WELCOME

Introduction to Machine Learning

DBDA.X408.(33)

Instructor:

Bill Chen

UCSC Silicon Valley Extension
PROFESSIONAL EDUCATION

UCSC Silicon Valley Extension

E: xch375@ucsc.edu

Week 5

Decision Tree.

Boosting.

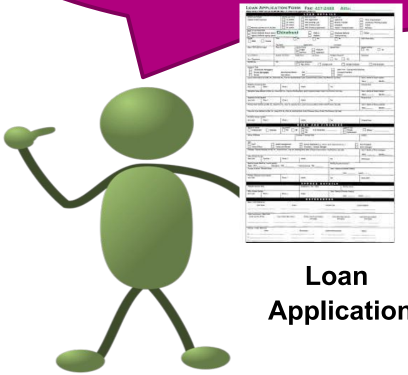
Support Vector Machines.

Colab Demonstration.

Decision Tree

What makes a loan risky?

I want a to buy a new house!



Loan Application

Credit History ★★★★

Income ★★★

Term ★★★★★

Personal Info ★★★

Credit history explained

Did I pay previous loans on time?

Example: excellent, good, or fair

Credit History
★★★★

Income
★★★

Term
★★★★★

Personal Info
★★★

Credit history explained

What's my income?

Example:
$80K per year

Credit History
★★★★

Income
★★★

Term
★★★★★

Personal Info
★★★

Credit history explained

How soon do I need to pay the loan?

Example: 3 years, 5 years,…

Credit History
★★★★

Income
★★★

Term
★★★★★

Personal Info
★★★

Credit history explained

Credit History
★★★★

Income
★★★

Term
★★★★★

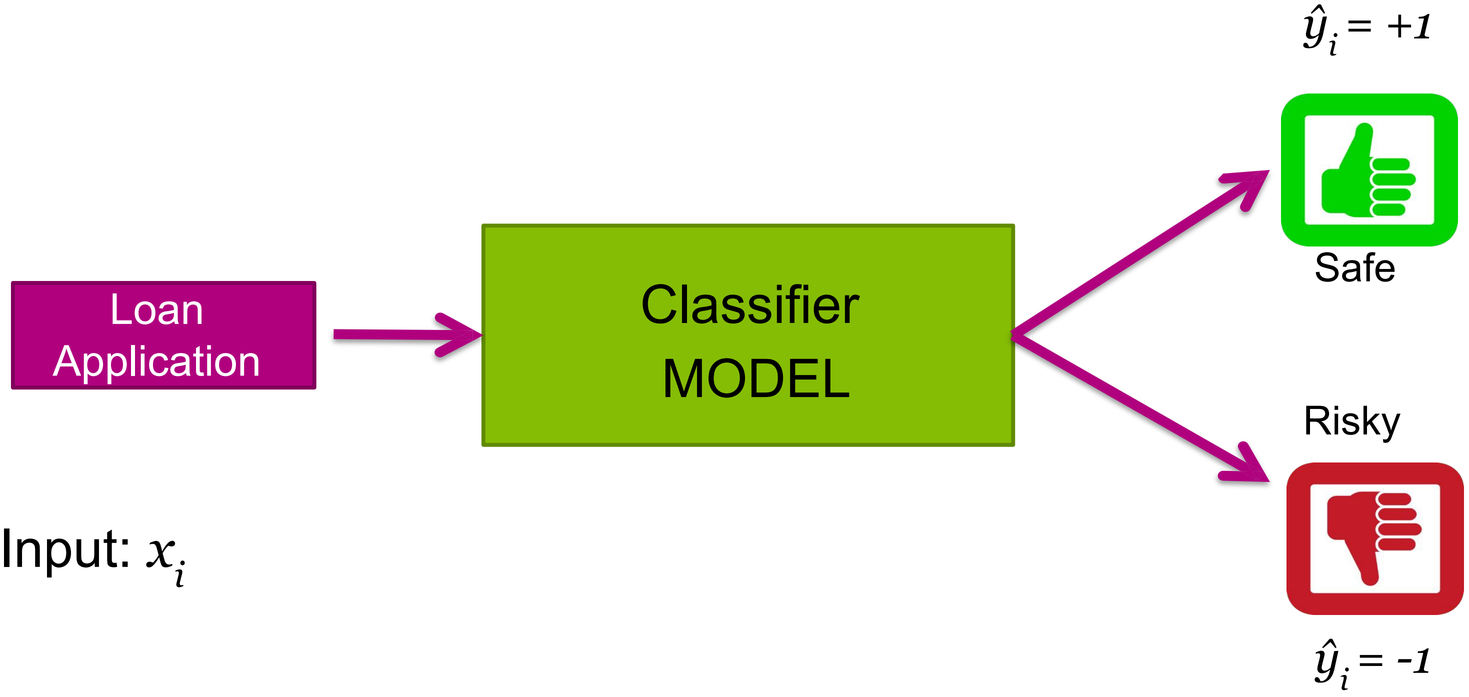
Personal Info
★★★

Age, reason for the loan, marital status,…

Example: Home loan for a married couple

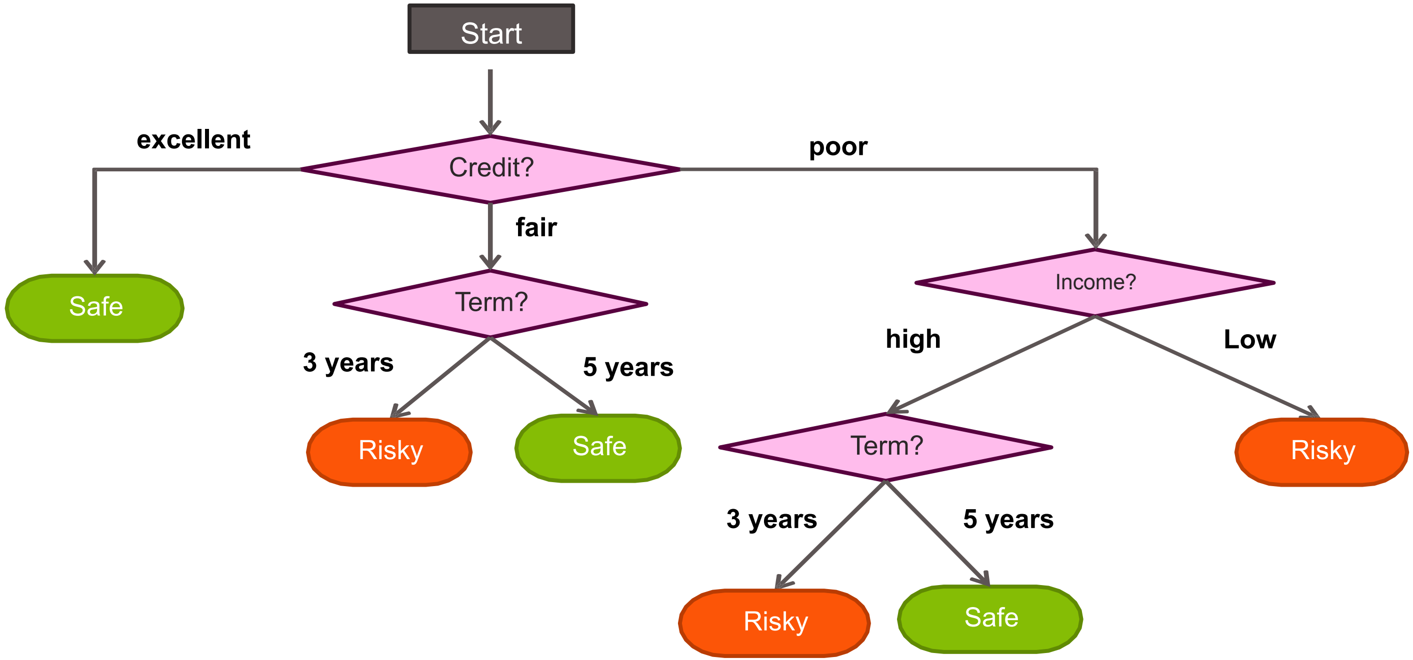
Classifier Review

Output: $\hat{y}$ Predicted class



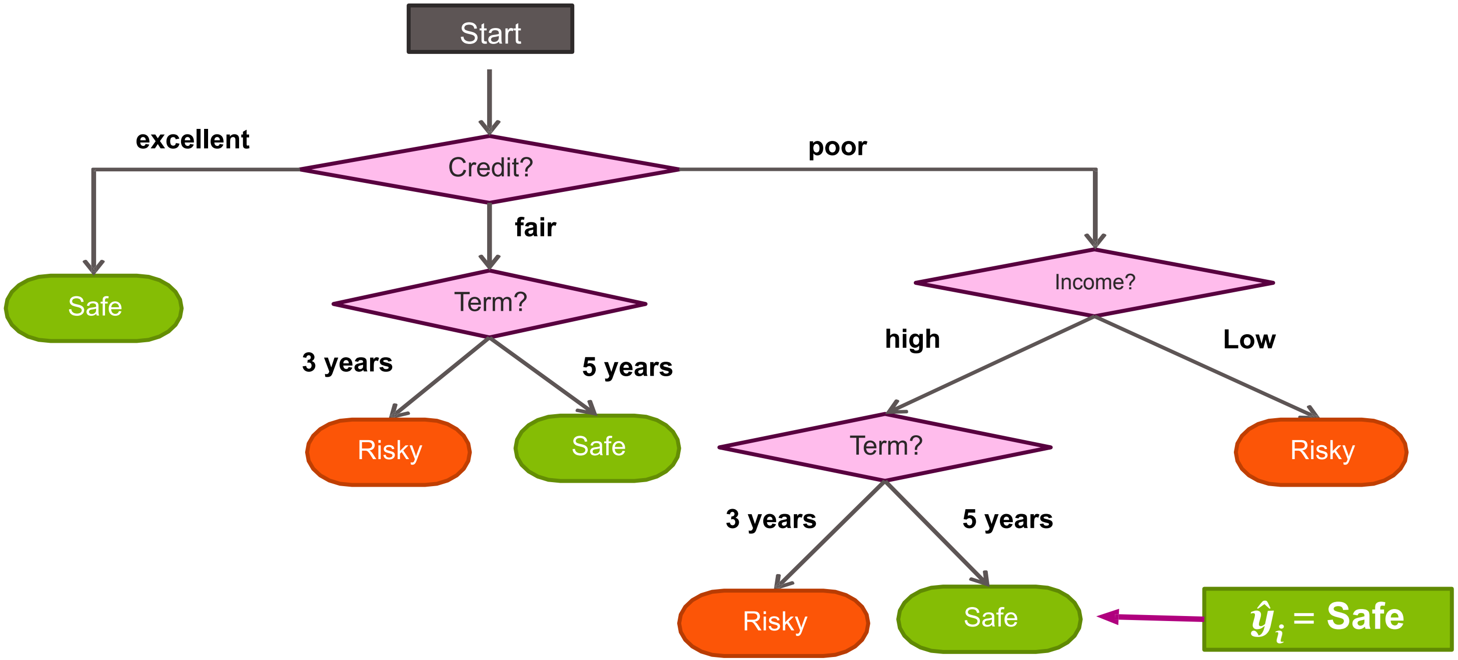
Input: x_i

Decision Tree



Decision Tree

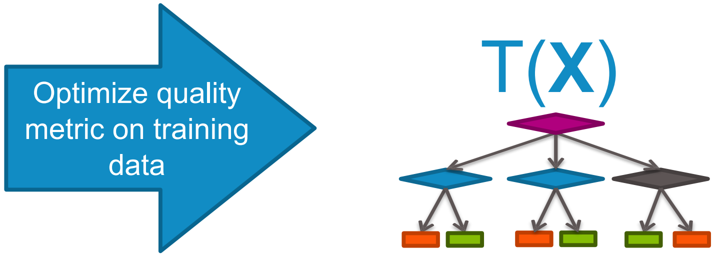
$x_i = (Credit = poor, Income = high, Term = 5\ years)$



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

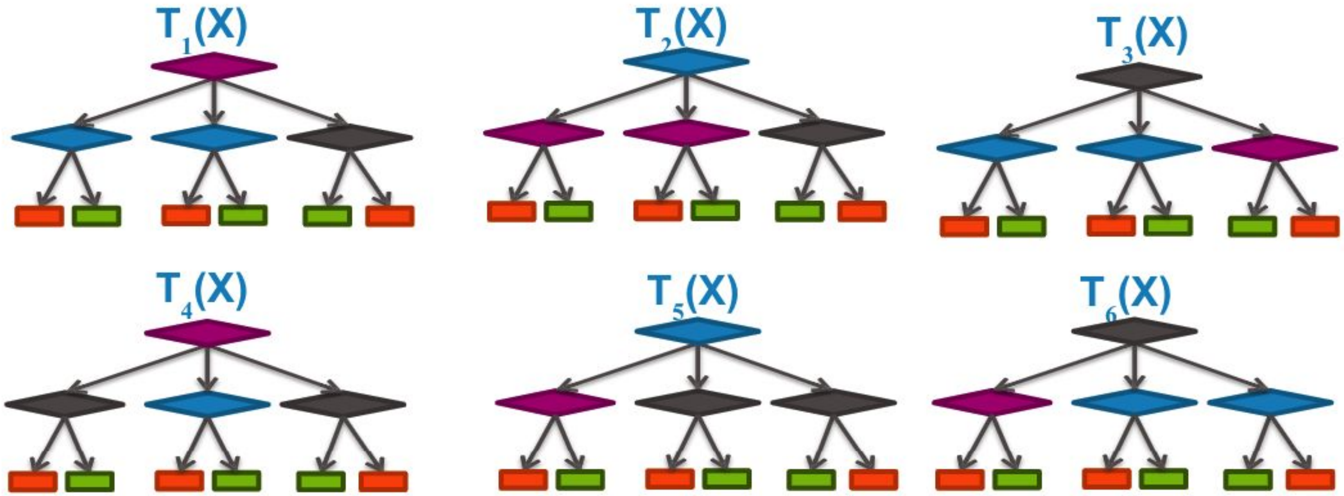
Error measures fraction of mistakes:

Error = # incorrect predictions / # example

- 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]

