# Decision Trees: Overfitting

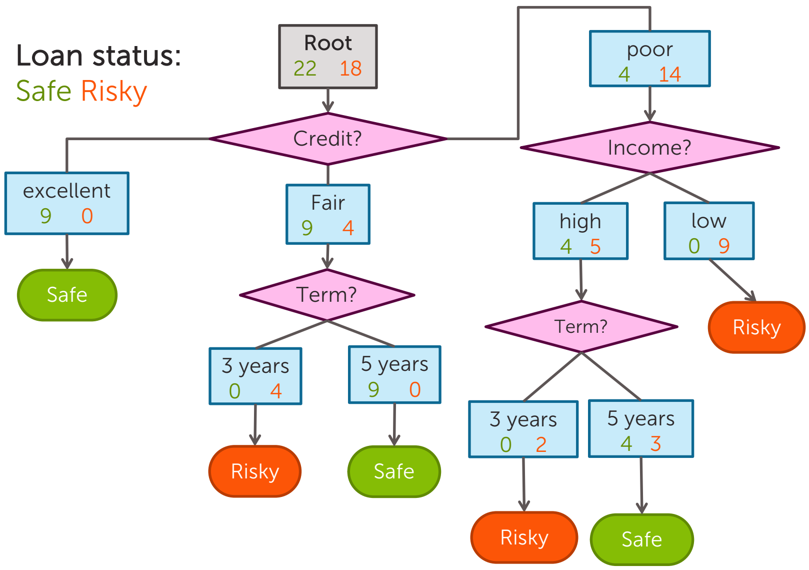
STAT/CSE 416: Machine Learning
Emily Fox
University of Washington
April 26, 2018



## Decision trees recap

# Decision tree recap

Loan status:
Safe Risky




For each leaf node, set
**ŷ = majority value**



# 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, make predictions
  - Step 4: Otherwise, go to Step 2 & continue (recurse) on this split

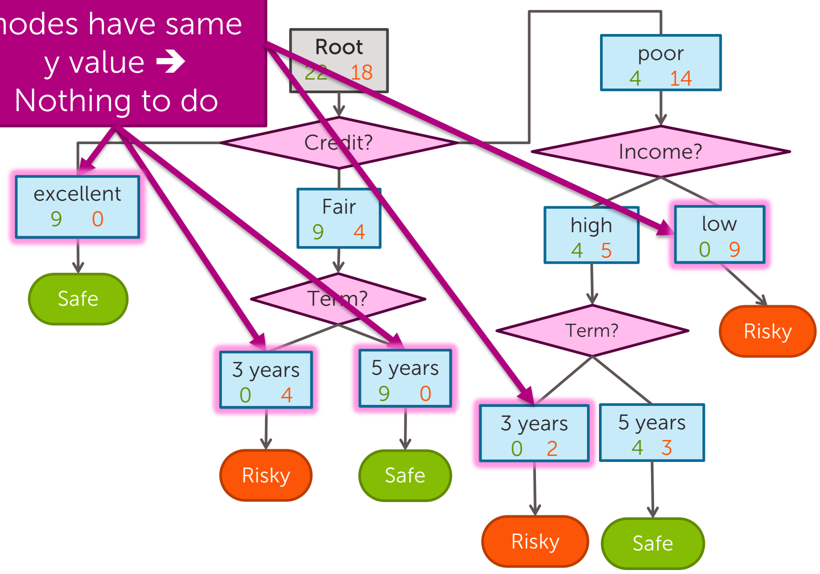
Pick feature split leading to lowest classification error

Stopping conditions 1 & 2

Recursion

## Stopping condition 1: All data agrees on y

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

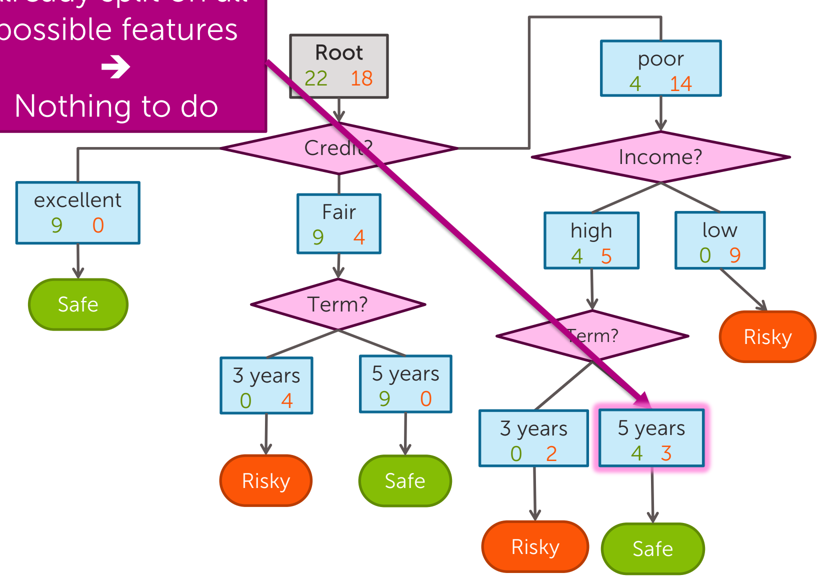






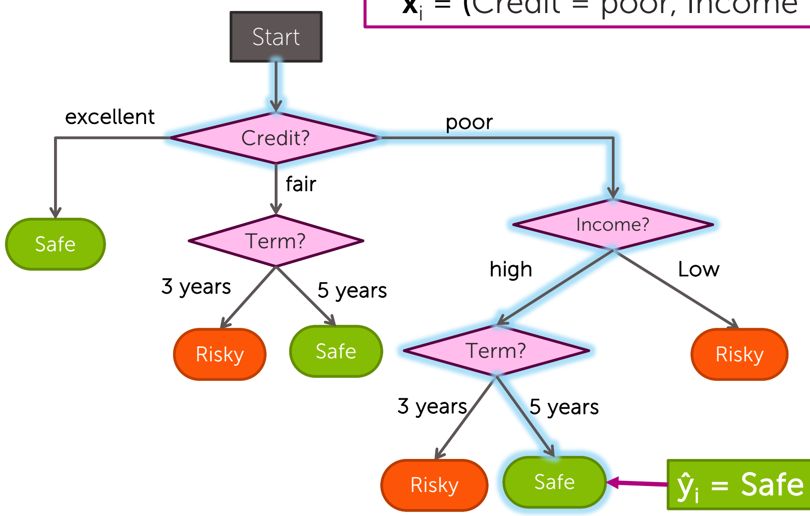

## Stopping condition 2: Already split on all features

Already split on all possible features ➔ Nothing to do



# Scoring a loan application

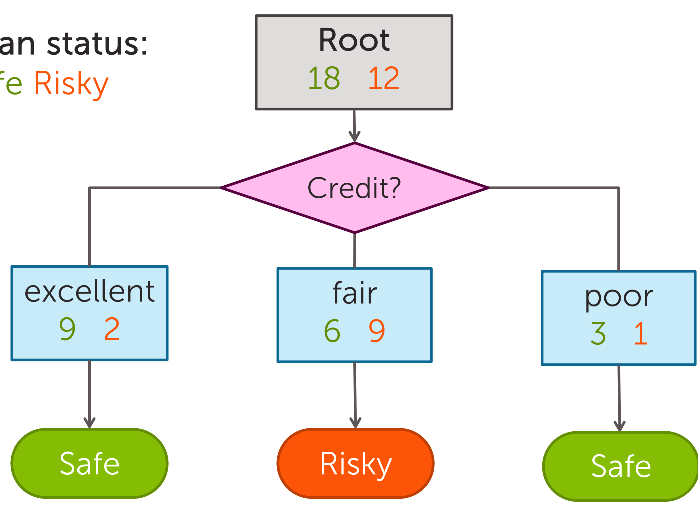
$\mathbf{x}_i$ = (Credit = poor, Income = high, Term = 5 years)





