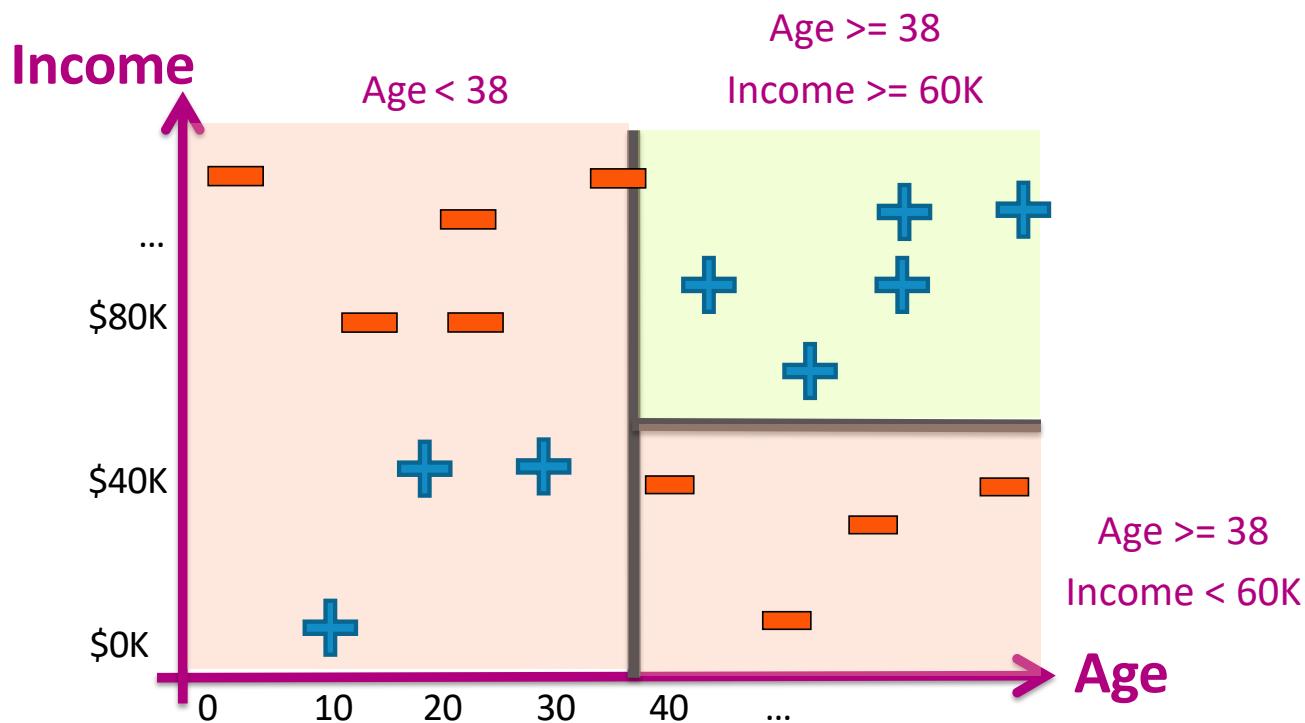
## Split on Age $\geq 38$



## Depth 2: Split on Income $\geq \$60K$

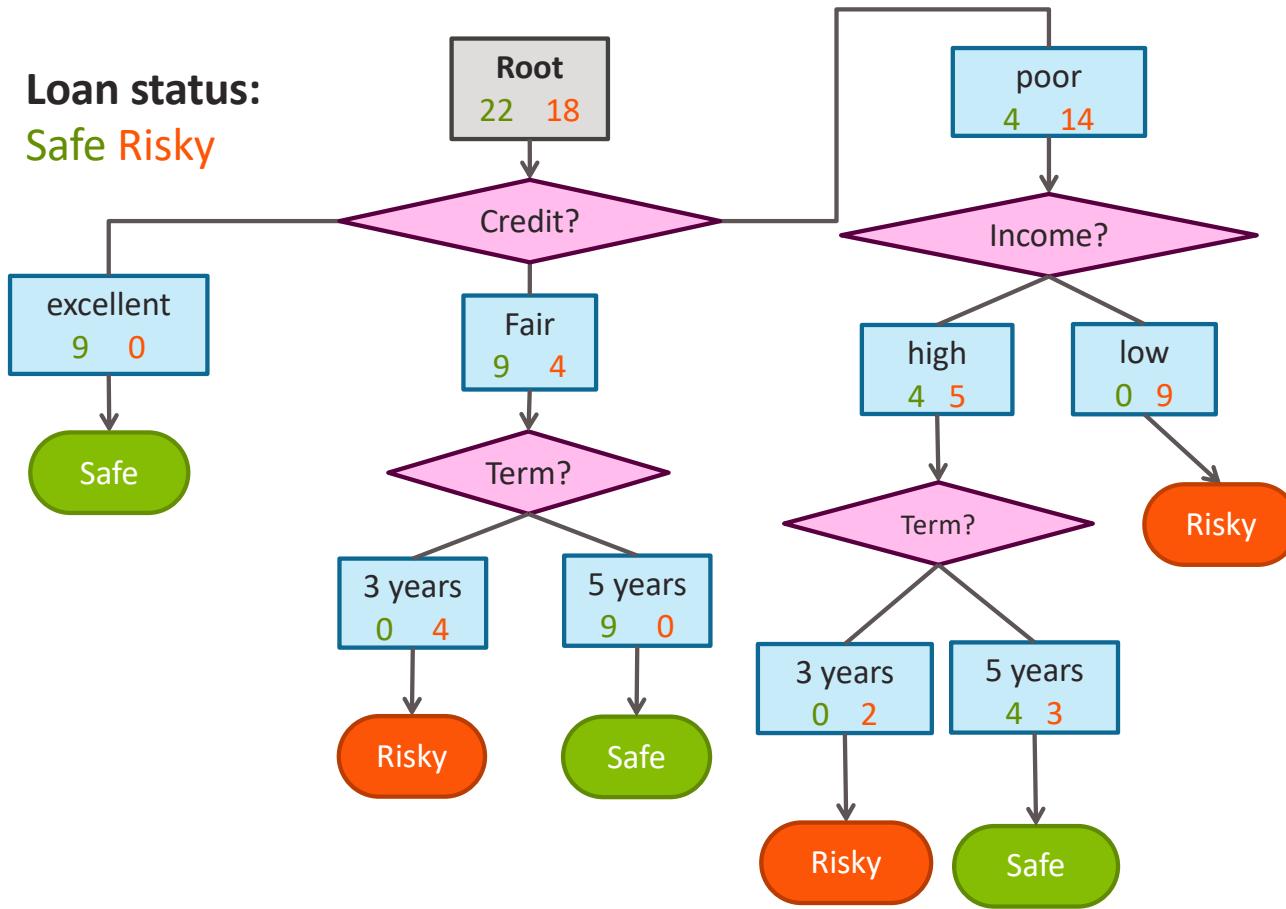


# Each split partitions the 2-D space



# Overfitting in decision trees

# Decision tree recap

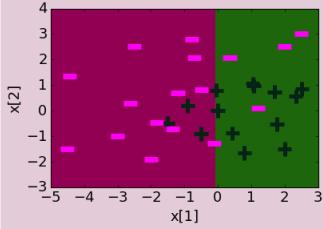
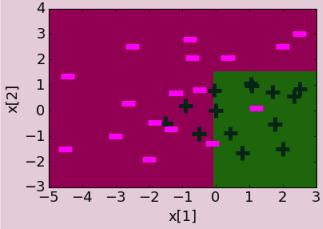
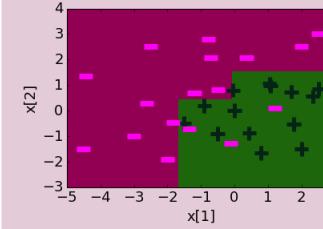
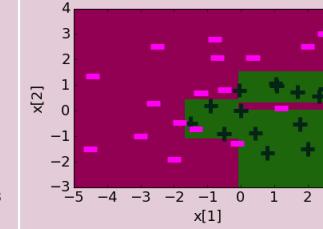
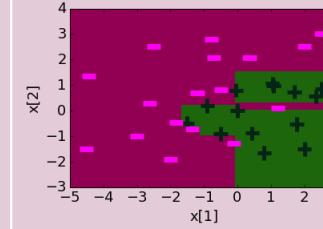


For each leaf node,  
set  
 $\hat{y}$  = majority value

# What happens when we increase depth?

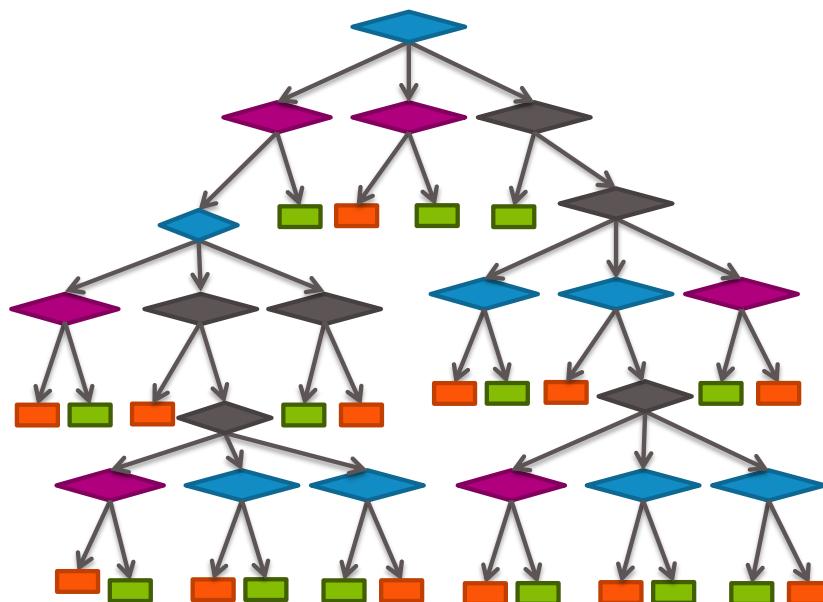
Training error reduces with depth



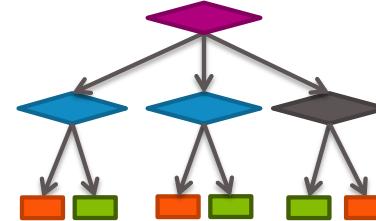
Tree depth	depth = 1	depth = 2	depth = 3	depth = 5	depth = 10
Training error	0.22	0.13	0.10	0.03	0.00
Decision boundary					

# How do we regularize? In this context, simplicity

Complex Tree



Simpler Tree



# Two approaches to picking simpler trees

1. **Early Stopping:**

Stop the learning algorithm **before** tree becomes too complex

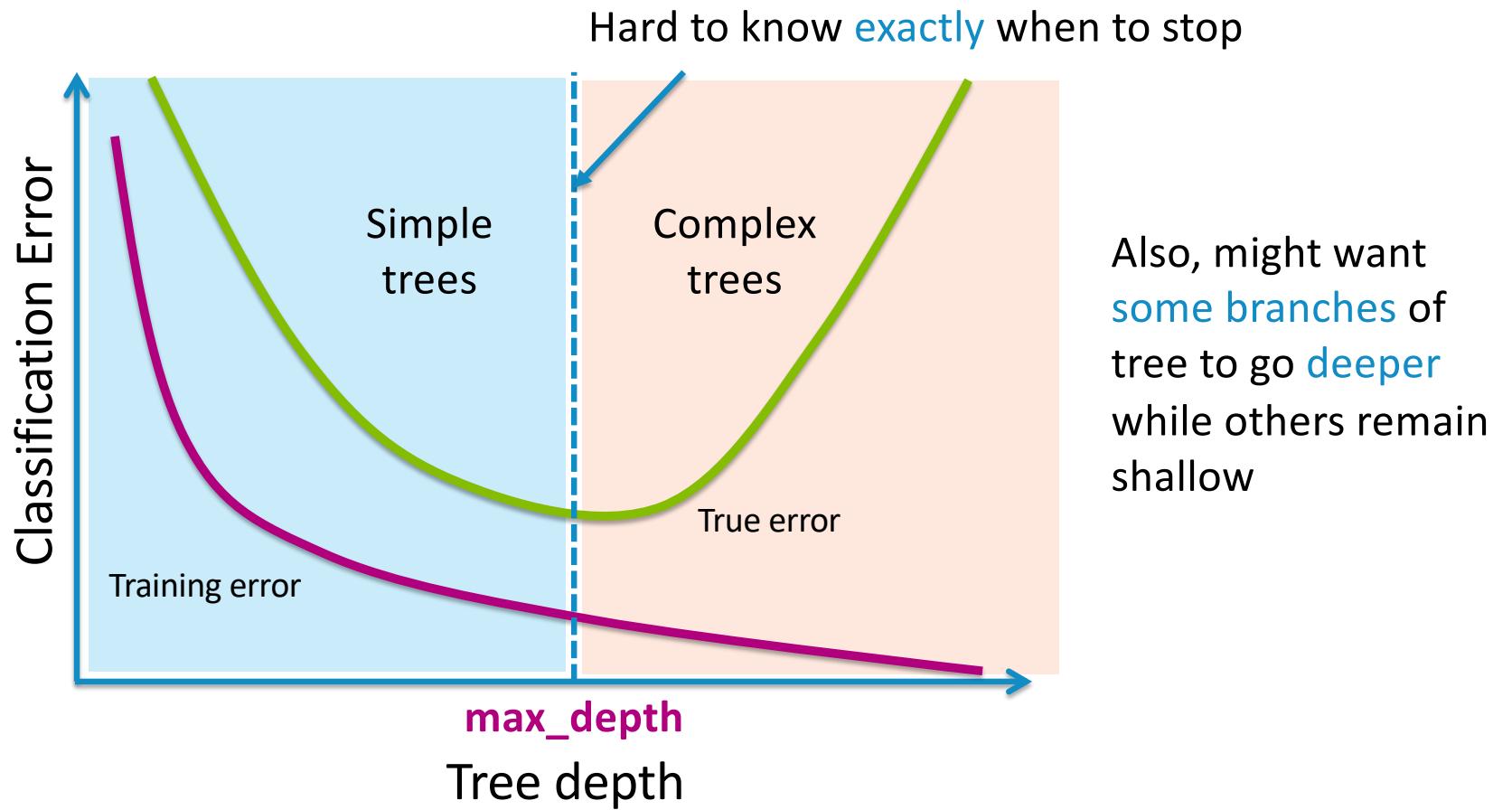
2. **Pruning:**

Simplify the tree **after** the learning algorithm terminates

# Technique 1: Early stopping

- **Stopping conditions (recap):**
  1. All examples have the same target value
  2. No more features to split on
- **Early stopping conditions:**
  1. Limit tree depth (choose  $\text{max\_depth}$  using validation set)
  2. Do not consider splits that do not cause a sufficient decrease in classification error
  3. Do not split an intermediate node which contains too few data points