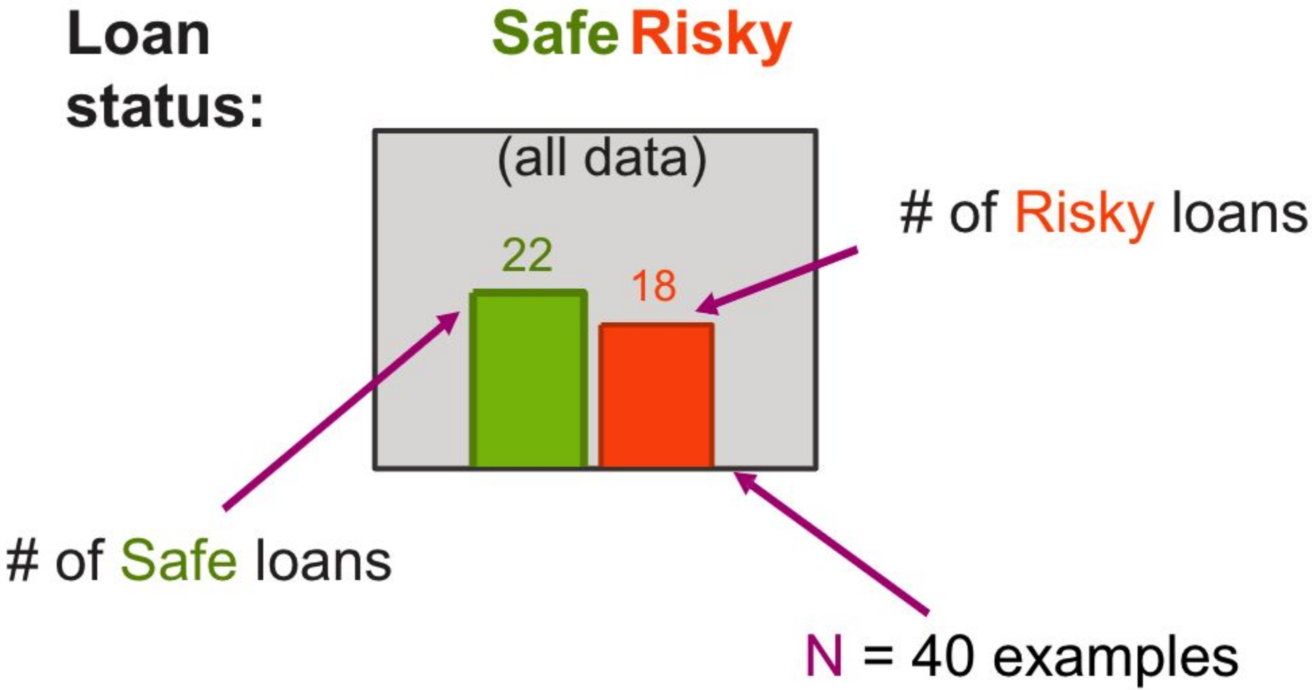
Greedy Decision Tree Learning

Our Training Data Table

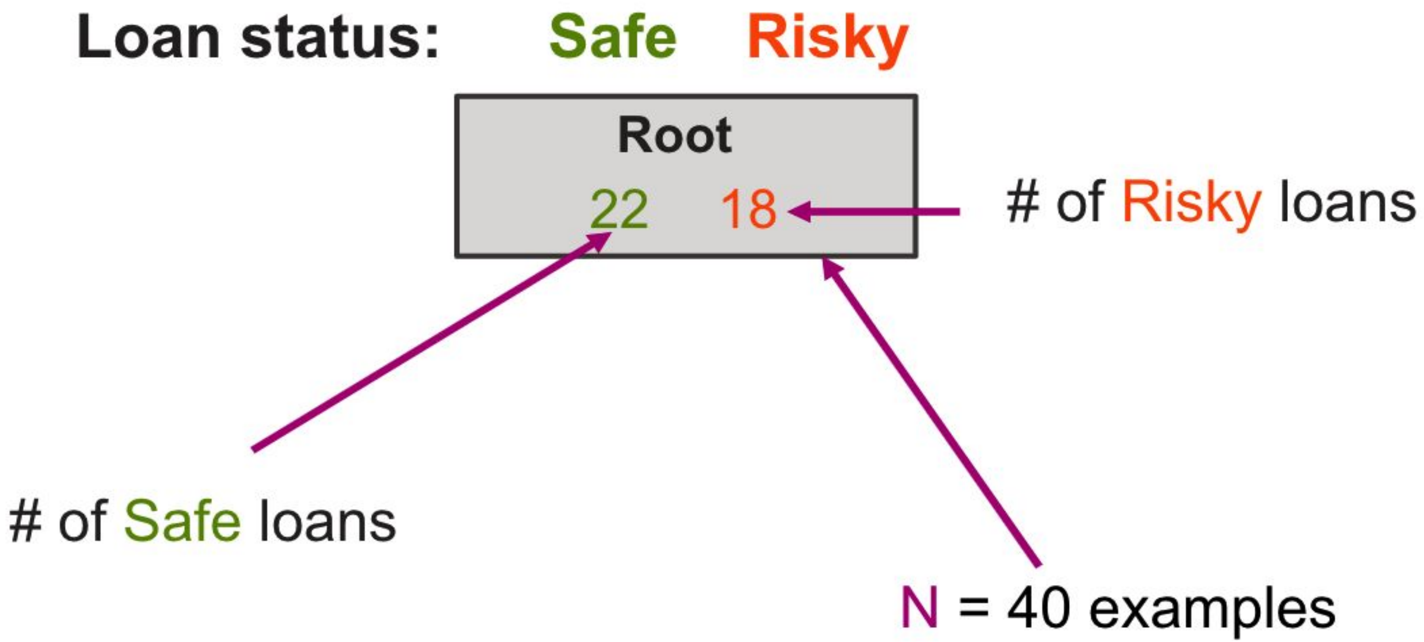
Assume N = 40, 3 features

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

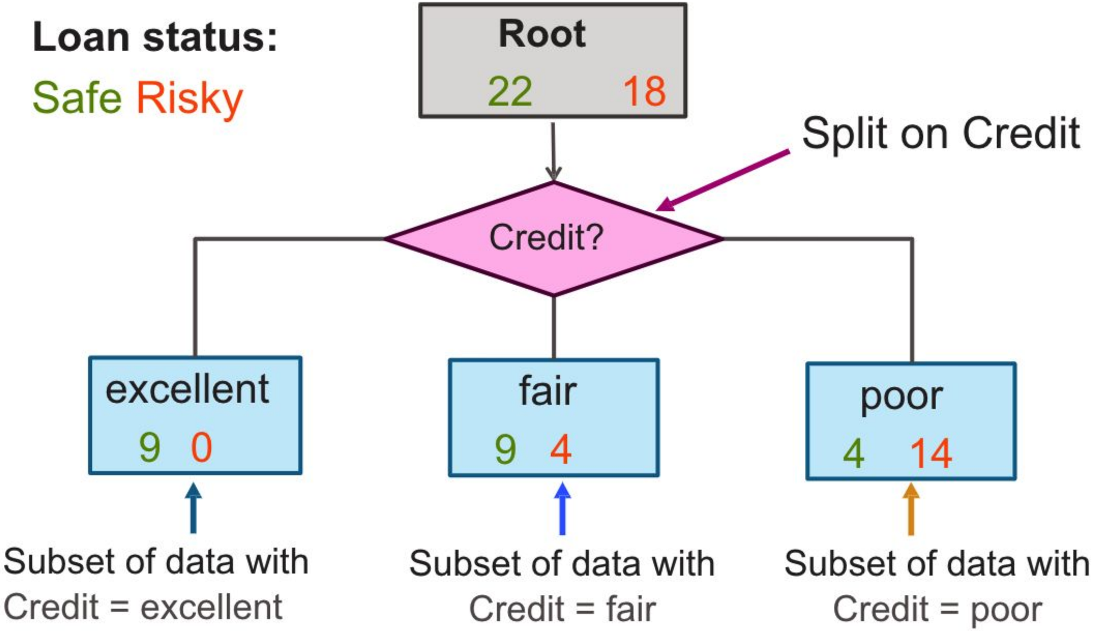
Start With All The Data



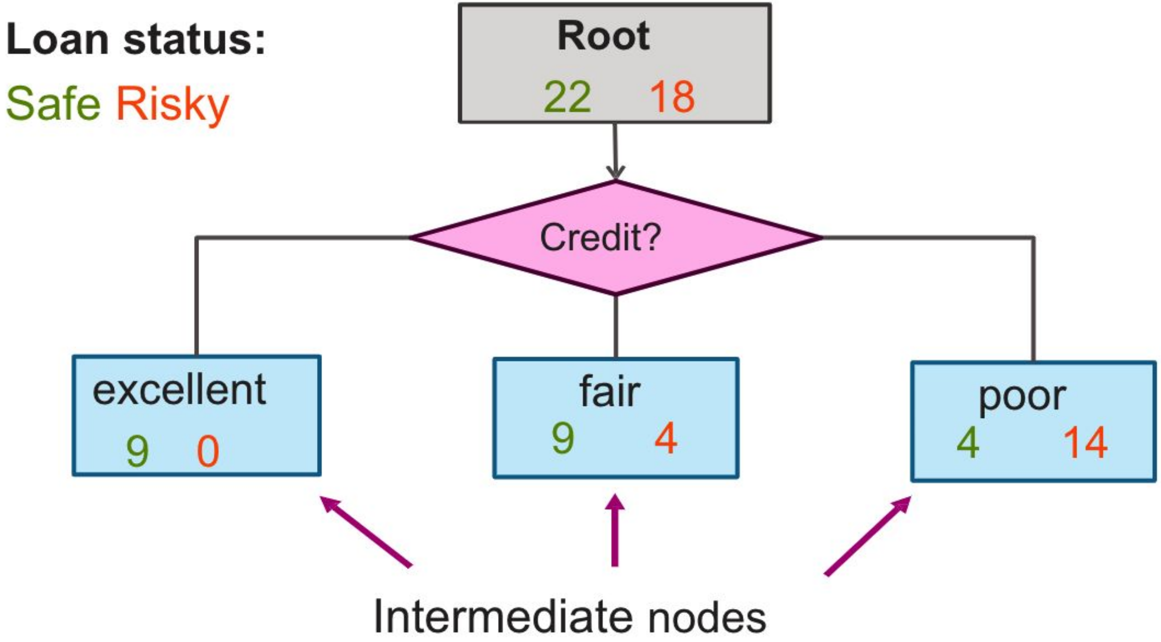
Compact Visual Notation: Root Node



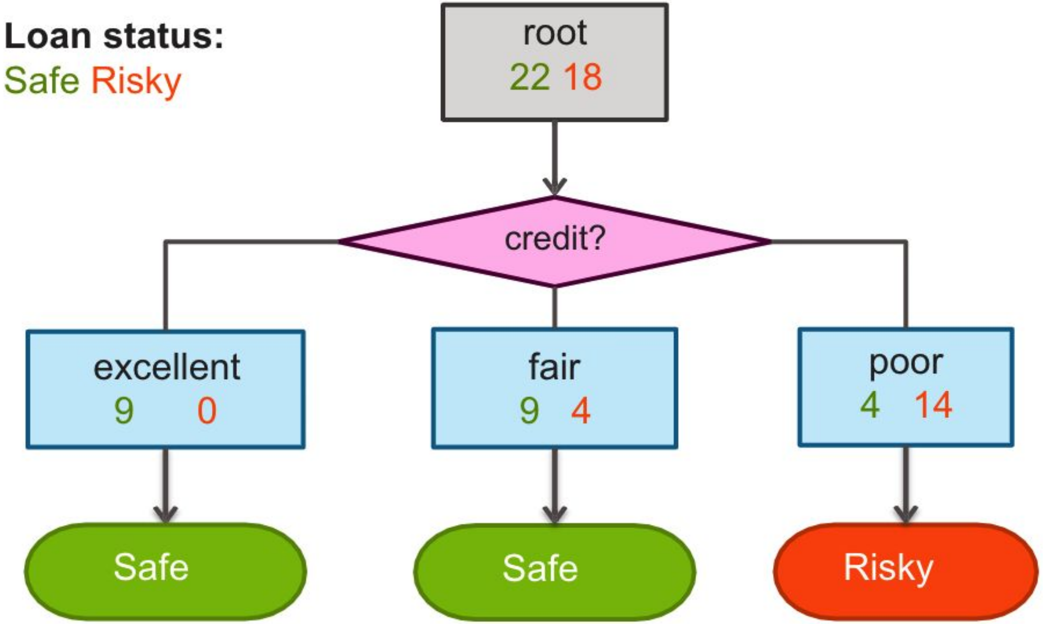
Decision Stump: Single Level Lree



Visual Notation: Intermediate Nodes



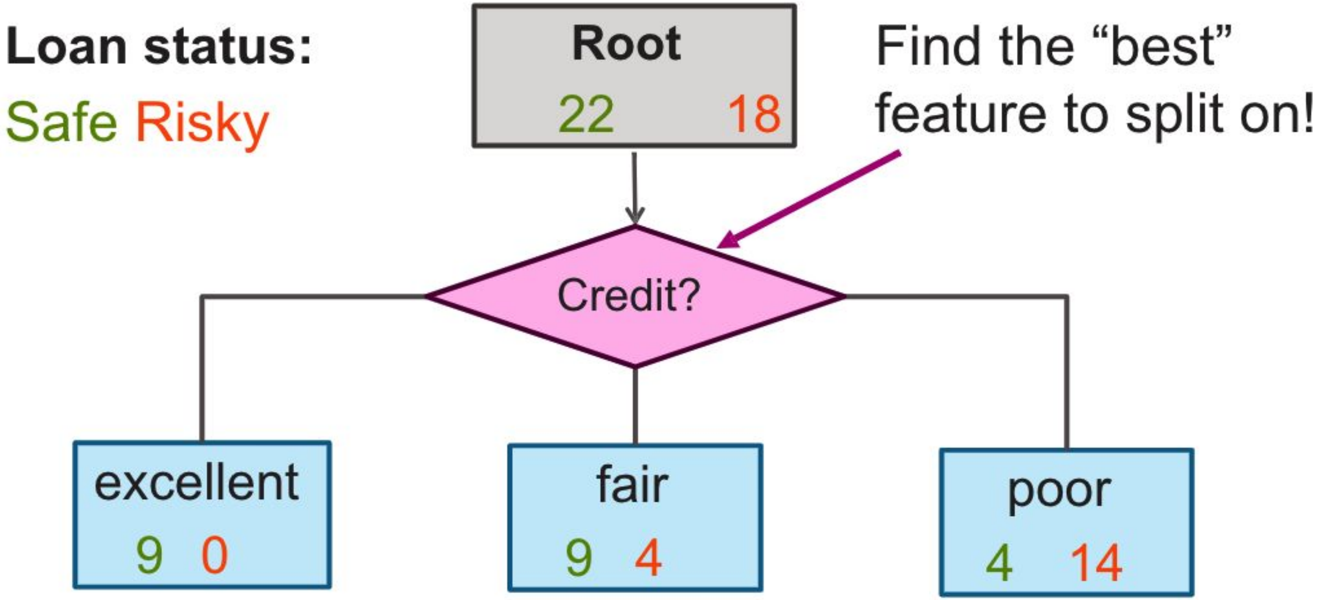
Making Predictions With a Decision Stump



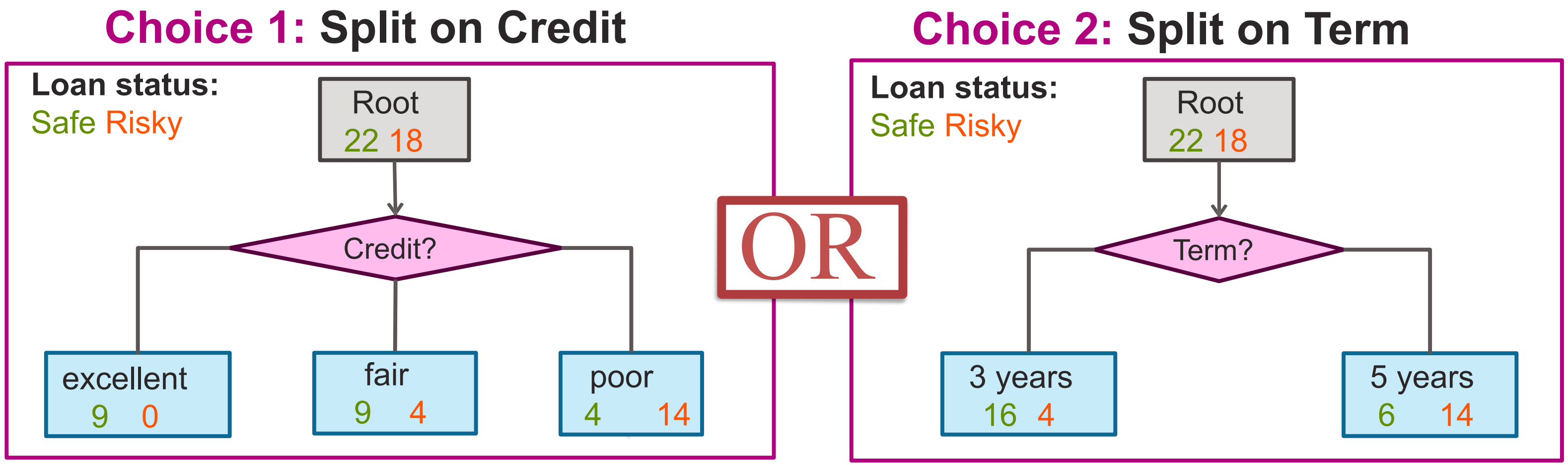
For each intermediate node, set
$\hat{y}$ = majority value

Selecting The Best Feature To Split On

How do we learn a decision stump?

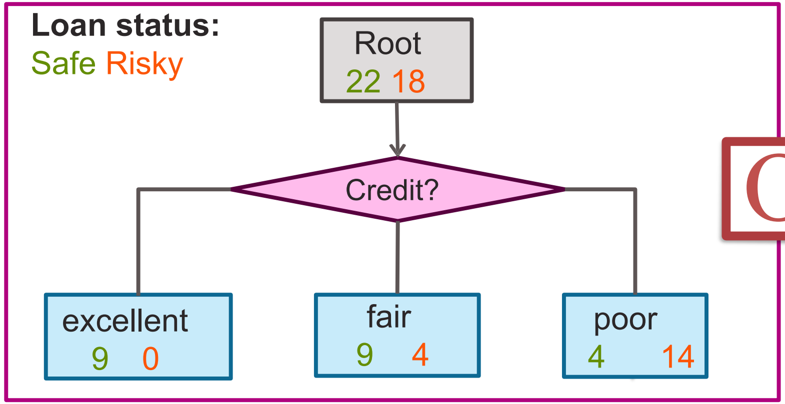


How do we select the best feature?



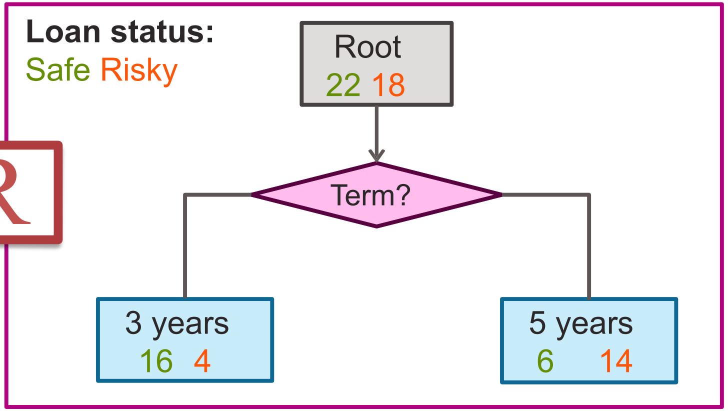
How do we select the best feature?

Choice 1: Split on Credit

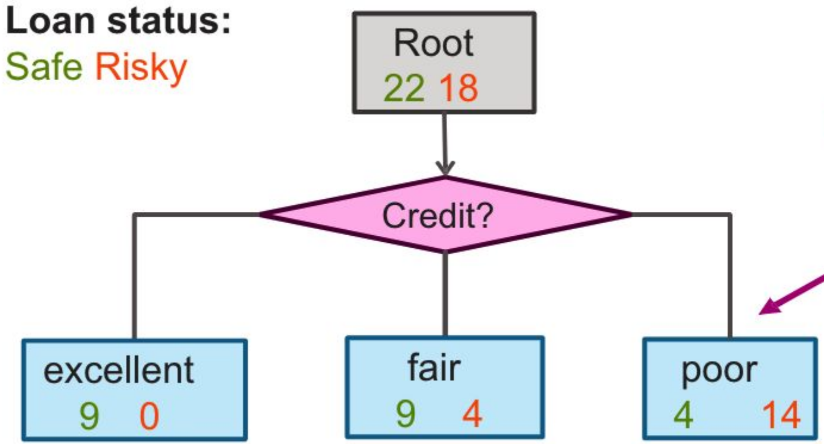


OR

Choice 2: Split on Term



How do we measure effectiveness of a split?



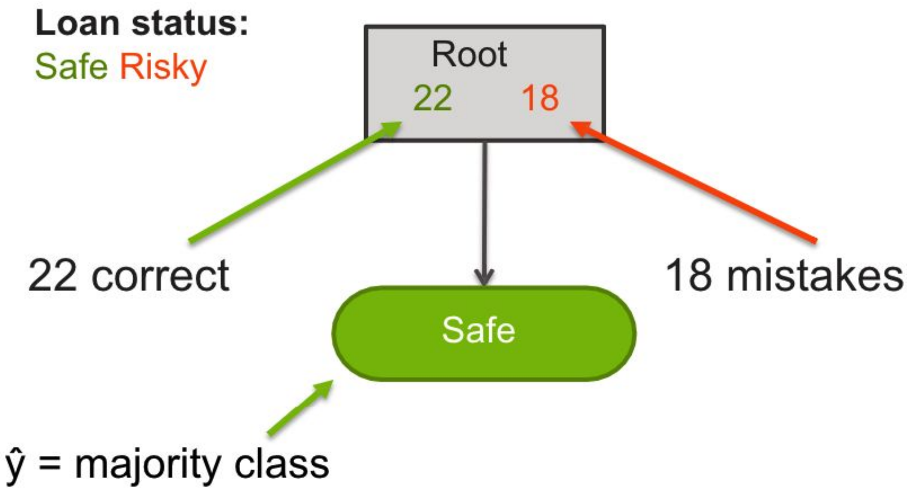
Idea: Calculate classification error of this decision stump

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

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

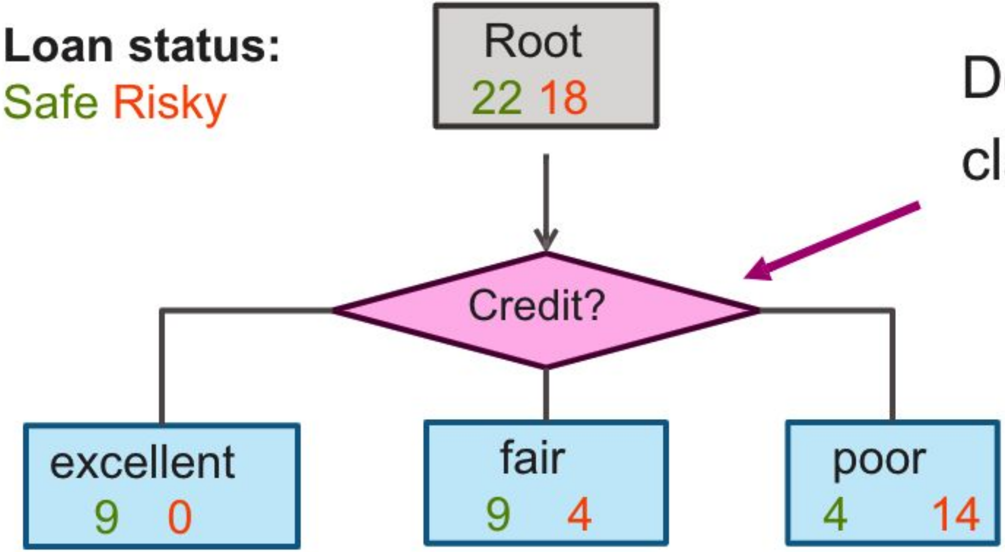


Tree	Classification error
(root)	0.45

How to do the calculation?