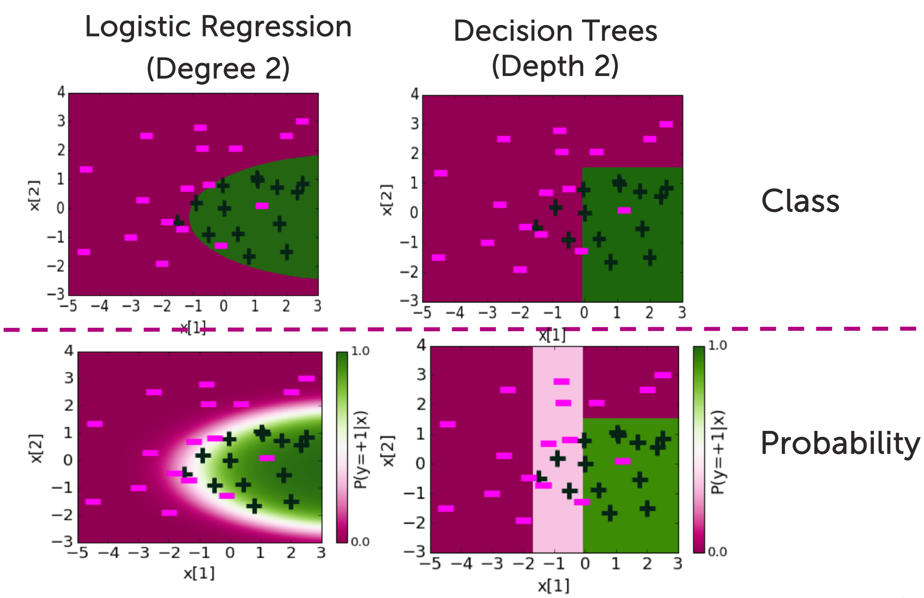

# Predicting probabilities with decision trees

Loan status:
Safe Risky




$$P(y = \text{Safe} \mid \mathbf{x}) = \frac{3}{3+1} = 0.75$$

# Comparison with logistic regression





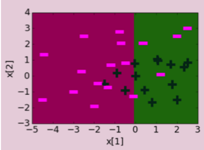
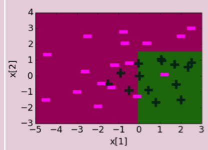
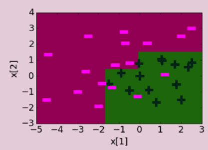
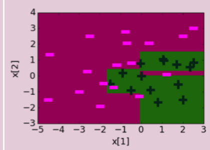
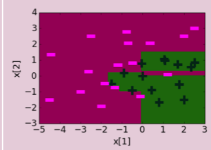



# Overfitting in decision trees

## What happens when we increase depth?

big warning!

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 |  |  |  |  |  |



## 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

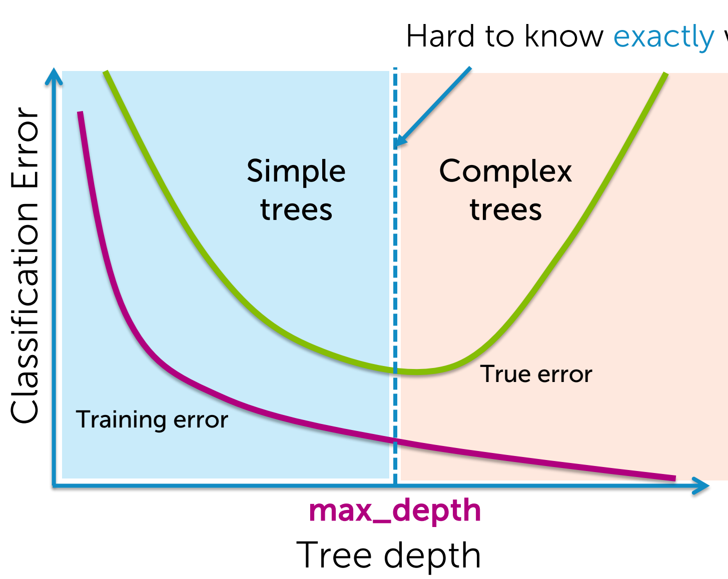
- Stopping conditions (recap):
  1. All examples have the same target value
  2. No more features to split on

- Early stopping conditions:
  1. Limit tree depth (choose *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




Also, might want some branches of tree to go deeper while others remain shallow

# 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 may occur right after "useless" splits
  - Saw this with "xor" example





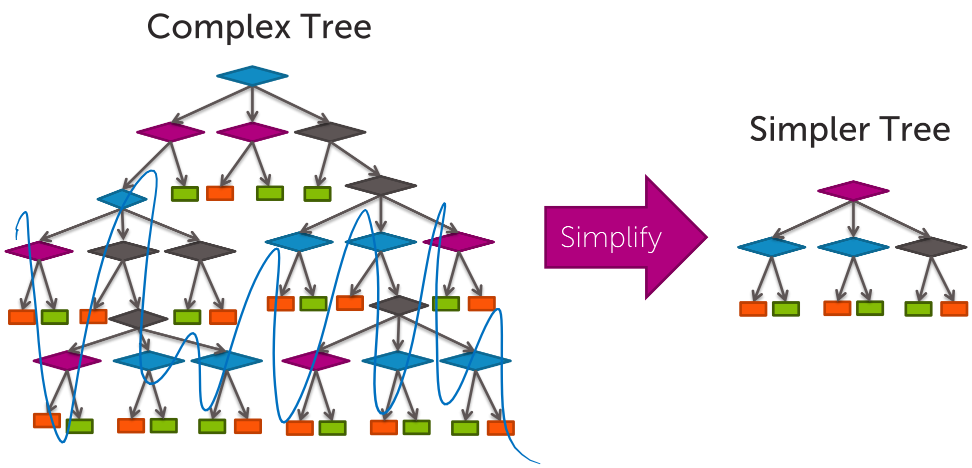
# Two approaches to picking simpler trees