# Challenge with early stopping condition 1



# Early stopping condition 2: Pros and Cons

- Pros:
  - A reasonable heuristic for early stopping to avoid useless splits
- Cons:
  - **Too short sighted:** We may miss out on “good” splits that occur right after “useless” splits
  - Saw this with “**xor**” example

# Two approaches to picking simpler trees

## 1. Early Stopping:

Stop the learning algorithm **before** tree becomes too complex

## 2. Pruning:

Simplify the tree **after** the learning algorithm terminates

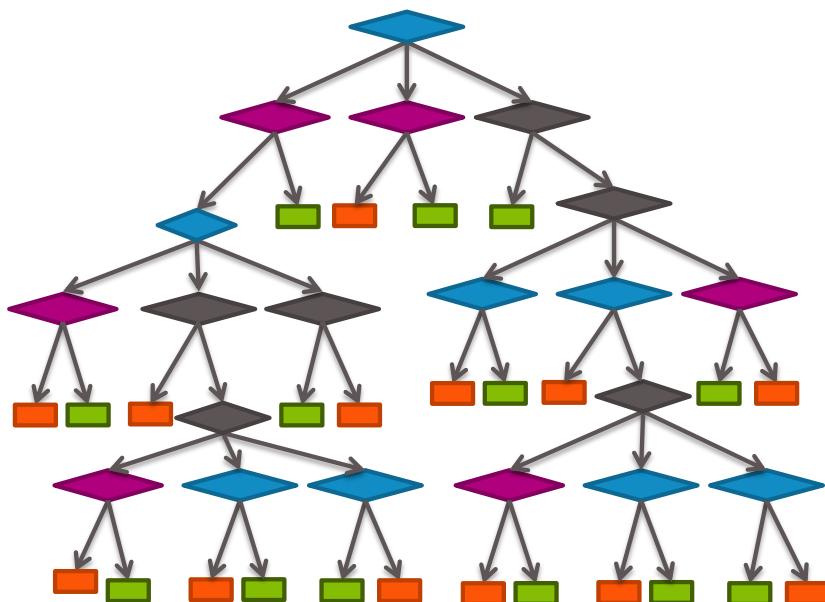
Complements early stopping



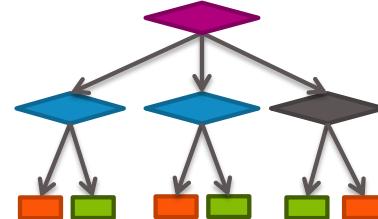
# Pruning: *Intuition*

Train a complex tree, simplify later

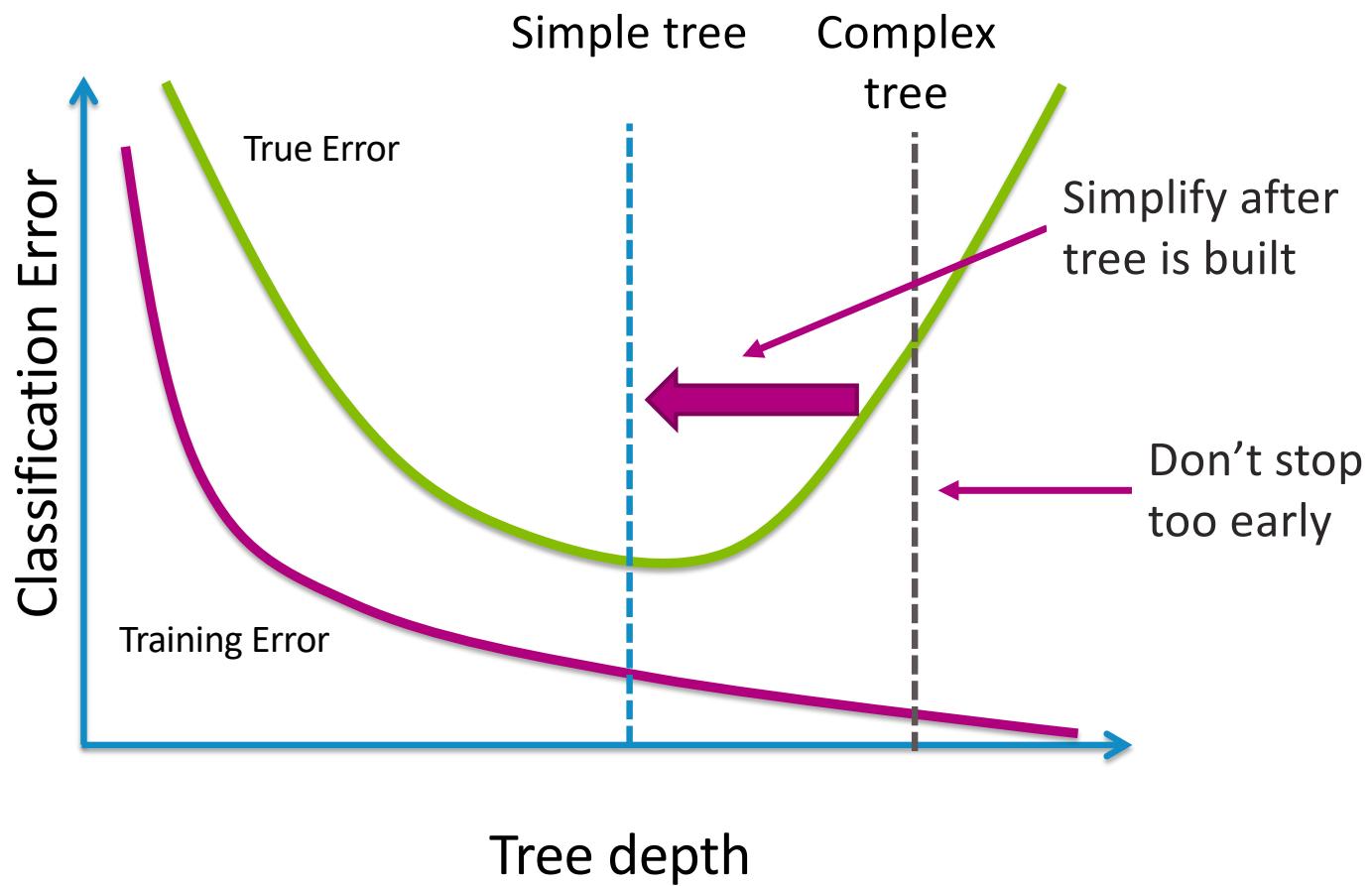
Complex Tree



Simpler Tree



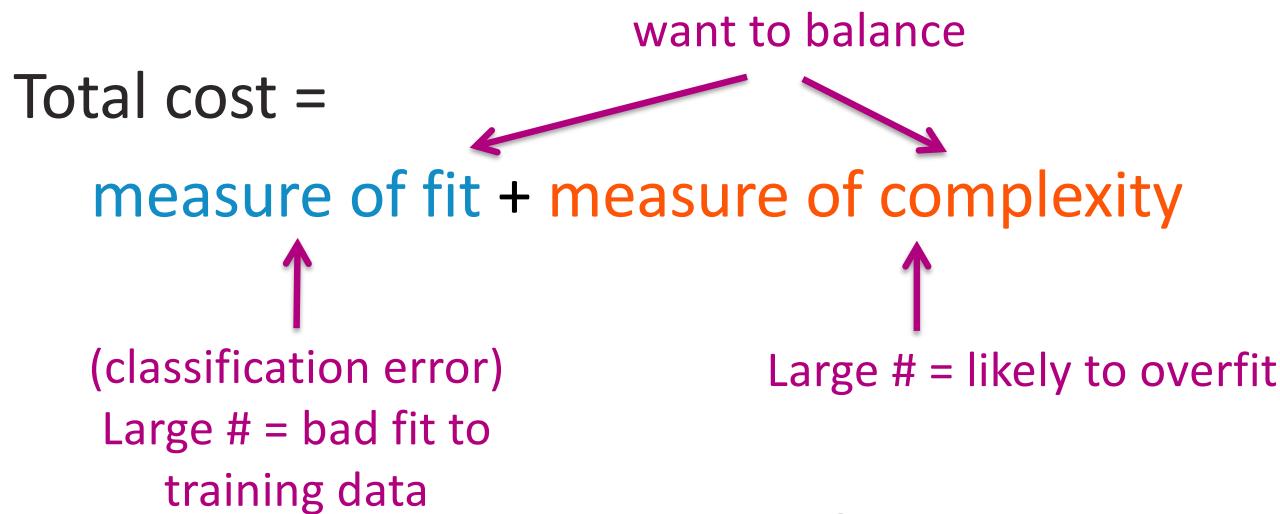
# Pruning motivation



# Scoring trees: Desired total quality format

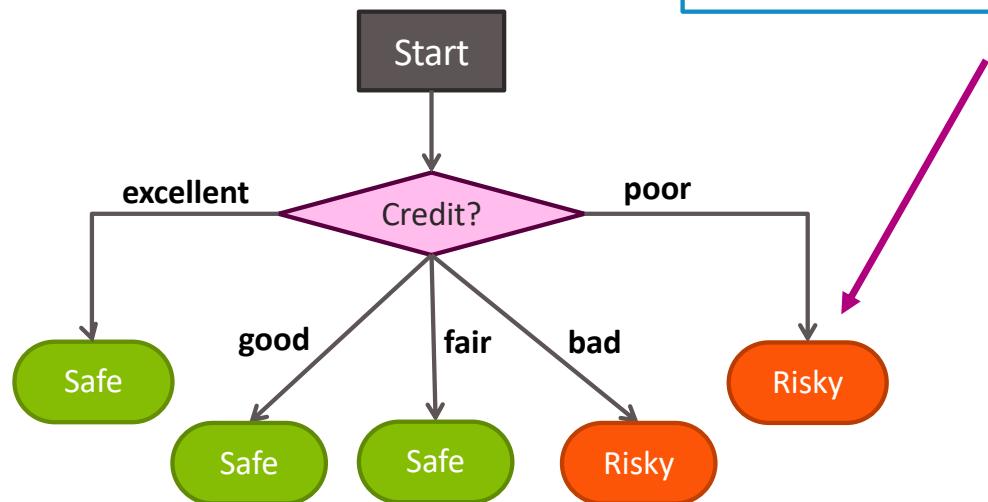
Want to balance:

- i. How well tree fits data
- ii. Complexity of tree



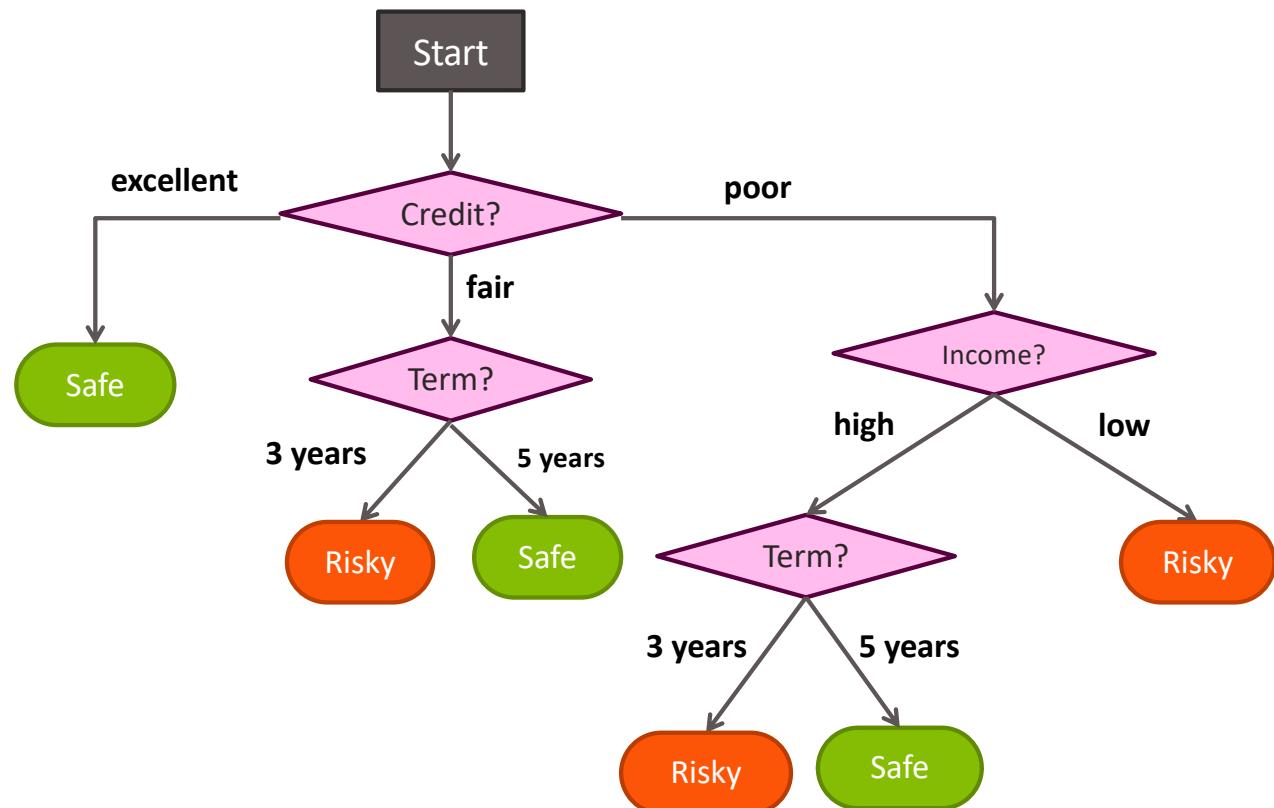
# Simple measure of complexity of tree

$$L(T) = \# \text{ of leaf nodes}$$

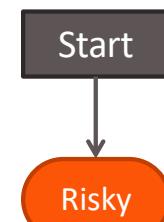


# Balance simplicity & predictive power

Too complex, risk of overfitting



Too simple, high classification error



# Balancing fit and complexity

$$\text{Total cost } C(T) = \text{Error}(T) + \lambda L(T)$$

  
tuning parameter

If  $\lambda=0$ : build out full tree.

If  $\lambda=\infty$ : won't ever split.

If  $\lambda$  in between:

# Having said all that...

Vanilla decision trees just don't really work very well.



# Bagging

CSE 446: Machine Learning

# Suppose we want to reduce variance

Recall definition of variance  $\mathbb{E}[(f_D(x) - \mathbb{E}_D[f_D(x)])^2]$

If we had lots of **independent** data sets, we could average their results and get the variance to approach 0

Given  $m$  independent data sets  $(D_1, \dots, D_m)$ , each of size  $n$ , learn  $m$  classifiers  $(f_1, \dots, f_m)$ , where  $f_i$  trained on  $D_i$ .

By law of large numbers  $\frac{1}{m} \sum_i f_{D_i}(x) \rightarrow \mathbb{E}_D(f_D(x))$ .

But we don't have lots of data sets

# Bootstrap Aggregation (aka bagging)

- A technique for addressing high variance (e.g. in decision trees)
  - Given sample  $D$ , draw  $m$  samples ( $D_1, \dots, D_m$ ) of size  $n$ , with replacement.
  - Train a classifier on each.
  - Use average (for regression) or majority vote (for classification).

# Advantages of bagging

- Easy to implement
- Reduces variance
- Gives information about uncertainty of prediction.
- Provides estimate of test error, “out of bag” error without using validation set.

Let  $S_i = \{k \mid (\mathbf{x}_i, y_i) \notin D_k\}$  and define  $\tilde{f}_i(\mathbf{x}) = \frac{1}{|S_i|} \sum_{k \in S_i} f_k(\mathbf{x}_i)$

Then out of bag error is:  $\frac{1}{n} \sum_{(\mathbf{x}_i, y_i) \in D} \ell(\tilde{f}_i(\mathbf{x}_i), h_i)$

# Bagging produces an ensemble classifier

# Ensemble classifier in general

- Goal:
  - Predict output  $y$ 
    - Either +1 or -1
  - From input  $x$
- Learn ensemble model:
  - Classifiers:  $f_1(x), f_2(x), \dots, f_T(x)$
  - Coefficients:  $\hat{w}_1, \hat{w}_2, \dots, \hat{w}_T$
- Prediction:

$$\hat{y} = sign \left( \sum_{t=1}^T \hat{w}_t f_t(\mathbf{x}) \right)$$

©2017 Emily Fox



# Random Forests

CSE 446: Machine Learning

# One of the most useful bagged algorithms

- Given data set  $D$ , draw  $m$  samples ( $D_1, \dots, D_m$ ) of size  $n$  from  $D$ , with replacement.
- For each  $D_j$ , train a full decision tree  $h_j$  with one **crucial modification**:
  - *Before each split, randomly subsample  $k$  features (without replacement) and only consider these for your split.*
- Final classifier obtained by averaging.

# One of best, most popular and easiest to use algorithms

- Only two hyperparameters  $m$  and  $k$ .
- Extremely insensitive to both.
  - Classification: good choice for  $k$  is  $\sqrt{d}$  (where  $d$  is number of features)
  - Regression: good choice for  $k$  is  $d/3$
  - Set  $m$  as large as you can afford.
- Works really well!
- Features can be of different scale, magnitude, etc.
  - Very convenient with heterogeneous features.



# Boosting

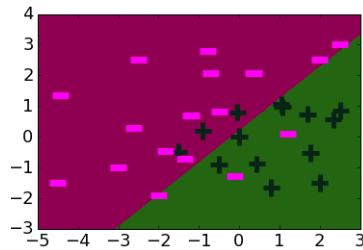
CSE 446: Machine Learning

Emily Fox

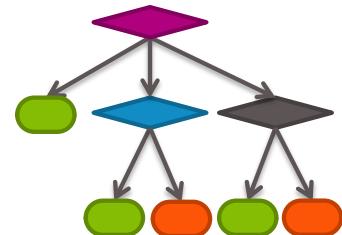
University of Washington

January 30, 2017

# Simple (weak) classifiers are good!



Logistic regression w.  
simple features



Shallow  
decision trees

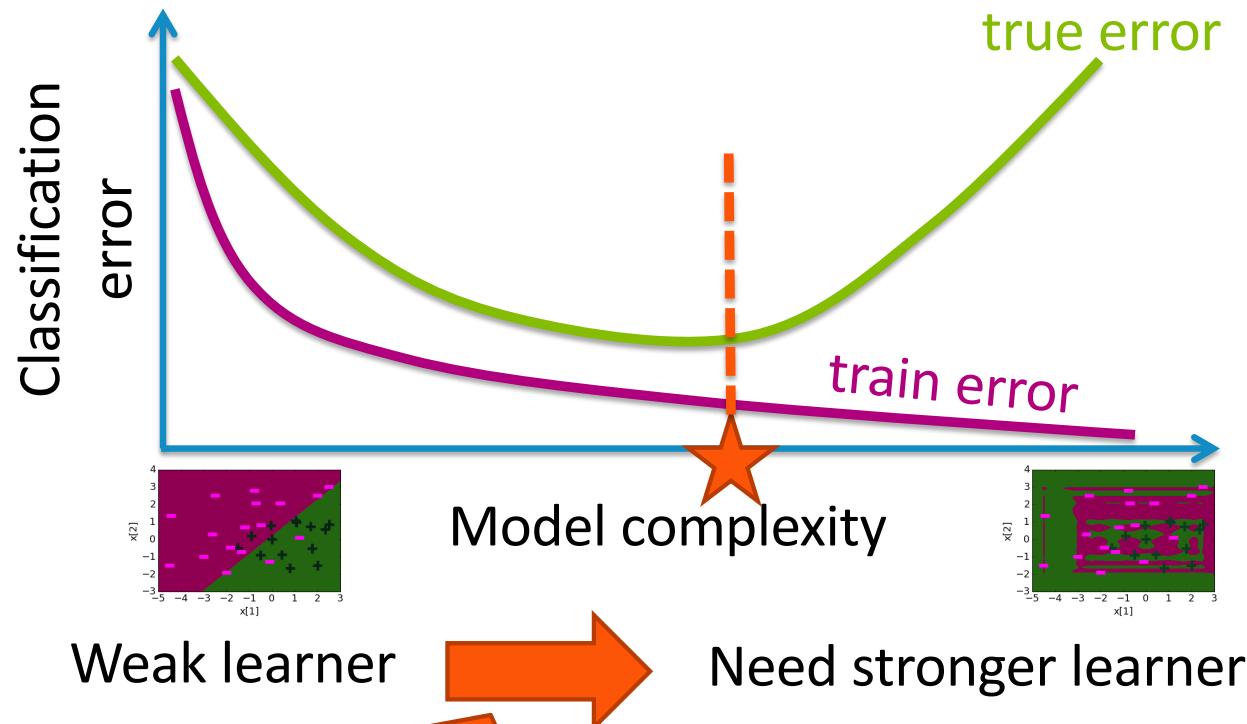


Decision  
stumps

Low variance. Learning is fast!

But high bias...

# Finding a classifier that's just right



Option 1: add more features or depth  
Option 2: ?????

# Boosting question

“Can a set of weak learners be combined to create a stronger learner?” *Kearns and Valiant (1988)*



Yes! *Schapire (1990)*



Boosting

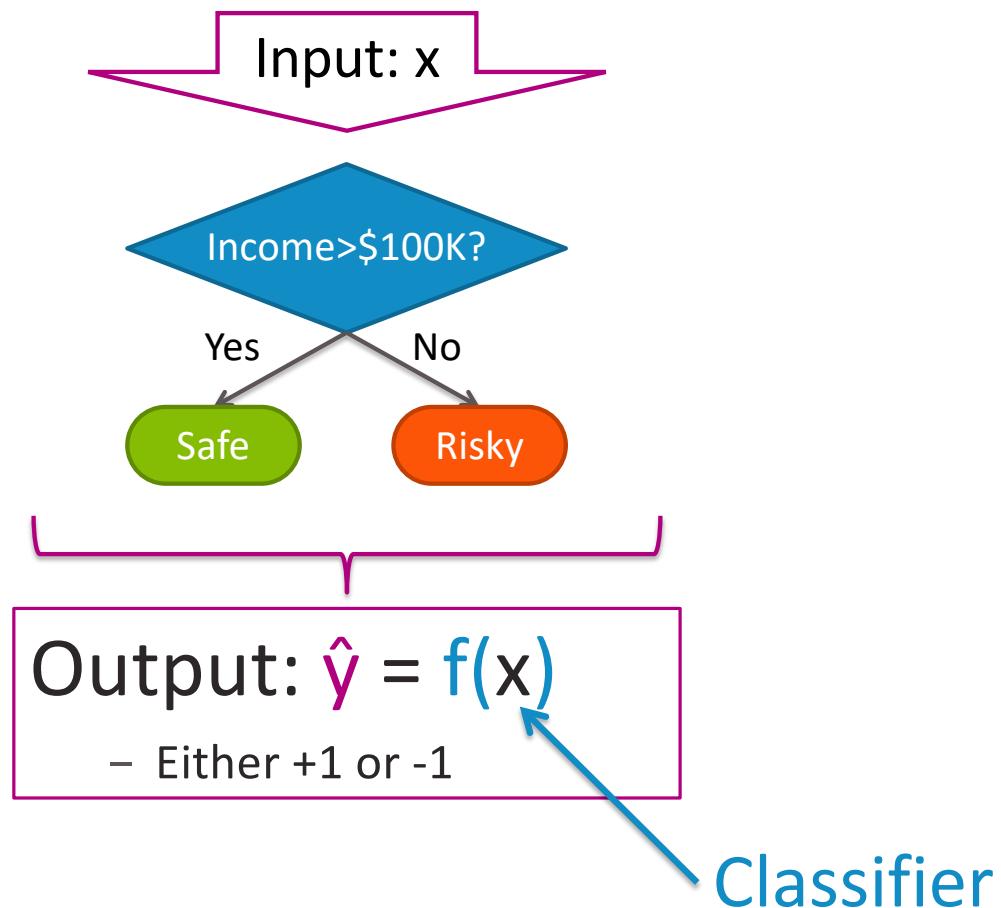


Amazing impact:

- simple approach
- widely used in industry
- wins most Kaggle competitions

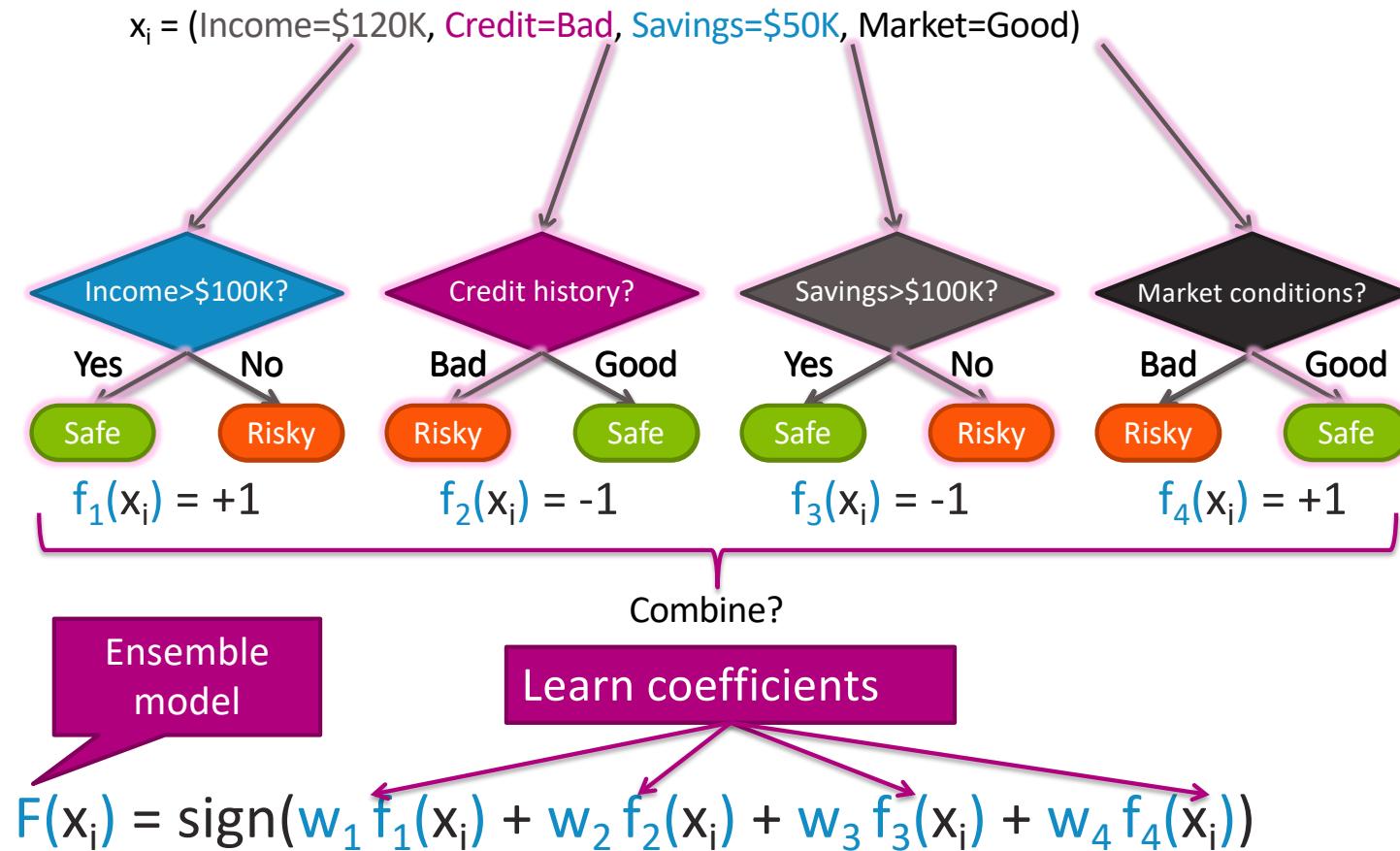
# Ensemble classifier

# A single classifier

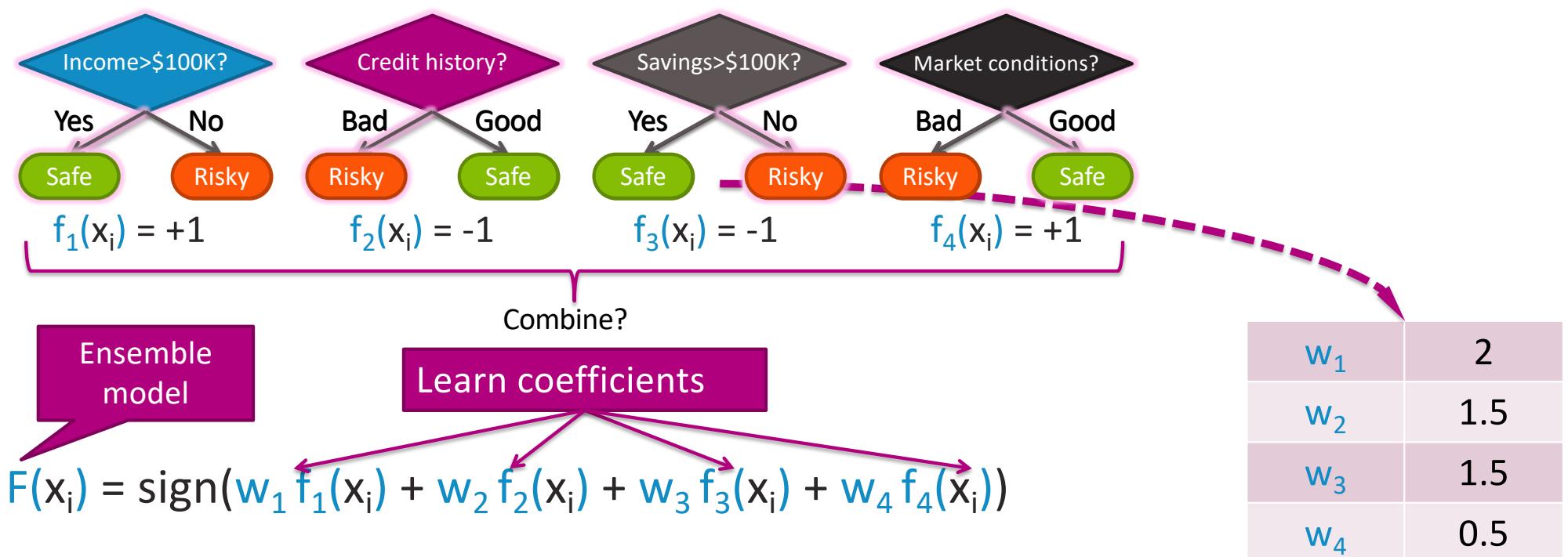




# Ensemble methods: Each classifier “votes” on prediction



# Prediction with ensemble



# Ensemble classifier in general

- Goal:
  - Predict output  $y$ 
    - Either +1 or -1
  - From input  $x$
- Learn ensemble model:
  - Classifiers:  $f_1(x), f_2(x), \dots, f_T(x)$
  - Coefficients:  $\hat{w}_1, \hat{w}_2, \dots, \hat{w}_T$
- Prediction:

$$\hat{y} = sign \left( \sum_{t=1}^T \hat{w}_t f_t(\mathbf{x}) \right)$$

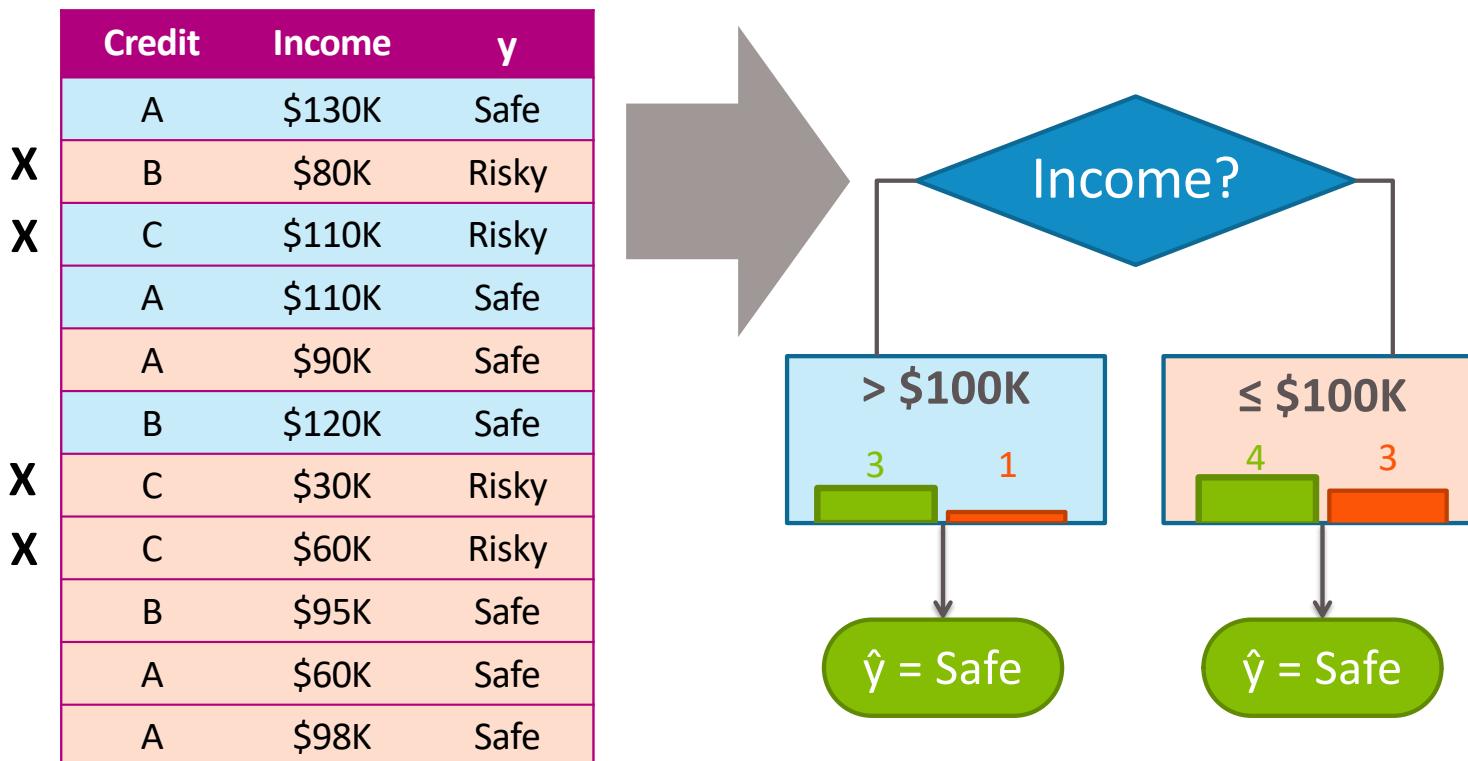
©2017 Emily Fox

# Boosting

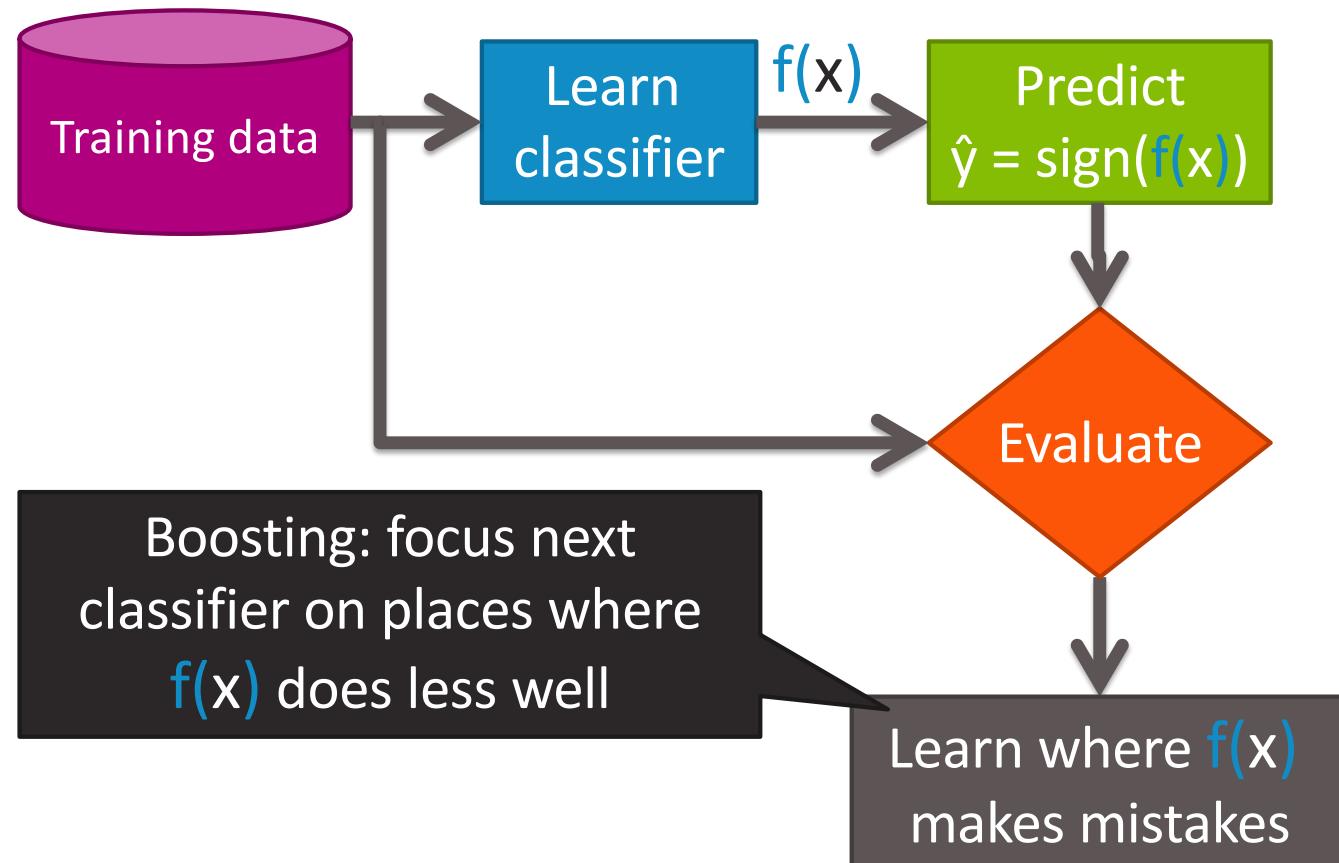
# Training a classifier



# Learning decision stump



# Boosting = Focus learning on “hard” points



# Learning on weighted data:

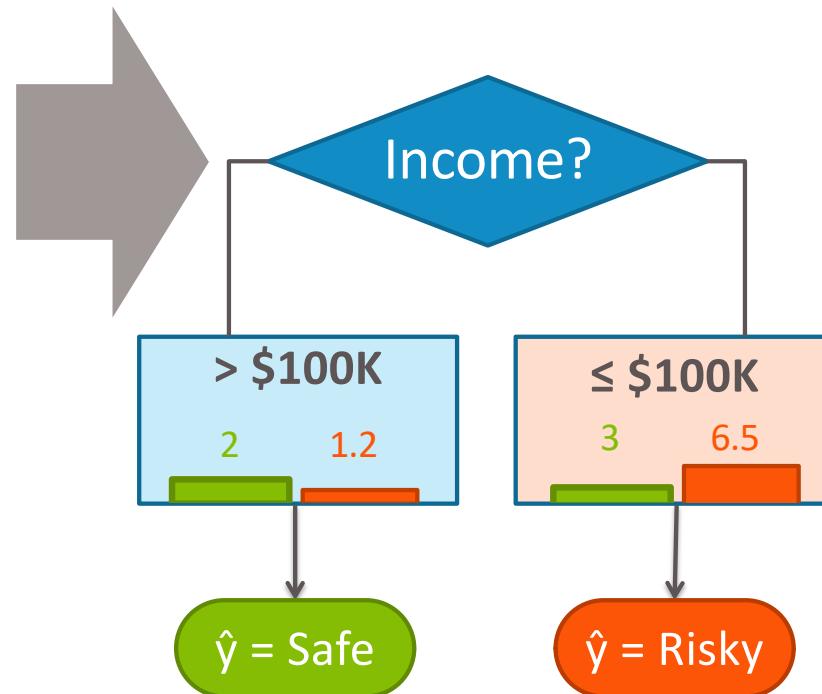
*More weight on “hard” or more important points*