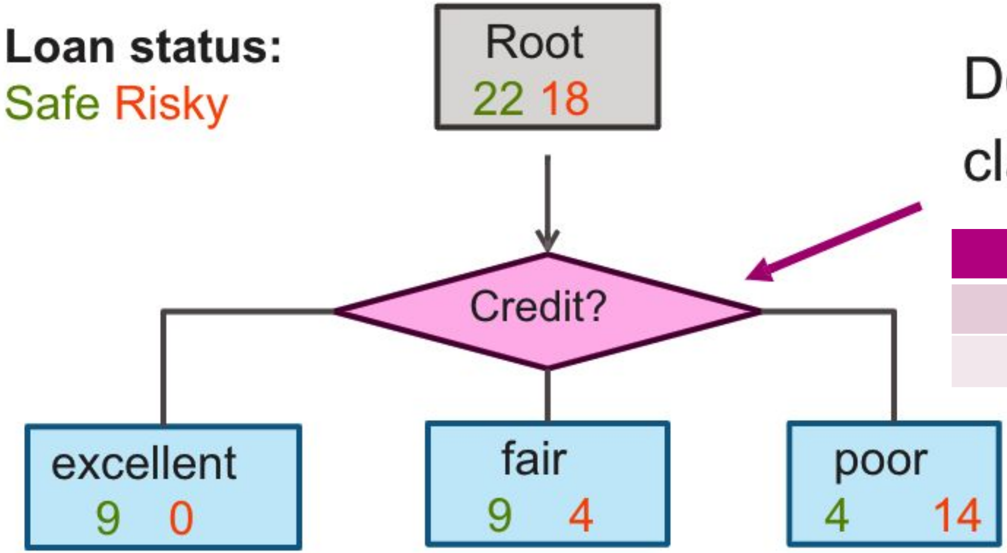
Choice 1: Split on Credit history?

Loan status:
Safe Risky



Does a split on Credit reduce classification error below 0.45?

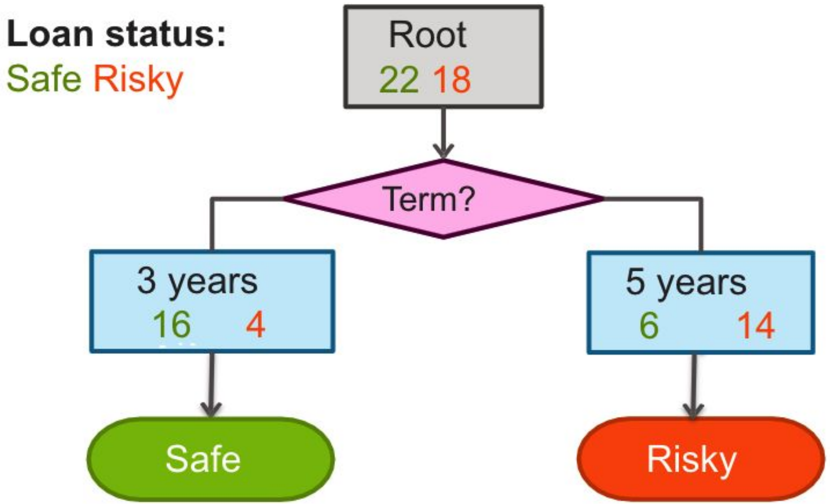
Choice 1: Split on Credit history?



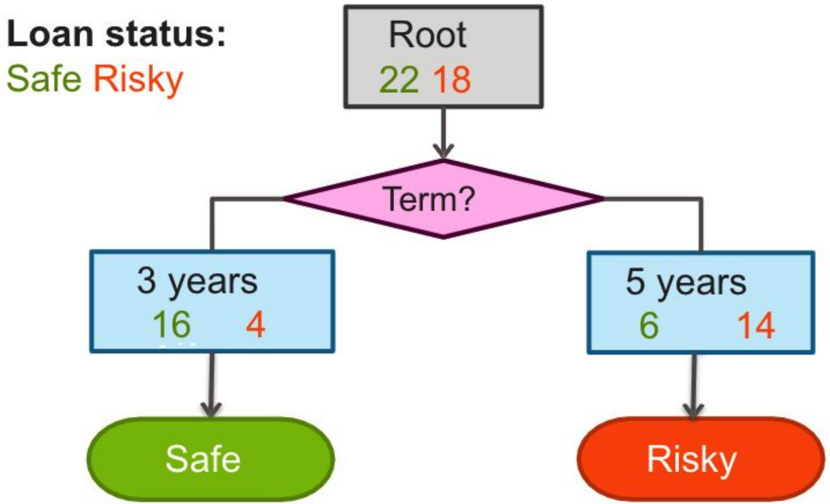
Does a split on Credit reduce classification error below 0.45?

Tree	Classification error
(root)	0.45
Split on credit	0.2

Choice 2: Split on Term?



Choice 2: Split on Term?

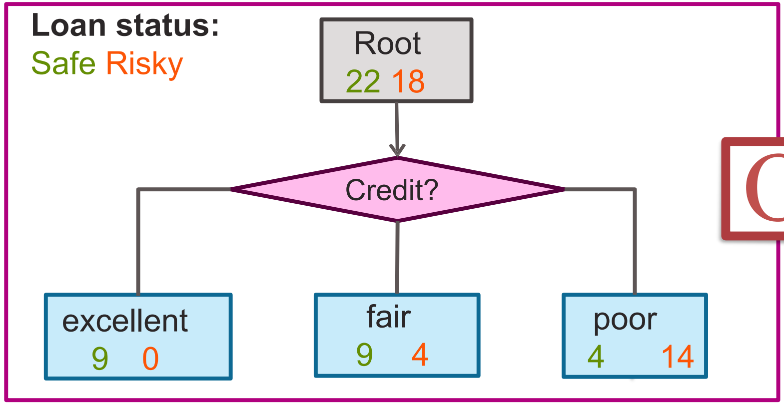


Tree	Classification error
(root)	0.45
Split on credit	0.2
Split on term	0.25

Choice 1 or Choice 2?

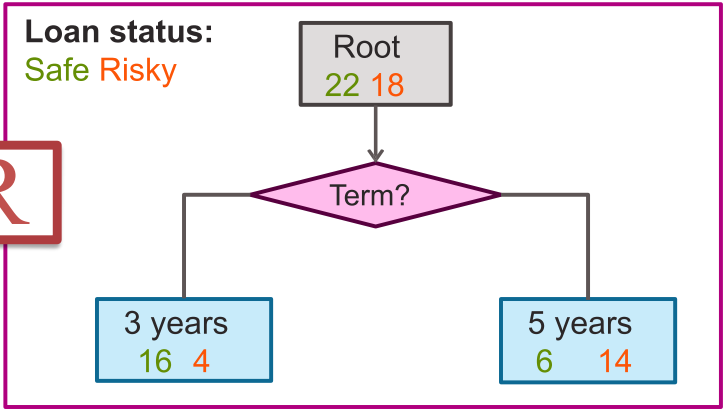
Tree	Classification error
(root)	0.45
Split on credit	0.2
Split on term	0.25

Choice 1: Split on Credit



OR

Choice 2: Split on Term

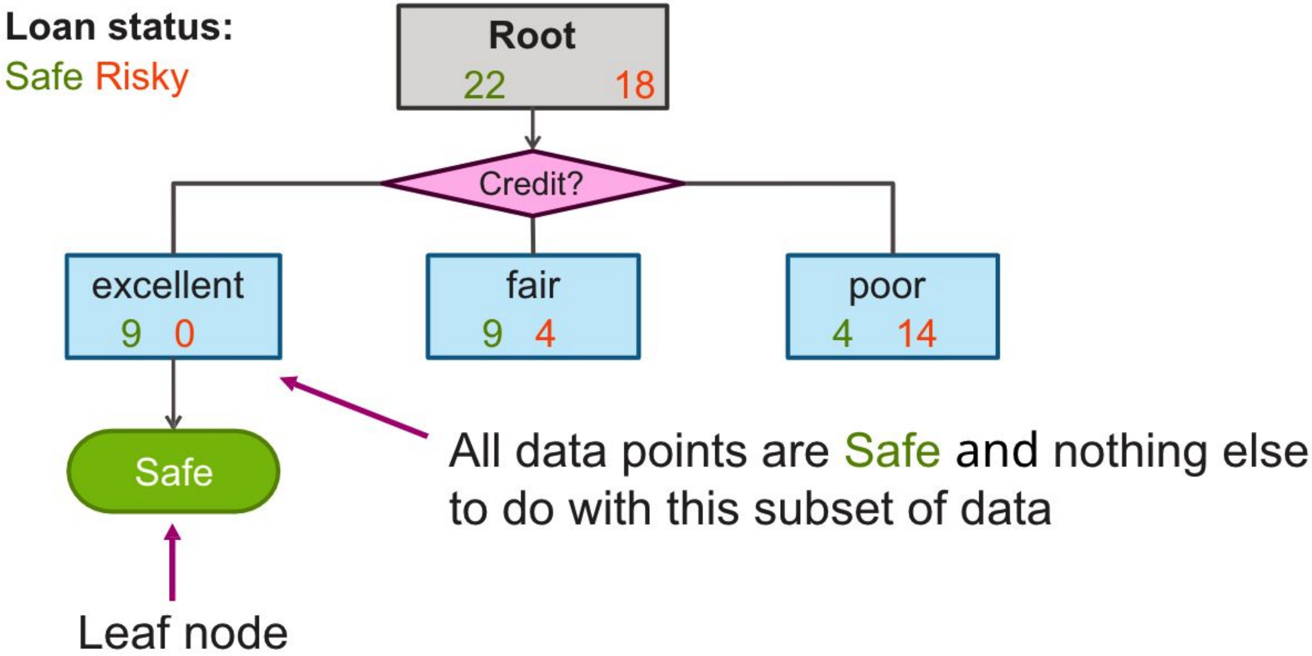


Feature Split Selection Algorithm

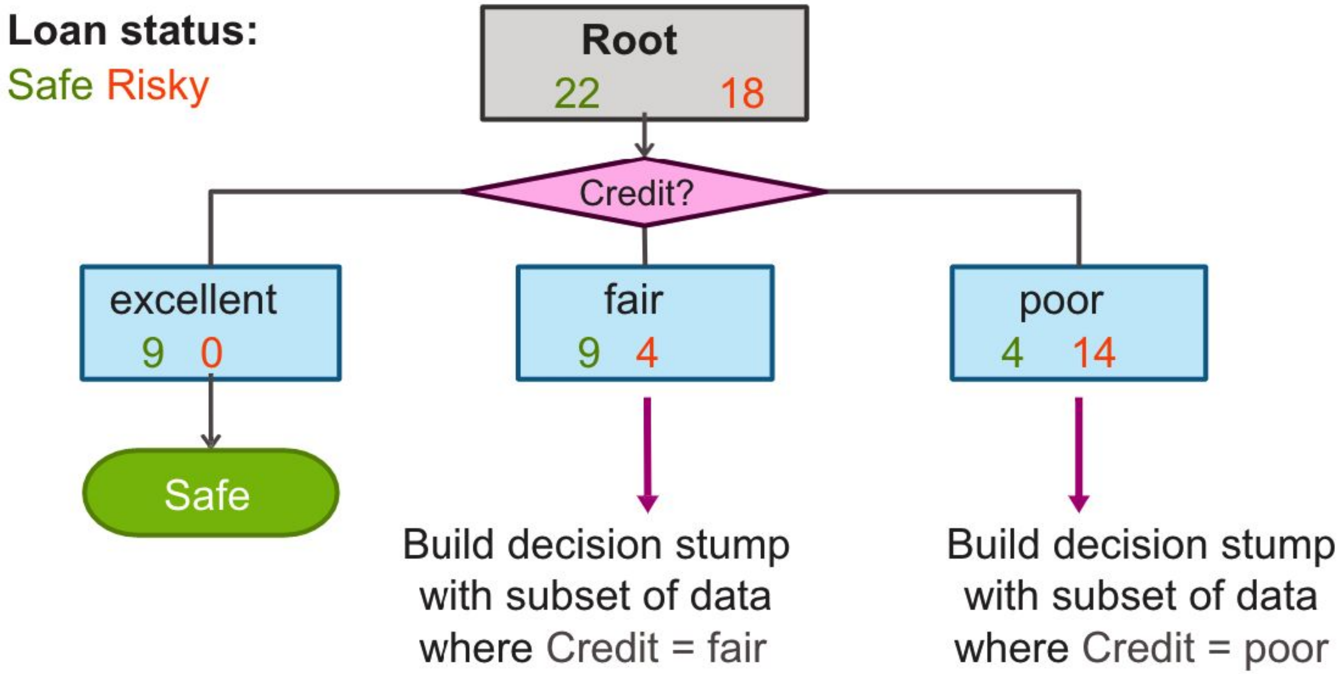
- Given a subset of data M (a node in a tree)
- For each feature $h_i(x)$:
  1. Split data of M according to feature $h_i(x)$
  2. Compute classification error of split
- Chose feature $h^*(x)$ with lowest classification error

Recursion & Stopping conditions

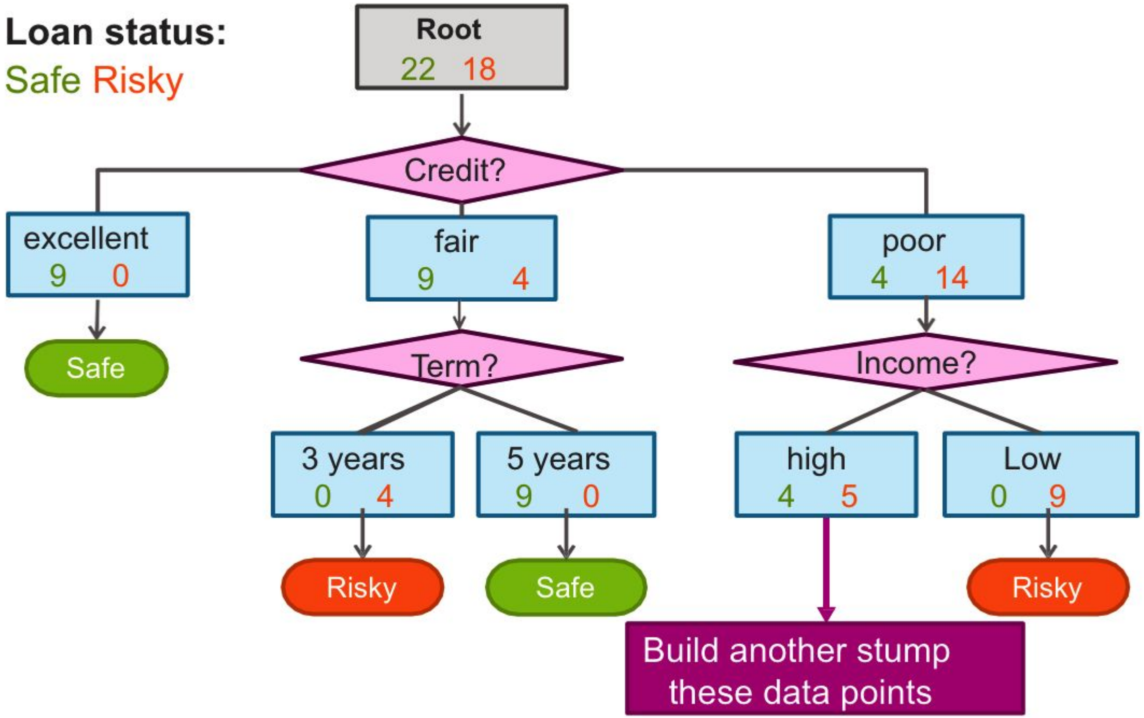
Can We Reuse The Data After Split?



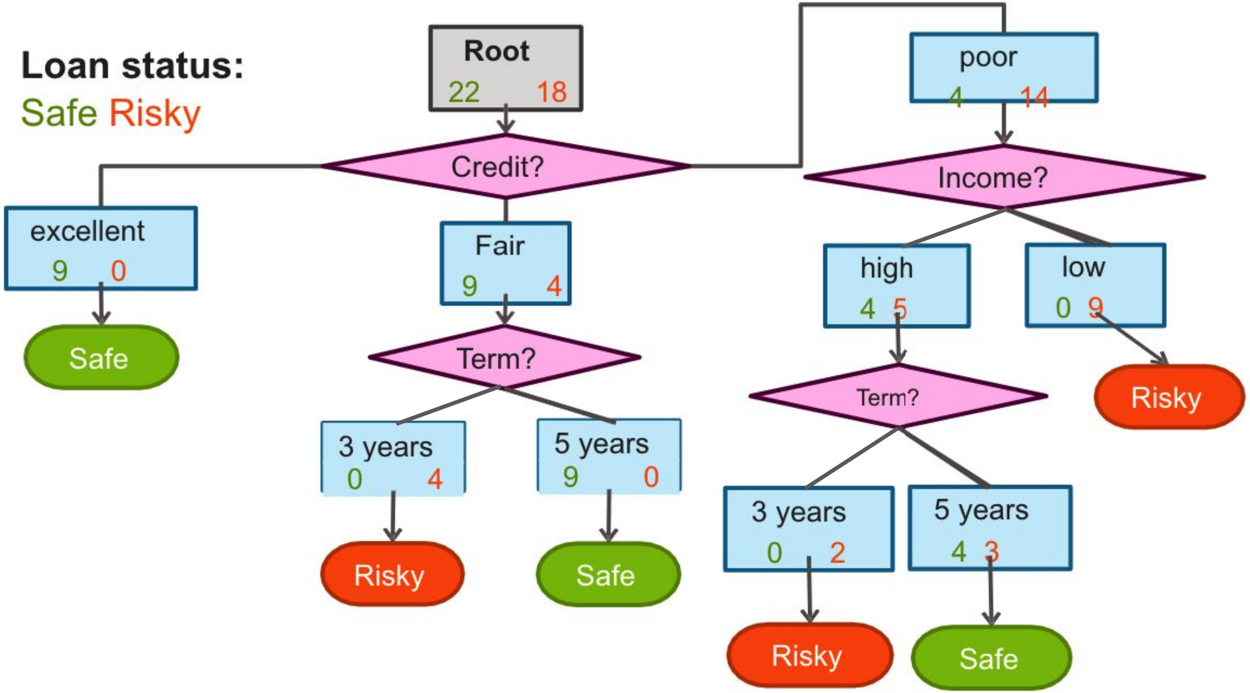
Tree Learning = Recursive Stump Learning



