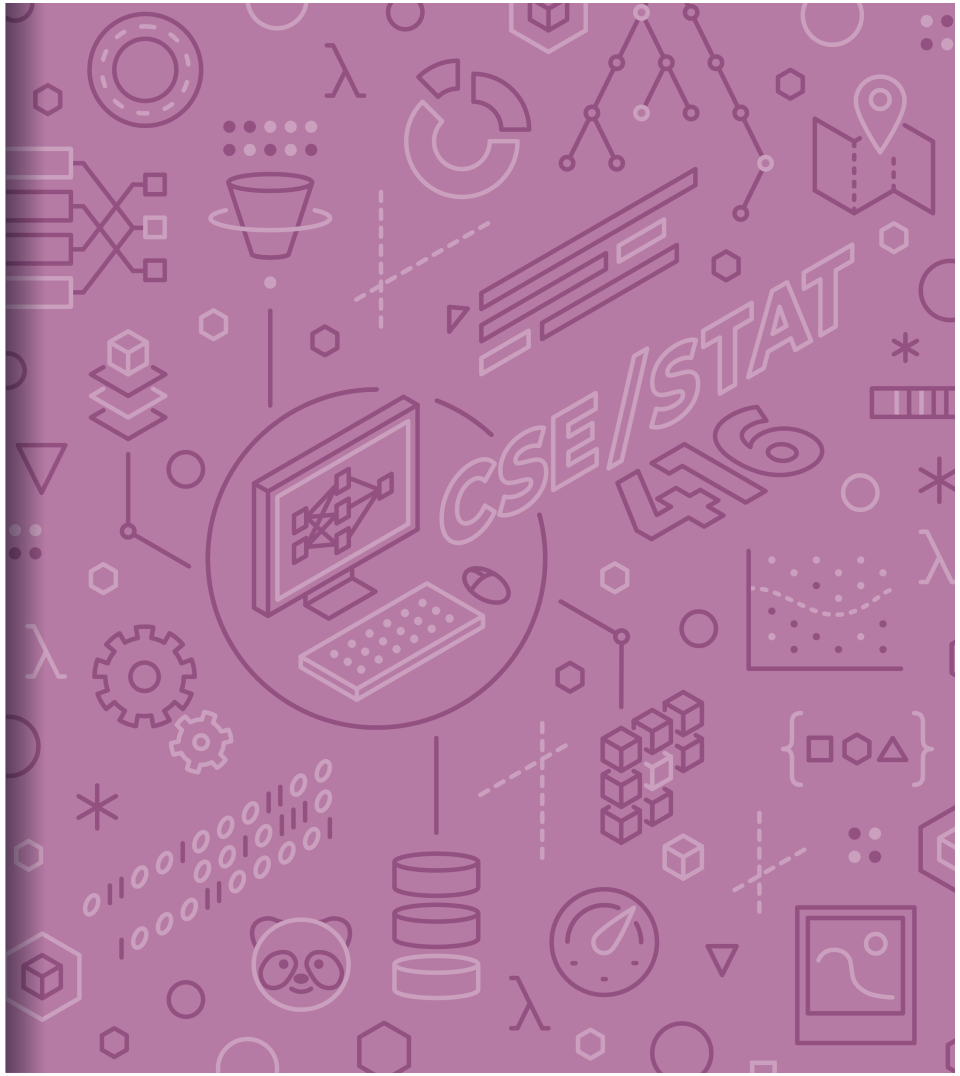
CSE/STAT 416

Ensemble Methods

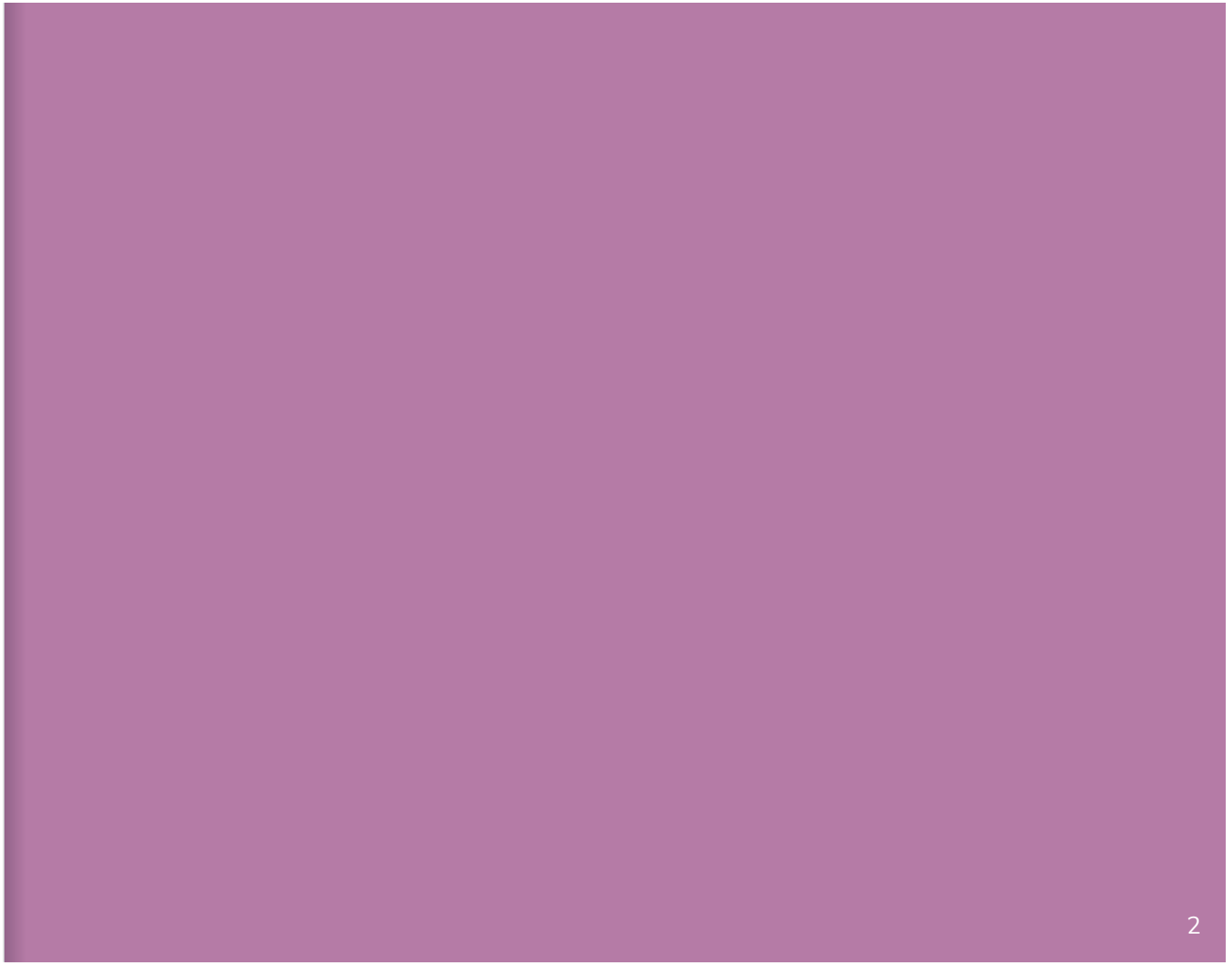
Tanmay Shah
Paul G. Allen School of Computer Science & Engineering
University of Washington

April 26 , 2024



Decision Trees Recap

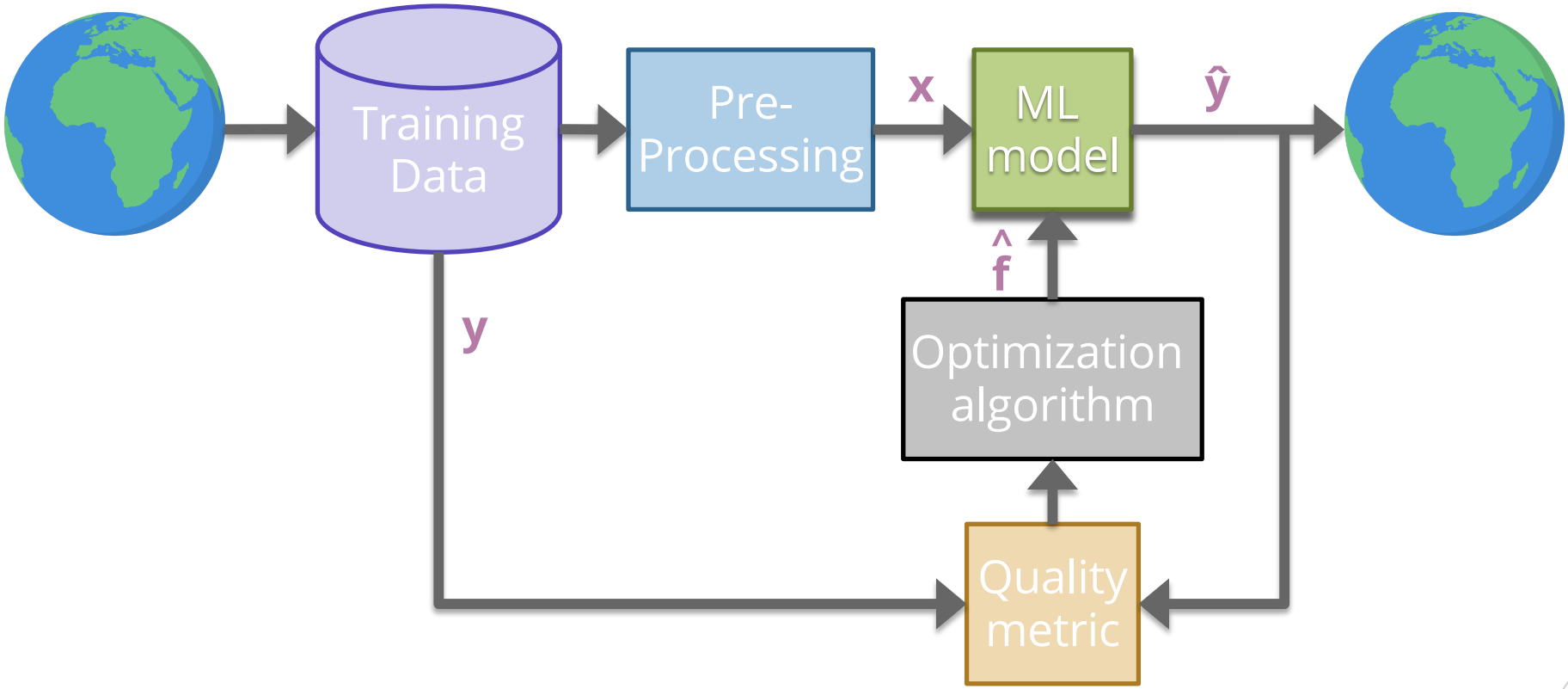
Video 1



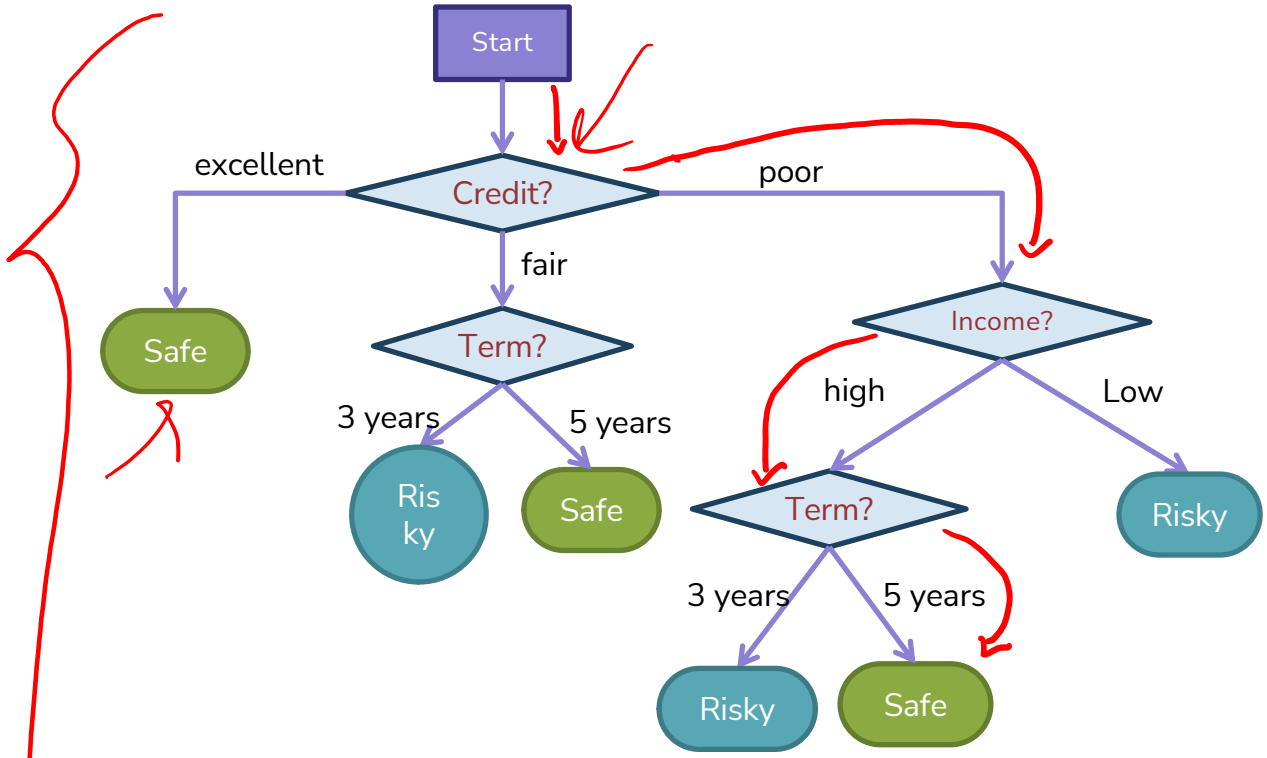
Roadmap So Far

1. Housing Prices - Regression
   - Regression Model
   - Assessing Performance
   - Ridge Regression
   - LASSO
2. Sentiment Analysis – Classification
   - Classification Overview
   - Logistic Regression
   - Naïve Bayes
   - Decision Trees
   - Ensemble Methods