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*

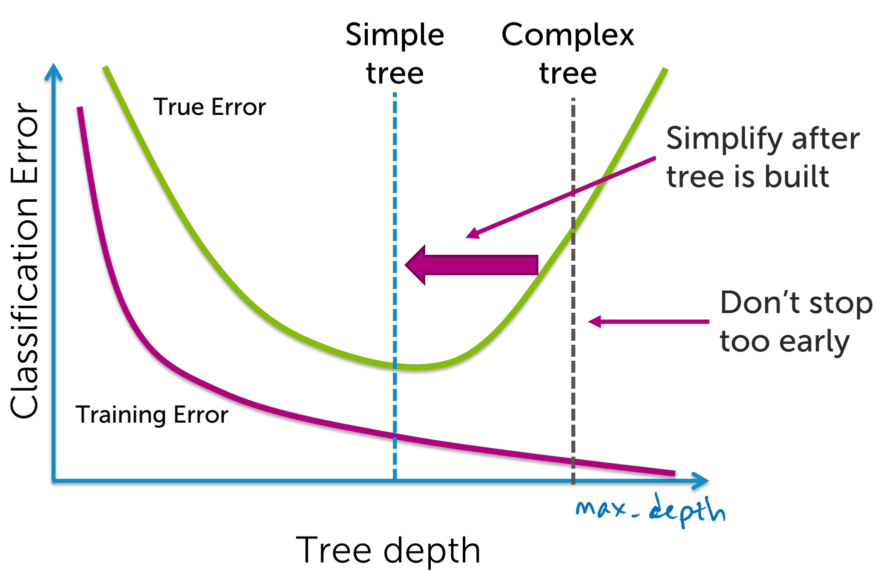
Train a complex tree, simplify later








# Pruning motivation



# Scoring trees: Desired total quality format

Want to balance:

i. How well tree fits data

ii. Complexity of tree

want to balance

Total cost =

measure of fit + measure of complexity

(classification error)
Large # = bad fit to training data

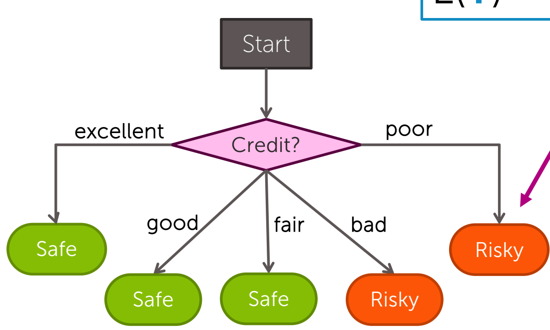
Large # = likely to overfit





# Simple measure of complexity of tree

L(**T**) = # of leaf nodes

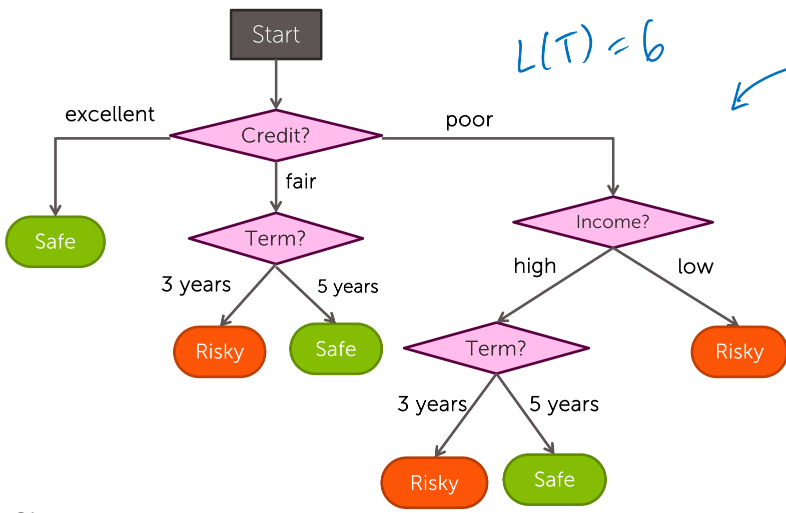



L(T) = 5

## Balance simplicity & predictive power

Too complex, risk of overfitting

$L(T) = 6$




solution in between

Too simple, high classification error


$L(T) = 1$





## Balancing fit and complexity

Total cost C(**T**) = Error(**T**) + λ L(**T**)

tuning parameter

If λ=0: standard decision tree learning

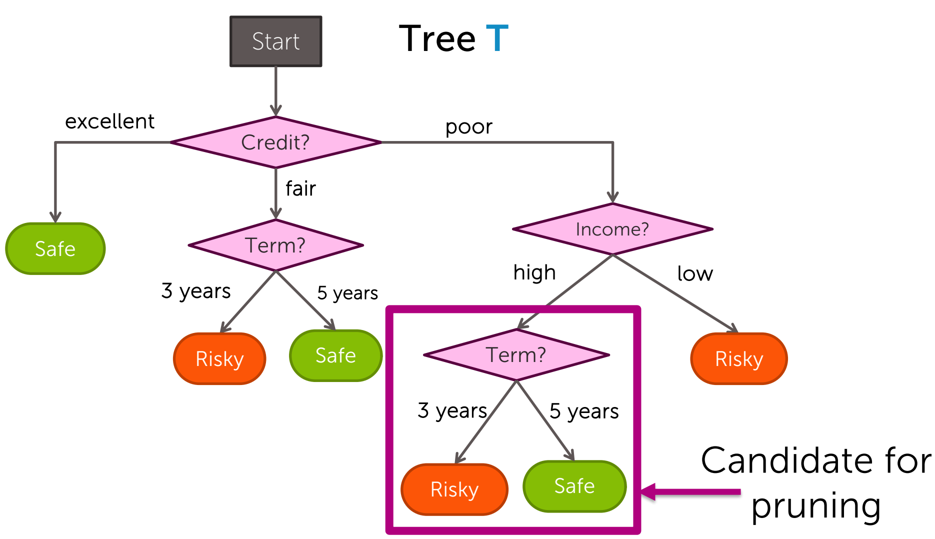
If λ=∞: ∞ penalty → $\boxed{\text{root}} \to 0$ $\hat{y}$ = majority class (of all training data)