- Weighted dataset:
  - Each  $x_i, y_i$  weighted by  $\alpha_i$ 
    - More important point = higher weight  $\alpha_i$
- Learning:
  - Data point  $i$  counts as  $\alpha_i$  data points
    - E.g.,  $\alpha_i = 2 \rightarrow$  count point twice

# Learning a decision stump on weighted data

Increase weight  $\alpha$  of harder/misclassified points

X	Credit	Income	y	Weight $\alpha$
X	A	\$130K	Safe	0.5
X	B	\$80K	Risky	1.5
X	C	\$110K	Risky	1.2
X	A	\$110K	Safe	0.8
X	A	\$90K	Safe	0.6
X	B	\$120K	Safe	0.7
X	C	\$30K	Risky	3
X	C	\$60K	Risky	2
X	B	\$95K	Safe	0.8
X	A	\$60K	Safe	0.7
X	A	\$98K	Safe	0.9



# Learning from weighted data in general

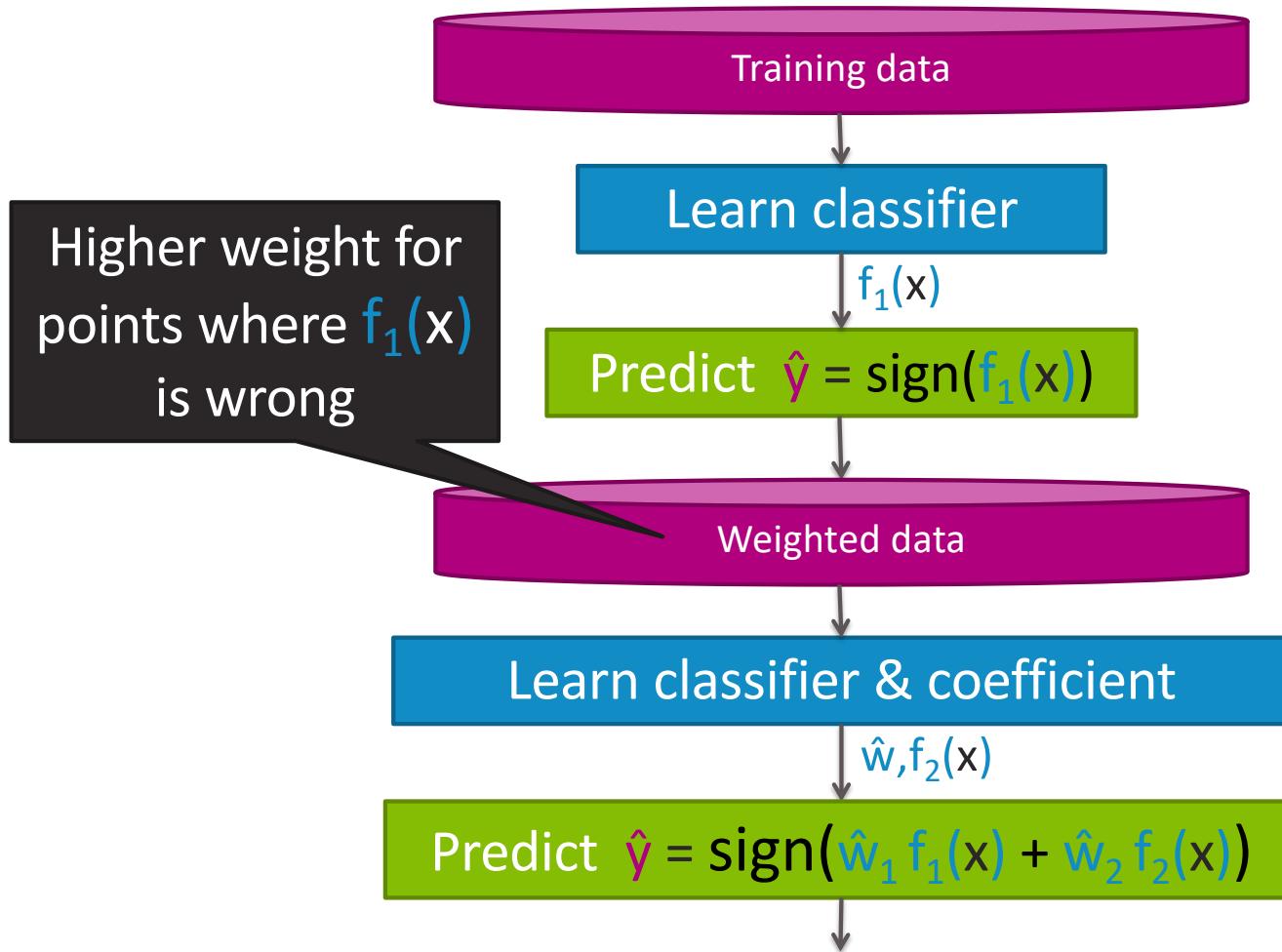
- Learning from weighted data generally no harder.
  - Data point  $i$  counts as  $\alpha_i$  data points
- E.g., gradient descent for logistic regression:

$$\mathbf{w}_j^{(t+1)} \leftarrow \mathbf{w}_j^{(t)} + \eta \sum_{i=1}^N \alpha_i(\mathbf{x}_i) \left( \mathbb{1}[y_i = +1] - P(y = +1 \mid \mathbf{x}_i, \mathbf{w}^{(t)}) \right)$$

Sum over data points

Weigh each point by  $\alpha_i$

# Boosting = Greedy learning ensembles from data



# AdaBoost

# AdaBoost: learning ensemble

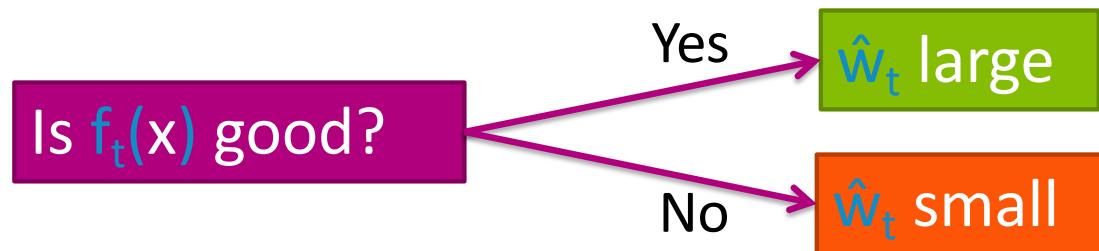
[Freund & Schapire 1999]

- Start with same weight for all points:  $\alpha_i = 1/N$
- For  $t = 1, \dots, T$ 
  - Learn  $f_t(x)$  with data weights  $\alpha_i$
  - Compute coefficient  $\hat{w}_t$
  - Recompute weights  $\alpha_i$
- Final model predicts by:

$$\hat{y} = sign \left( \sum_{t=1}^T \hat{w}_t f_t(\mathbf{x}) \right)$$

# Computing coefficient $\hat{w}_t$

## AdaBoost: Computing coefficient $\hat{w}_t$ of classifier $f_t(x)$



- $f_t(x)$  is good  $\rightarrow f_t$  has low training error
- Measuring error in weighted data?
  - Just weighted # of misclassified points

# Weighted classification error

Learned classifier

$$\hat{y} = +$$

Data point

(Badly was @ Keat, ,  $\alpha=0.15$ )

Mistake!

Weight of correct  
Weight of mistakes

102

005

Hide label

# Weighted classification error

- Total weight of mistakes:

$$\sum_i \alpha_i \mathbb{1}(\hat{y}_i \neq y_i)$$

- Total weight of all points:

$$\sum_{i=1}^n \alpha_i$$

- Weighted error measures fraction of weight of mistakes:

$$\text{weighted\_error} = \frac{\text{Total weight of all mistakes}}{\text{Total weight of all data}}$$

– Best possible value is 0.0

Worst possible value is 1.0

Actual worst is 0.5

## AdaBoost:

Formula for computing coefficient  $\hat{w}_t$  of classifier  $f_t(x)$

$$\hat{w}_t = \frac{1}{2} \ln \left( \frac{1 - \text{weighted\_error}(f_t)}{\text{weighted\_error}(f_t)} \right)$$

Is $f_t(x)$ good?	weighted_error( $f_t$ ) on training data	$\frac{1 - \text{weighted\_error}(f_t)}{\text{weighted\_error}(f_t)}$	$\hat{w}_t$
Yes	0.01	$(1-0.01)/0.01 = 99$	$0.5 \ln(99) = 2.3$
No	0.5	$(1-0.5)/0.5 = 1$	$0.5 \ln(1) = 0$
	0.99	$(1-0.99)/0.99 = 0.01$	$0.5 \ln(0.01) = -2.3$

Last one is a terrible classifier, but  $-f_t(x)$  is great!

# AdaBoost: learning ensemble

- Start with same weight for all points:  $\alpha_i = 1/N$

- For  $t = 1, \dots, T$

- Learn  $f_t(x)$  with data weights  $\alpha_i$

- Compute coefficient  $\hat{w}_t$

- Recompute weights  $\alpha_i$

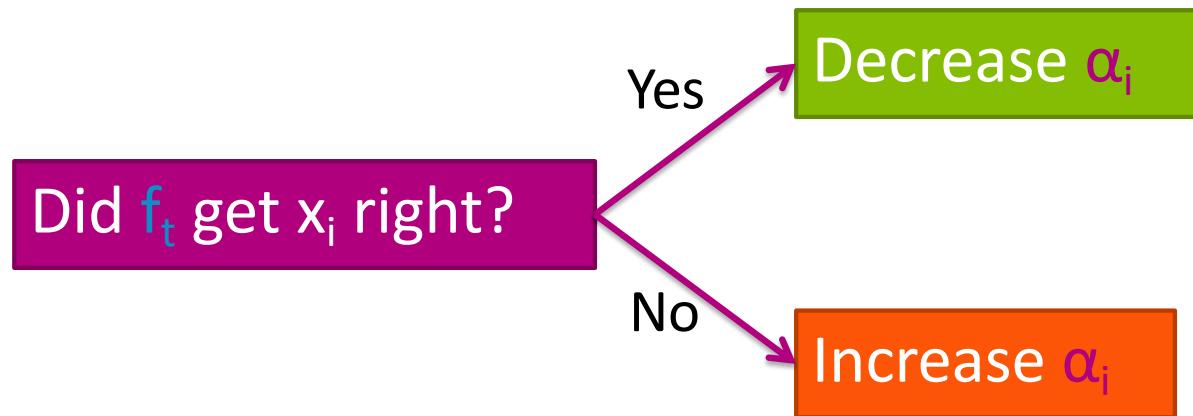
$$\hat{w}_t = \frac{1}{2} \ln \left( \frac{1 - \text{weighted\_error}(f_t)}{\text{weighted\_error}(f_t)} \right)$$

- Final model predicts by:

$$\hat{y} = \text{sign} \left( \sum_{t=1}^T \hat{w}_t f_t(\mathbf{x}) \right)$$

Recompute weights  $\alpha_i$

AdaBoost: Updating weights  $\alpha_i$  based on where classifier  $f_t(x)$  makes mistakes



# AdaBoost: Formula for updating weights $\alpha_i$

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t}, & \text{if } f_t(x_i) = y_i \\ \alpha_i e^{\hat{w}_t}, & \text{if } f_t(x_i) \neq y_i \end{cases}$$

Did  $f_t$  get  $x_i$  right?