ML Pipeline


Training Data
Pre-Processing
x
ML model
ŷ
f̂
y
Optimization algorithm
Quality metric

Decision Trees

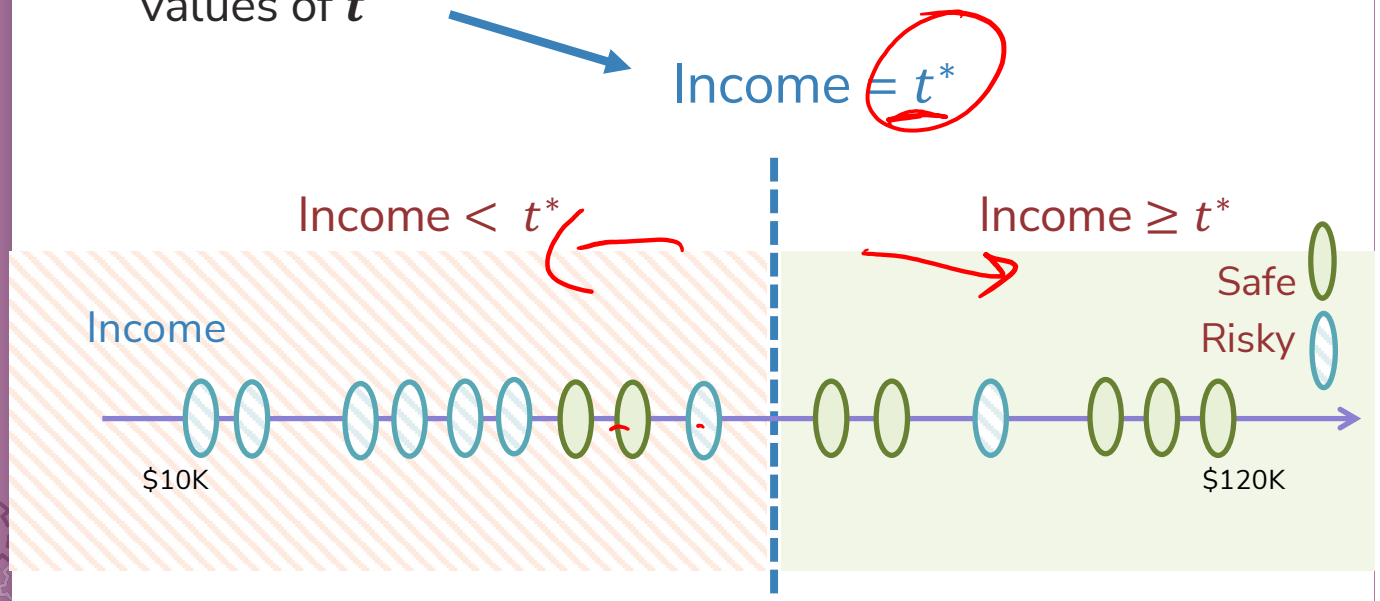


- **Branch/Internal node:** splits into possible values of a feature
- **Leaf node:** final decision (the class value)

Best threshold?

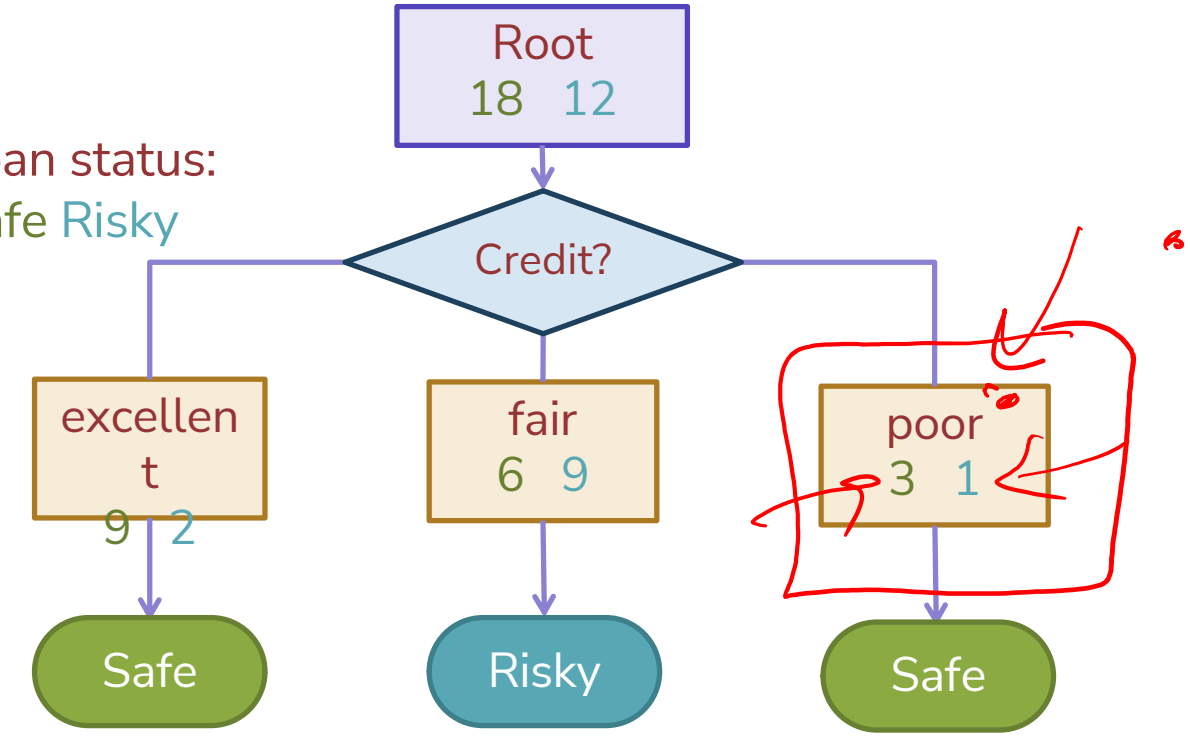
Similar to our simple, threshold model when discussing Fairness!

Infinite possible values of t



Predicting probabilities

Loan status:
Safe Risky

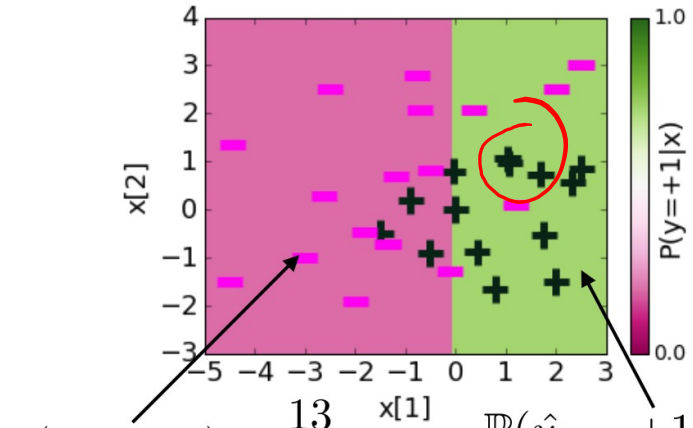


P(y = Safe | x)

= 3/4

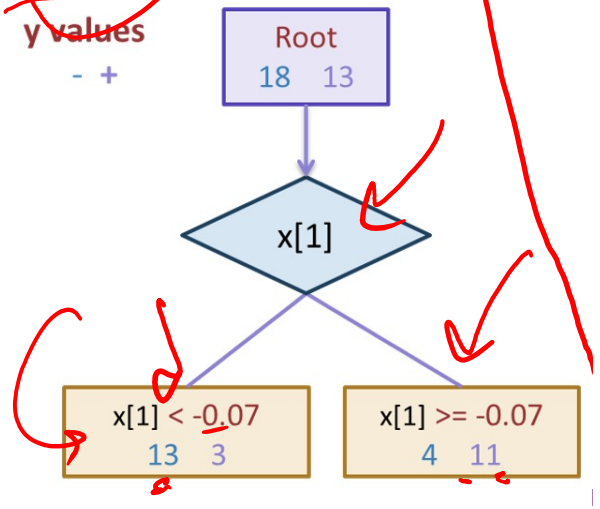
= 0.75

Probabilities (Depth 1)

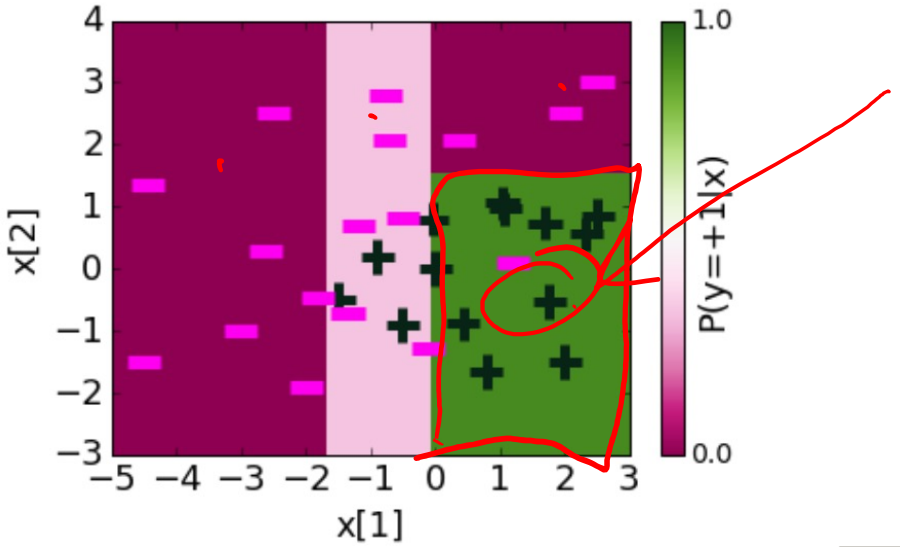


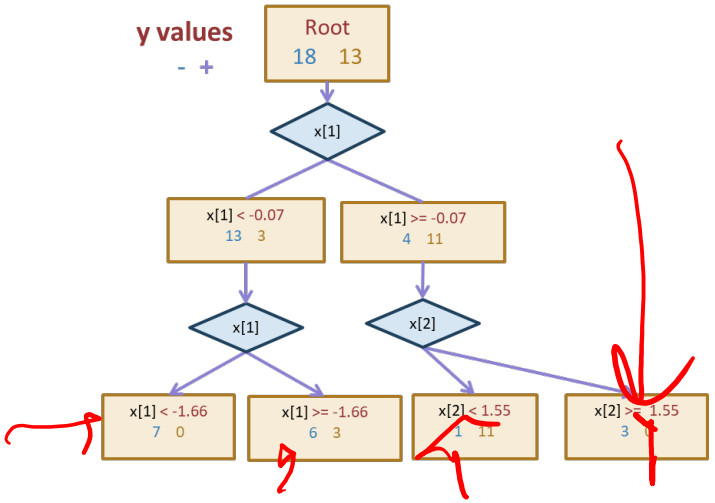
$$\mathbb{P}(\hat{y}_i = -1) = \frac{13}{16}$$

$$\mathbb{P}(\hat{y}_i = +1) = \frac{11}{15}$$

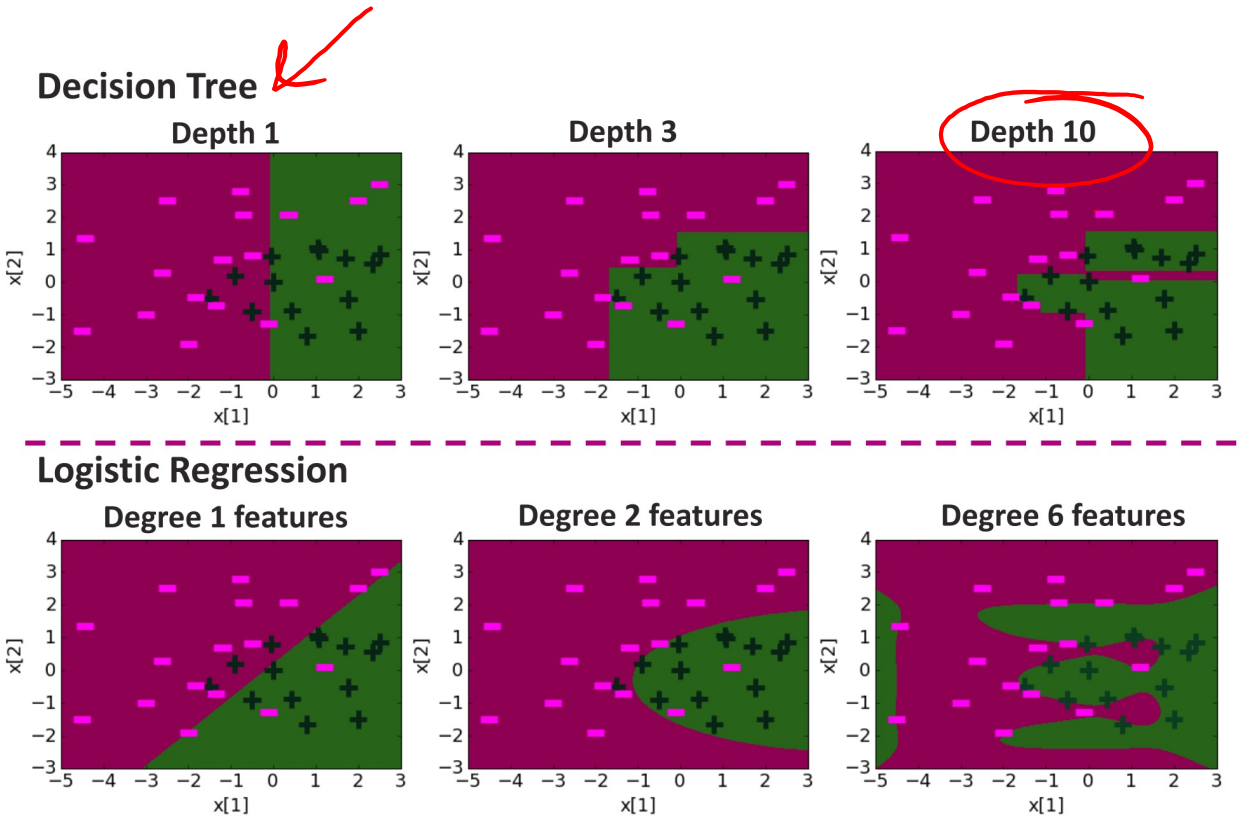


Depth 2


x[2]
x[1]
P(y=+1|x)
1.0
0.0


y values
- +
Root
18 13
x[1]
x[1] < -0.07
13 3
x[1] >= -0.07
4 11
x[1]
x[2]
x[1] < -1.66
7 0
x[1] >= -1.66
6 3
x[2] < 1.55
1 11