If λ in between: balance of fit + complexity

# Tree pruning algorithm

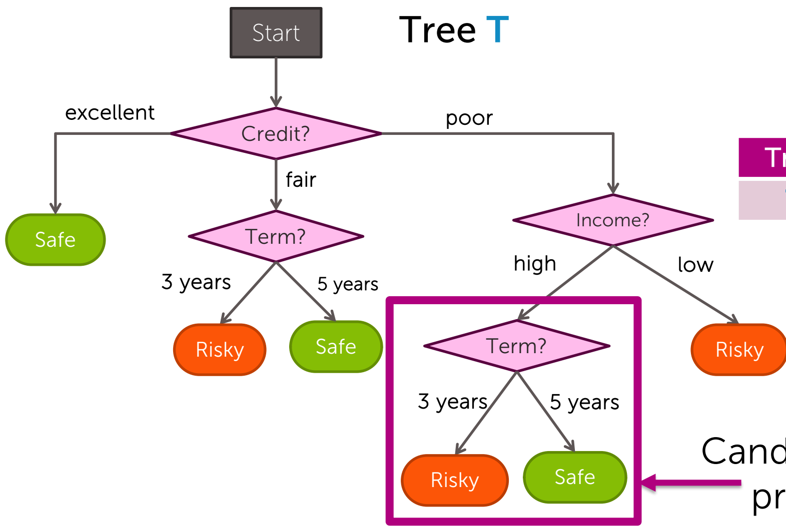


# Step 1: Consider a split



# Step 2: Compute total cost C(T) of split

Tree T




$\lambda = 0.3$ for example

| Tree | Error | #Leaves | Total |
|---|---|---|---|
| T | 0.25 | 6 | 0.43 |

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

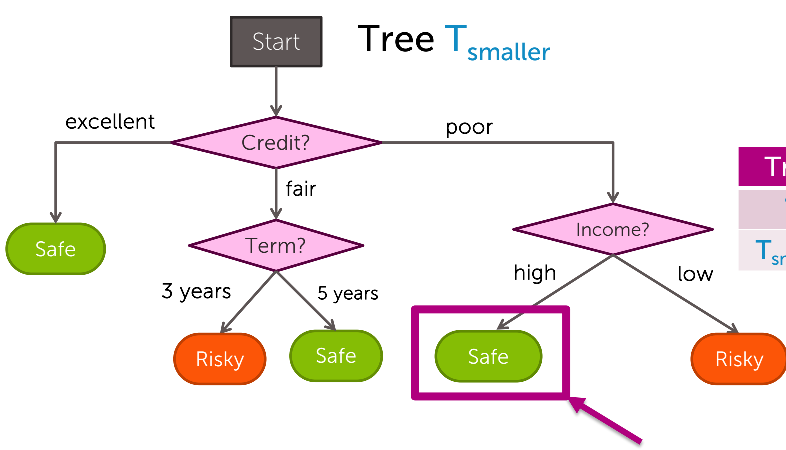
Candidate for pruning





# Step 2: “Undo” the splits on Tsmaller

Tree $T_{smaller}$




$\lambda = 0.3$

| Tree | Error | #Leaves | Total |
|---|---|---|---|
| T | 0.25 | 6 | 0.43 |
| $T_{smaller}$ | 0.26 | 5 | 0.41 |

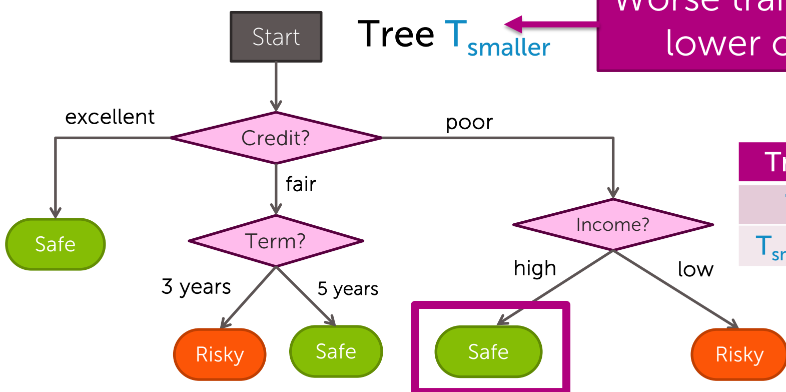
$$C(T) = \text{Error}(T) + \lambda\, L(T)$$

Replace split by leaf node?

# Prune if total cost is lower: C($T_{smaller}$) ≤ C($T$)

Tree $T_{smaller}$

Worse training error but lower overall cost




$\lambda = 0.3$

| Tree | Error | #Leaves | Total |
|---|---|---|---|
| $T$ | 0.25 | 6 | 0.43 |
| $T_{smaller}$ | 0.26 | 5 | 0.41 |

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

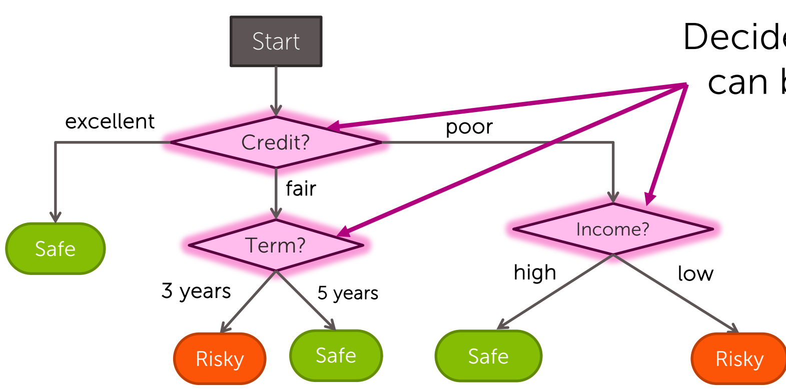
Replace split by leaf node? ← YES!





# Step 5: Repeat Steps 1-4 for every split

Decide if each split can be "pruned"



# Summary of overfitting in decision trees



## What you can do now…

- Identify when overfitting in decision trees
- Prevent overfitting with early stopping
  - Limit tree depth
  - Do not consider splits that do not reduce classification error
  - Do not split intermediate nodes with only few points
- Prevent overfitting by pruning complex trees
  - Use a total cost formula that balances classification error and tree complexity
  - Use total cost to merge potentially complex trees into simpler ones

# Boosting

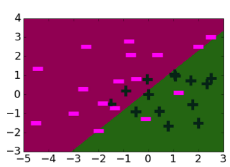
STAT/CSE 416: Machine Learning

Emily Fox
University of Washington

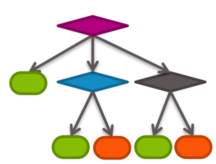
April 26, 2018



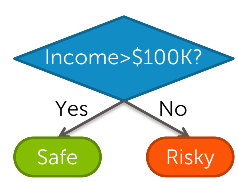
## Simple (weak) classifiers are good!



Logistic regression w. simple features



Shallow decision trees

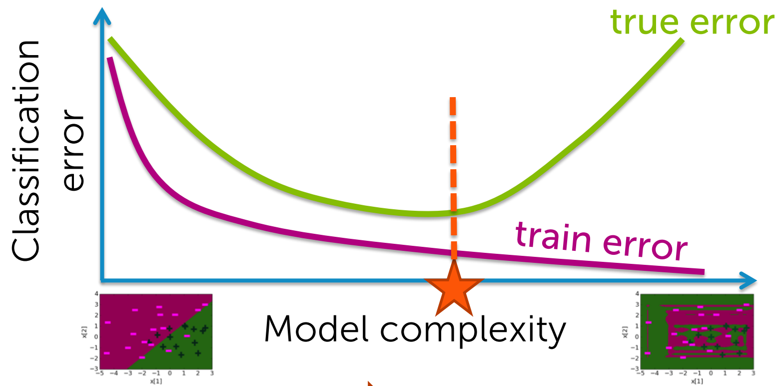



Decision stumps

Low variance. Learning is fast!