Second Level



Final Decision Tree



Simple greedy decision tree learning

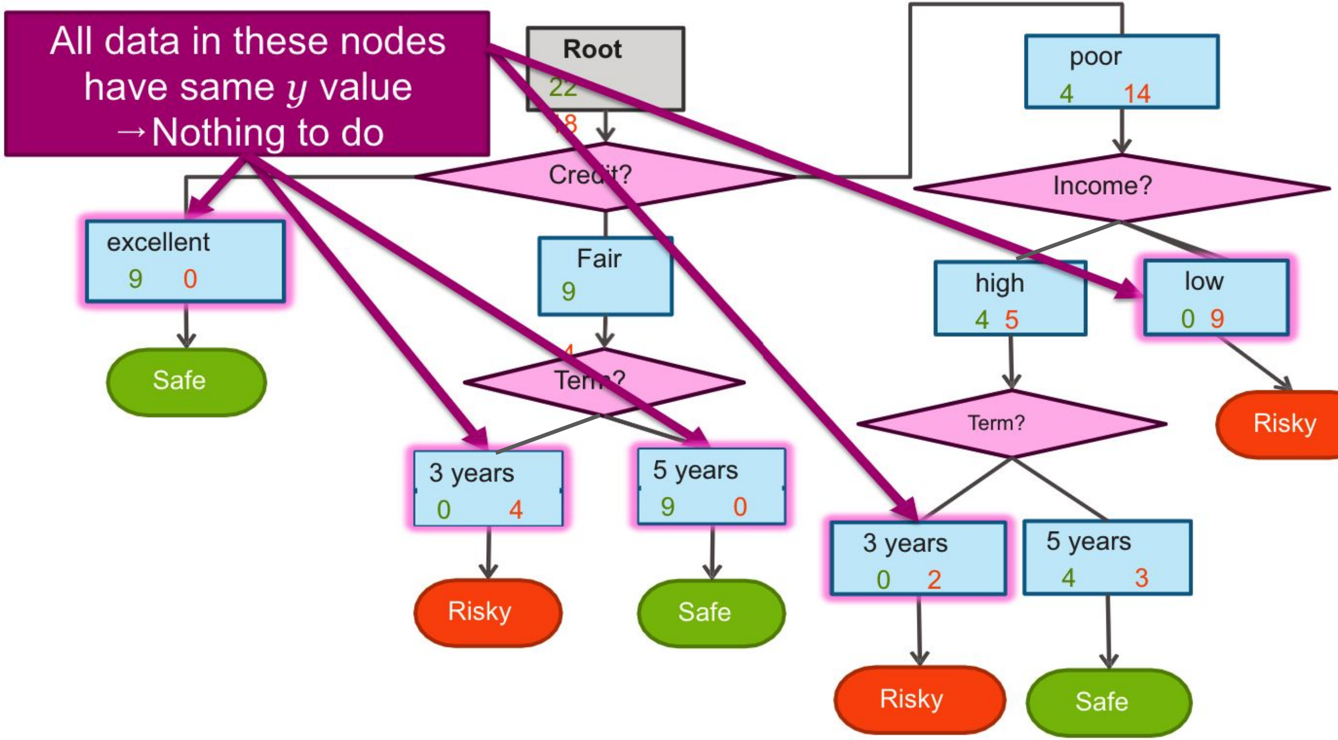
Pick best feature to split on

Learn decision stump with this split

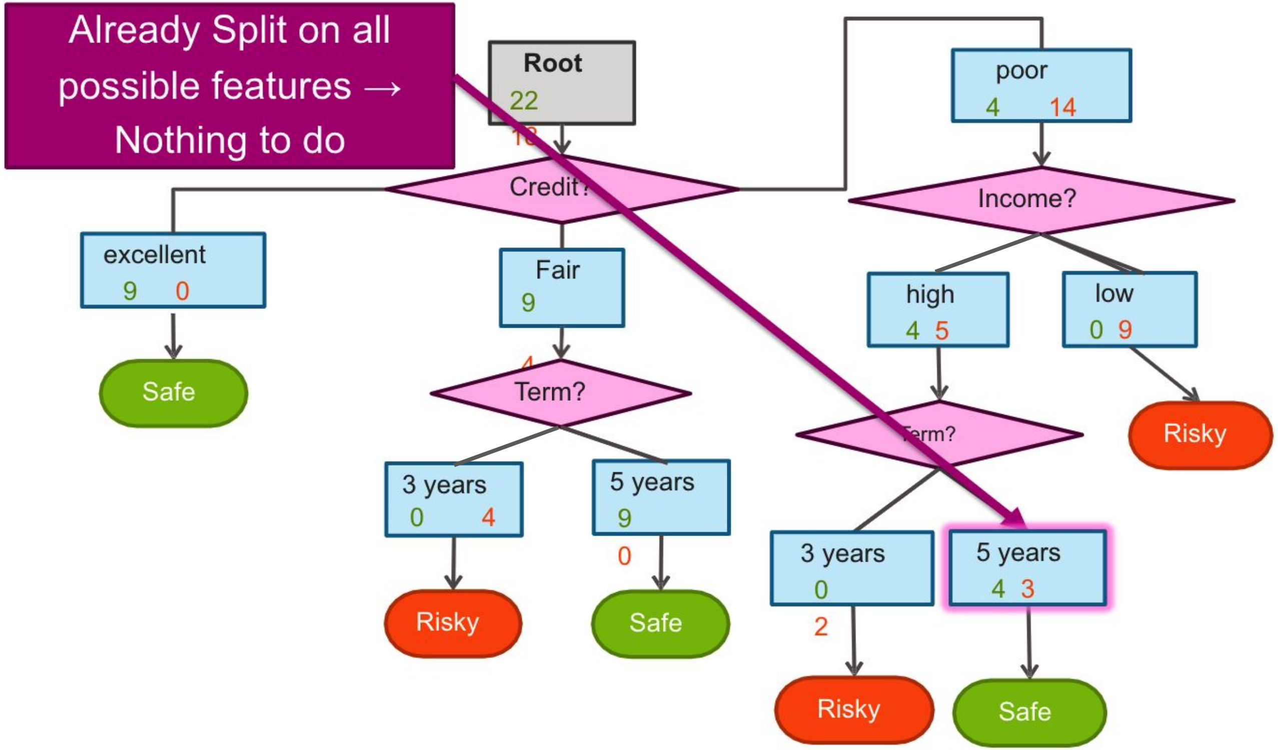
For each leaf of decision stump, recurse

When do we stop???

Stopping Condition 1: All Data Agrees On y



Stopping Condition 2: Already Split On All Features



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

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???

$y = x_{[1]}$ xor $x_{[2]}$

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

y values
True False

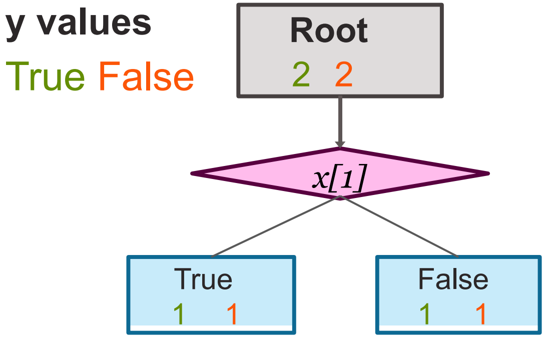
Root
2 2

Tree	Classification error
(root)	0.5

Consider Split On $x_{[1]}$

$y = x_{[1]}$ xor $x_{[2]}$

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



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

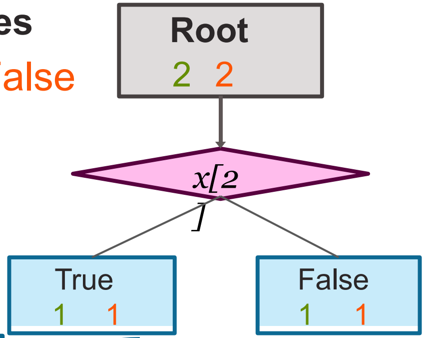
Consider Split On $x_{[2]}$

$y = x_{[1]}$ xor $x_{[2]}$

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

y values

True False



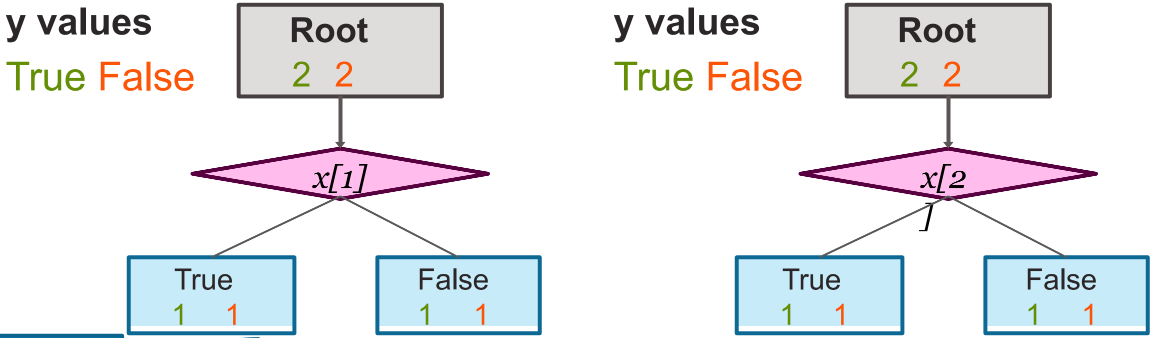
Neither features improve training error… Stop now???

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

Consider Split On $x_{[2]}$

$y = x_{[1]}$ xor $x_{[2]}$

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



Neither features improve training error… Stop now???

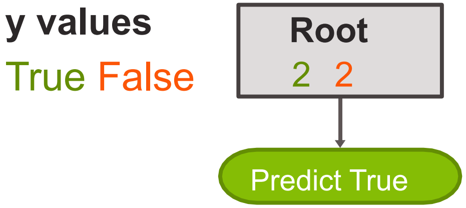
Tree	Classification error
(root)	0.5
Split on x[1]	0.5
Split on x[2]	0.5

Final Tree With Stopping Condition 3

$y = x_{[1]}$ 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



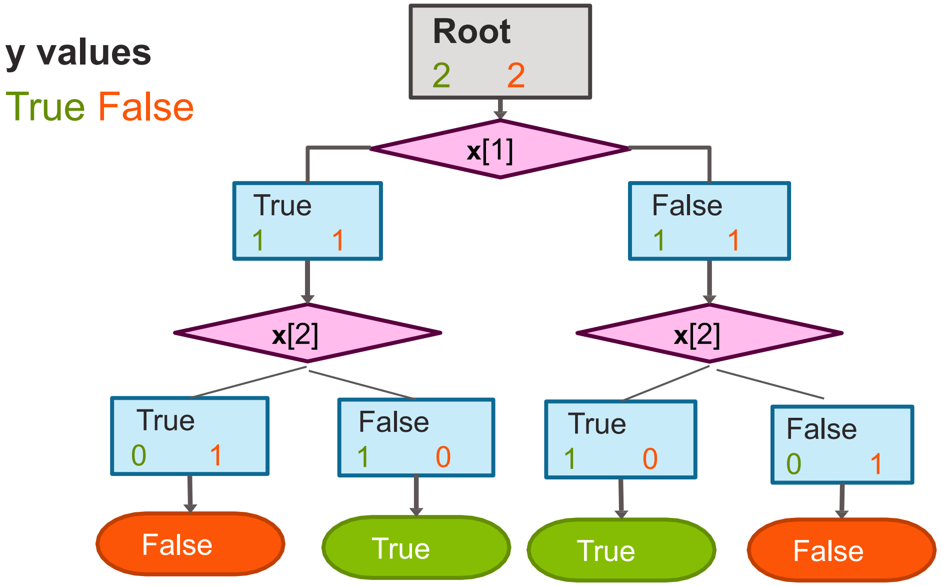
Without Stopping Condition 3

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

$y = x_{[1]}$ 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	

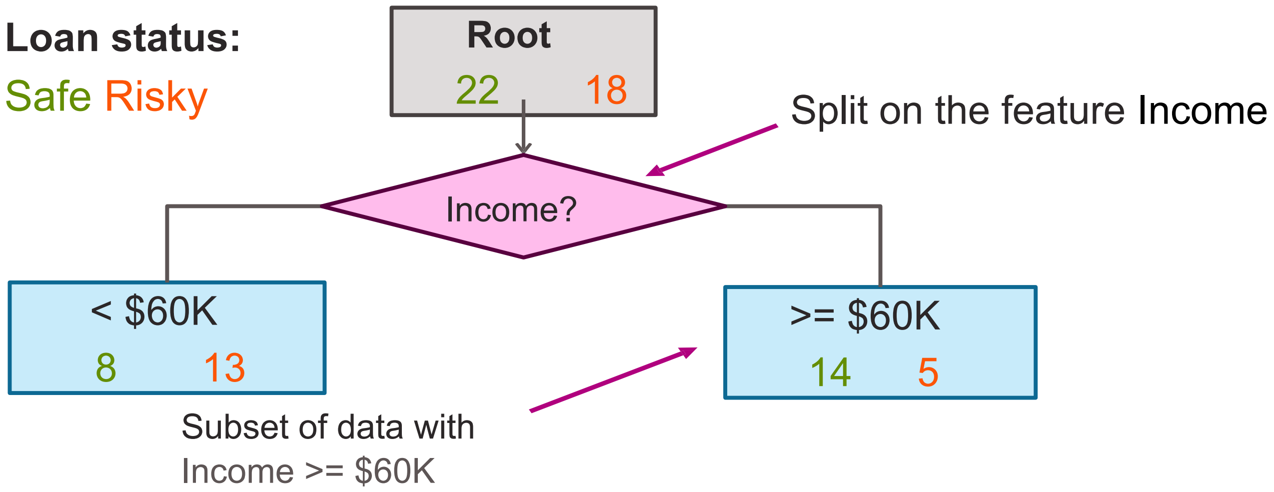


Decision Tree Learning: Real Valued Features

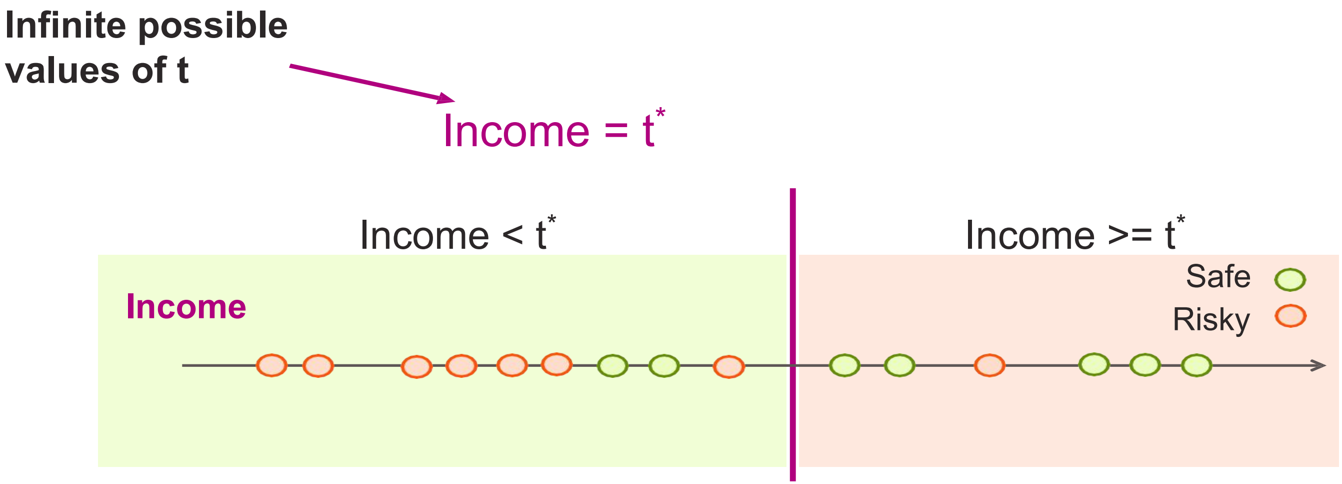
How Do We Use Real Values Inputs?

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

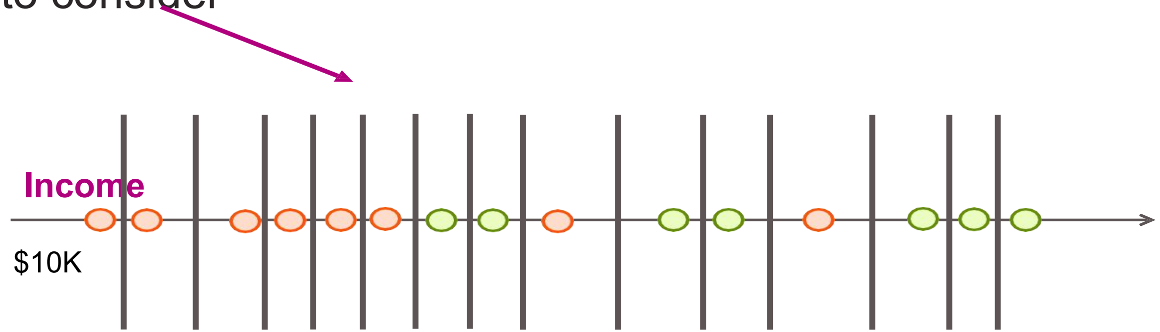


Finding The Best Threshold Split



Only Need To Consider Mid-points

Finite number of
splits to consider



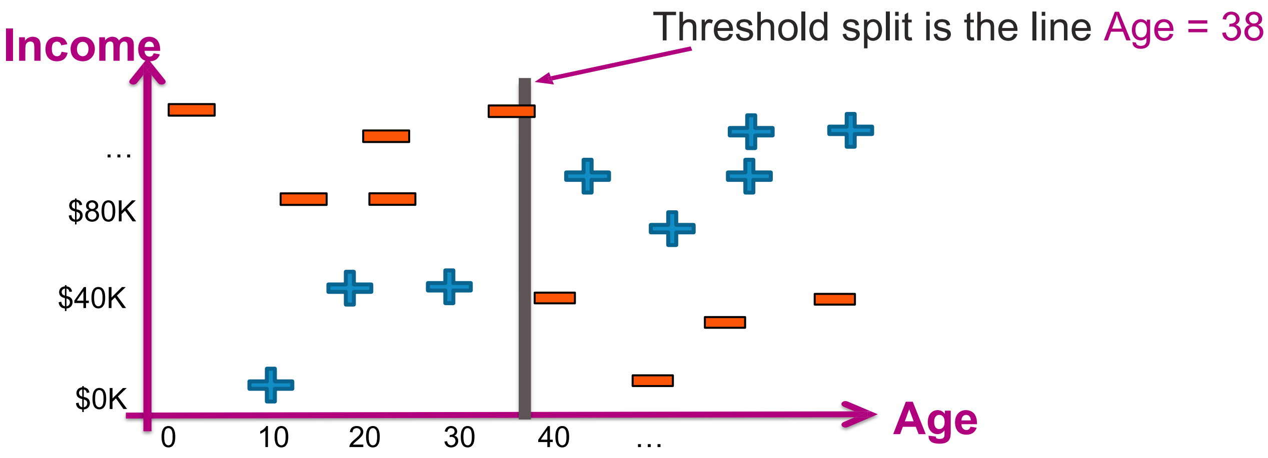
Threshold split selection algorithm

- Step 1: Sort the values of a feature $h_j(x)$:

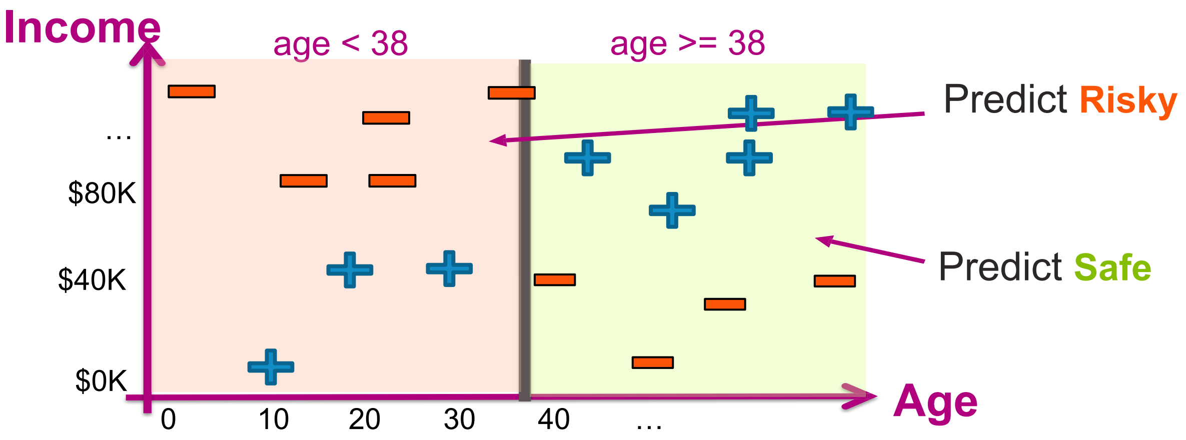
  Let $\{v_1, v_2, v_3, \ldots v_N\}$ denote sorted values