Compare Decision Boundaries



Overfitting

Deep decision trees are prone to overfitting

- Decision boundaries are interpretable but not stable
- Small change in the dataset leads to big difference in the outcome

Overcoming Overfitting:

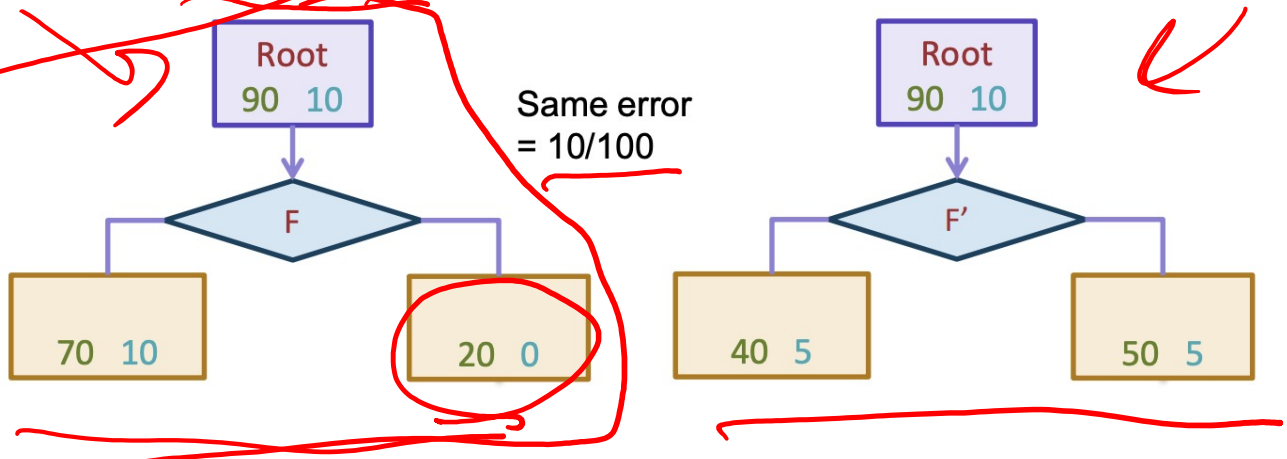
- Early stopping
  - Fixed length depth
  - Stop if error does not considerably decrease
- Pruning
  - Grow full length trees
  - Prune nodes to balance a complexity penalty

In Practice

Trees can be used for classification or regression (CART)

- Classification: Predict majority class for root node
- Regression: Predict average label for root node

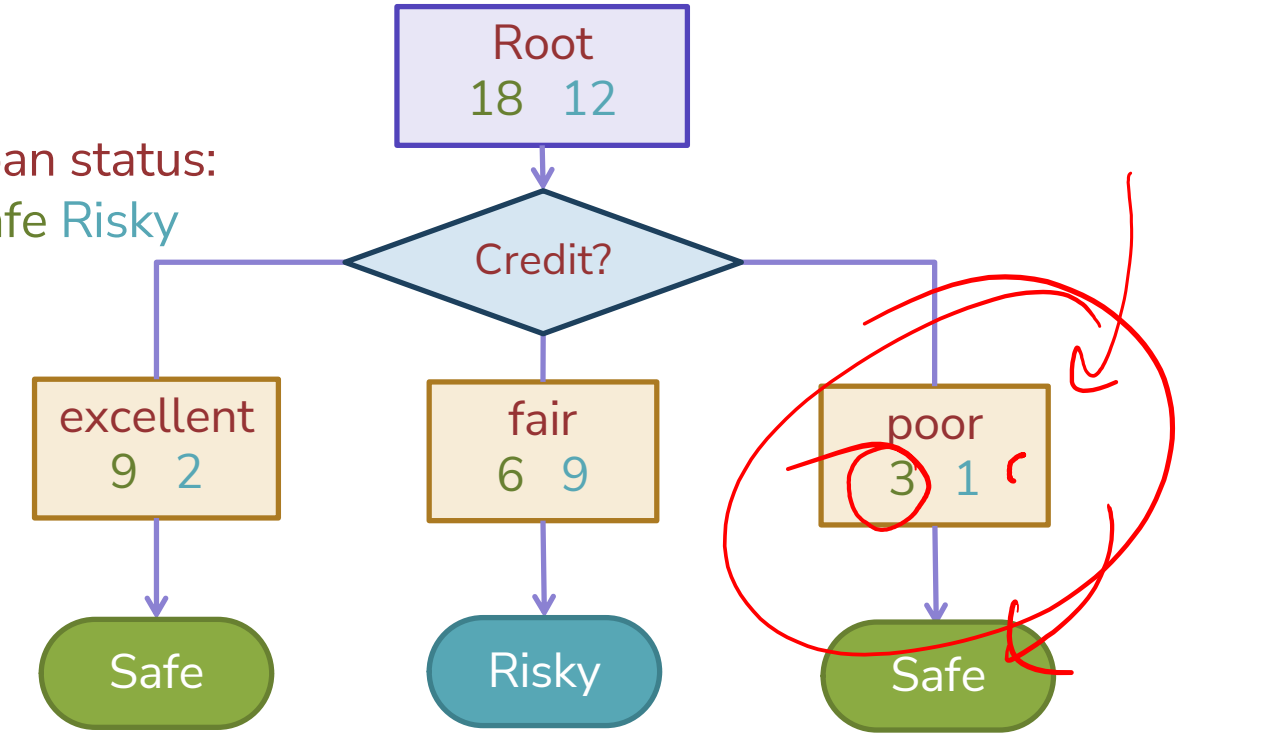
In practice, we don't minimize classification error but instead some more complex metric to measure quality of split such as **Gini Impurity** or **Information Gain** (not covered in 416)



Can also be used to predict probabilities

Predicting probabilities

Loan status:
Safe Risky



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

Early Stopping

Stopping Rules:

- 1) All data in the subset have the same label
- 2) No more features left to split

Early Stopping Rule

- Only grow up to a max depth hyperparameter (choose via validation)
- Don't split if there is not a sufficient decrease in error
- **Require a minimum number of examples in a leaf node**
  - Will use this on HW

Decision Tree Overview

Super Simple: Interpretable model that is understandable by people without too much ML experience.

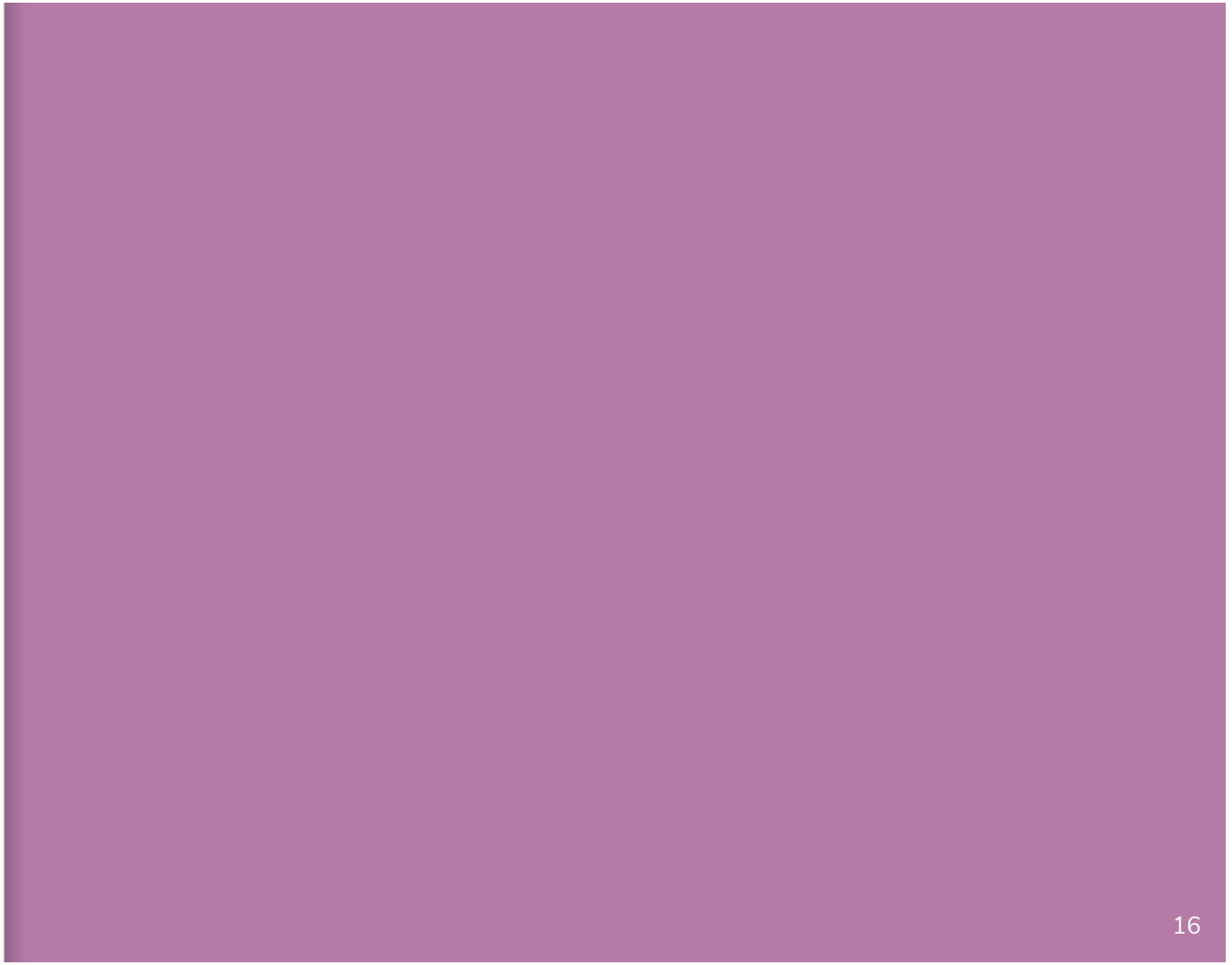
Very Efficient: It actually isn't too hard to train a tree

Depth Matters

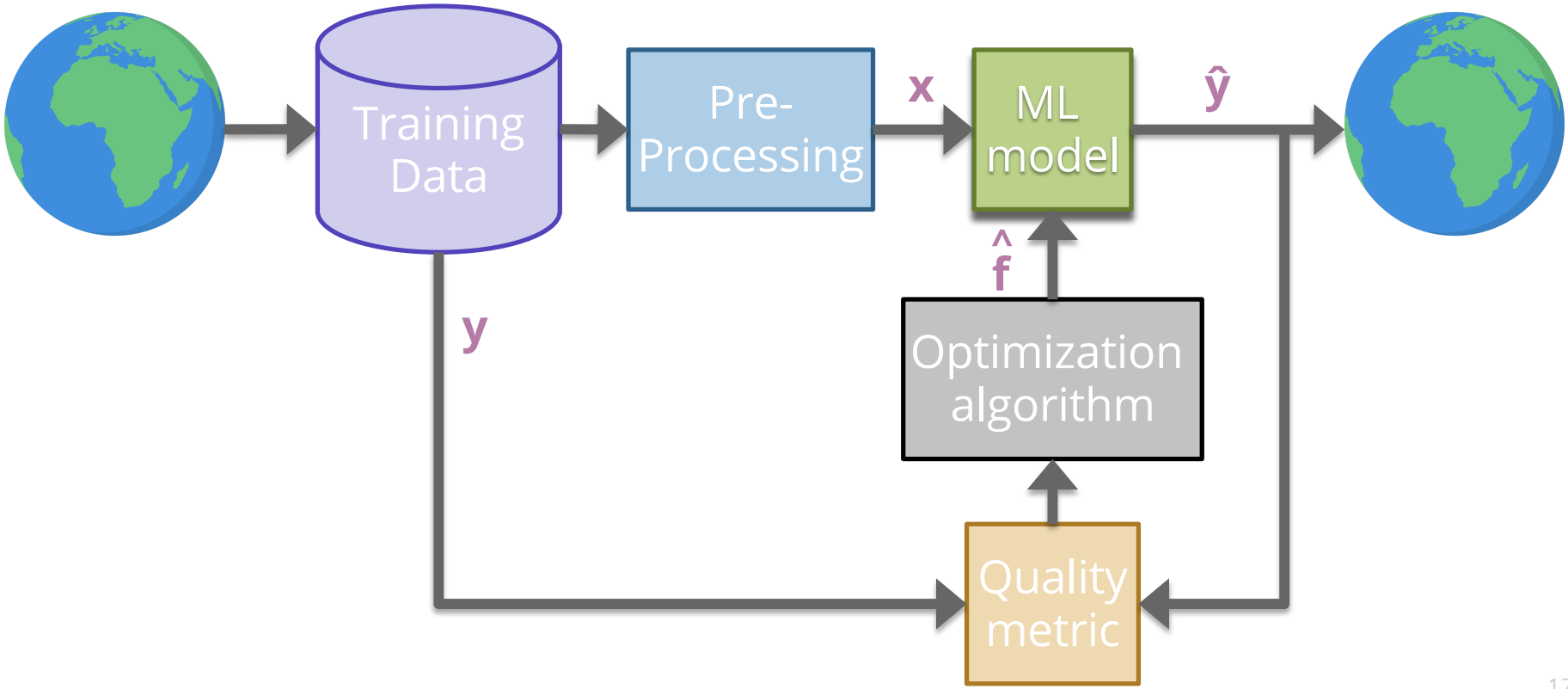
- Too small, it is too weak to learn the function (high bias)
- Too tall, it is likely to overfit to the data (high variance)
- Even by choosing depth appropriately, trees tend to not be the best performing models

Random Forests

Video 2



ML Pipeline


Training Data
Pre-Processing
x
ML model
ŷ
f̂
y
Optimization algorithm
Quality metric

Ensemble Method

Instead of switching to a brand new type of model that is more powerful than trees, what if we instead tried to make the tree into a more powerful model.