But high bias…

# Finding a classifier that's just right




Weak learner → Need stronger learner

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

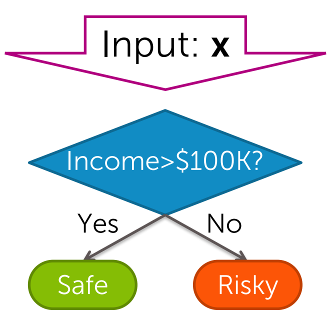
↓

**Amazing impact**: • simple approach • widely used in industry • wins most Kaggle competitions

# Ensemble classifier



# A single classifier




Output: $\hat{y} = f(\mathbf{x})$

– Either +1 or -1

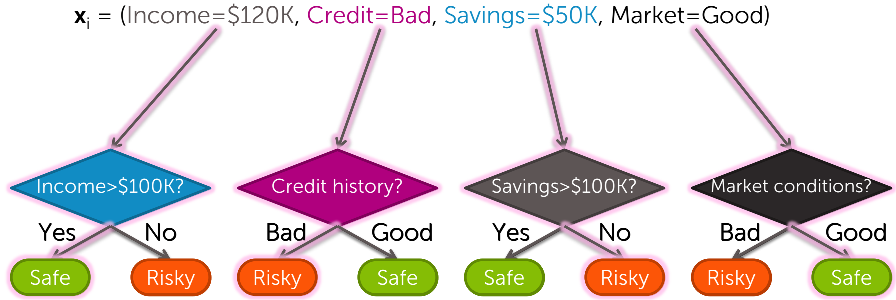
Classifier






# **Ensemble methods**: Each classifier "votes" on prediction

$\mathbf{x}_i$ = (Income=$120K, Credit=Bad, Savings=$50K, Market=Good)




$f_1(\mathbf{x}_i) = +1$ $f_2(\mathbf{x}_i) = -1$ $f_3(\mathbf{x}_i) = -1$ $f_4(\mathbf{x}_i) = +1$

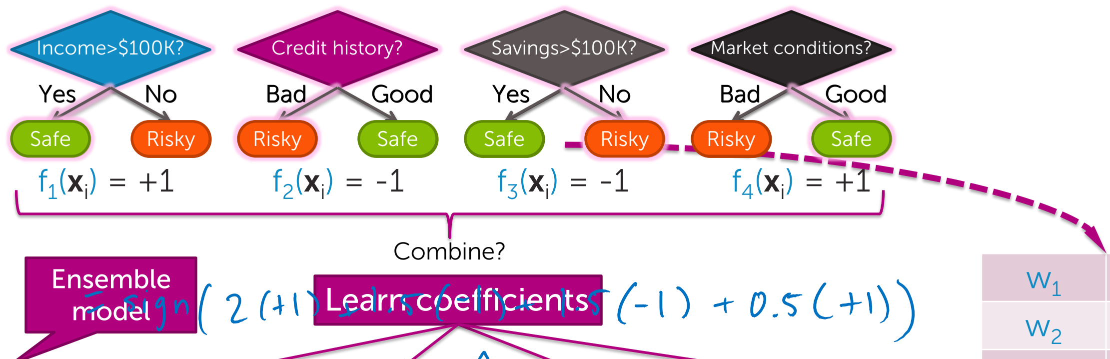
Combine?

Ensemble model

Learn coefficients

$$F(\mathbf{x}_i) = \text{sign}(w_1 f_1(\mathbf{x}_i) + w_2 f_2(\mathbf{x}_i) + w_3 f_3(\mathbf{x}_i) + w_4 f_4(\mathbf{x}_i))$$

# Prediction with ensemble




= sign( 2(+1) + 1.5(-1) + 1.5(-1) + 0.5(+1) )

$F(\mathbf{x}_i) = \text{sign}(w_1 f_1(\mathbf{x}_i) + w_2 f_2(\mathbf{x}_i) + w_3 f_3(\mathbf{x}_i) + w_4 f_4(\mathbf{x}_i))$

| | |
|---|---|
| $w_1$ | 2 |
| $w_2$ | 1.5 |
| $w_3$ | 1.5 |
| $w_4$ | 0.5 |





# Ensemble classifier in general

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

total # of classifiers

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

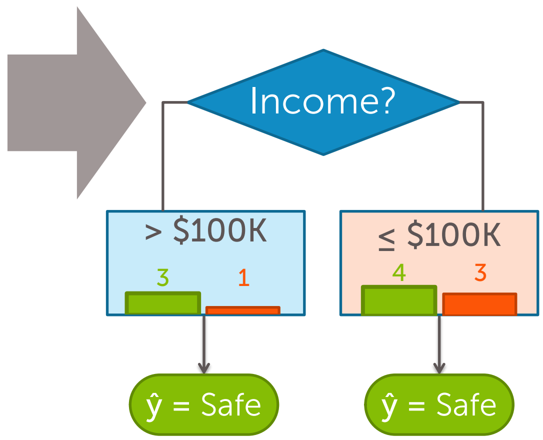
# Boosting



# Training a classifier

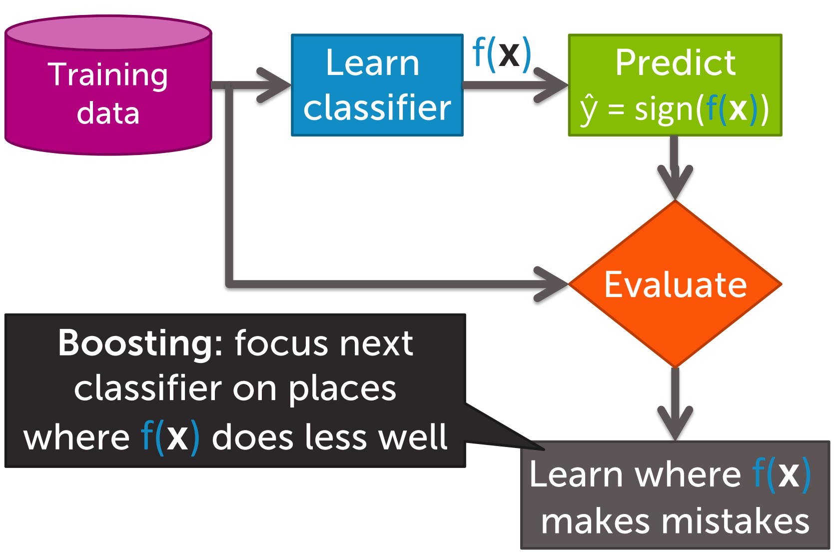
# Learning decision stump

| 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 |




43 ©2018 Emily Fox STAT/CSE 416: Intro to Machine Learning

# Boosting = Focus learning on "hard" points




44 ©2018 Emily Fox STAT/CSE 416: Intro to Machine Learning

# Learning on weighted data:
*More weight on "hard" or more important points*

- Weighted dataset:
  - Each $\mathbf{x}_i$,$y_i$ weighted by $\alpha_i$
    - More important point = higher weight $\alpha_i$
- Learning:
  - Data point i counts as $\alpha_i$ data points
    - E.g., $\alpha_i = 2$ ➔ count point twice