- Step 2:
  - For $i = 1 \ldots N\text{-}1$
    - Consider split $t_i = (v_i + v_{i+1}) / 2$
    - Compute classification error for threshold split $h_j(x) >= t_i$
  - Chose the t^* with the lowest classification error

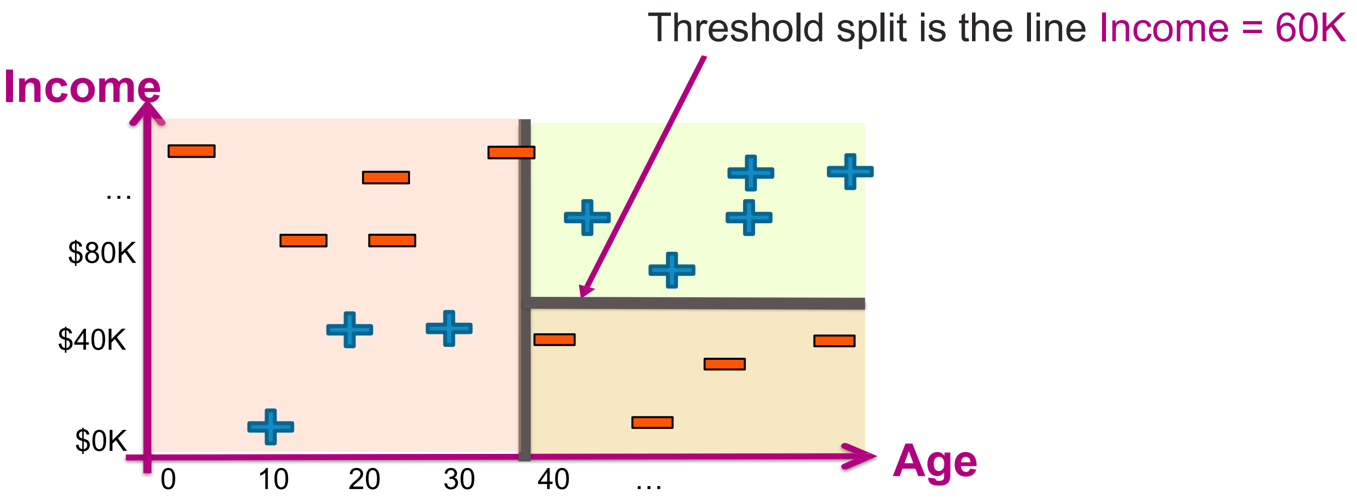
Visualizing the threshold split



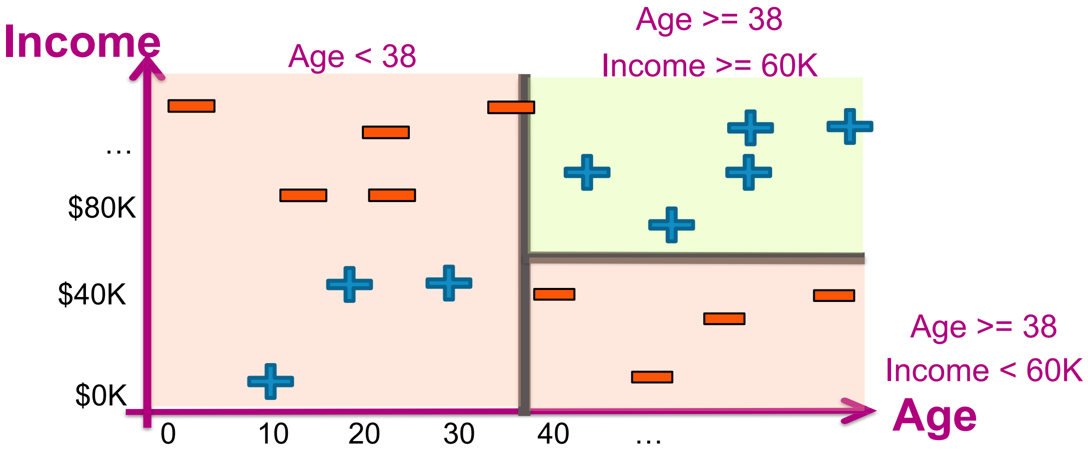
Split on Age >= 38



Depth 2: Split on Income >= $60K



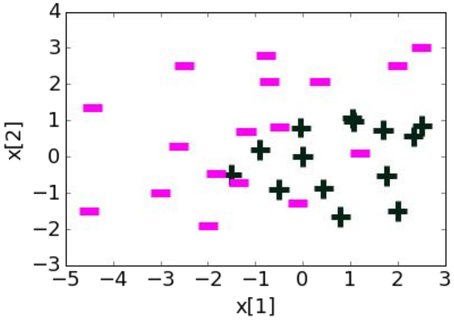
Each split partitions the 2-D space

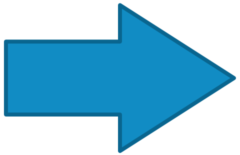


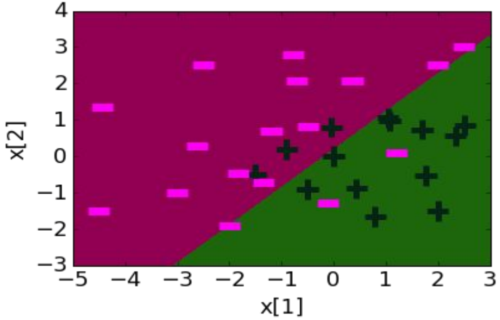
Decision trees vs logistic regression: Example

Logistic Regression

Feature	Value	Weight Learned
$h_0(x)$	1	0.22
$h_1(x)$	x[1]	1.12
$h_2(x)$	x[2]	-1.07



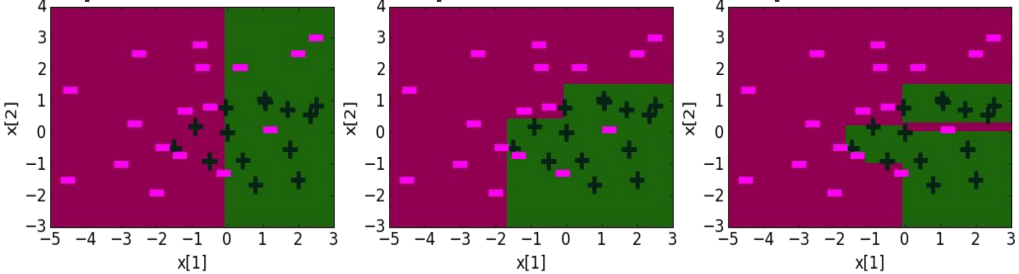




Comparing decision boundaries

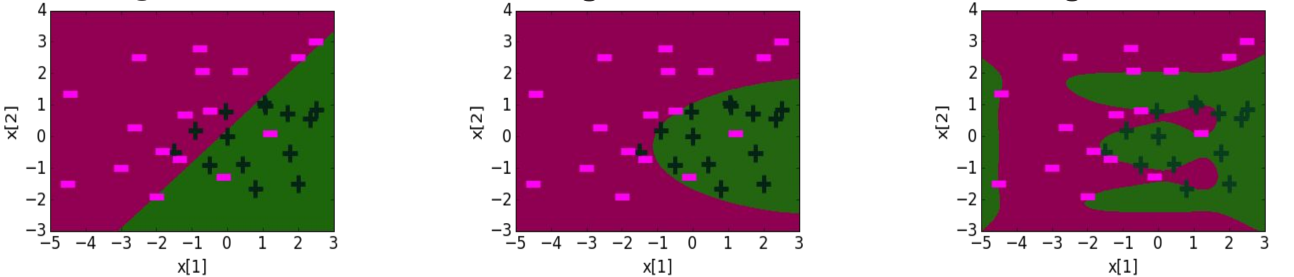
Decision Tree

Depth 1 | Depth 3 | Depth 10



Logistic Regression

Degree 1 features | Degree 2 features | Degree 6 features

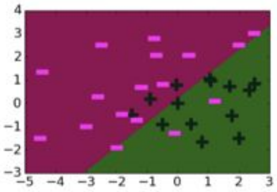


What you can do now

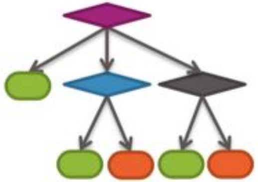
- Define a decision tree classifier
- Interpret the output of a decision trees
- Learn a decision tree classifier using greedy algorithm
- Traverse a decision tree to make predictions
    - Majority class predictions
- Tackle continuous and discrete features

Boosting

Simple (Weak) Classifiers Are Good



Logistic regression w. simple features



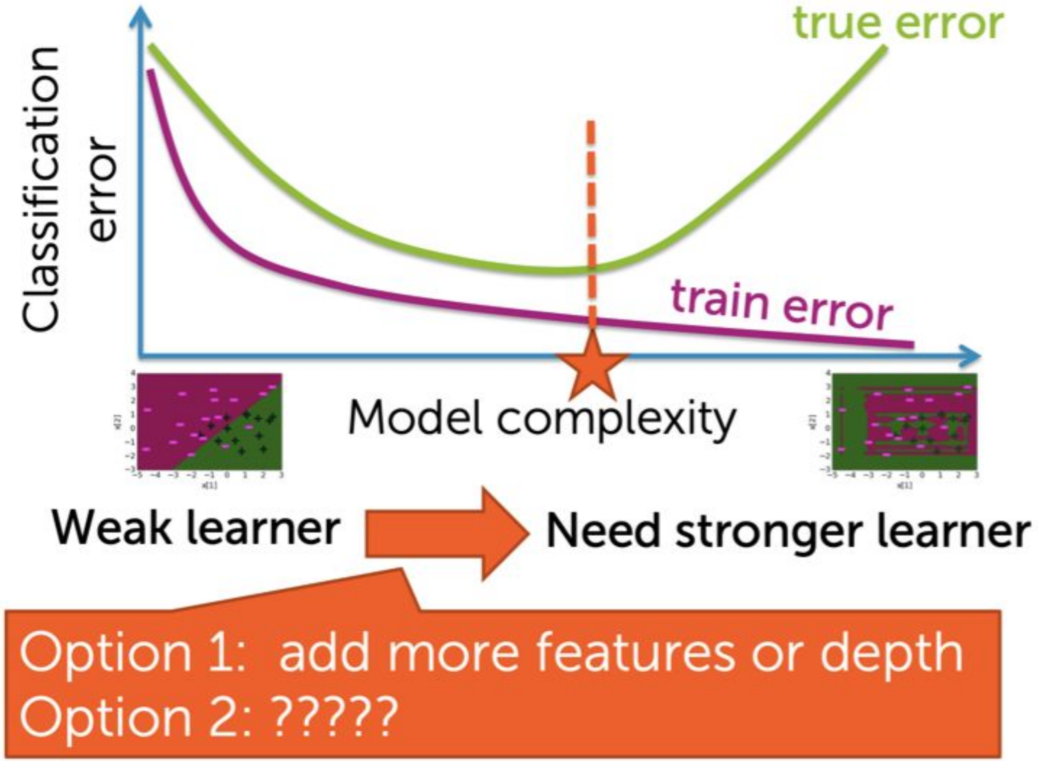
Shallow decision trees

Decision stumps

Low variance. Learning is fast!

But high bias...

FInding A Classifier That's Just Right



Boosting

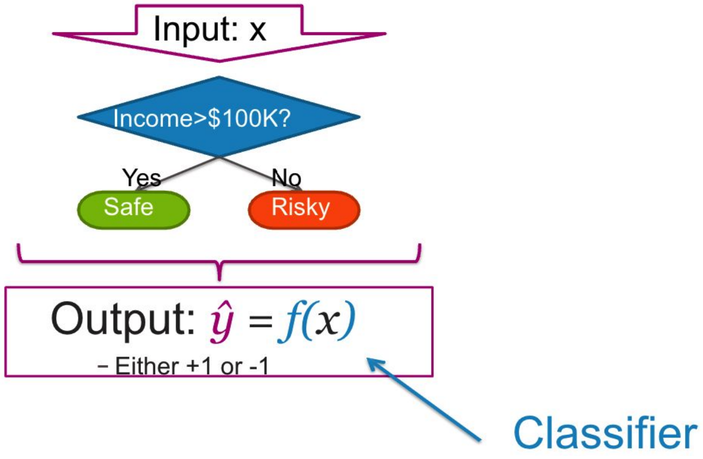
"Can a set of weak learners be combined to create a stronger learner?" *Kearns and Valiant (1988)*

Yes! *Schapire (1990)*

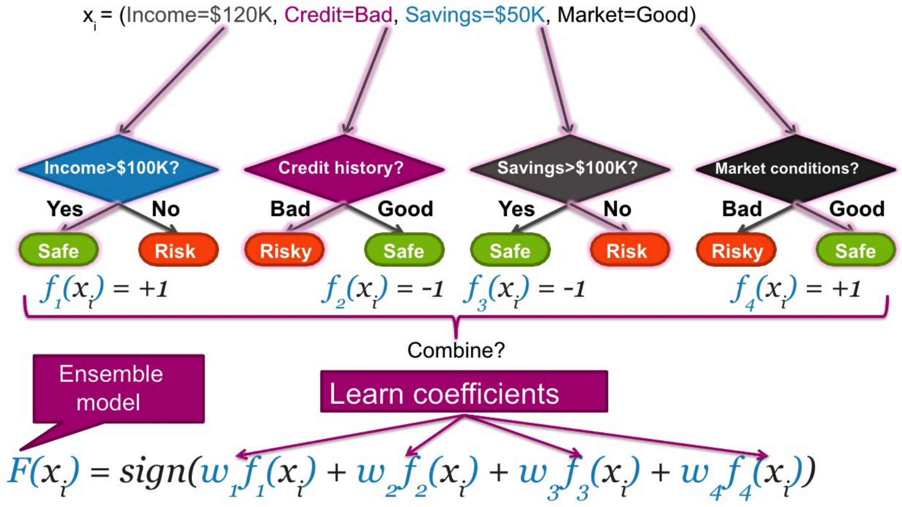
Boosting

Amazing impact: • simple approach • widely used in industry • wins most Kaggle competitions • great systems (e.g., XGBoost)

Ensemble Classifier



A single classifier



Ensemble methods: Each classifier "votes" on prediction

Ensemble Classifier in General

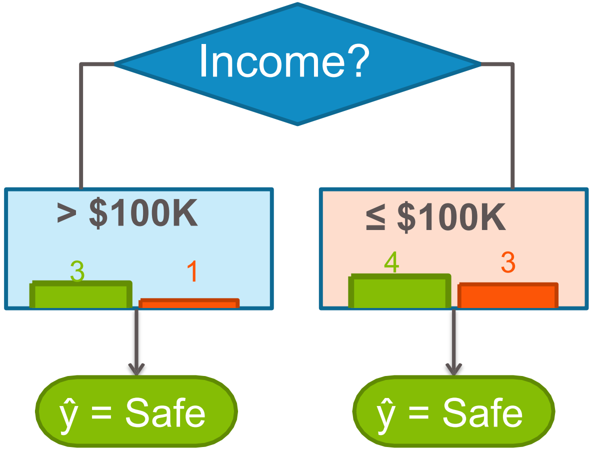
- Goal:
  - Predict output y
    - Either +1 or -1
  - From input x
- Learn ensemble model:
  - Classifiers: $f_1(x), f_2(x), \ldots, f_T(x)$
  - Coefficients: $\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_T$
- Prediction:

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

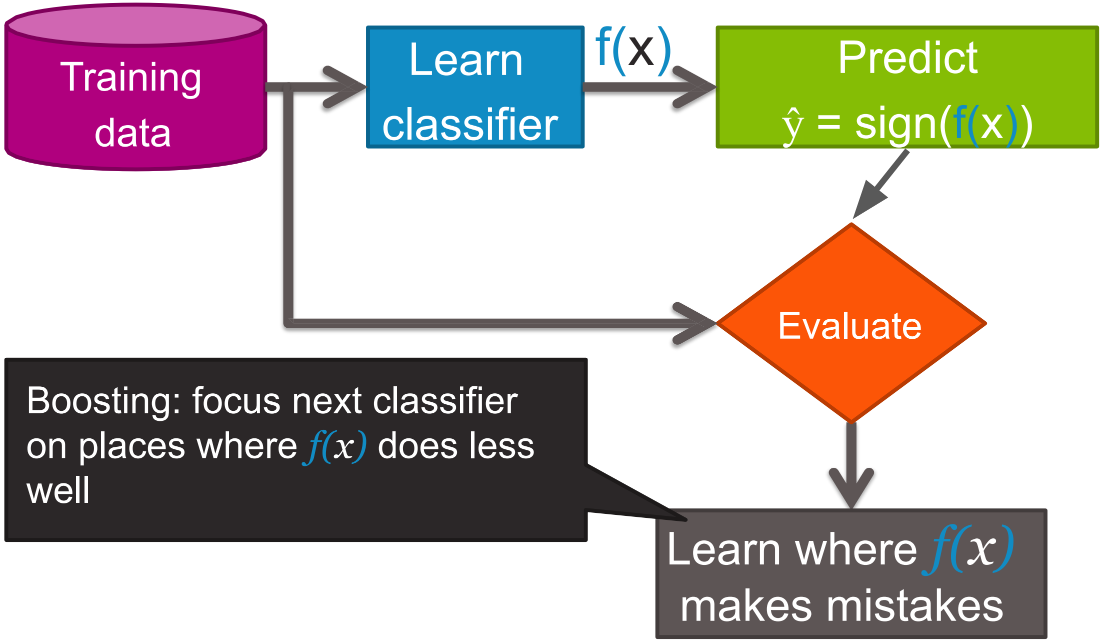
Training a Classifier

Learning Decision Stump

Credit	Income	y
A	$130K	Safe
B	$80K	Risky
C	$110K	Risky
A	$110K	Safe
A	$90K	Safe
B	$120K	Safe
C	$30K	Risky
C	$60K	Risky
B	$95K	Safe
A	$60K	Safe
A	$98K	Safe



Boosting = Focus Learning on “Hard” Points



Learning on Weighted Data:

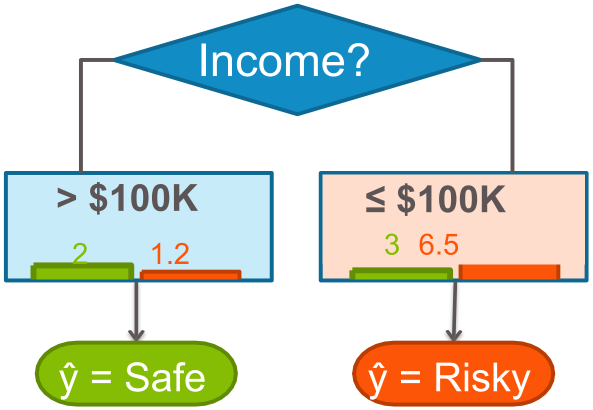
More weight on “hard” or more important points