What if we could combine many weaker models in such a way to make a more powerful model?

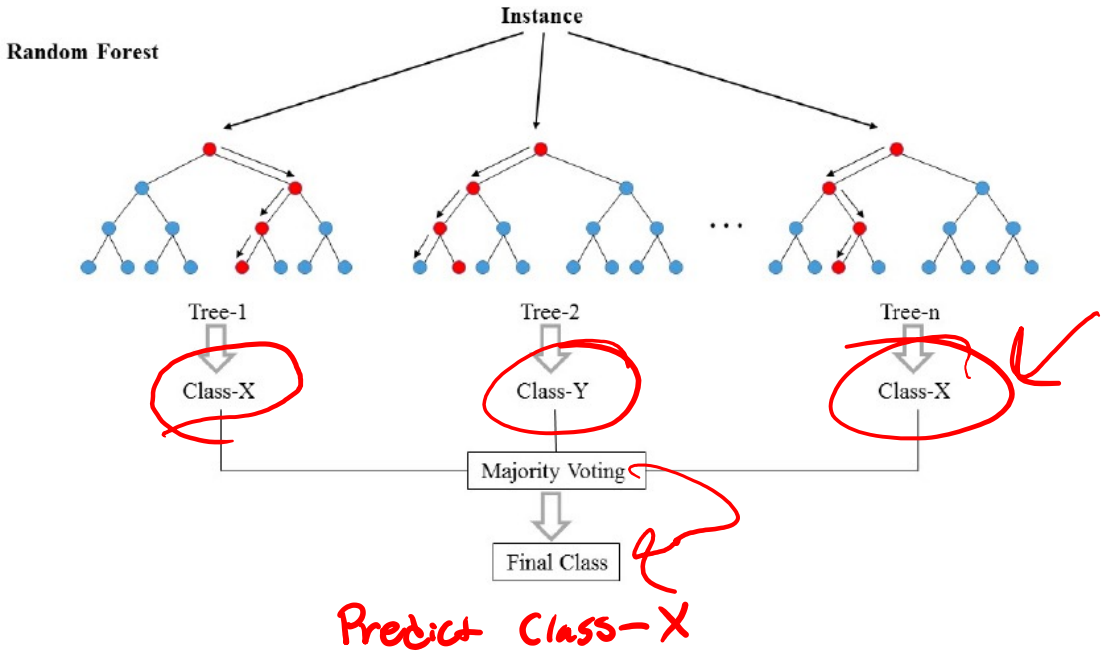
A **model ensemble** is a collection of (generally weak) models that are combined in such a way to create a more powerful model.

There are two common ways this is done with trees

- Random Forest (Bagging)
- AdaBoost (Boosting)

Overview

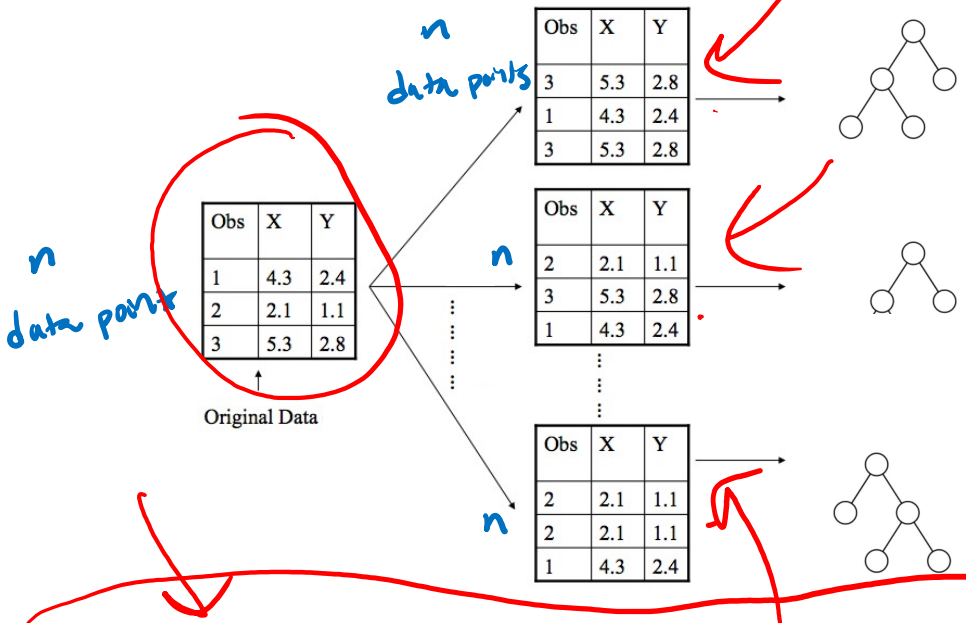
A **Random Forest** is a collection of T Decision Trees. Each decision tree casts a "vote" for a prediction and the ensemble predicts the majority vote of all of its trees.



Training Trees

If I just have one dataset, how could I learn more than one tree?

Solve this with **bootstrapping**! Can create many similar datasets by randomly sampling *with replacement*.



Technically, you also randomly select features too! $(m < D)$

Details

The Random Forest model is a specific type of ensemble model that uses **bagging** (bootstrapped aggregation).

When training the trees on the bootstrapped samples, we actually want to use very deep trees that overfit!

That sounds bad at first, but we are trying to take advantage of what it means to have a high variance model (low bias).

Remember that high variance models have low bias because if you "average out" over all the models you could learn, they will not have bias.

That is exactly what we are doing here! If we average over a bunch of high variance (overfit) models, to get an ensemble that has low bias and lower variance (if we add more trees)!

Random Forest Algorithm

Training

- Make T random samples of the training data that are the same size as the training data but are sampled with replacement
- Train a really tall tree on each sampled dataset (overfit)

Predict

- For a given example, ask each tree to predict what it thinks the label should be
- Take a majority vote over all trees

Application

Microsoft used Random Forests in their Kinect system to identify the “pose” of a person from the depth camera.

Real-Time Human Pose Recognition in Parts from Single Depth Images

Jamie Shotton Andrew Fitzgibbon Mat Cook Toby Sharp Mark Finocchio
Richard Moore Alex Kipman Andrew Blake
Microsoft Research Cambridge & Xbox Incubation

Abstract

We propose a new method to quickly and accurately predict 3D positions of body joints from a single depth image, using no temporal information. We take an object recognition approach, designing an intermediate body parts representation that maps the difficult pose estimation problem into a simpler per-pixel classification problem. Our large and highly varied training dataset allows the classifier to estimate body parts invariant to pose, body shape, clothing, etc. *Finally we generate confidence-scored 3D proposals of several body joints by reprojecting the classification result and finding local modes.*

The system runs at 200 frames per second on consumer hardware. Our evaluation shows high accuracy on both synthetic and real test sets, and investigates the effect of several training parameters. We achieve state of the art accuracy in our comparison with related work and demonstrate improved generalization over exact whole-skeleton nearest neighbor matching.

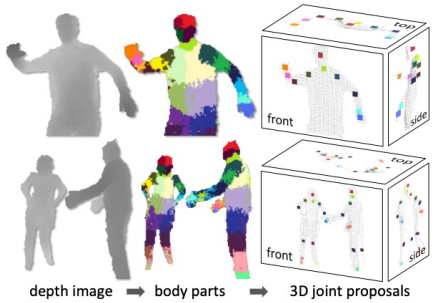


Figure 1. **Overview.** From an single input depth image, a per-pixel body part distribution is inferred. (Colors indicate the most likely part labels at each pixel, and correspond in the joint proposals). Local modes of this signal are estimated to give high-quality proposals for the 3D locations of body joints, even for multiple users.

Random Forest Overview

Use overfitting to our advantage! Averaging overfit models can help make a strong model.

Versatile: Works pretty well in a lot of cases and can serve many different purposes.

- Classification, regression, clustering, feature importance

Low Maintenance: Tends to require less hyper-parameter tuning. Good "out of the box" model.

- More trees is always better here (but takes longer).
- Some other hyperparameters, but they tend to have a small affect on performance.

Efficient: Trees can be learned in parallel!

Brain Break

Types of Features

Numeric: data described by a number (quantitative)

- **Discrete:** cannot be subdivided
  - e.g., number of bedrooms
- **Continuous:** can be subdivided
  - e.g., area of the house
- **Tricky Case**: house price? (don't divide further than penny)
  - **Rule of Thumb**: if the discreteness is caused by units of measurement, as opposed to the quantity being measured, treat it as continuous!

Categorical: data described by a category (qualitative)

- **Ordinal**: has an order
  e.g., school quality (good / okay / poor)
  e.g., survey response (agree / neutral / disagree)
- **Nominal**: doesn't have an order
  - e.g., nearest school type (public / private / charter)

Data Encodings

All ML models we've learnt so far require input features to be numbers!

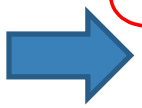
Ordinal: Assign each value to a number:

- e.g., good = 1, okay = 0, poor = -1

Nominal: One-hot encoding, make each value its own binary feature!

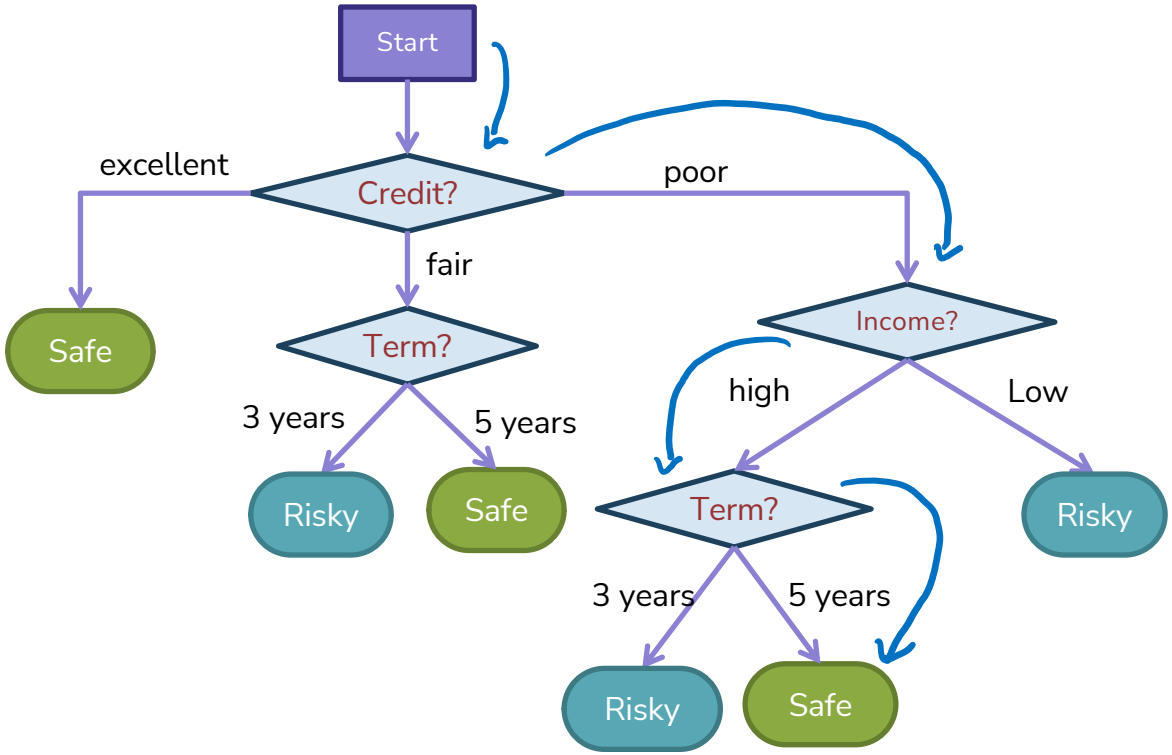
- In section, you saw a one-hot encoding of "County"

School	House Price
Public	$500K
Private	$750K
Charter	$600K
Public	$700K



School - Public	School - Private	School - Charter	House Price
1	0	0	$500K
0	1	0	$750K
0	0	1	$600K
1	0	0	$700K

Decision Trees



- **Branch/Internal node:** splits into possible values of a feature
- **Leaf node:** final decision (the class value)

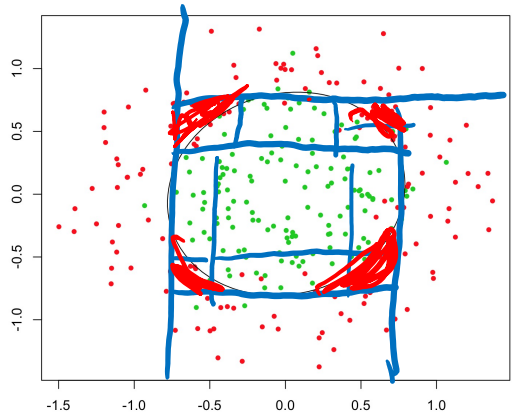
Pros/Cons Decision Tree

Pros:

- Easy to interpret
- Handles numeric and categorical variables without preprocessing*
  - In theory, scikit-learn still requires preprocessing
- No normalization required as it uses rule-based approach
- Can create non-linear decision boundaries
- Can readily do multi-class classification (unlike Logistic Regression)

Cons:

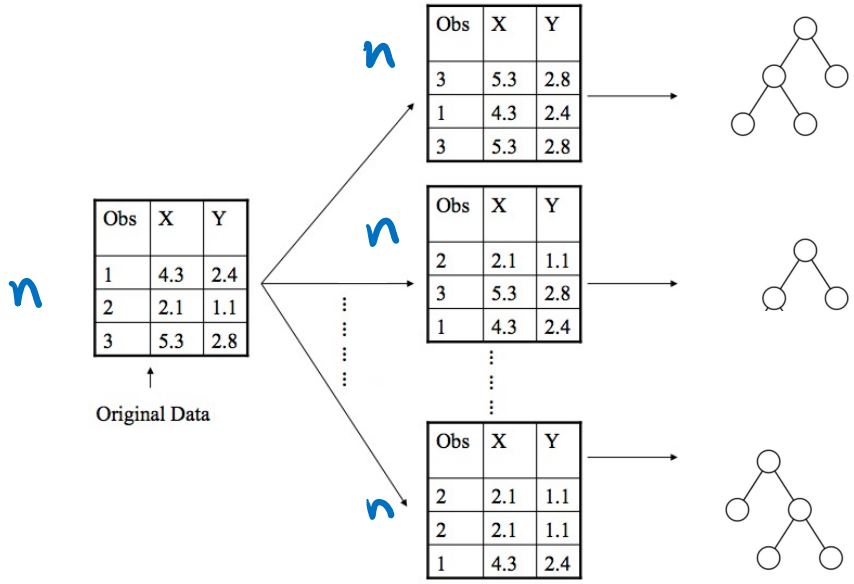
- Deep decision trees are prone to overfitting
- Only allows axis-parallel decision boundaries



Training Trees

If I just have one dataset, how could I learn more than one tree?