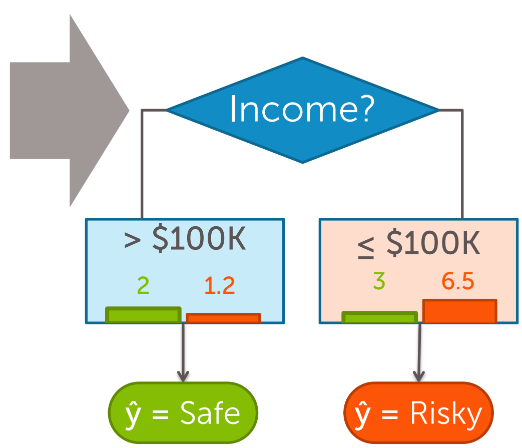




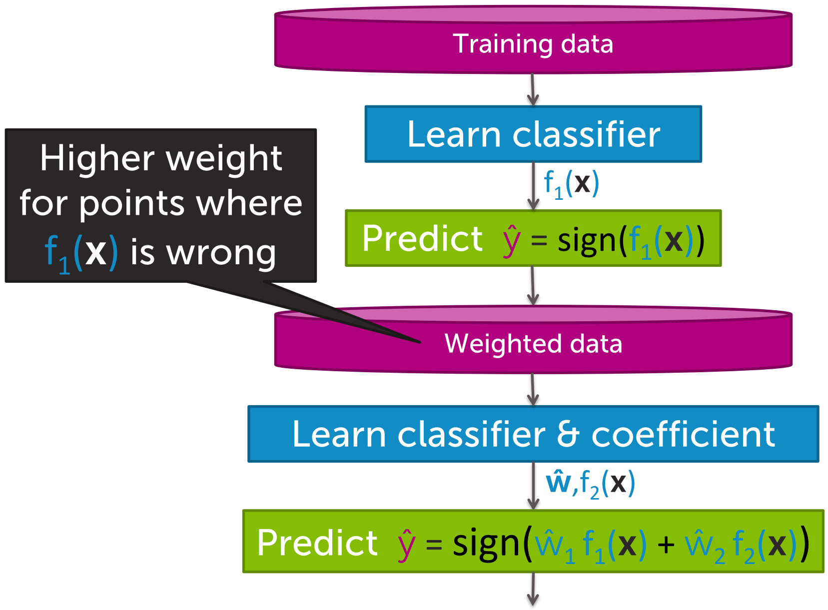
# Learning a decision stump on weighted data

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

©2018 Emily Fox STAT/CSE 416: Intro to Machine Learning

# AdaBoost: learning ensemble

*[Freund & Schapire 1999]*

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

- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$ with data weights $\alpha_i$
  - Compute coefficient $\hat{w}_t$ — Problem 1: How much do I trust $f_t$?
  - Recompute weights $\alpha_i$ — Problem 2: Weigh mistakes more

- Final model predicts by:

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



## Computing coefficient $\hat{w}_t$

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

Is $f_t(\mathbf{x})$ good? $\xrightarrow{\text{Yes}}$ $\hat{w}_t$ large

Is $f_t(\mathbf{x})$ good? $\xrightarrow{\text{No}}$ $\hat{w}_t$ small

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





## Weighted classification error

Learned classifier

$\hat{y}$ = +

**Data point**

(Sushi was great, +, α=1.2)
(Food was OK, +, α=0.5)

Correct!
Mistake!

| | |
|---|---|
| Weight of correct | 1.2 / 0 |
| Weight of mistakes | 0.5 / 0 |

Hide label

$$\text{weighted error} = \frac{\text{total weight of mistakes}}{\text{total weight}}$$

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

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

Is $f_t(\mathbf{x})$ good?

Yes → row 1; No → rows 2–3

| weighted_error($f_t$) on training data | $\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)}$ | $\hat{w}_t$ |
|---|---|---|
| 0.01 | $\frac{1-0.01}{0.01} = 99$ | 2.3 |
| 0.5 | 1 | 0 |
| 0.99 | 0.01 | -2.3 |

terrible classier, $1-\hat{f}_t$ is awesome!





## AdaBoost: learning ensemble

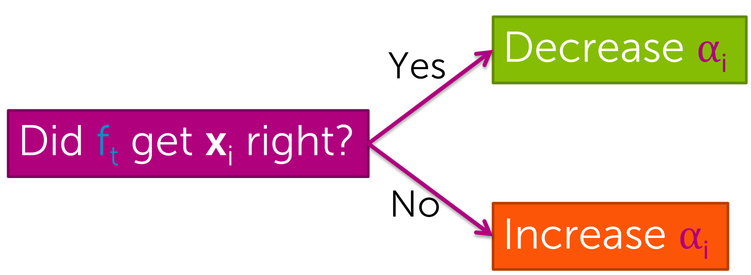
- Start with same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$ with data weights $\alpha_i$
  - Compute coefficient $\hat{w}_t$ ← $\hat{w}_t = \frac{1}{2} \ln \left( \frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)} \right)$
  - Recompute weights $\alpha_i$
- Final model predicts by:

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

## Recompute weights $\alpha_i$



## **AdaBoost**: Updating weights $\alpha_i$ based on where classifier $f_t(\mathbf{x})$ makes mistakes



# AdaBoost: Formula for updating weights $\alpha_i$

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

Did $f_t$ get $\mathbf{x}_i$ right? Yes / No

| $f_t(\mathbf{x}_i)=y_i$ ? | $\hat{w}_t$ | Multiply $\alpha_i$ by | Implication |
|---|---|---|---|
| yes | 2.3 | $e^{-2.3} = 0.1$ | decrease importance of $x_i, y_i$ |
| yes | 0 | $e^{-0} = 1$ | no change |
| no | 2.3 | $e^{2.3} = 9.98$ | increase importance of $x_i, y_i$ |
| no | 0 | $e^{0} = 1$ | no change |

# AdaBoost: learning ensemble

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

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

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

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

# AdaBoost: Normalizing weights $\alpha_i$

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

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

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

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

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

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

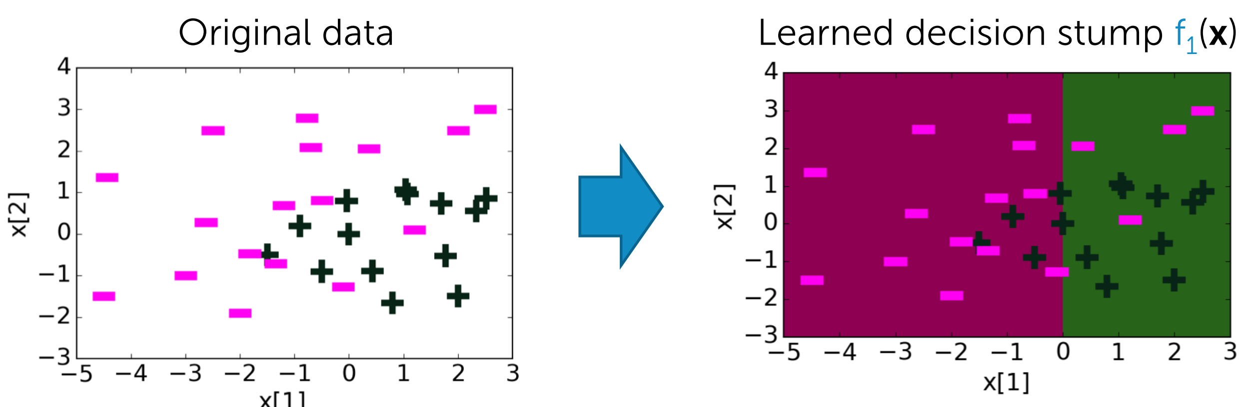
# AdaBoost example:
# **A visualization**