Yes

No

$f_t(x_i) = y_i ?$	$\hat{w}_t$	Multiply $\alpha_i$ by	Implication
correct	2.3	$e^{-2.3} = 0.1$	Decrease importance of $x_i, y_i$
correct	0	$e^{-0} = 1$	Keep importance same
mistake	2.3	$e^{2.3} = 9.98$	Increase importance
mistake	0	$e^{-0} = 1$	Keep importance same

# AdaBoost: learning ensemble

- Start with same weight for all points:  $\alpha_i = 1/N$

- For  $t = 1, \dots, T$

- Learn  $f_t(x)$  with data weights  $\alpha_i$

- Compute coefficient  $\hat{w}_t$

$$\hat{w}_t = \frac{1}{2} \ln \left( \frac{1 - \text{weighted\_error}(f_t)}{\text{weighted\_error}(f_t)} \right)$$

- Recompute weights  $\alpha_i$

- Final model predicts by:

$$\hat{y} = \text{sign} \left( \sum_{t=1}^T \hat{w}_t f_t(\mathbf{x}) \right)$$

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) = y_i \\ \alpha_i e^{\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

# AdaBoost: Normalizing weights $\alpha_i$

If  $x_i$  often mistake,  
weight  $\alpha_i$  gets very  
large

If  $x_i$  often correct,  
weight  $\alpha_i$  gets very  
small

Can cause numerical instability  
after many iterations

Normalize weights to  
add up to 1 after every iteration

$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^N \alpha_j}$$

# AdaBoost: learning ensemble

- Start with same weight for all points:  $\alpha_i = 1/N$

$$\hat{w}_t = \frac{1}{2} \ln \left( \frac{1 - \text{weighted\_error}(f_t)}{\text{weighted\_error}(f_t)} \right)$$

- For  $t = 1, \dots, T$

- Learn  $f_t(x)$  with data weights  $\alpha_i$

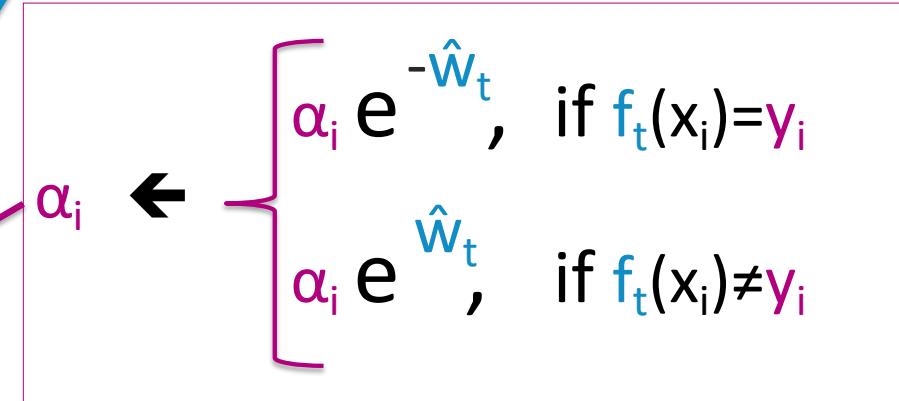
- Compute coefficient  $\hat{w}_t$

- Recompute weights  $\alpha_i$

- Normalize weights  $\alpha_i$

- Final model predicts by:

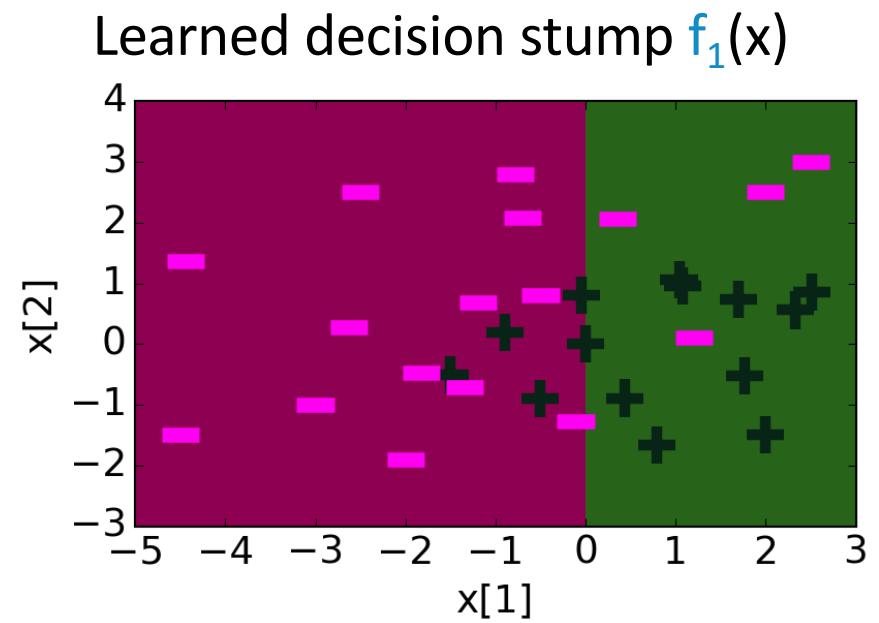
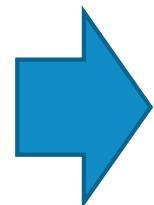
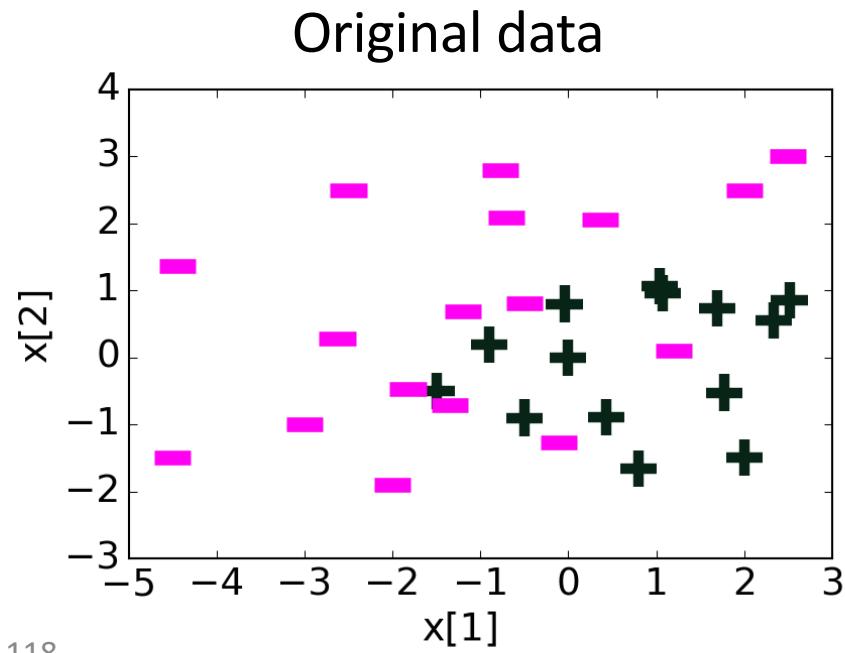
$$\hat{y} = \text{sign} \left( \sum_{t=1}^T \hat{w}_t f_t(\mathbf{x}) \right)$$



$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^N \alpha_j}$$

# AdaBoost example

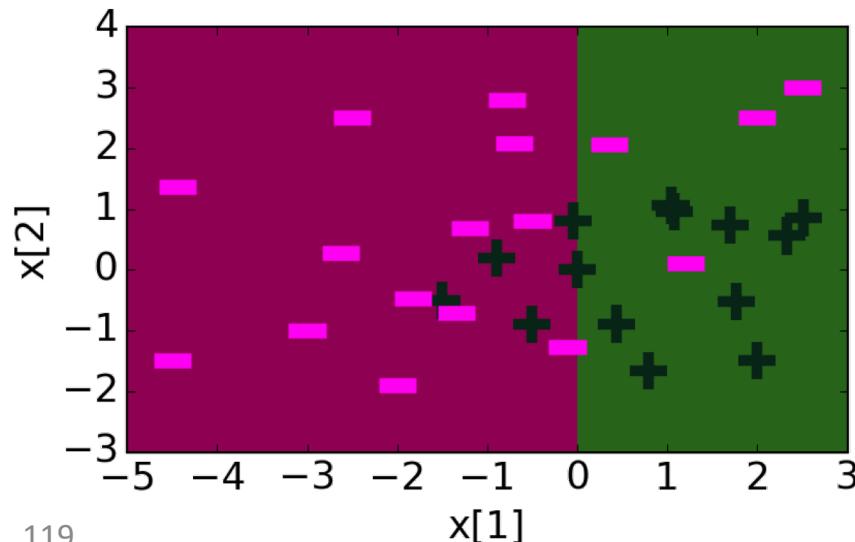
$t=1$ : Just learn a classifier on original data



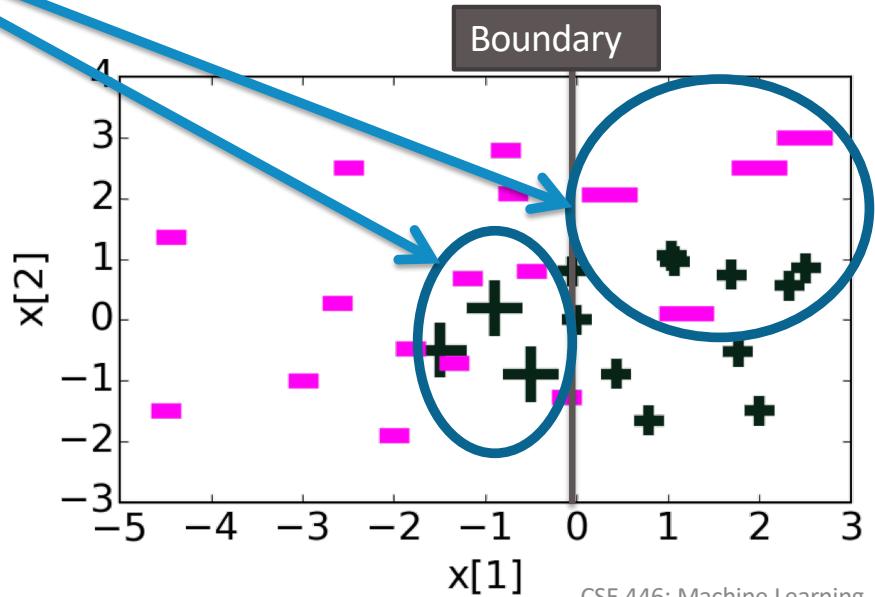
# Updating weights $\alpha_i$

Increase weight  $\alpha_i$   
of misclassified points

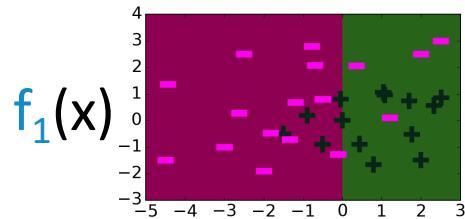
Learned decision stump  $f_1(x)$



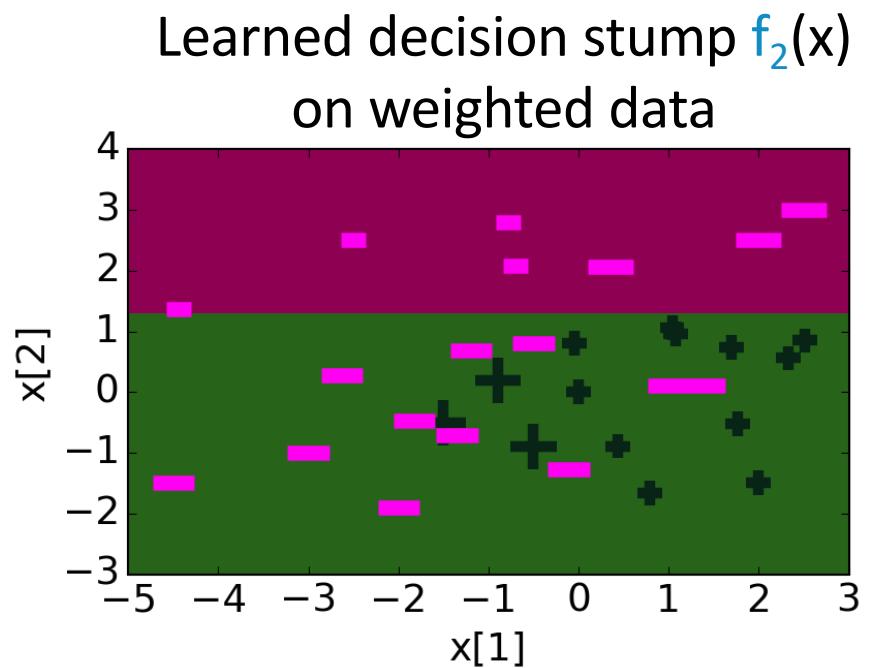
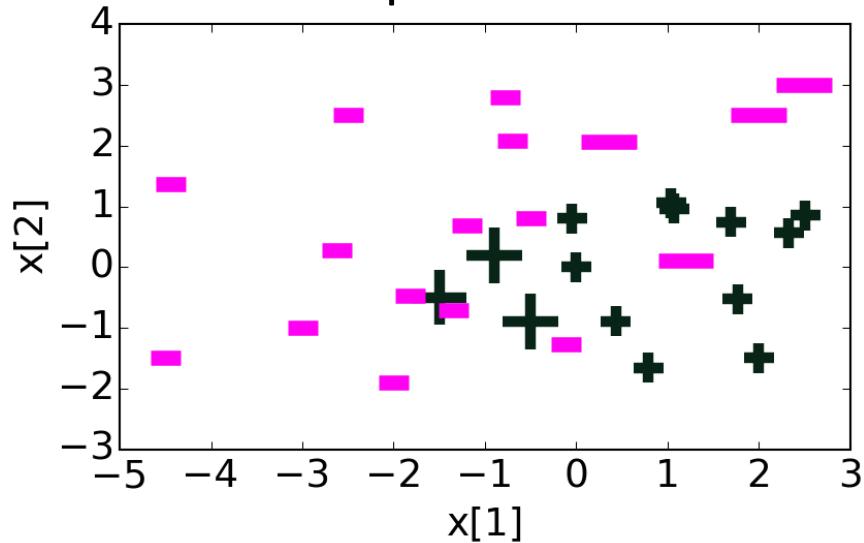
New data weights  $\alpha_i$