Solve this with **bootstrap sampling**! Can create many similar datasets by randomly sampling *with replacement*.



- Technically, you also randomly select features too!

Practical Details

Random Forests

Important that you are also randomly sampling features for each tree too! Yet another hyperparameter.

Author recommends first guesses:

- Classification: $\sqrt{D}$, Regression: $D/3$

Added Benefit: Out Of Bag (OOB) Error

- Can estimate future performance on *training data!*
- For each training point, only ask for predictions from trees that did not train on that point.
- Still a good general idea to have a test set anyways, but is an added benefit if you have a small amount of data.

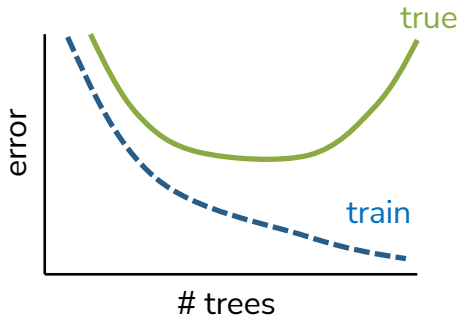
slido

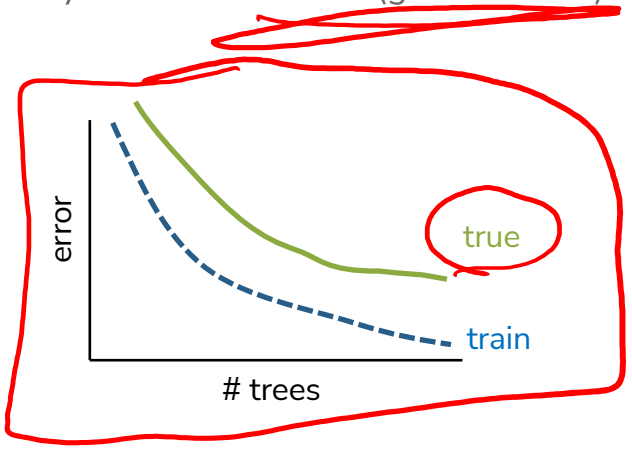
Think

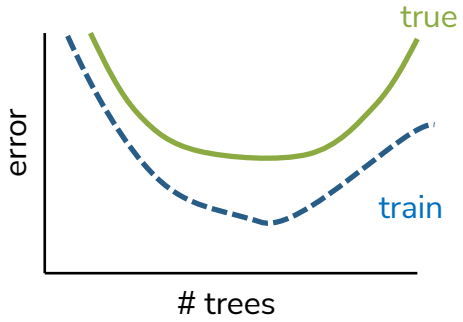
1 min

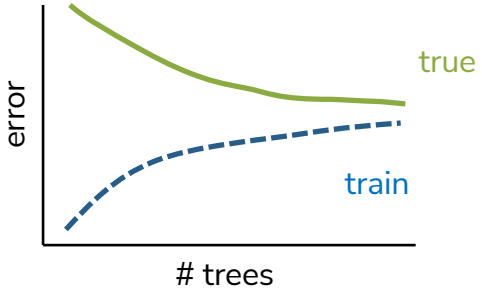
sli.do #cs416

Which of the following graphs do you think shows the training/true **error** curves for random forests as you increase # trees (general trend)?









slido

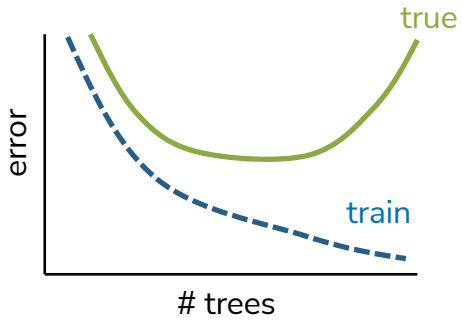
Group

2 min

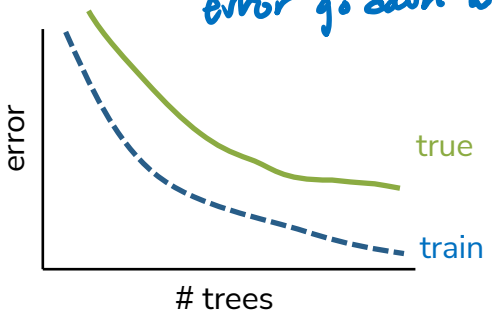
sli.do #cs416

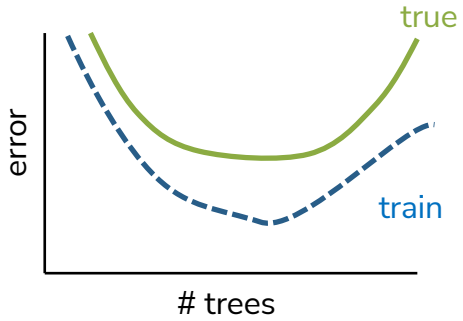
Recall, each tree is high variance / low bias!

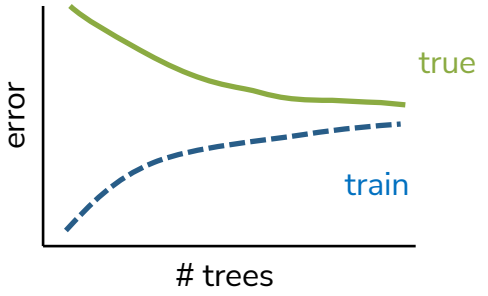
Which of the following graphs do you think shows the training/true **error** curves for random forests as you increase # trees (general trend)?



☆ Generally train/true error go down w/ # trees



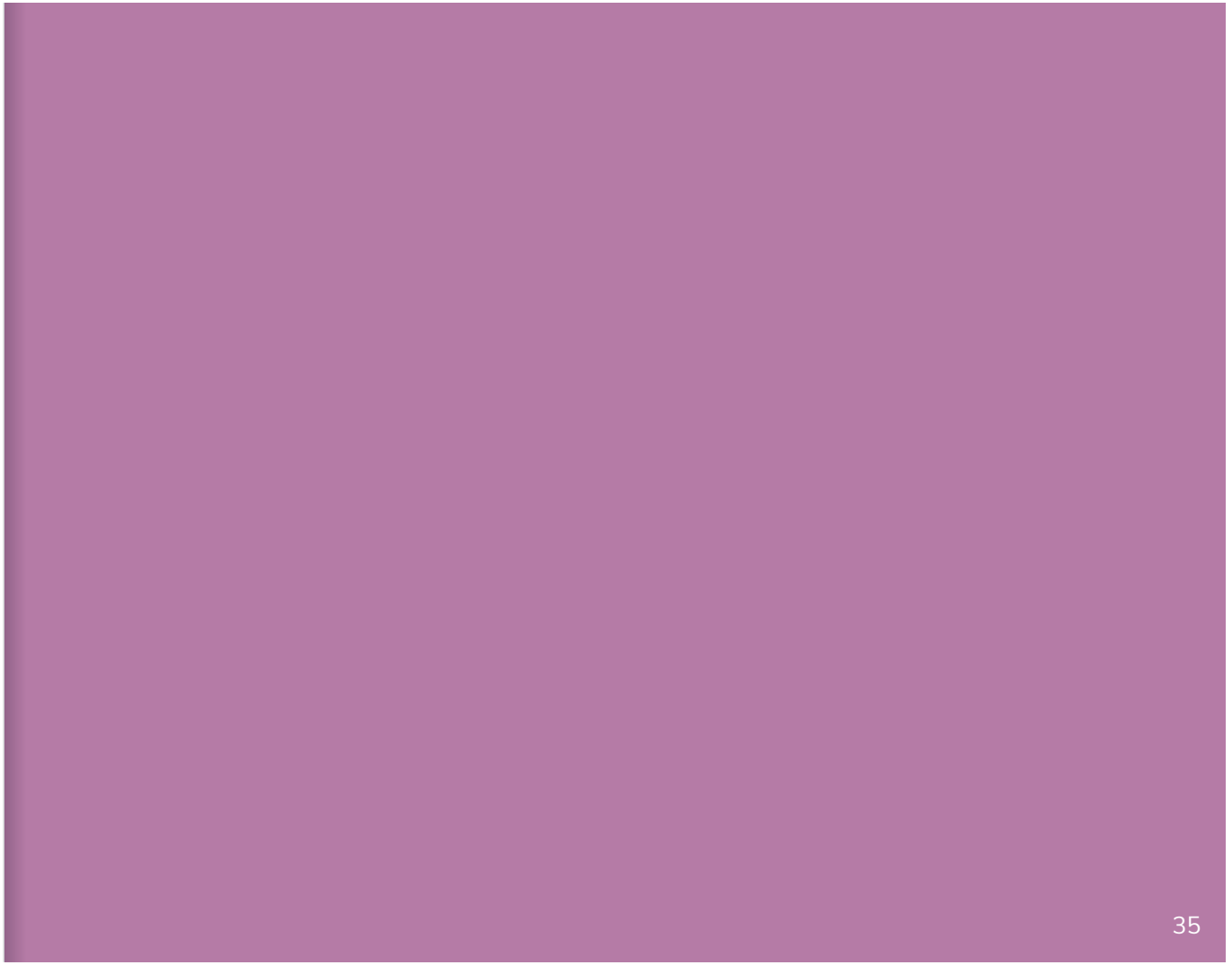




Brain Break

AdaBoost

Boosting



Background

A **weak learner** is a model that only does slightly better than random guessing.

Kearns and Valiant (1988, 1989):

> "Can a set of weak learners create a single strong learner?"

Schapire (1990)

> "Yes!"

AdaBoost Overview

AdaBoost is a model similar to Random Forest (an ensemble of decision trees) with three notable differences that impact how we train it quite severely.

Instead of using high depth trees that will overfit, we limit ourselves to **decision stumps**.

Instead of doing majority voting, each model in the ensemble gets a weight and we take a **weighted majority vote**

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

weight of stump t

decision stump t

Instead of doing random sampling with replacement, we **use the whole dataset and assign each datapoint a weight**, where high-weight datapoints were frequently misclassified by earlier models in the ensemble.

slido

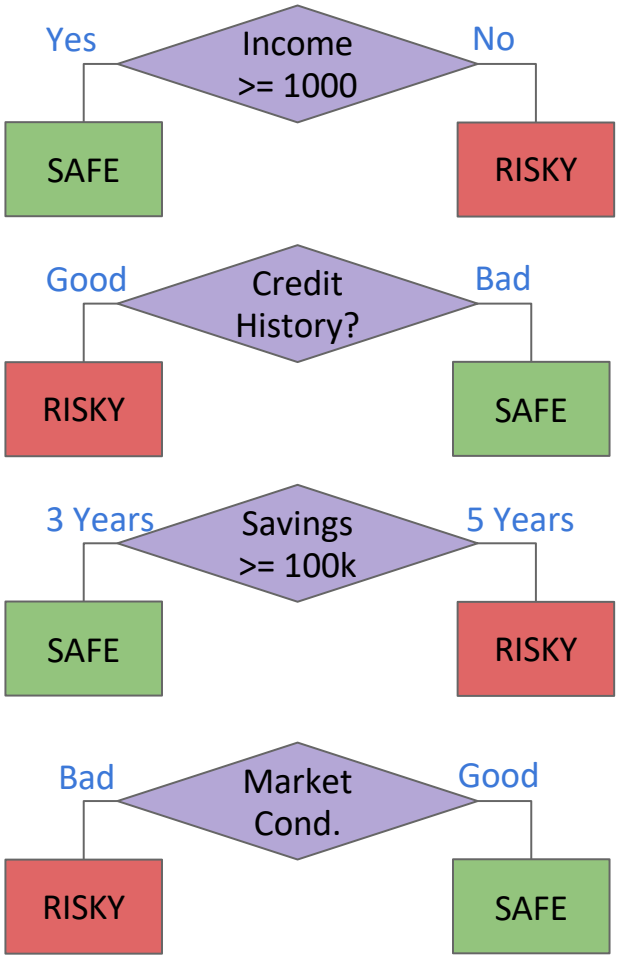
Group

2 min
sli.do #cs416

Recall the prediction rule for **weighted majority vote**.

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

What label will AdaBoost predict with these trees and weights?



$\hat{f}_1(x) = +1$

$\hat{f}_2(x) = -1$

$\hat{f}_3(x) = -1$

$\hat{f}_4(x) = +1$

Weight	Value
$\hat{w}_1$	2
$\hat{w}_2$	-1
$\hat{w}_3$	1.5
$\hat{w}_4$	0

$\hat{F}(x) = sign\left(\sum_{t=1}^{T} \hat{w}_t \hat{f}_t(x)\right)$

$= sign(2 \cdot 1 + (-1) \cdot (-1) + 1.5 \cdot (-1) + 0 \cdot (-1))$

$= sign(1.5)$

$= +1$

Training AdaBoost

With AdaBoost, training is going to look very different.