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

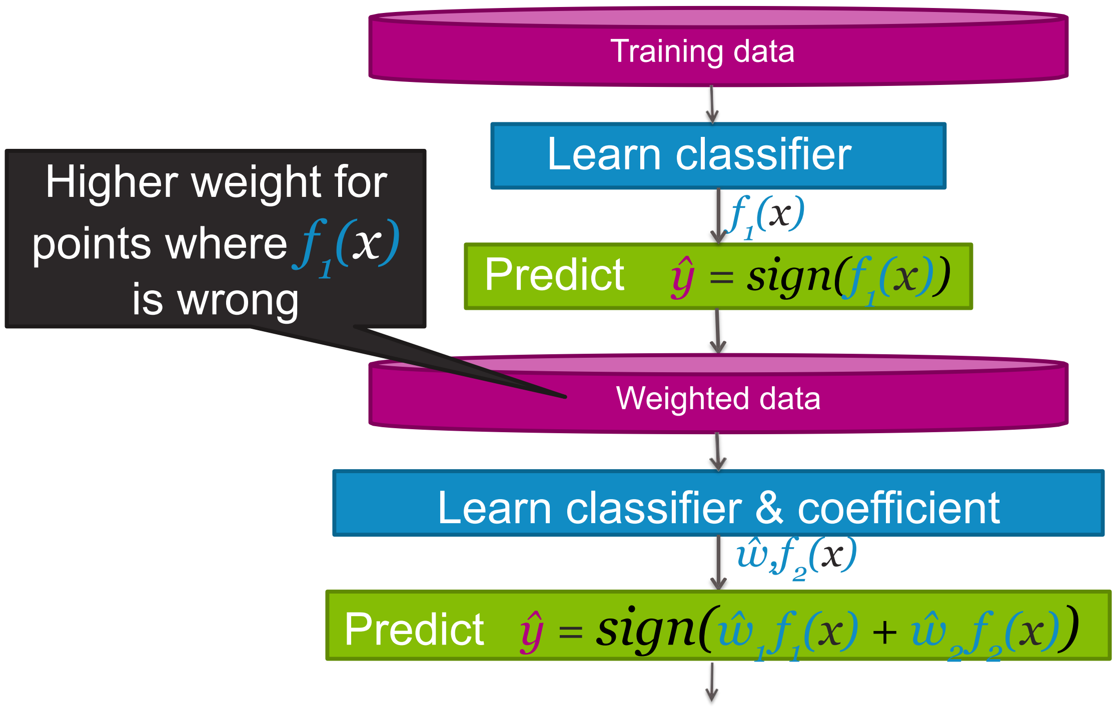
Learning a Decision Stump on Weighted Data

Increase weight **α** of harder/misclassified points

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



Boosting = Greedy Learning Ensembles from Data



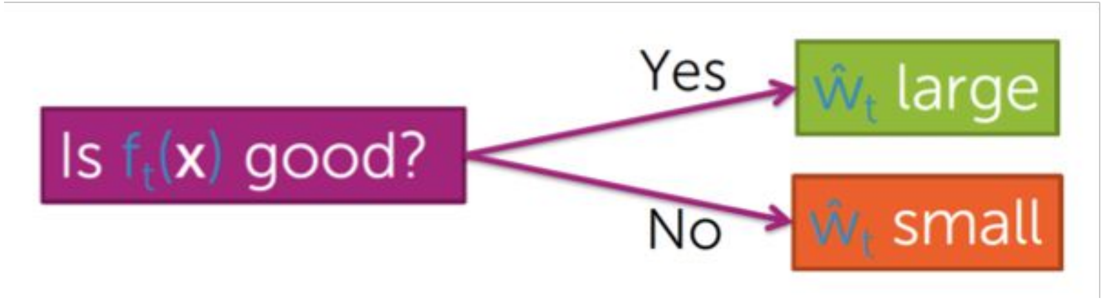
AdaBoost Algorithm

AdaBoost: learning ensemble [Freund & Schapire 1999]

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

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

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

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

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

Is $f_t(x)$ good?

Yes

No

weighted_error(f_t) on training data	$\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)}$	$\hat{w}_t$

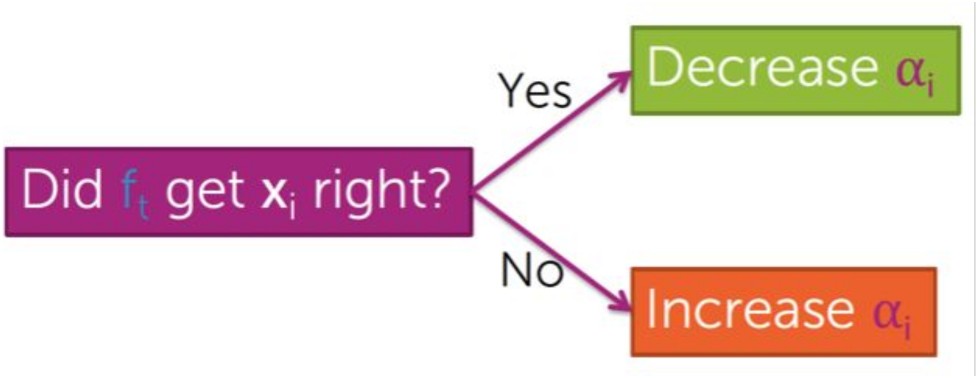
AdaBoost: Learning Ensemble

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

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

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

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



AdaBoost: Formula for Updating Weights α_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 α_i by	Implication

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 αi
  - Compute coefficient $\hat{w}_t$
  - Recompute weights α_i
- Final model predicts by:

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

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

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

AdaBoost: Normalizing Weights α_i

If $\mathbf{x}_i$ often mistake, weight α_i gets very **large**

If $\mathbf{x}_i$ often correct, weight α_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: $\alpha_i = 1/N$
- For $t = 1,\ldots,T$
    - Learn $f_t(x)$ with data weights αi
    - Compute coefficient $\hat{w}_t$
    - Recompute weights α_i
    - Normalize weights α_i
- Final model predicts by:

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

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

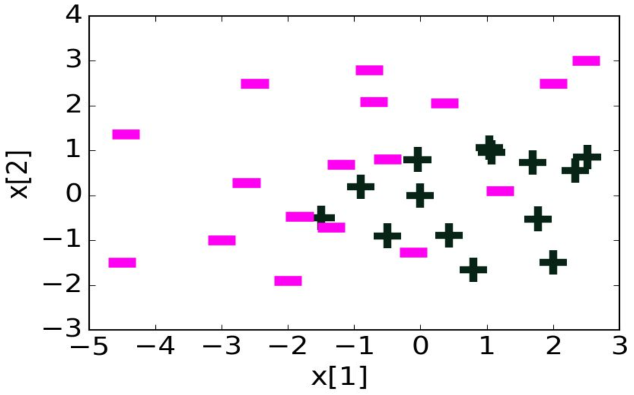
$$\alpha_i \leftarrow \begin{cases} \alpha_i\, e^{-\hat{w}_t}, & \text{if } f_t(x_i) = y_i \\ \alpha_i\, e^{\hat{w}_t}, & \text{if } f_t(x_i) \neq y_i \end{cases}$$

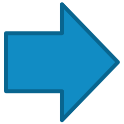
$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j}$$

AdaBoost Example: A Visualization

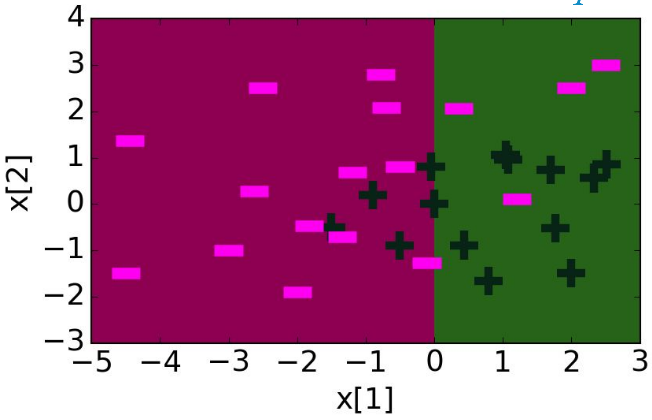
t=1: Just Learn a Classifier on Original Data

Original data

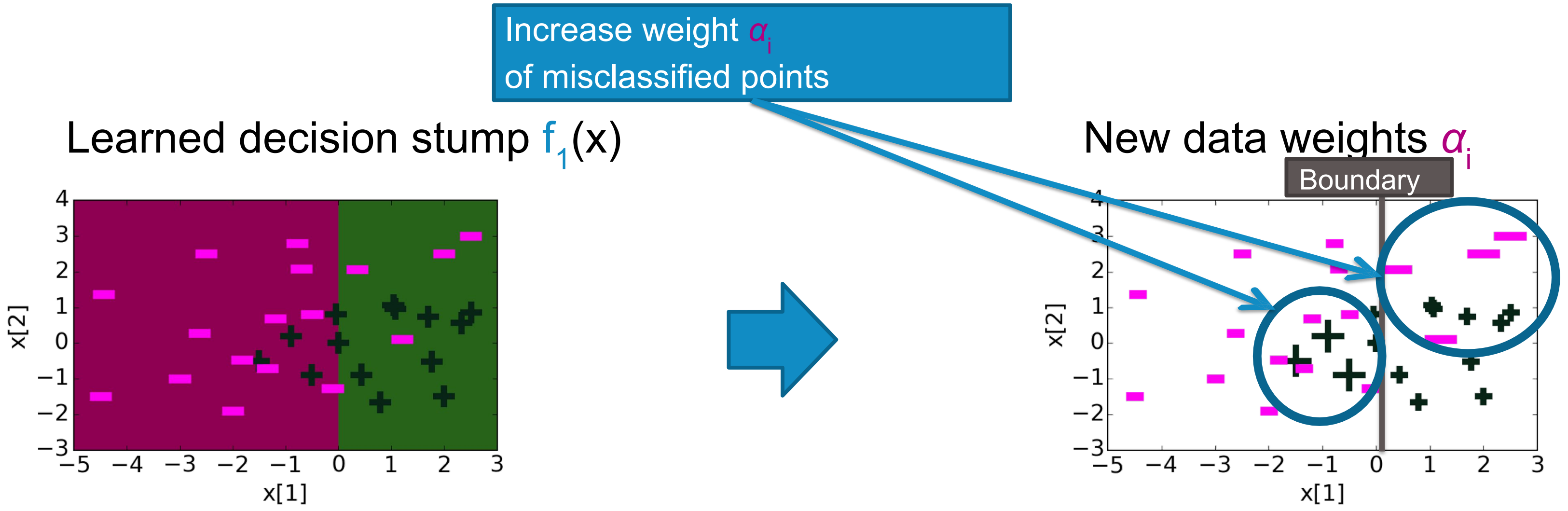




Learned Decision Stump $f_1(x)$

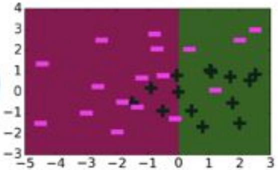


Updating Weights α_i



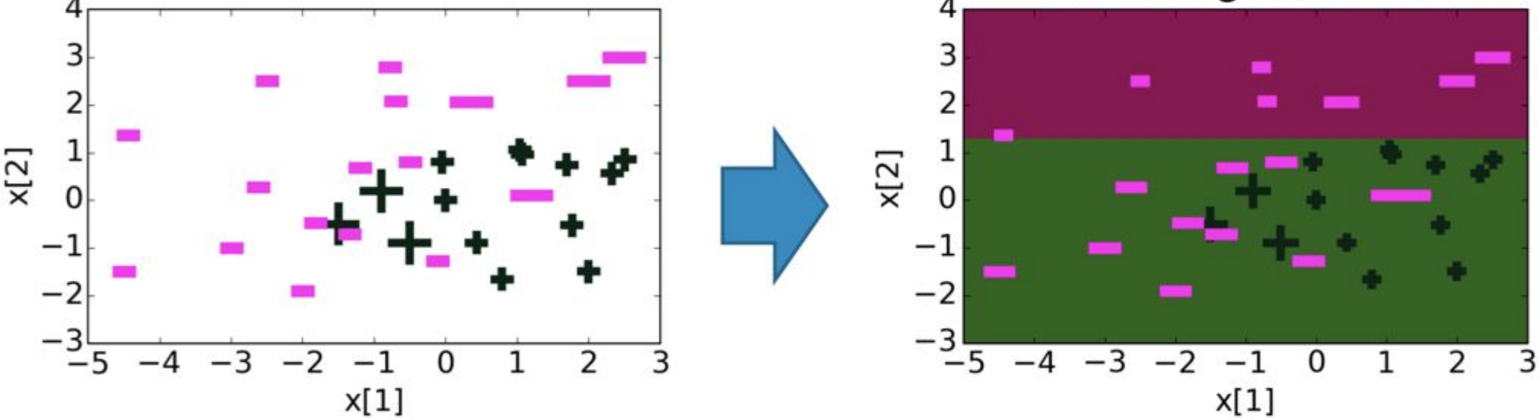
$t=2$: Learn Classifier on Weighted Data

$f_1(\mathbf{x})$

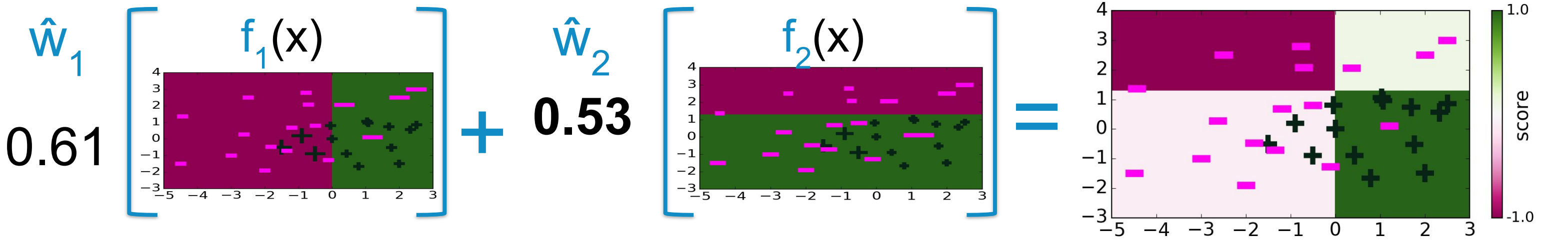


Weighted data: using α_i chosen in previous iteration

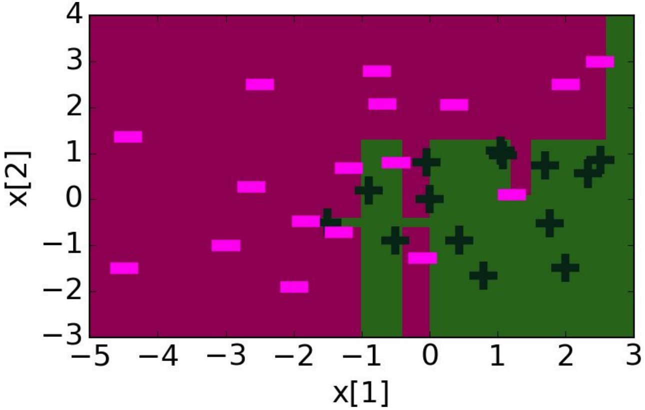
Learned decision stump $f_2(\mathbf{x})$ on weighted data



Ensemble Becomes Weighted Sum of Learned Classifiers



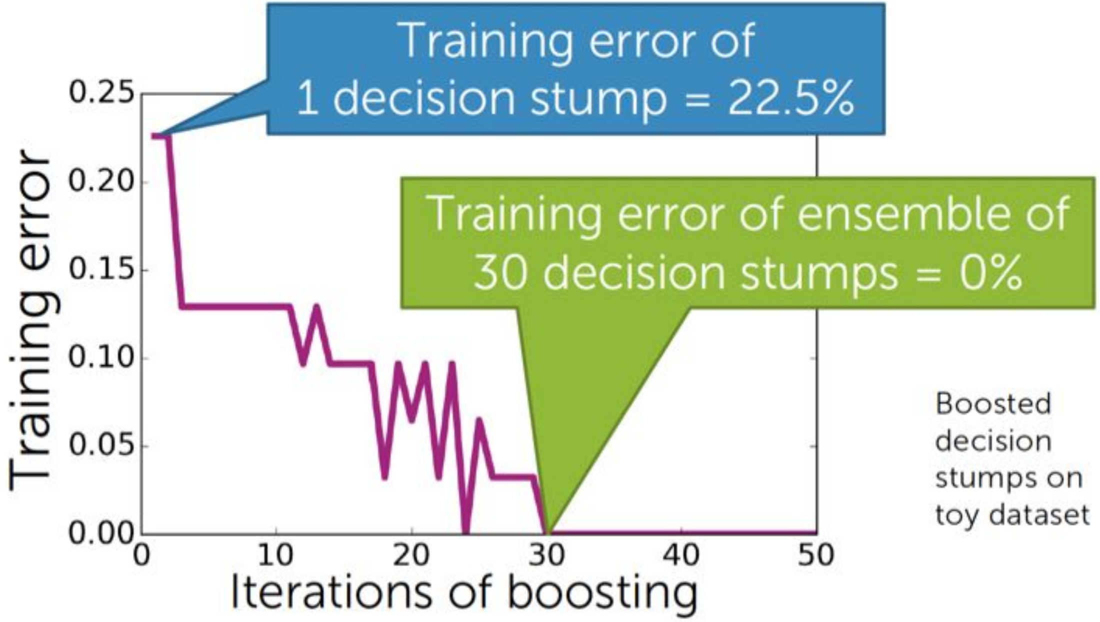
Decision Boundary of Ensemble Classifier after 30 Iterations



training_error = 0

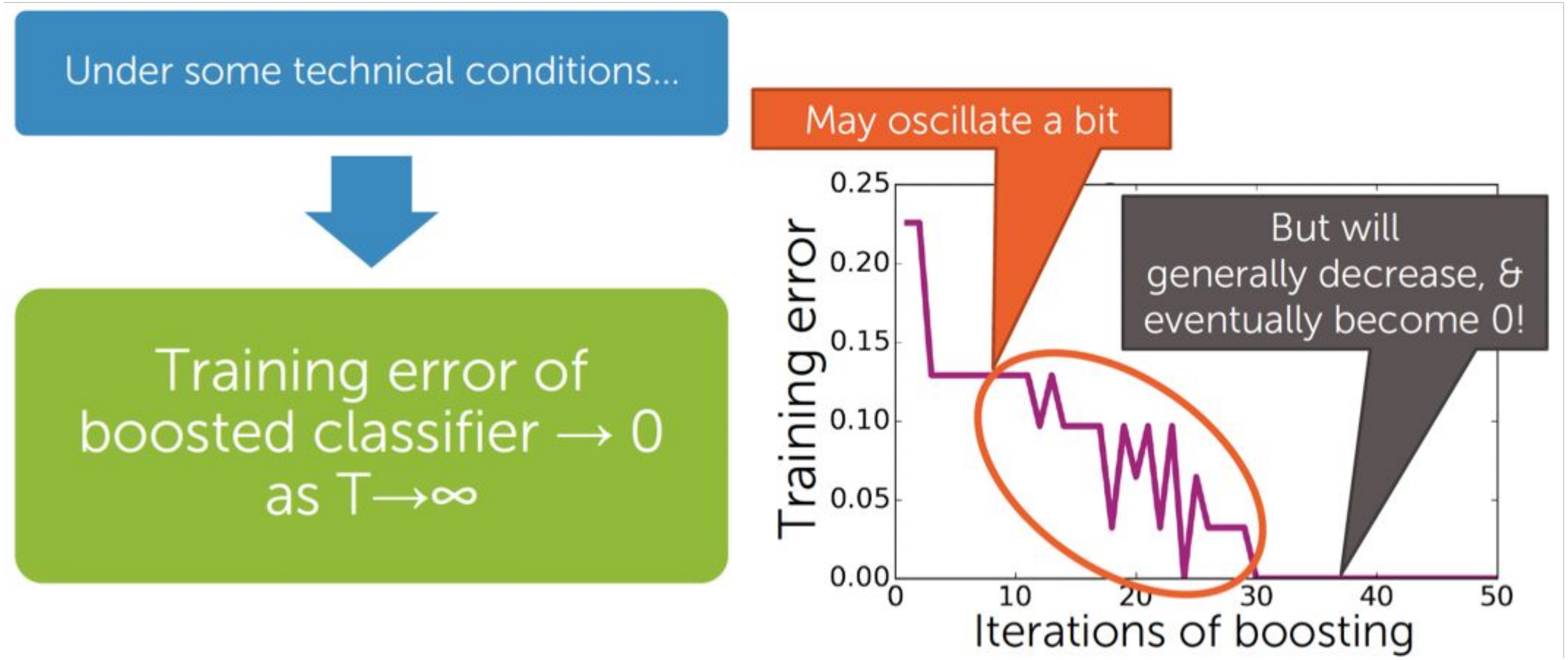
Boosting Convergence & Overfitting

After Some Iterations, Training Error of Boosting Goes to Zero!!!