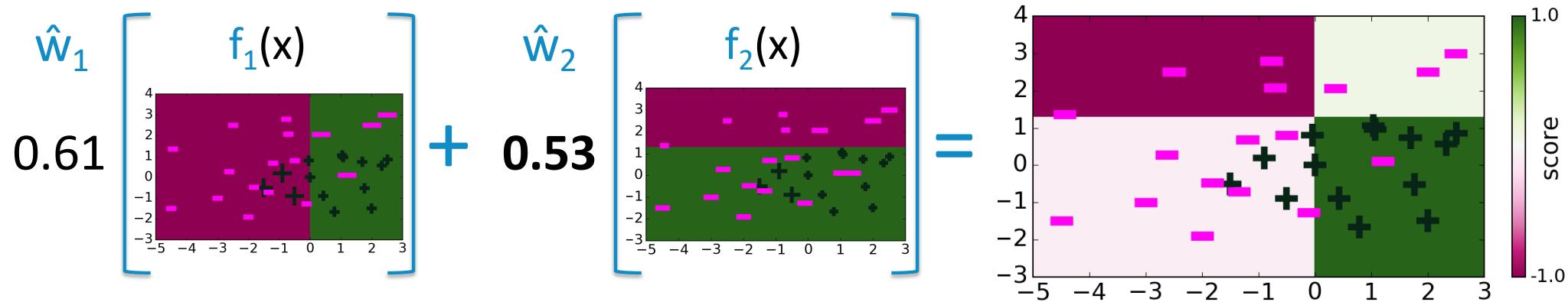
## $t=2$ : Learn classifier on weighted data



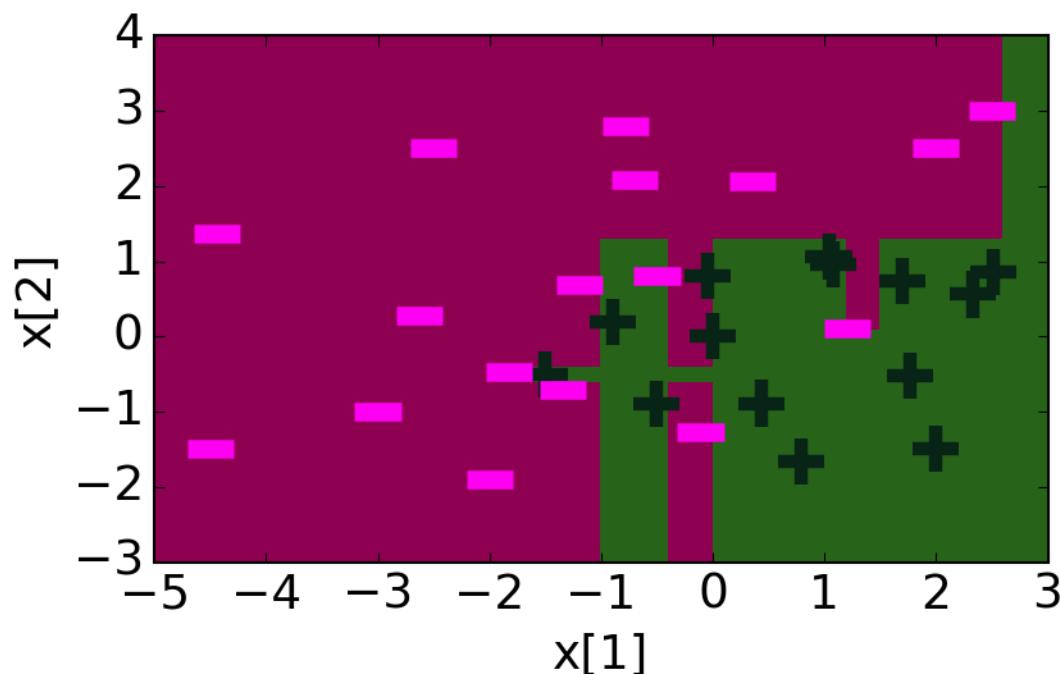
Weighted data: using  $\alpha_i$   
chosen in previous iteration



# Ensemble becomes weighted sum of learned classifiers



# Decision boundary of ensemble classifier after 30 iterations



training\_error = 0

# Boosting convergence & overfitting

# Boosting question revisited

“Can a set of weak learners be combined to  
create a stronger learner?” *Kearns and Valiant (1988)*

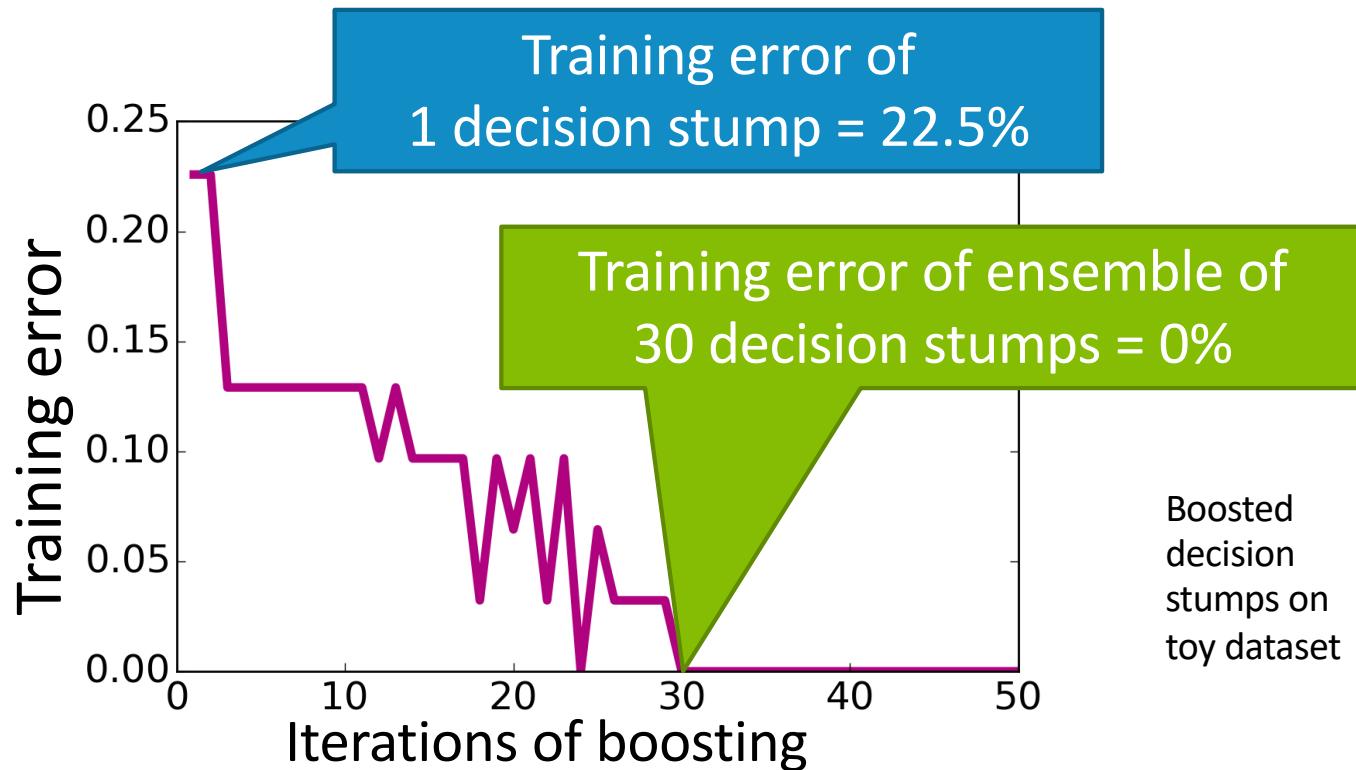


Yes! *Schapire (1990)*



Boosting

After some iterations,  
training error of boosting goes to zero!!!

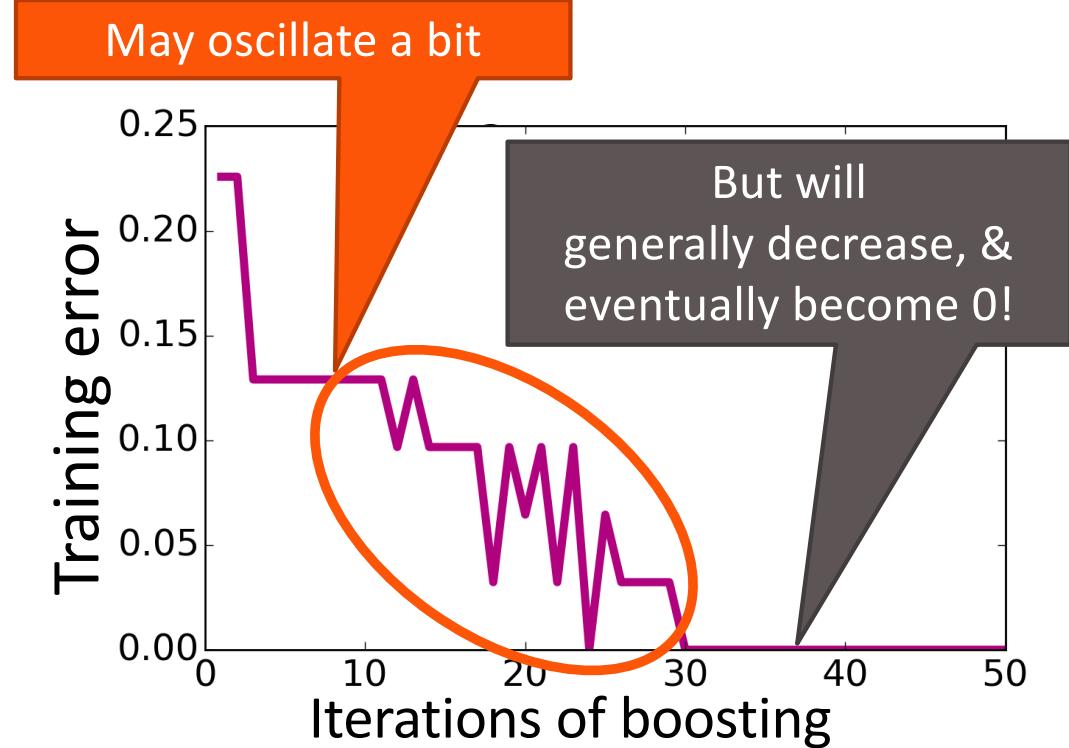


# AdaBoost Theorem

Under some technical conditions...



Training error of  
boosted classifier  $\rightarrow 0$   
as  $T \rightarrow \infty$



# Condition of AdaBoost Theorem

Under some technical conditions...



Training error of  
boosted classifier  $\rightarrow 0$   
as  $T \rightarrow \infty$

Condition = At every  $t$ ,  
can find a weak learner with  
 $\text{weighted\_error}(f_t) < 0.5$



Not always possible

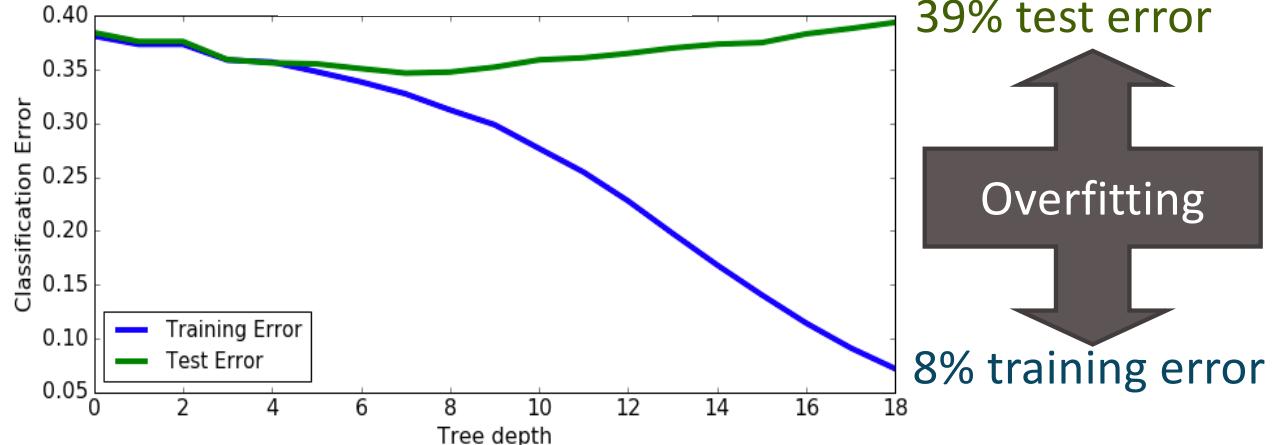
Extreme example:  
No classifier can  
separate a +1  
on top of -1