We train each model **in succession**, where we use the errors of the previous model to affect how we learn the next one.

To do this, we will need to keep track of two types of weights

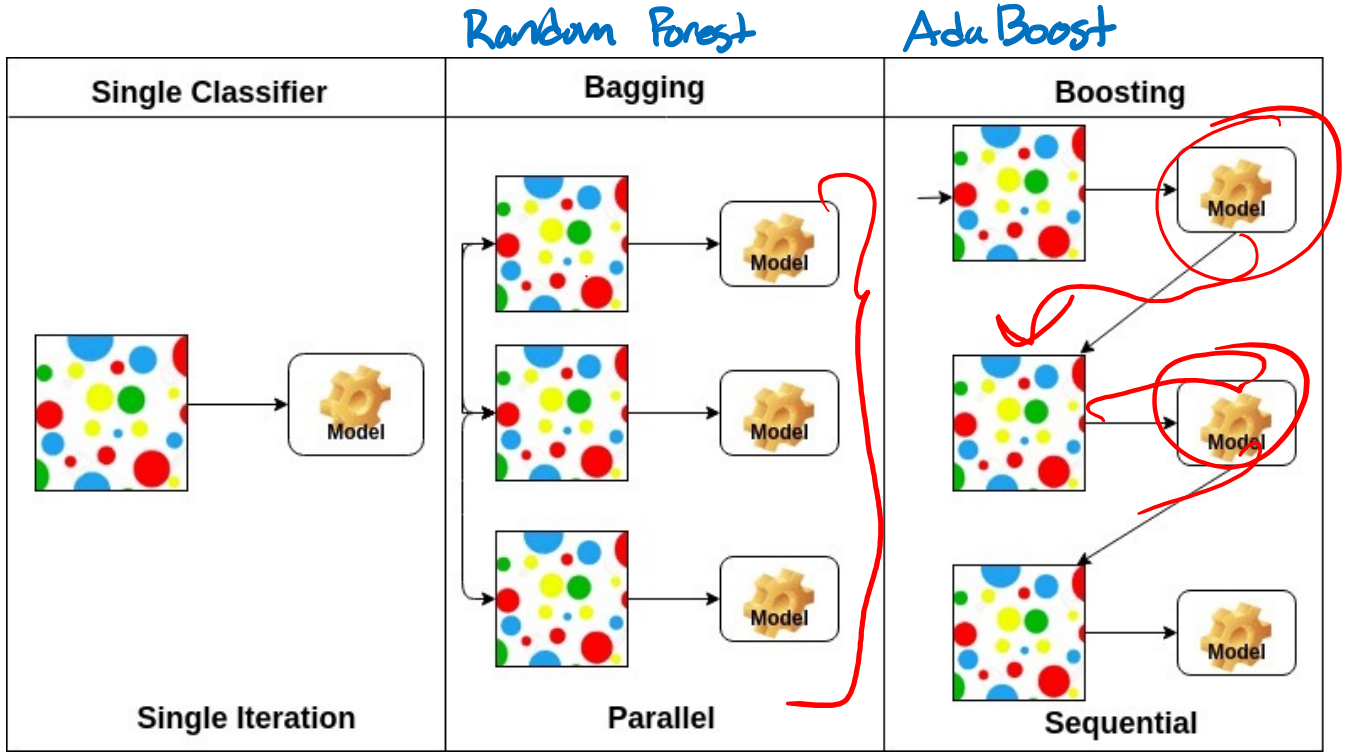
The first are the $\hat{w}_t$ that we will use as the end result to weight each model.

- **Intuition**: An accurate model within the ensemble should have a high weight

We will also introduce a weight α_i for each example in the dataset that we update each time we train a new model

- **Intuition**: We want to put more weight on examples that seem hard to classify correctly

Boosting (AdaBoost) vs. Bagging (Random Forrest)



AdaBoost
Ada Glance

Train

for t in $[1, 2, \ldots, T]$:

- Learn $\hat{f}_t(x)$ based on data weights $\alpha_{i,t}$
- Compute model weight $\hat{w}_t$
- Compute data weights $\alpha_{i,t+1}$

Predict

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

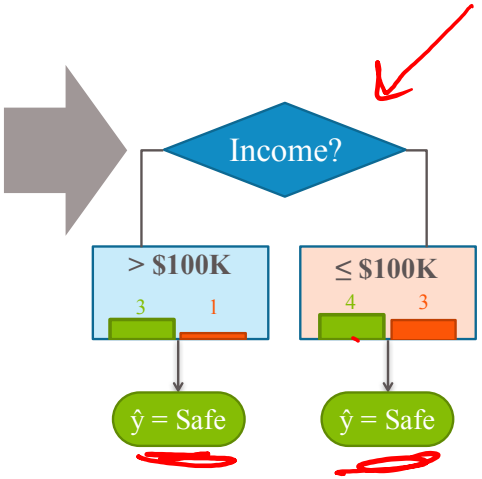
Weighted Data α_i

Start with a dataset and train our first model (a decision stump)

For all the things it gets wrong, increase the weight of that example. For each one that's right, decrease its weight.

– Correct --- Incorrect

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



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

Learning w/ Weighted Data

Before, when we learned decision trees we found the split that minimized classification error.

Now, we want to minimize **weighted classification error**

$$WeightedError(f_t) = \frac{\sum_{i=1}^{n} \alpha_{i,t} \mathbb{I}\{\hat{f}_t(x_i) \neq y_i\}}{\sum_{i=1}^{n} \alpha_{i,t}}$$

Classification Error = $\frac{\#\text{ mistakes}}{\#\text{ examples}}$

vs.

Weighted Error = $\frac{\sum \text{ total weight of mistakes}}{\sum \text{ total weight of all examples}}$

If an example x_2 has weight $\alpha_2 = 3$, this means getting that example wrong is the same as getting 3 examples wrong!

This will most likely change which split is optimal!

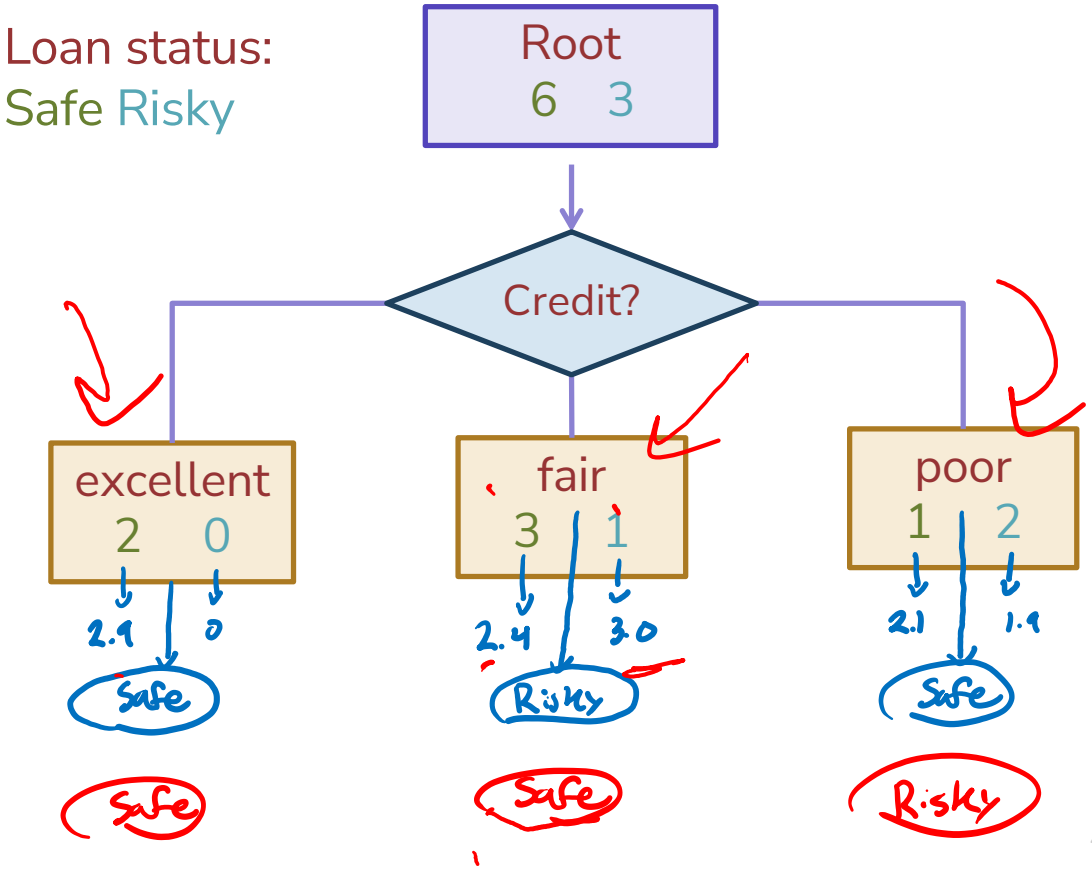
Learning w/ Weighted Data

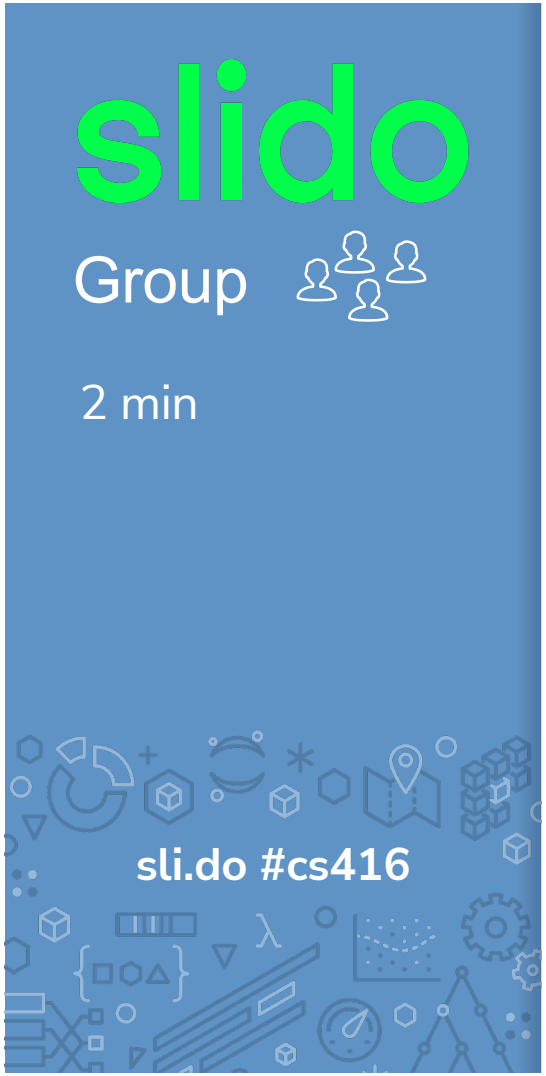
Credit	y	weight
excellent	safe	1.2
fair	risky	3.0
fair	safe	0.5
poor	risky	0.9
excellent	safe	0.9
fair	safe	0.7
poor	risky	1.0
poor	safe	2.1
fair	safe	1.2

Weighted:

Unweighted:

We also set leaf node predictions to be the **class with larger total weight**, not the class with more instances.





Consider the following weighted dataset, what is the weighted classification error of the optimal decision stump (just one split)?

We want to use the TumorSize and IsSmoker to predict if a patient's tumor is malignant.

TumorSize	IsSmoker	Malignant	Weight
Small	No	No	0.5
Small	Yes	Yes	1.2
Large	No	No	0.3
Large	Yes	Yes	0.5
Small	Yes	No	3.3

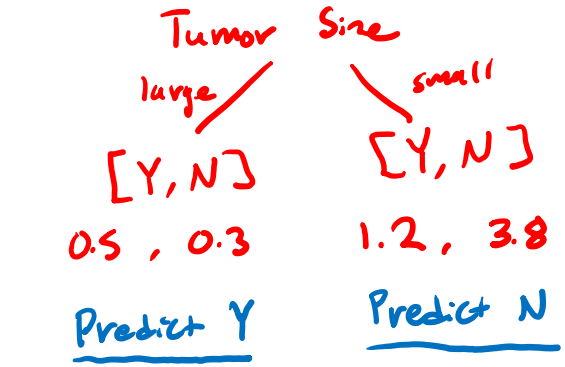
slido

Group

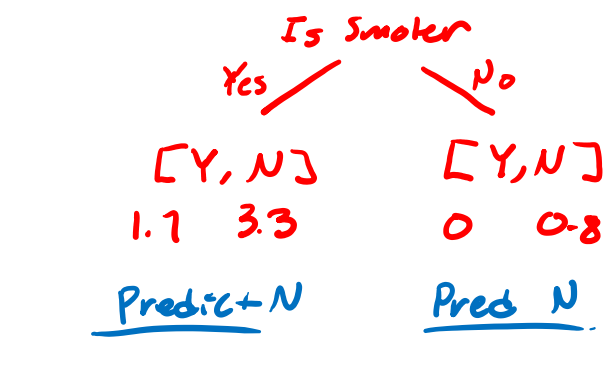
0 min

sli.do #cs416

TumorSize	IsSmoker	Malignant	Weight
Small	No	No	0.5
Small	Yes	Yes	1.2
Large	No	No	0.3
Large	Yes	Yes	0.5
Small	Yes	No	3.3