Boosted decision stumps on toy dataset

AdaBoost Theorem



Condition of AdaBoost Theorem

Under some technical conditions...

Training error of boosted classifier → 0 as T→∞

Condition = At every t, can find a weak learner with 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

AdaBoost Theorem More Formally

Training error of final classifier is bounded by:

$$\frac{1}{N}\sum_{i=1}^{N}\mathbb{I}[F(x_i) \neq y_i] \leq \frac{1}{N}\sum_{i=1}^{N}\exp(-y_i \mathrm{score}(x_i))$$

Where $\mathrm{score}(x) = \sum_t \hat{w}_t f_t(x)$; $F(x) = sign(\mathrm{score}(x))$

AdaBoost Theorem More Formally

Training error of final classifier is bounded by:

$$Z_t = \sum_{i=1}^{N} \alpha_{i,t} \exp(-\hat{w}_t y_i f_t(x_i))$$

$$\frac{1}{N}\sum_{i=1}^{N} \mathbb{I}[F(x_i) \neq y_i] \leq \frac{1}{N}\sum_{i=1}^{N} \exp(-y_i \text{score}(x_i)) = \prod_{t=1}^{T} Z_t$$

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

Where $\text{score}(x) = \sum_t \hat{w}_t f_t(x)$; $F(x) = sign(\text{score}(x))$

AdaBoost Theorem More Formally

If we minimize $\prod_{t=1}^{T} Z_t$, we minimize our training error

We can tighten this bound greedily by choosing $\hat{w}_t$, f_t on each iteration to minimize:

$$Z_t = \sum_{i=1}^{N} \alpha_{i,t} \exp(-\hat{w}_t y_i f_t(x_i))$$

For boolean target function, this is accomplished by [Freund & Schapire '97]:

$$\hat{w}_t = \frac{1}{2} \ln \left(\frac{1 - \epsilon_t}{\epsilon_t} \right)$$

AdaBoost Theorem More Formally

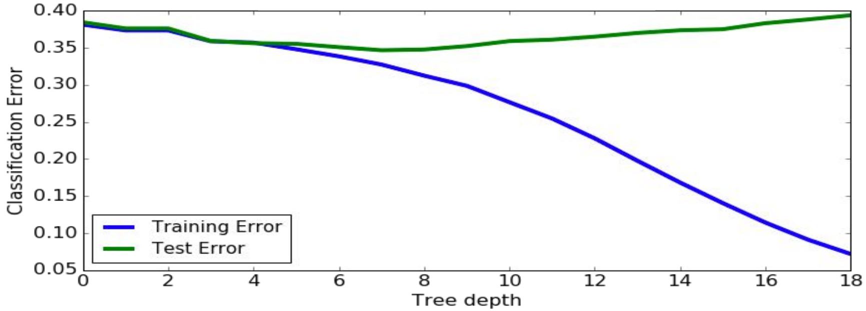
If each classifier is (at least slightly) better than random

$$weighted\_error(f_t) = \epsilon_t < 0.5$$

AdaBoost will achieve zero training error (exponentially fast):

$$\frac{1}{N}\sum_{i=1}^{N}\mathbb{I}[F(x_i) \neq y_i] \leq \prod_{t=1}^{T} Z_t \leq \exp\left(-2\sum_{t=1}^{T}(1/2 - \epsilon_t)^2\right)$$

Decision trees on loan data

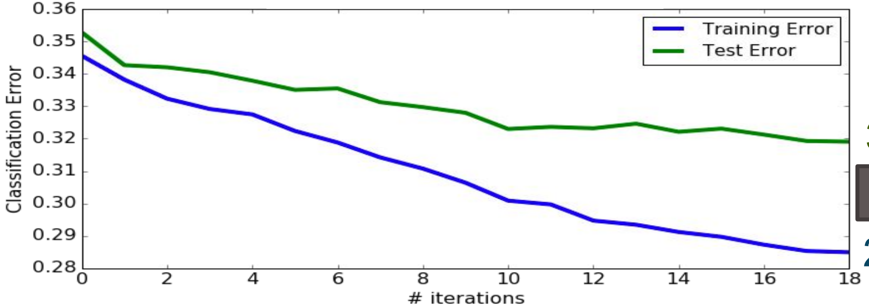


39% test error

Overfitting

8% training error

Boosted decision stumps on loan data

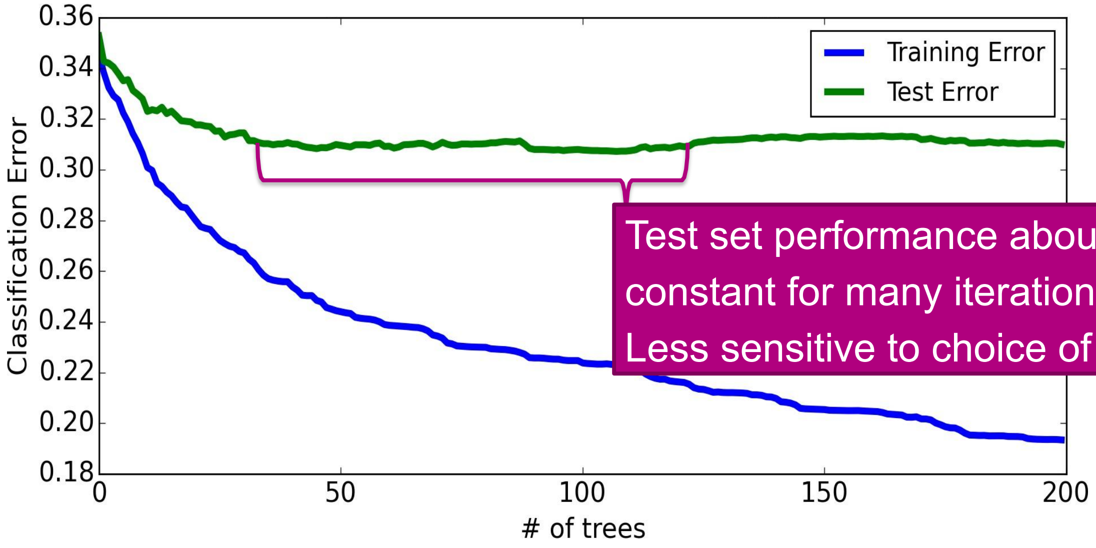


32% test error

Better fit & lower test error

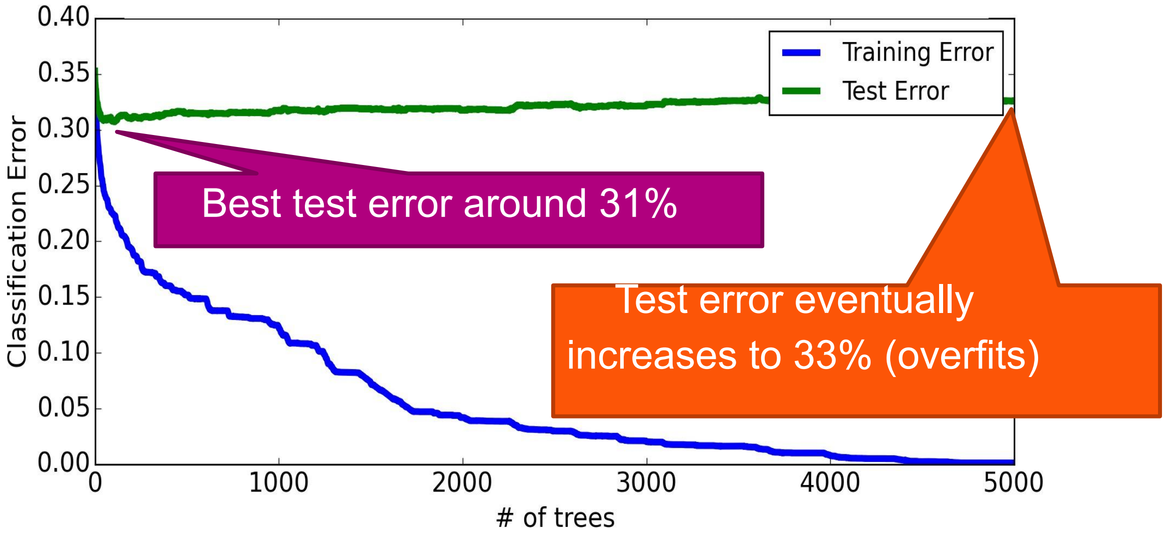
28.5% training error

Boosting Tends to Be Robust to Overfitting



Test set performance about constant for many iterations → Less sensitive to choice of T

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



Summary of Boosting

There are hundreds of variants of boosting, most important:

Gradient boosting

- Like AdaBoost, but useful beyond basic classification
- Great implementations available (e.g., XGBoost)

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

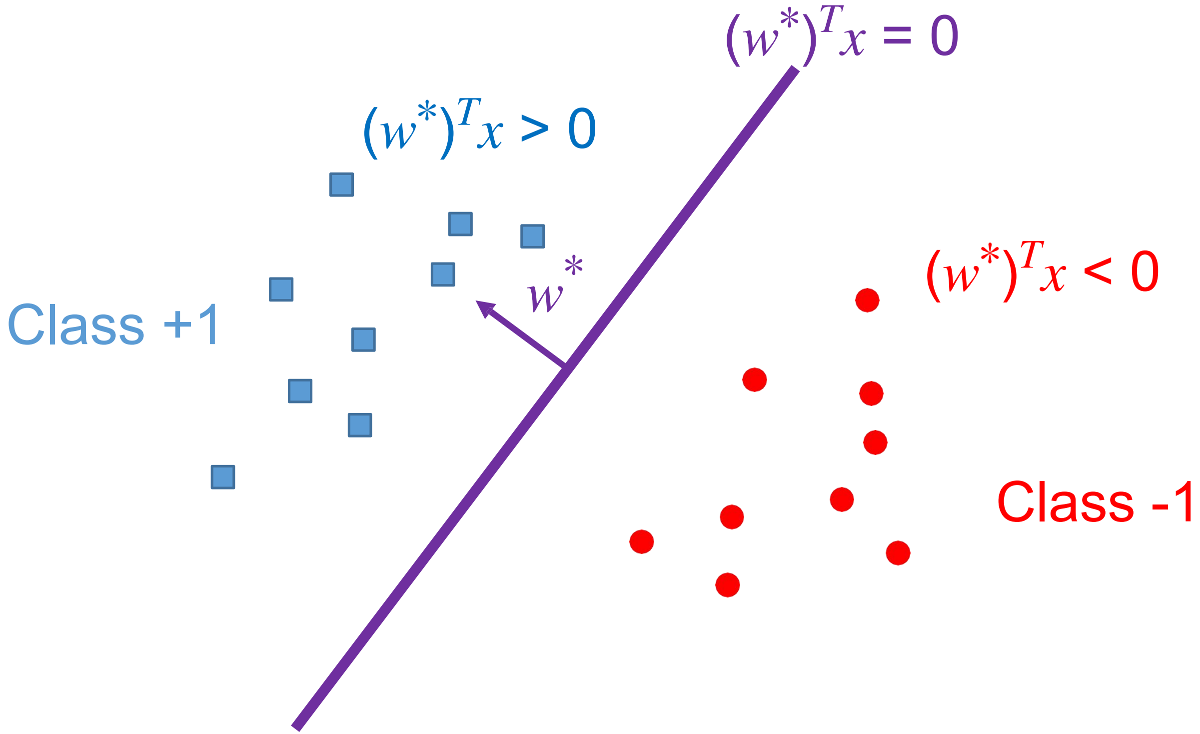
- Coefficients chosen manually, with boosting, with bagging, or others

What You Can Do Now…

- Identify notion ensemble classifiers
- Formalize ensembles as weighted combination of simpler classifiers
- Outline the boosting framework – sequentially learn classifiers on weighted data
- Describe the AdaBoost algorithm
  - Learn each classifier on weighted data
  - Compute coefficient of classifier
  - Recompute data weights
  - Normalize weights
- Implement AdaBoost to create an ensemble of decision stumps

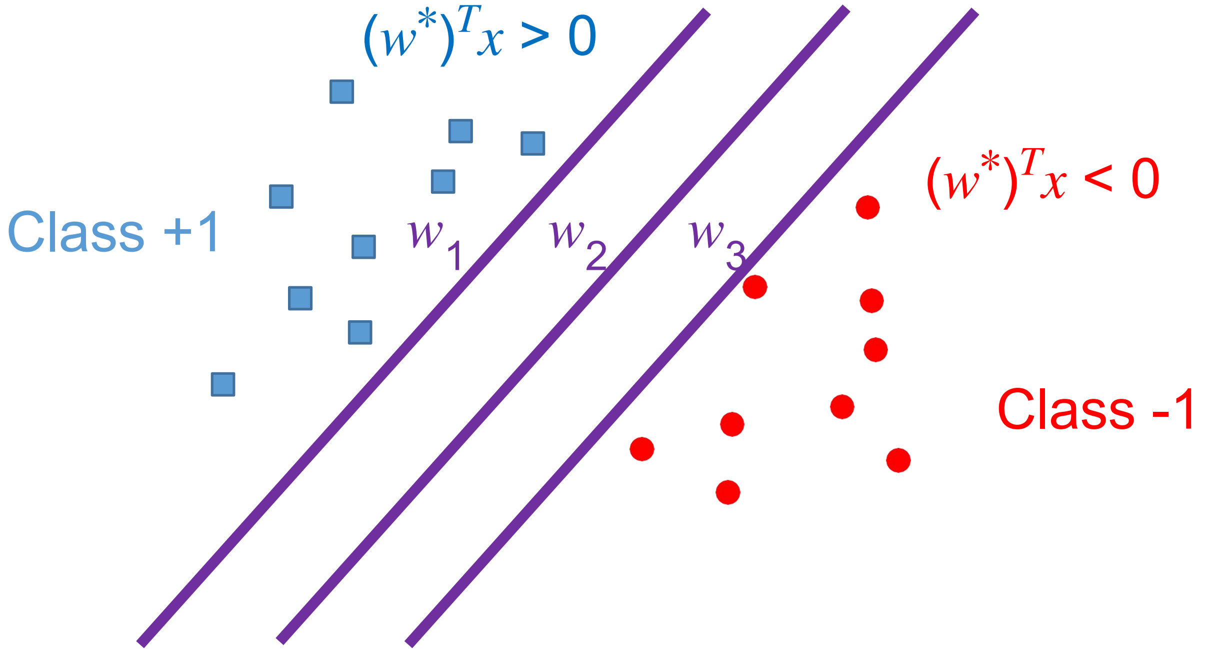
Support Vector Machine

Motive: Linear Classification



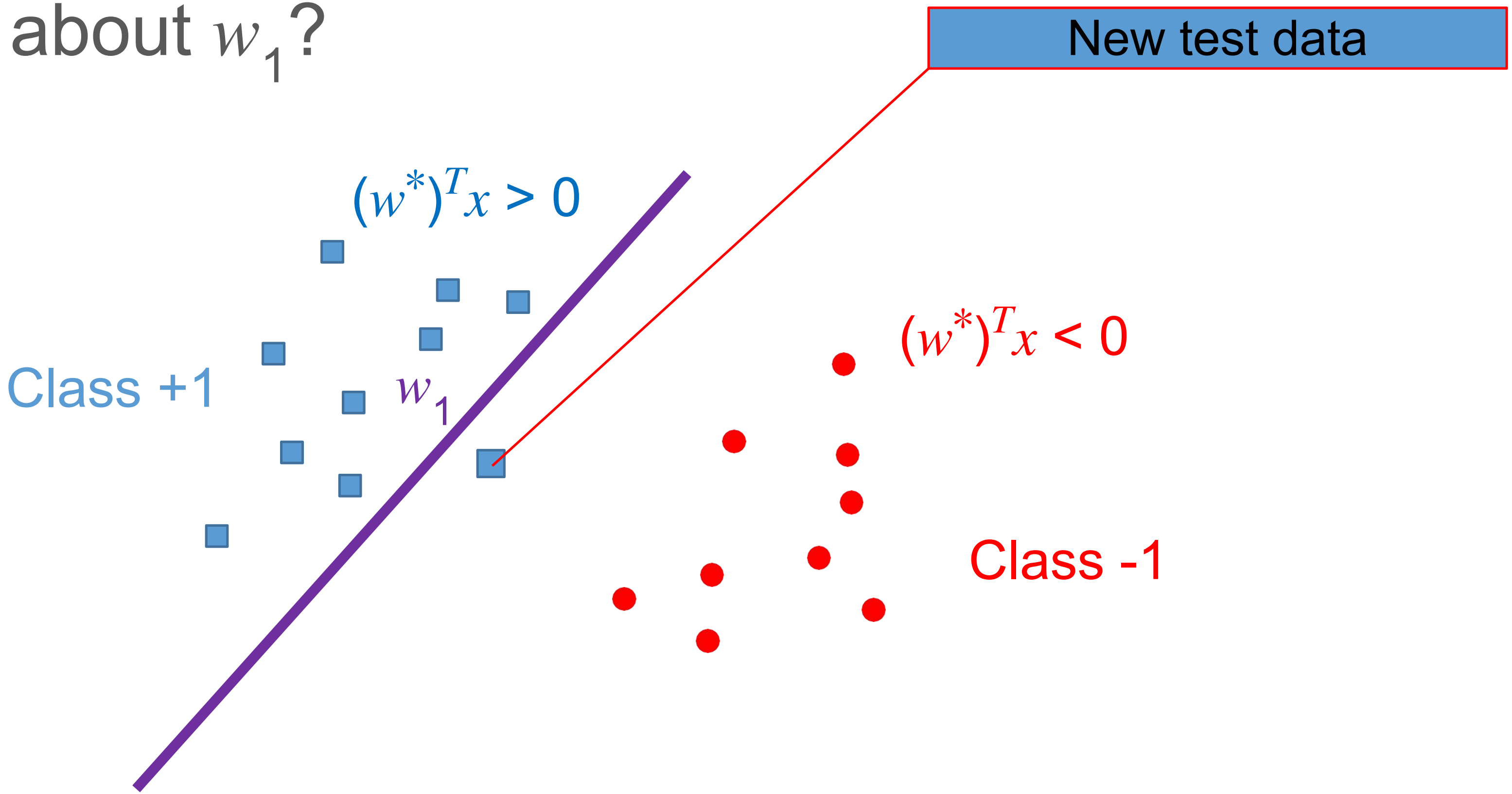
Assume perfect separation between the two classes

Motive: Multiple Optimal Solutions?