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

all pts have same weight

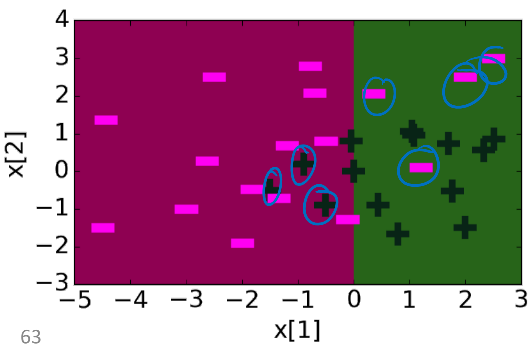
standard learning



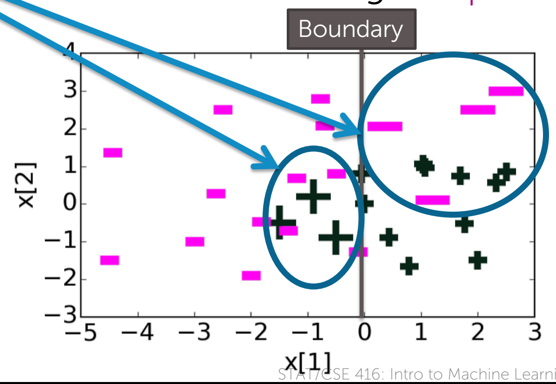
# Updating weights $\alpha_i$

Increase weight $\alpha_i$ of misclassified points

Learned decision stump $f_1(\mathbf{x})$




New data weights $\alpha_i$

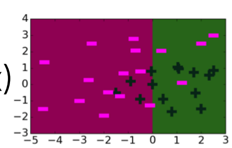




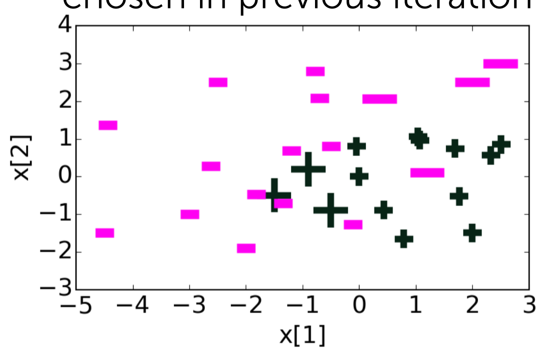



# t=2: Learn classifier on weighted data

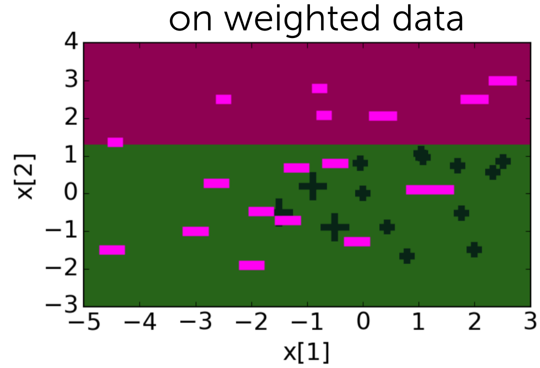
$f_1(\mathbf{x})$



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




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

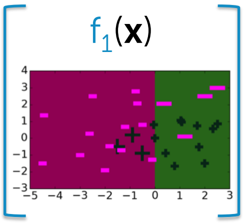
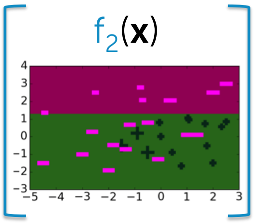


# Ensemble becomes weighted sum of learned classifiers

$\hat{y} = \text{sign}(\underbrace{\hat{w}_1 f_1(x) + \hat{w}_2 f_2(x)}_{\text{score}(x)})$

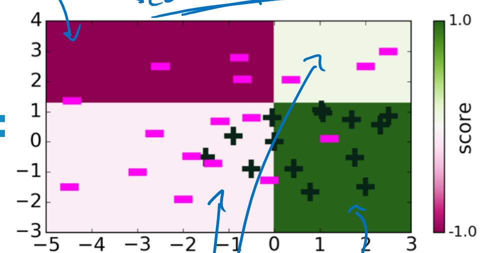
Score(x) =

$\hat{w}_1$ $f_1(\mathbf{x})$

0.61  + $\hat{w}_2$ $f_2(\mathbf{x})$ 0.53  =

definite $\hat{y} = -1$

score space



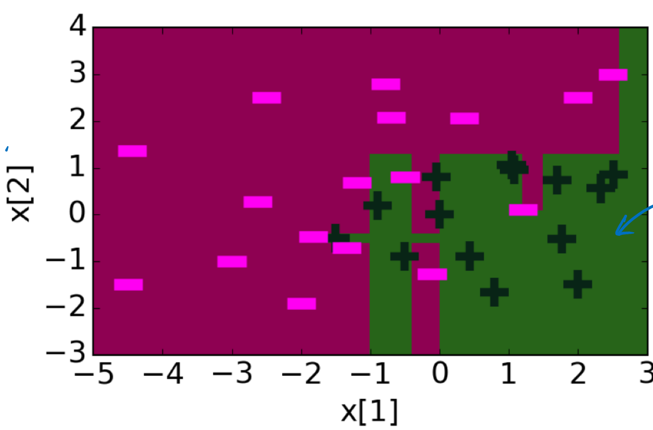

uncertain

definite $\hat{y} = +1$

# Decision boundary of ensemble classifier after 30 iterations

(30 classifiers, T=30)




decision boundary is crazy!

↓

probably overfitting

training_error = 0

AdaBoost example:
**Boosted decision stumps step-by-step**



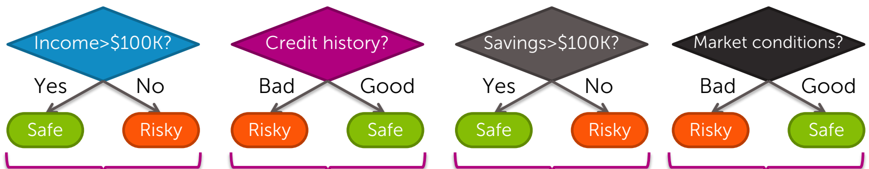
# Boosted decision stumps

- Start same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$: pick decision stump with lowest weighted training error according to $\alpha_i$
  - Compute coefficient $\hat{w}_t$
  - Recompute weights $\alpha_i$
  - Normalize weights $\alpha_i$
- Final model predicts by:

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

# Finding best next decision stump $f_t(\mathbf{x})$

Consider splitting on each feature:



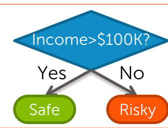

weighted_error = 0.2

weighted_error = 0.35

weighted_error = 0.3

weighted_error = 0.4

$f_t =$




$$\hat{w}_t = \frac{1}{2} \ln\left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)}\right) = 0.69$$



# Boosted decision stumps

- Start same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$: pick decision stump with lowest weighted training error according to $\alpha_i$
  - Compute coefficient $\hat{w}_t$
  - Recompute weights $\alpha_i$
  - Normalize weights $\alpha_i$
- Final model predicts by:

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

# Updating weights $\alpha_i$


$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t} = \alpha_i e^{-0.69} = \alpha_i/2, & \text{if } f_t(x_i) = y_i \\ \alpha_i e^{\hat{w}_t} = \alpha_i e^{0.69} = 2\,\alpha_i, & \text{if } f_t(x_i) \neq y_i \end{cases}$$

| Credit | Income | y | ŷ | Previous weight α | New weight α |
|---|---|---|---|---|---|
| A | $130K | Safe | Safe | 0.5 | 0.5/2 = 0.25 |
| B | $80K | Risky | Risky | 1.5 | 0.75 |
| C | $110K | Risky | Safe | 1.5 | 2 * 1.5 = 3 |
| A | $110K | Safe | Safe | 2 | 1 |
| A | $90K | Safe | Risky | 1 | 2 |
| B | $120K | Safe | Safe | 2.5 | 1.25 |
| C | $30K | Risky | Risky | 3 | 1.5 |
| C | $60K | Risky | Risky | 2 | 1 |
| B | $95K | Safe | Risky | 0.5 | 1 |
| A | $60K | Safe | Risky | 1 | 2 |
| A | $98K | Safe | Risky | 0.5 | 1 |