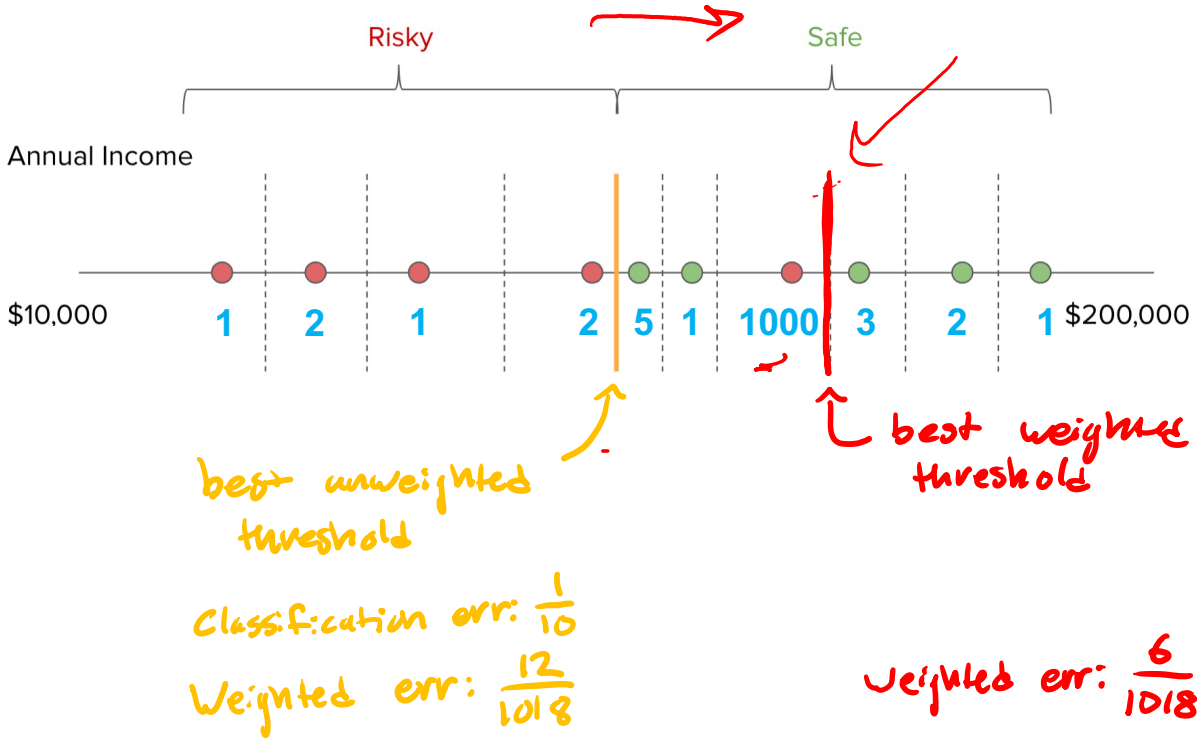
Weighted Error: $\frac{0.3 + 1.2}{5.8} \approx 0.26$



Weighted Error : $\frac{1.7 + 0}{5.8} \approx 0.29$

Real Valued Features

The algorithm is more or less the same, but now we need to account for weights



Brain Break

AdaBoost Ada Glance

Train

for t in $[1, 2, \ldots, T]$:

- Learn $\hat{f}_t(x)$ based on data weights $\alpha_{i,t}$
- Compute model weight $\hat{w}_t$
- Compute data weights $\alpha_{i,t+1}$

Predict

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

Model Weights $\hat{w}_t$

Model weight: $-\infty < \hat{w} < \infty$

Goal: Want to have high weight for models that are very accurate, and low weight for models that are not.

The specific formula used for AdaBoost

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

Great classifier ($WeightedError(\hat{f}_t) = 0.01$)

- $\hat{w}_t = \frac{1}{2}\ln\left(\frac{1-0.01}{0.01}\right) = \frac{1}{2}\ln(99) = 2.3$

Meh classifier ($WeightedError(\hat{f}_t) = 0.5$)

- $\hat{w}_t = \frac{1}{2}\ln\left(\frac{1-0.5}{0.5}\right) = \frac{1}{2}\ln(1) = 0$

Awful classifier ($WeightedError(\hat{f}_t) = 0.99$)

- $\hat{w}_t = \frac{1}{2}\ln\left(\frac{1-0.99}{0.99}\right) = \frac{1}{2}\ln\left(\frac{1}{99}\right) = -2.3$

Computing $\alpha_{i,t+1}$

Goal: Increase the weights of data examples that were hard to classify. If we got it wrong, increase the weight, otherwise decrease it.

start w/ old weight

downweight datapoint i if correctly classified by stump t

$$\alpha_{i,t+1} \leftarrow \begin{cases} \alpha_{i,t} e^{-\hat{w}_t}, & \text{if } \hat{f}_t(x_i) = y_i \\ \alpha_{i,t} e^{\hat{w}_t}, & \text{if } \hat{f}_t(x_i) \neq y_i \end{cases}$$

upweight datapoint i if misclassified by stump t

AdaBoost
Ada Glance

Model Weight →

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

Train

for t in $[1, 2, \ldots, T]$:

- Learn $\hat{f}_t(x)$ based on data weights $\alpha_{i,t}$
- Compute model weight $\hat{w}_t$
- Compute data weights $\alpha_{i,t+1}$

Data weight ↓

$$\alpha_{i,t+1} \leftarrow \begin{cases} \alpha_{i,t} e^{-\hat{w}_t}, & \text{if } \hat{f}_t(x_i) = y_i \\ \alpha_{i,t} e^{\hat{w}_t}, & \text{if } \hat{f}_t(x_i) \neq y_i \end{cases}$$

Predict

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

Normalizing $\alpha_{i,t}$

Generally, the weights for some points get really large/small in magnitude due to how the data is laid out.

Numbers in wildly different scales can often cause problems due to finite precision of computers when it comes to real numbers.

Generally, we normalize the data weights so they sum to 1 to prevent them from getting too small or too big.

$$\alpha_{i,t+1} \leftarrow \frac{\alpha_{i,1+1}}{\sum_{j=1}^{n} \alpha_{j,t+1}}$$

AdaBoost
Ada Glance

$\alpha_{i,1} = \frac{1}{n}$

Train

for t in $[1, 2, \ldots, T]$:

1. - Learn $\hat{f}_t(x)$ based on data weights $\alpha_{i,t}$
2. - Compute model weight $\hat{w}_t$
3. - Compute data weights $\alpha_{i,t+1}$

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

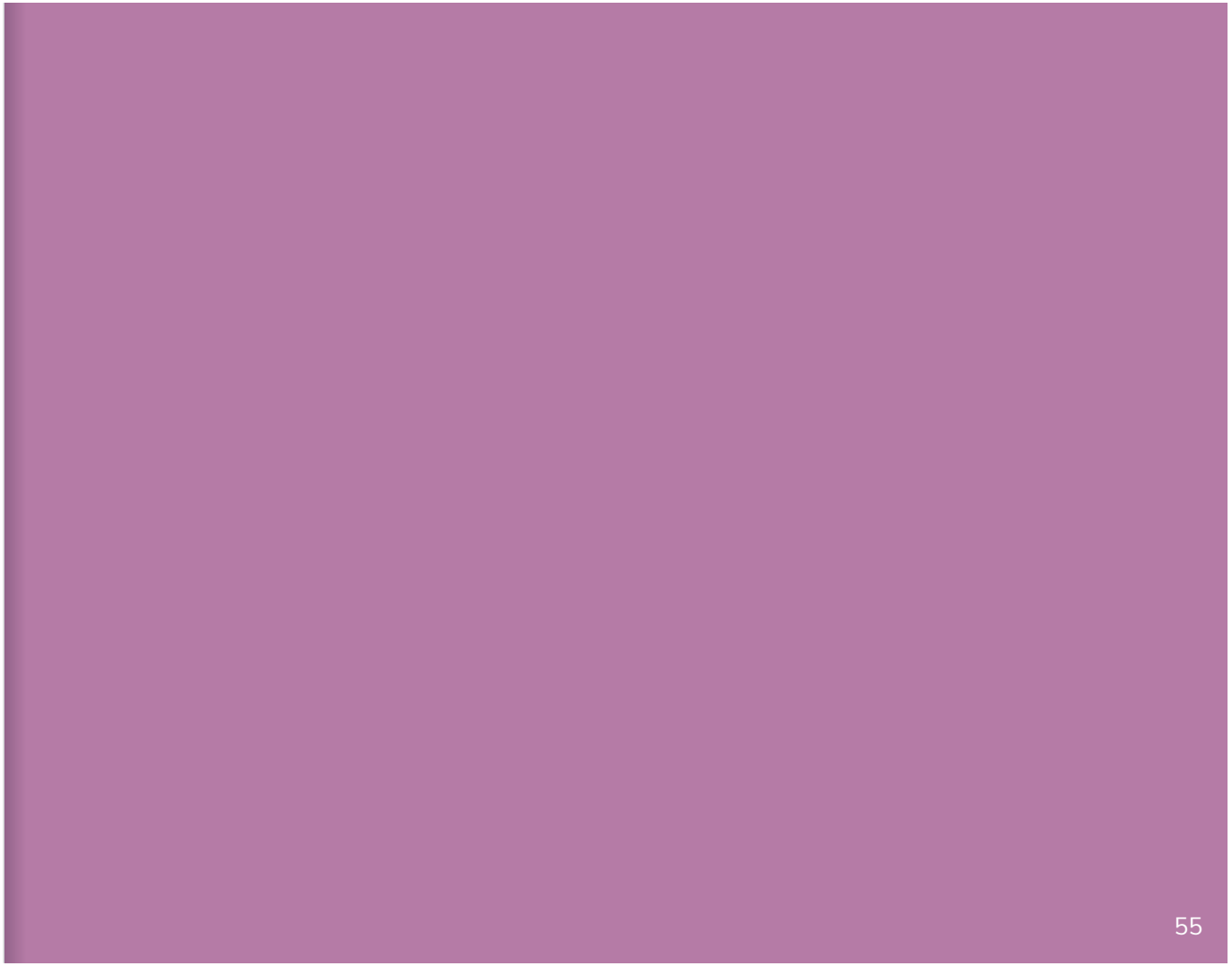
3a. $$\alpha_{i,t+1} \leftarrow \begin{cases} \alpha_{i,t}e^{-\hat{w}_t}, & \text{if } \hat{f}_t(x_i) = y_i \\ \alpha_{i,t}e^{\hat{w}_t}, & \text{if } \hat{f}_t(x_i) \neq y_i \end{cases}$$

3b. $$\alpha_{i,t+1} \leftarrow \frac{\alpha_{i,t+1}}{\sum_{j=1}^{n} \alpha_{j,t+1}}$$

Predict

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

Visualizing AdaBoost



$t = 1$ Learn a Classifier

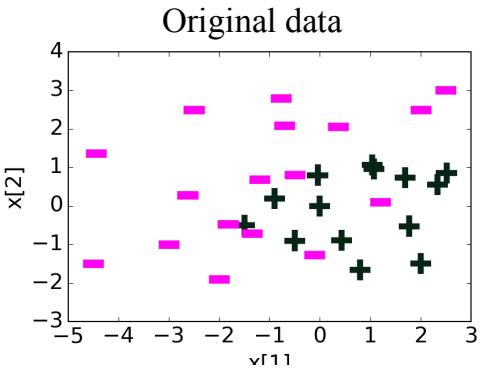
Start with all data having same weight $\alpha_{i,1} = 1/n$

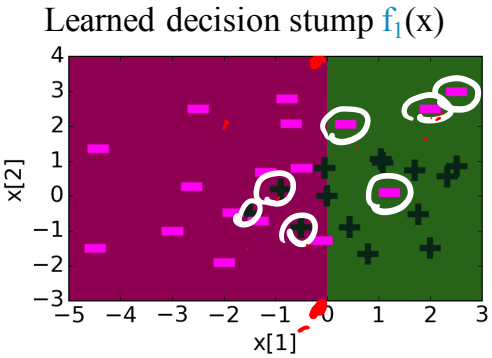
Learn a decision stump that minimizes weighted error

With all the same weights, this is the same as before!

Visualization: size of point i correlated w/ $\alpha_{i,t}$

$\hat{f}_1(x) = \cdots$





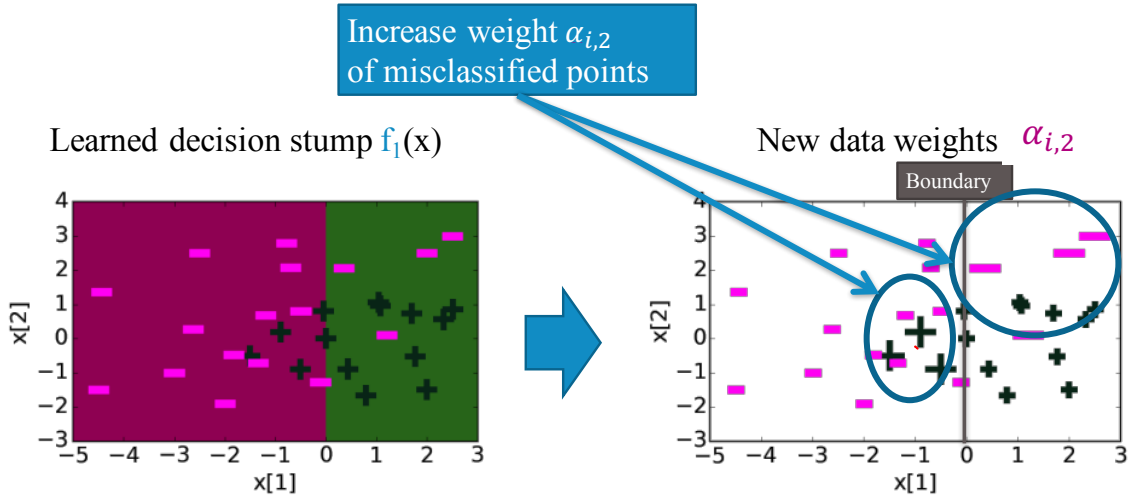
Calculate $\hat{w}_1 \approx 0.61$

7 points misclassified

$t = 1$ Update Data Weights

Compute new weights $\alpha_{i,2}$ based on the errors of $\hat{f}_1$

The points with more weight are drawn larger



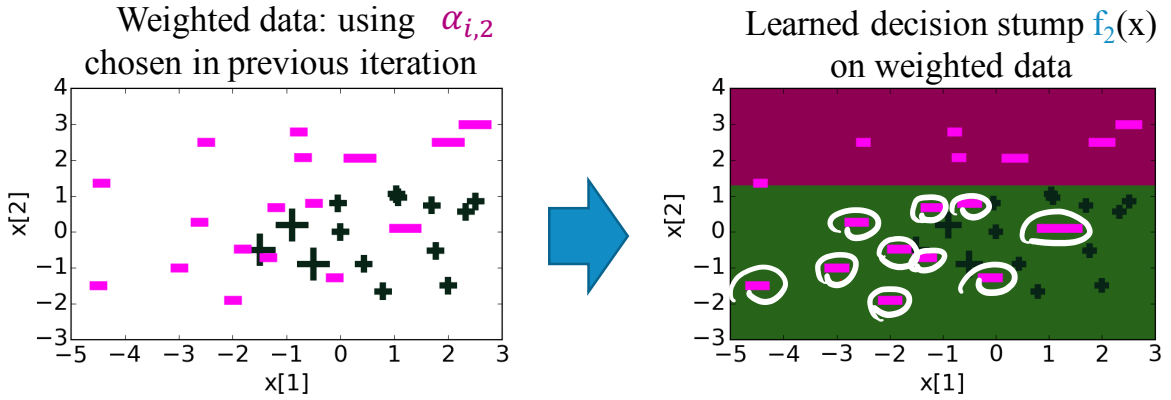
$t = 2$ Learn a Classifier

Now use new weights to learn best stump that minimizes weighted classification error.