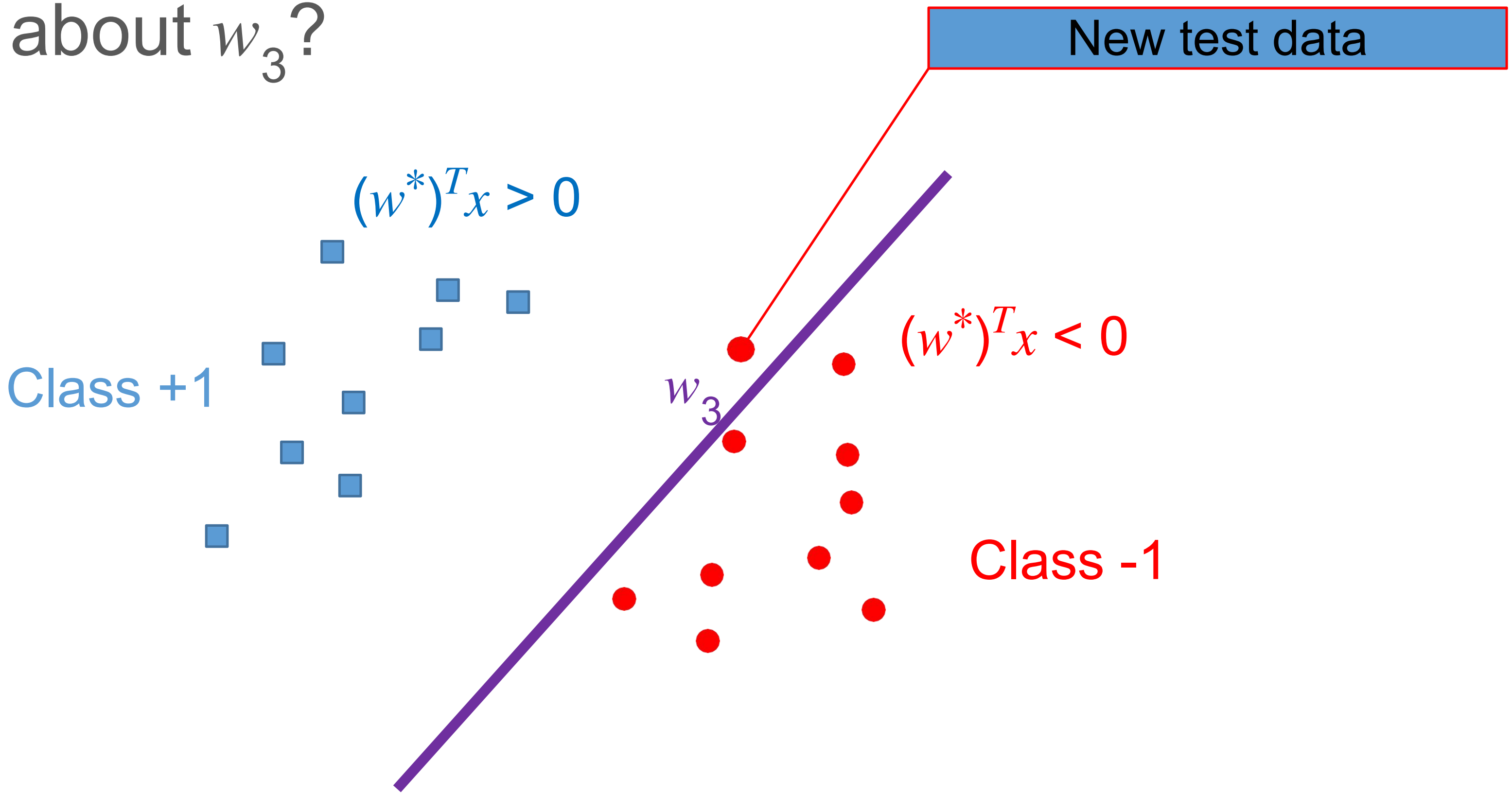
Same on empirical loss; Different on test/expected loss

What about w_1?



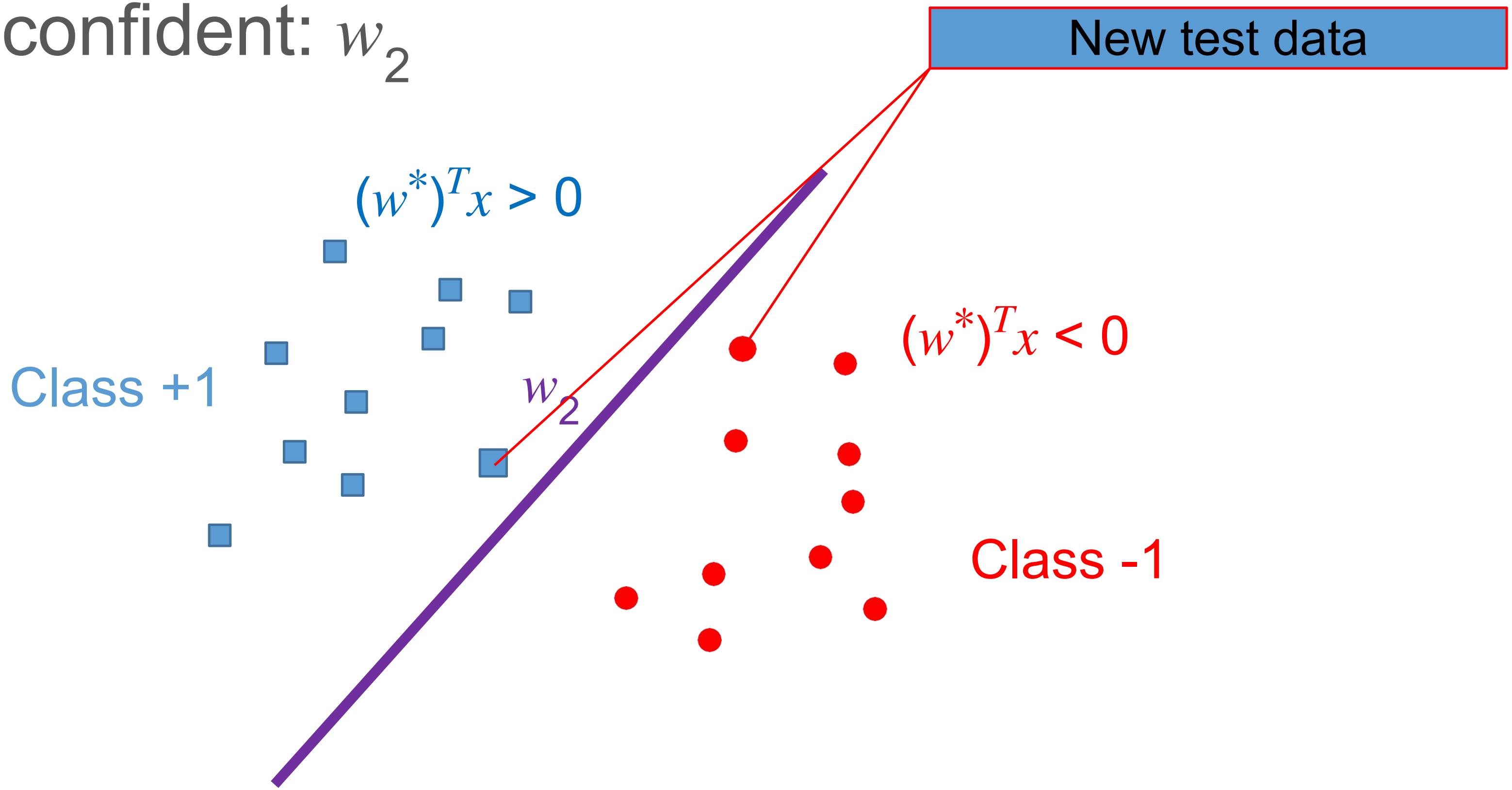
Same on empirical loss; Different on test/expected loss

What about w_3?



Same on empirical loss; Different on test/expected loss

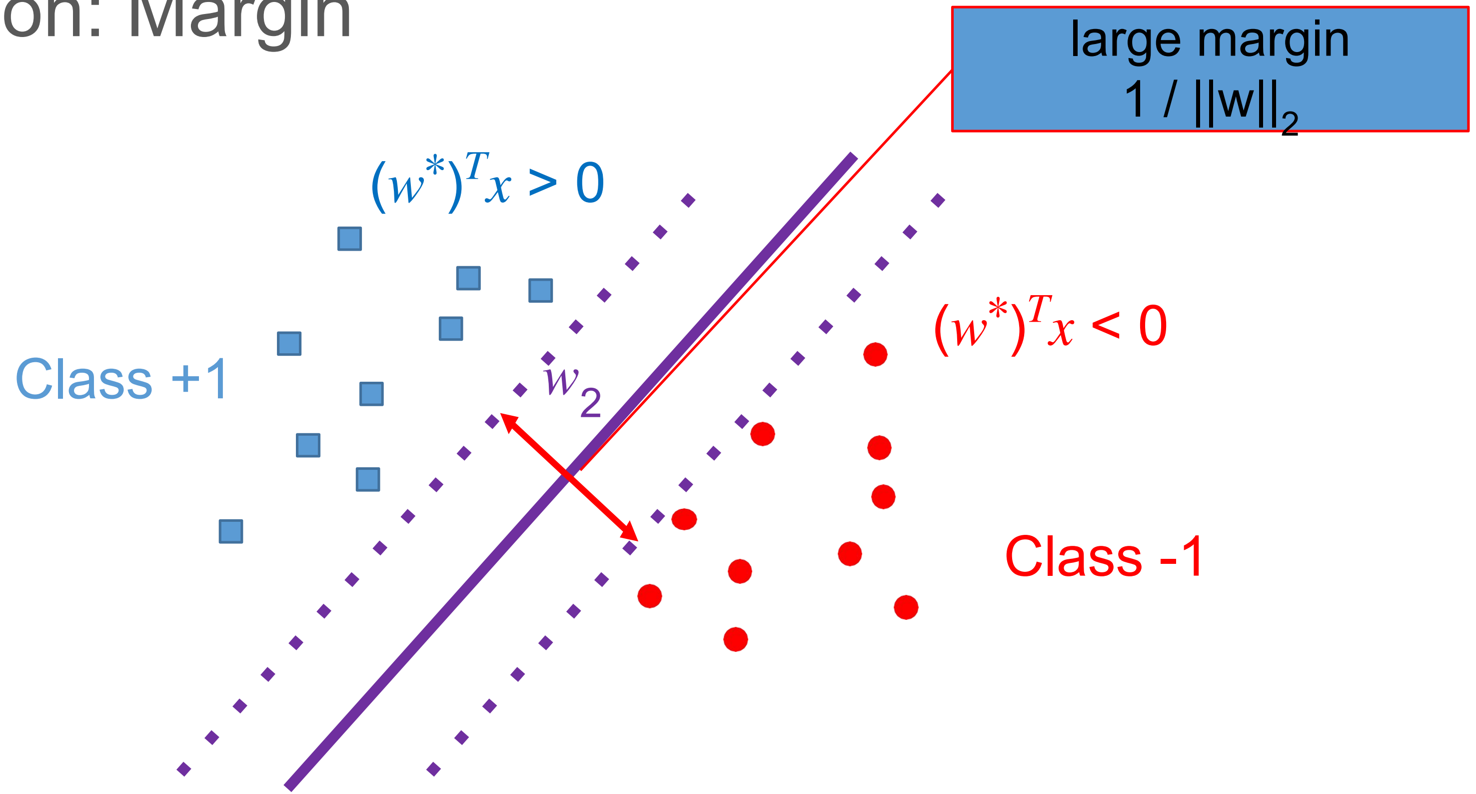
Most confident: w_2



Same on empirical loss; Different on test/expected loss

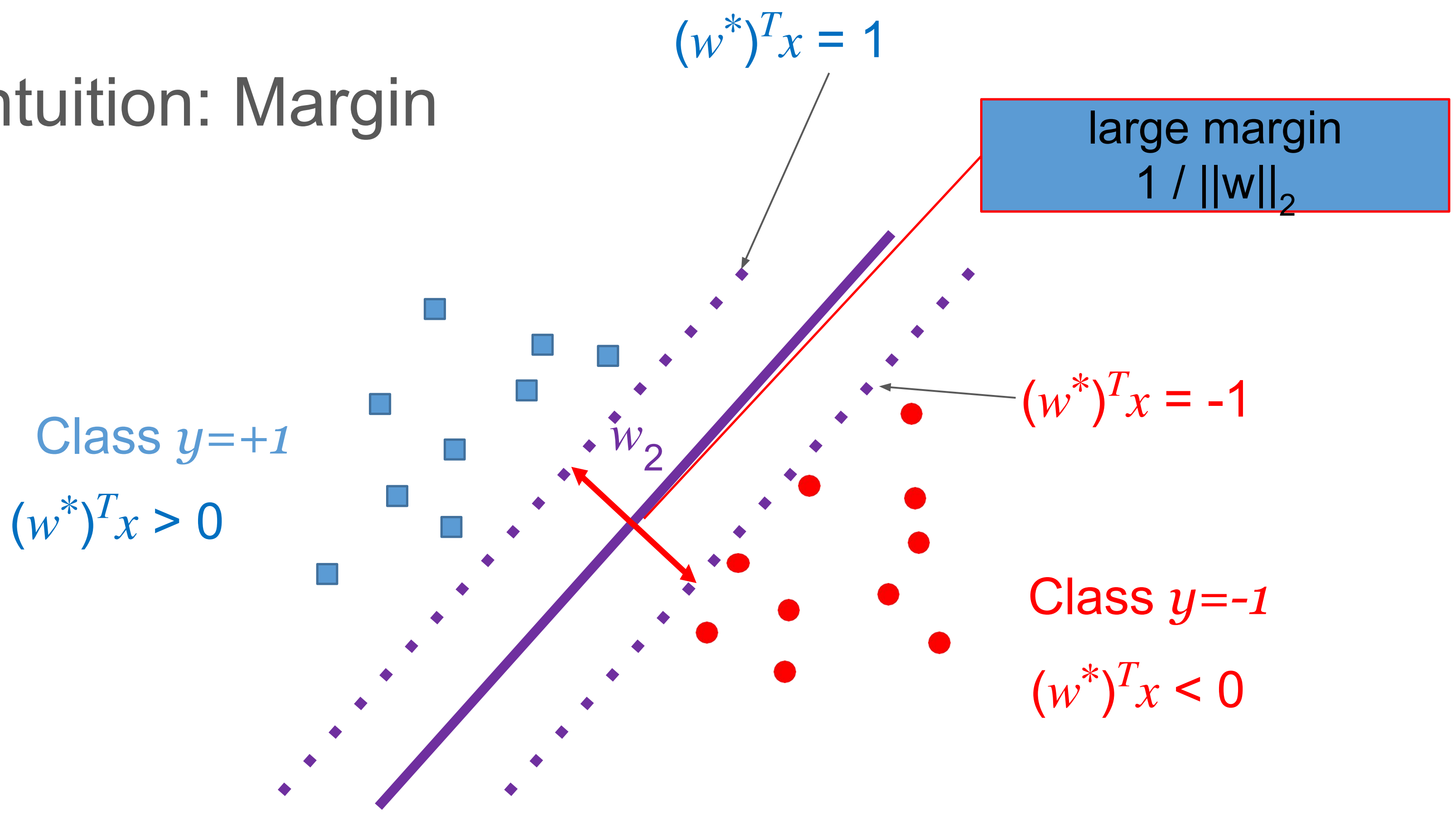
$(w^*)^T x > 0$

Intuition: Margin



Same on empirical loss; Different on test/expected loss

Intuition: Margin

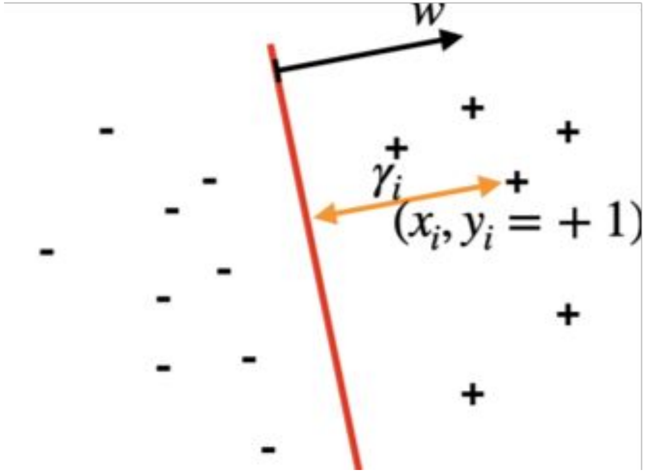


Same on empirical loss; Different on test/expected loss

SVM: Geometric Margin

- Given a set of training examples $\{(x_i, y_i)\}^n_{i=1}$
- A linear classifier will be $(w^*)^T x = 0$
- We define functional margin of w with respect to a training example (x_i, y_i) as the distance from the point (x_i, y_i) to the decision boundary, which is

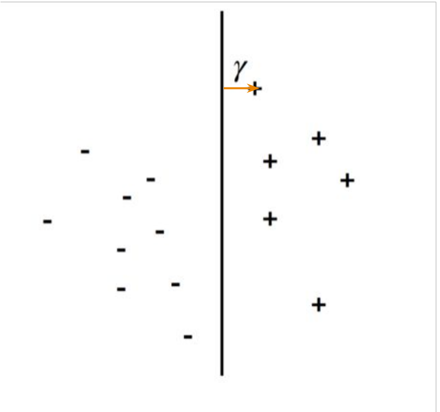
$$\gamma_i = y_i \frac{(w^T x_i + b)}{\|w\|_2}$$



Maximum Margin Classifier

We propose the following optimization problem, maximize the margin.

$$\gamma_i = y_i \frac{(w^T x_i + b)}{\|w\|_2}$$



This is hard to do:

- maximize the numerator
- minimize the denominator

How about we just try to do this if we assume γ is a lower bound on the margin?

$$\frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \quad \text{for all } i \in \{1,\ldots,n\}$$

Maximum Margin Classifier

Still hard to optimize, so we reparameterization if we condition the numerator is >=1:

$$\frac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \quad \text{for all } i \in \{1, \ldots, n\} \longrightarrow \begin{array}{l} \dfrac{y_i(w^T x_i + b)}{\|w\|_2} \geq \gamma \quad \text{for all } i \in \{1, \ldots, n\} \\ \|w\|_2 = \dfrac{1}{\gamma} \end{array}$$

Conditioning on $y_i(w^T x_i + b) \geq 1 \quad \text{for all } i \in \{1, \ldots, n\}$

So we can see we want to maximize $\frac{1}{\|w\|_2}$, which is minimize $\|w\|_2^2$

This is is a quadratic problem with linear constraints, easy to solve

Don't Forget It's Still a Classifier

We want to have a loss function that conditioning on the $\ell(\hat{y}, y)$

So the overall SVM algorithm is try to optimize:

$$\lambda||w||_2^2 + \sum_{i=1}^{n} \ell(\hat{y}, y)$$