# Boosting convergence & overfitting

# Boosting question revisited

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

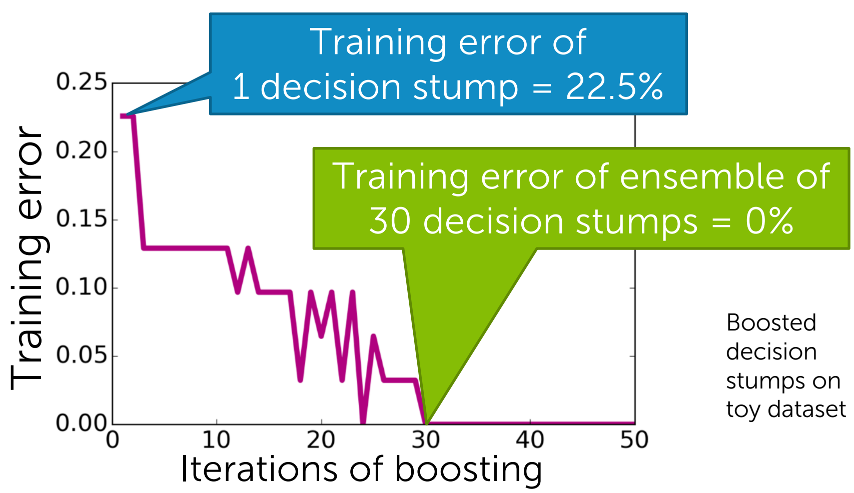
Yes! *Schapire (1990)*

Boosting





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

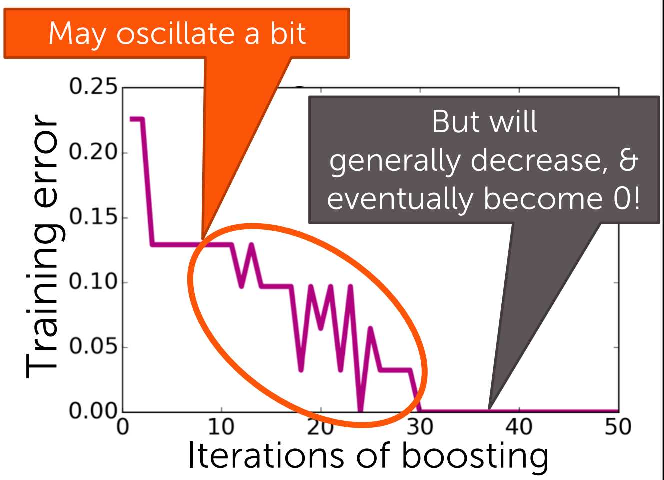



Boosted decision stumps on toy dataset

# AdaBoost Theorem

Under some technical conditions…

Training error of boosted classifier → 0 as T→∞






# Condition of AdaBoost Theorem

Under some technical conditions…

Training error of boosted classifier → 0 as T→∞

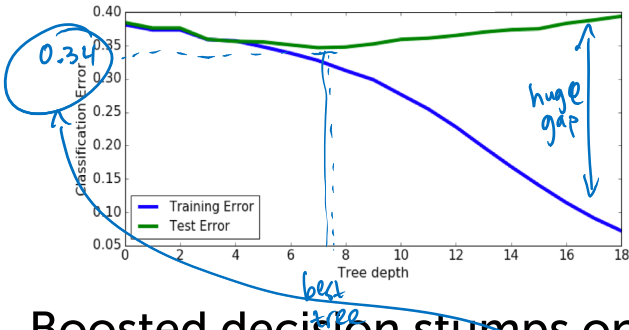
Condition = At every t, can find a weak learner with weighted_error($f_t$) < 0.5

Not always possible

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

Nonetheless, boosting often yields great training error

# Decision trees on loan data

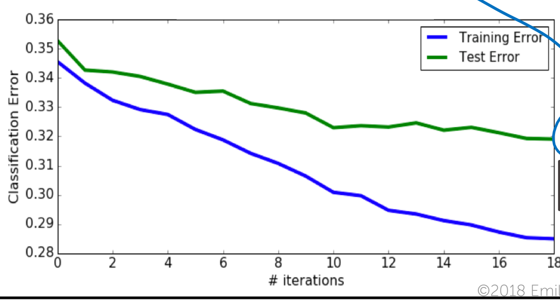



39% test error

Overfitting

8% training error

# Boosted decision stumps on loan data




32% test error

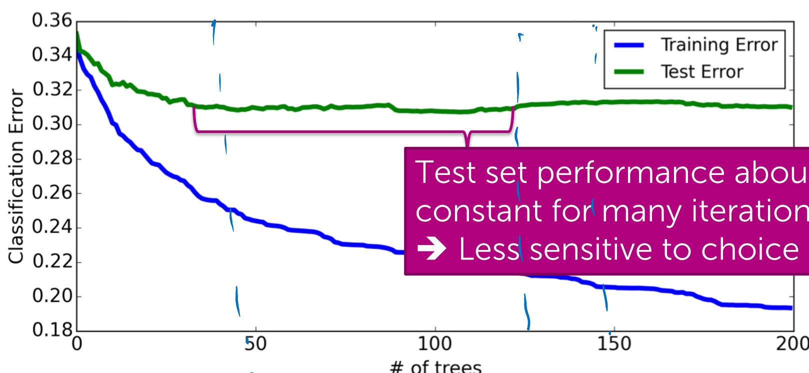
Better fit & lower test error

28.5% training error





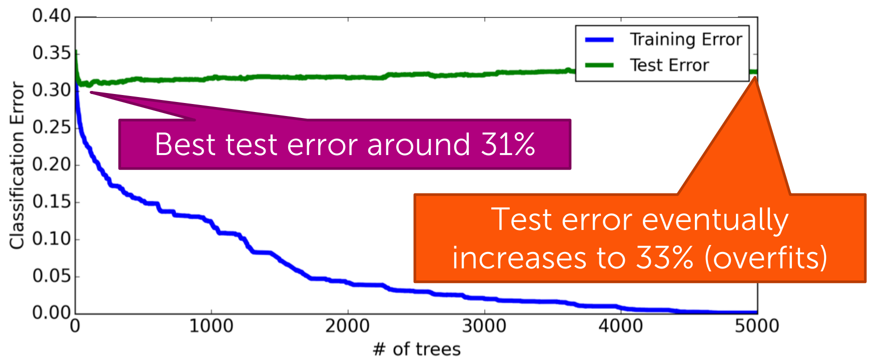
# Boosting tends to be robust to overfitting




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

any of these values of T would be fine

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








# How do we decide when to stop boosting?

Choosing T ?

Like selecting parameters in other ML approaches (e.g., λ in regularization)

- Not on training data — Not useful: training error improves as T increases
- Never ever ever ever on test data
- Validation set — If dataset is large
- Cross-validation — For smaller data

# 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

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