Nonetheless, boosting often  
yields great training error

# Overfitting in boosting

## Decision trees on loan data

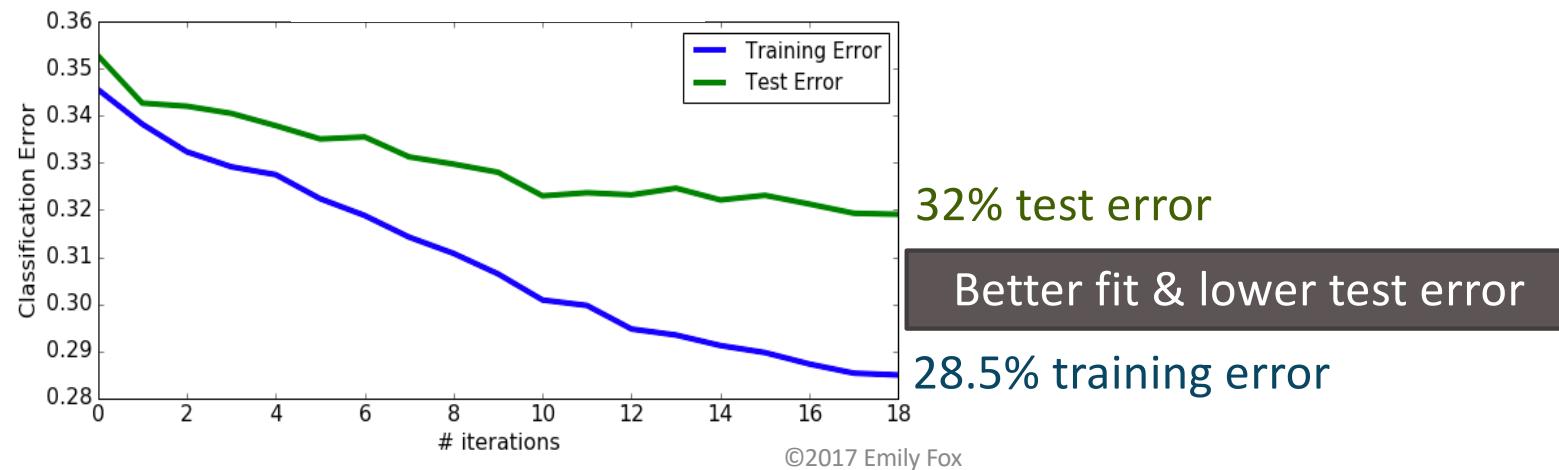


39% test error

Overfitting

8% training error

## Boosted decision stumps on loan data

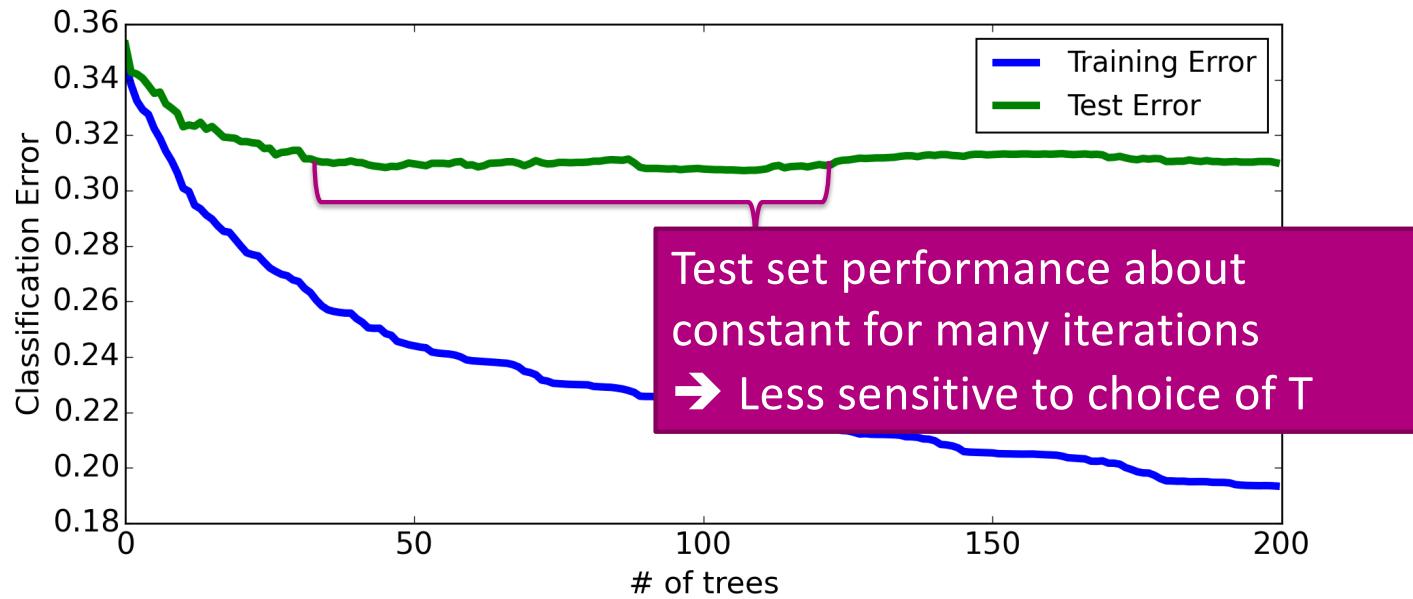


32% test error

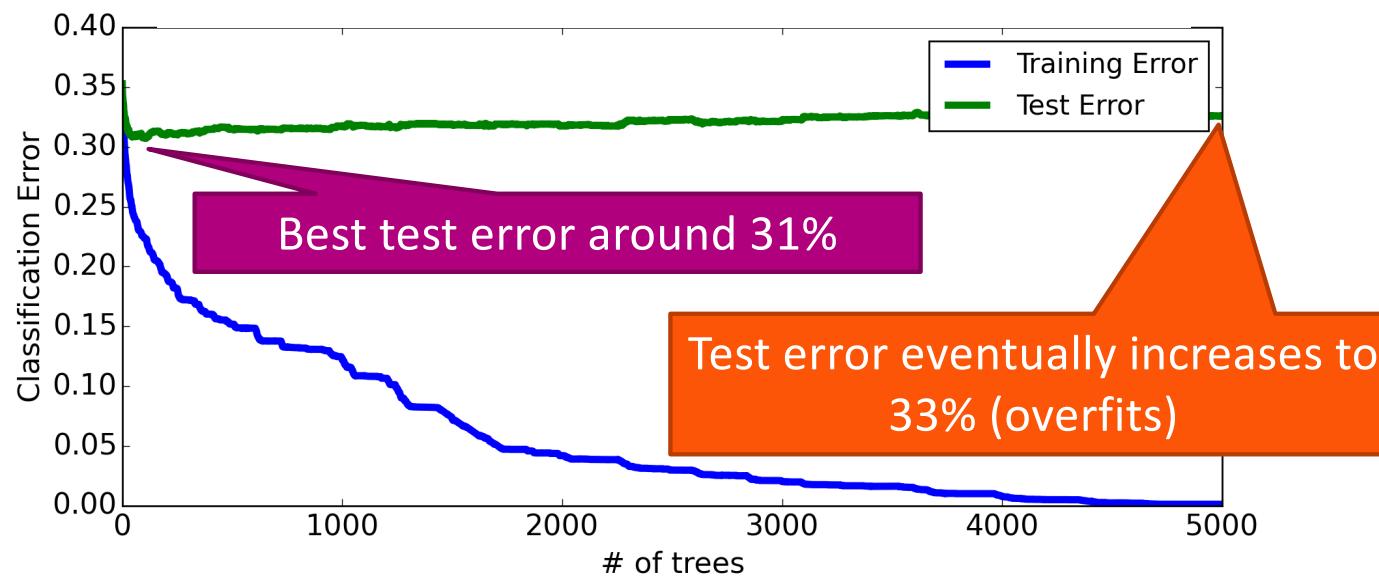
Better fit & lower test error

28.5% training error

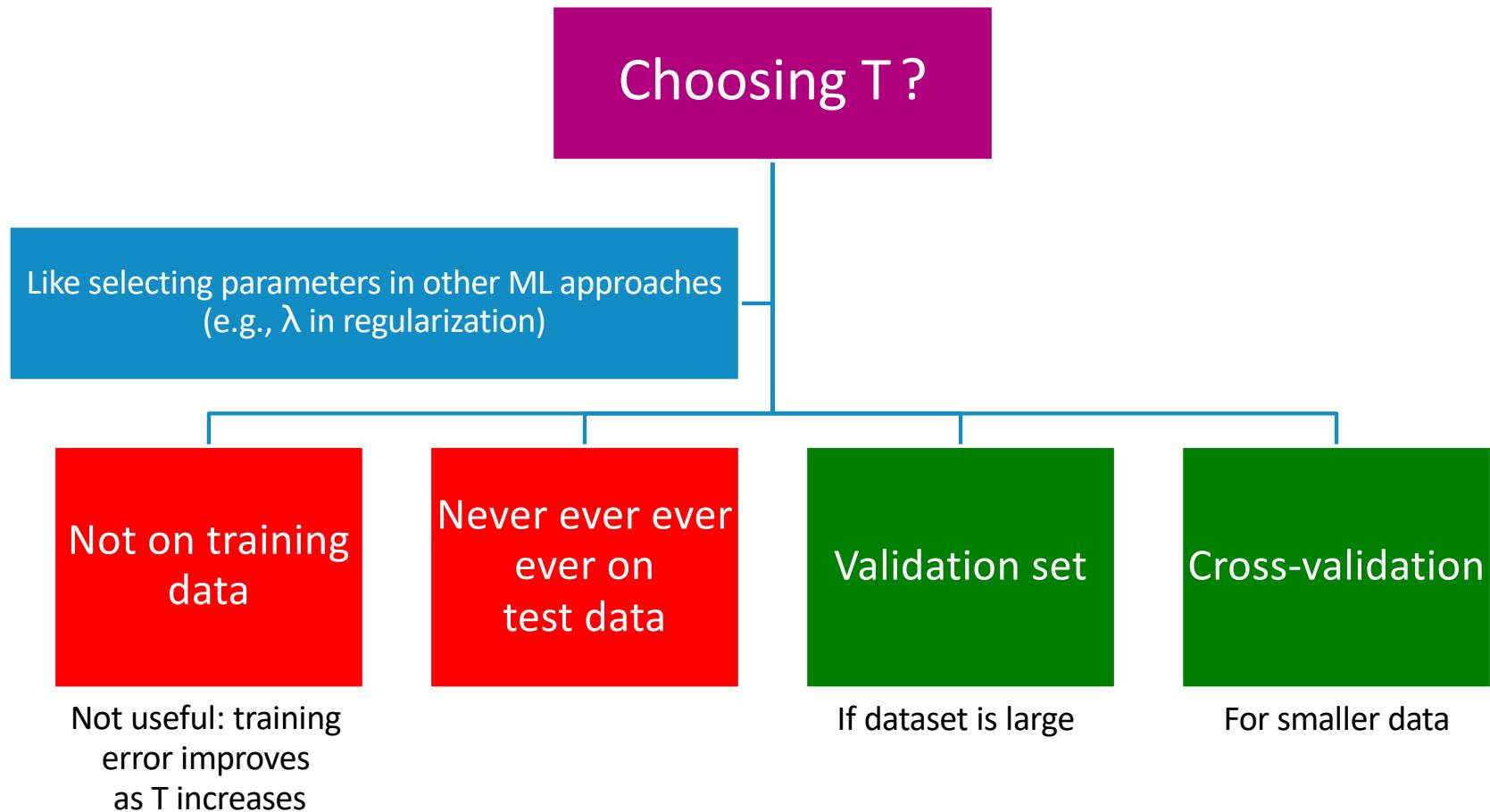
# Boosting tends to be robust to overfitting



But boosting will eventually overfit,  
so must choose max number of components T



# How do we decide when to stop boosting?



# Summary of boosting

# Variants of boosting and related algorithms

There are hundreds of variants of boosting, most important:

Gradient  
boosting

- Like AdaBoost, but useful beyond basic classification
- XGBoost (Tianqi Chen, UW)

# Variants of boosting and related algorithms

There are hundreds of variants of boosting, most important:

Gradient  
boosting

- Like AdaBoost, but useful beyond basic classification
- XGBoost (Tianqi Chen, UW)

Many other approaches to learn ensembles, most important:

Random  
forests

- Bagging: Pick random subsets of the data
  - Learn a tree in each subset
  - Average predictions
- Simpler than boosting & easier to parallelize
- Typically higher error than boosting for same # of trees (# iterations T)

# Impact of boosting (*spoiler alert... HUGE IMPACT*)

Amongst most useful ML methods ever created

Extremely useful in  
computer vision

- Standard approach for face detection, for example

Used by most winners of  
ML competitions  
(Kaggle, KDD Cup,...)

- Malware classification, credit fraud detection, ads click through rate estimation, sales forecasting, ranking webpages for search, Higgs boson detection,...

Most deployed ML systems use model  
ensembles

- Coefficients chosen manually, with boosting, with bagging, or others

# What you should know about today's material for final

- What a decision tree is, basic greedy approach to constructing them, what the decision boundary will look like if data is real-valued (axis-aligned partitions)
- Mitigating overfitting and high variance by limiting depth, pruning tree, bagging and random forests.
- Understand idea of ensemble classifier
- Outline the boosting framework – sequentially learn classifiers on weighted data. Key idea: put more weight on data points where making error.
- Understand the AdaBoost algorithm at a high level
  - Learn each classifier on weighted data
  - Compute coefficient of classifier
  - Recompute data weights
  - Normalize weights