$\hat{f}_2(x) = \cdots$

Calculate $\hat{w}_2 \approx 0.53$

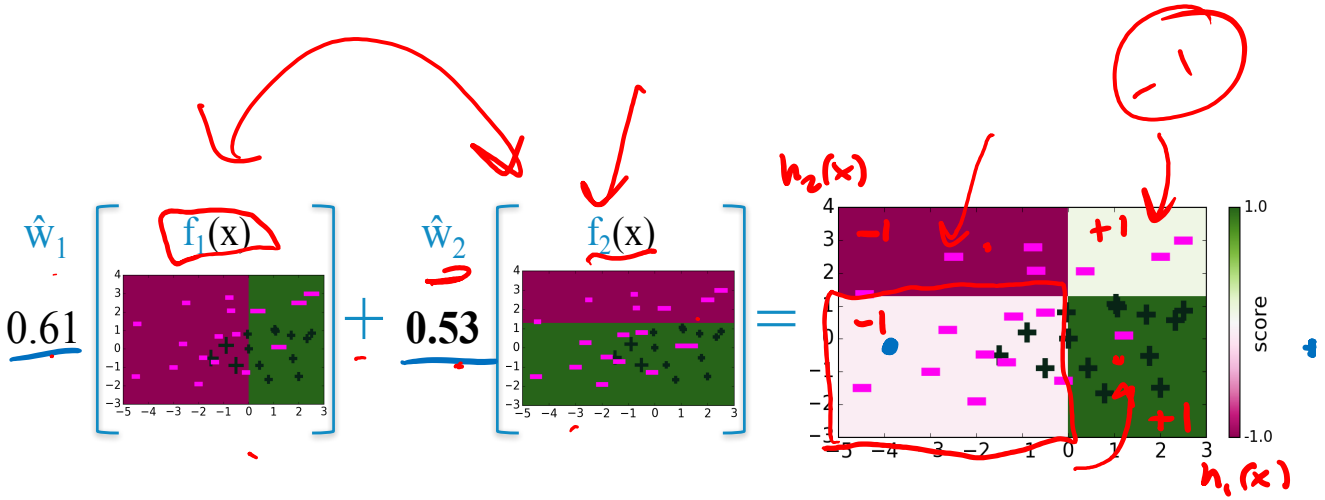
Then update weights based on errors.



Made more errors, but less weighted error

AdaBoost Ensemble

If we plot what the predictions would be for each point, we get something that looks like this:



$sign(0.61 \cdot (-1) + 0.53 \cdot (+1))$
$= sign(-0.08) = -1$

slido

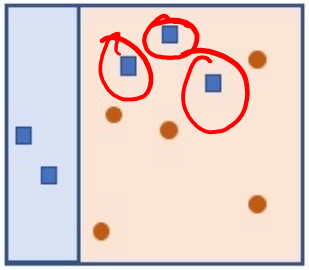
Group

1 min

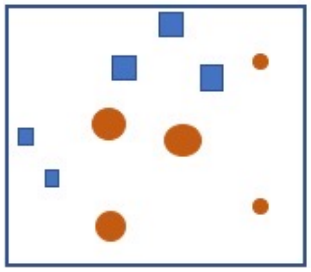
sli.do #cs416

$\alpha_{i,1} = 1/n$

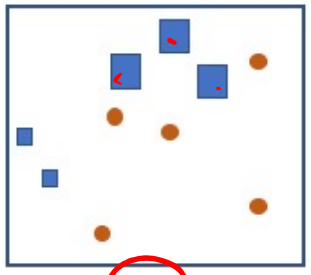
Say AdaBoost learned the below classifier at time $t = 1$.



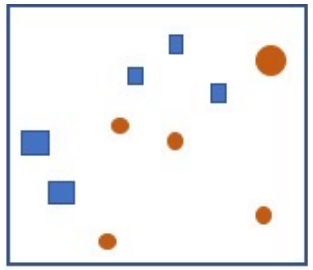
Which of the following images represent the reweighted points for time $t = 2$?



(A)



(B)



(C)

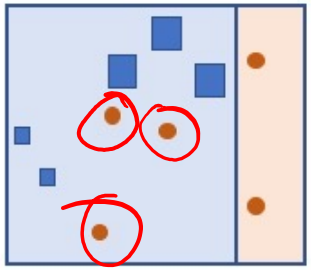
slido

Group

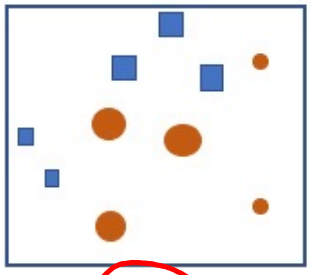
1 min

sli.do #cs416

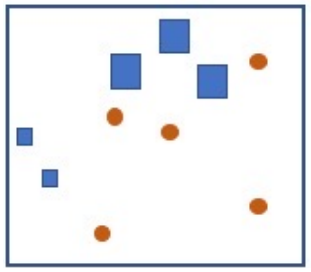
Say AdaBoost learned the below classifier at time $t = 2$.



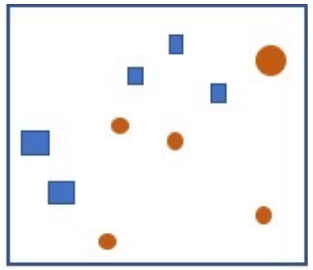
Which of the following images represent the reweighted points for time $t = 3$?



(A)



(B)



(C)

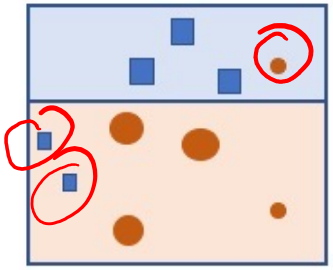
slido

Group

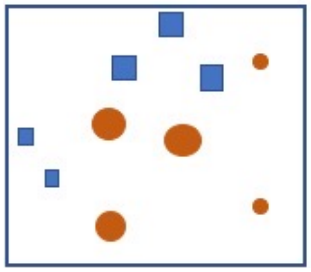
30 sec

sli.do #cs416

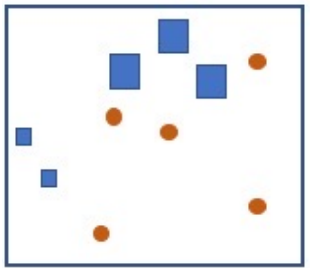
Say AdaBoost learned the below classifier at time $t = 3$.



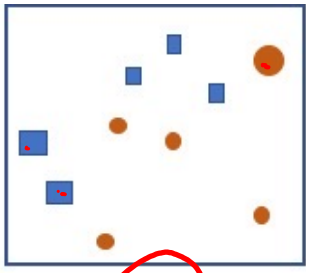
Which of the following images represent the reweighted points for time $t = 4$?



(A)



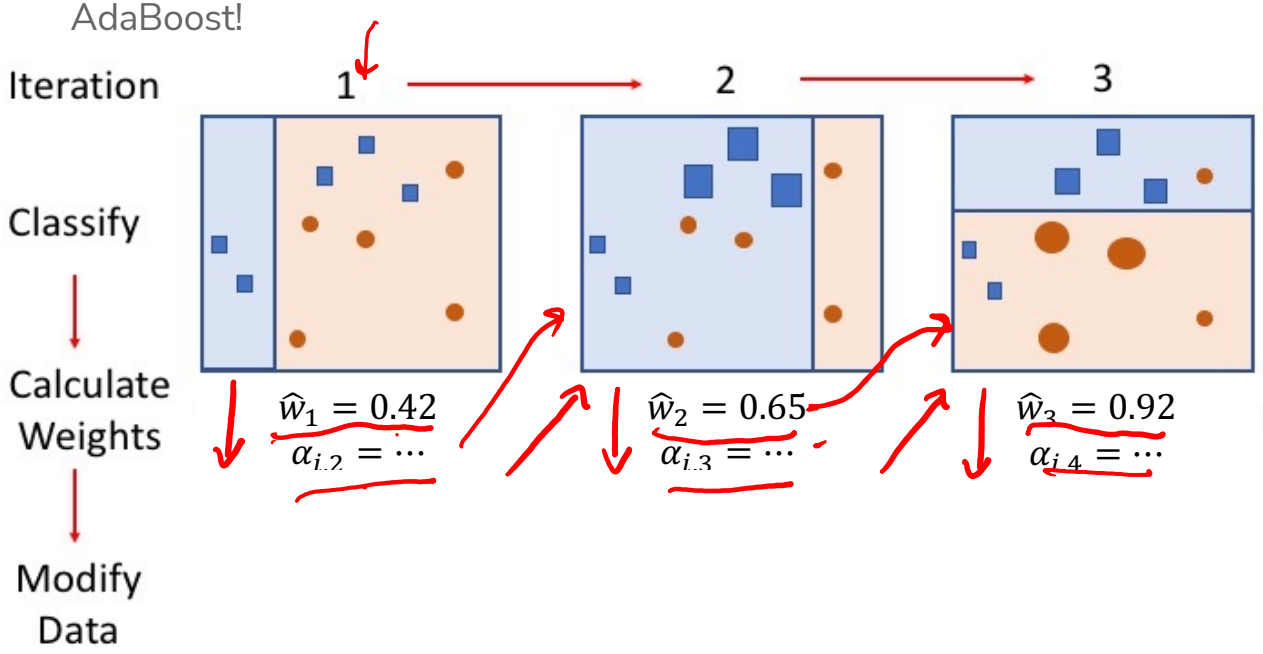
(B)



(C)

AdaBoost Example

You have now worked through a complete example of training AdaBoost!



Source: A Tutorial on Boosting (Freund and Schapire)

What about predicting?

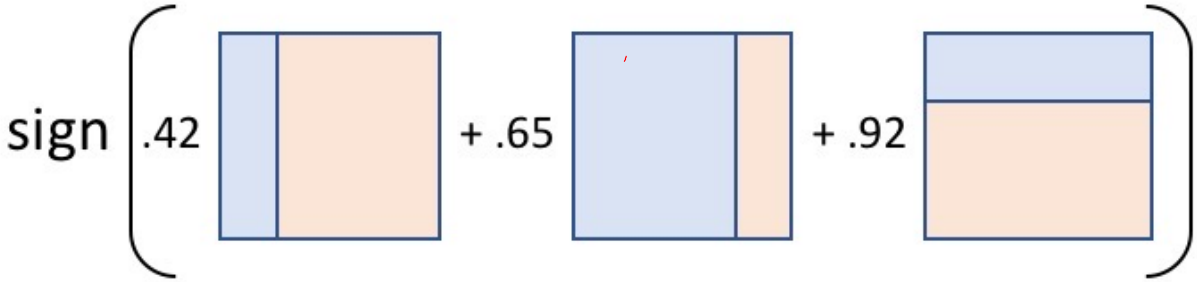
slido

Think

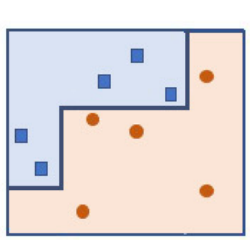
1 min

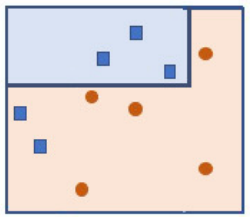
sli.do #cs416

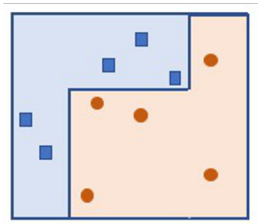
Consider the following ensemble and weights from the AdaBoost example we've been working through.

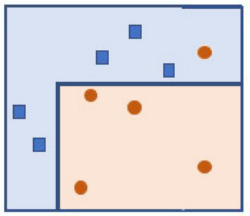


Which of the following is the final decision boundary?


(A)


(B)


(C)


(D)

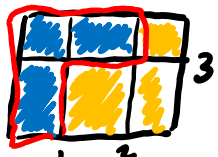
slido

Group

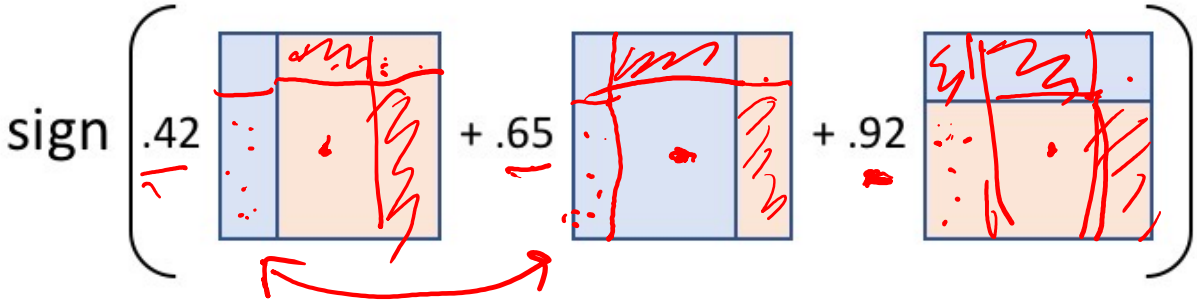
2 min

sli.do #cs416

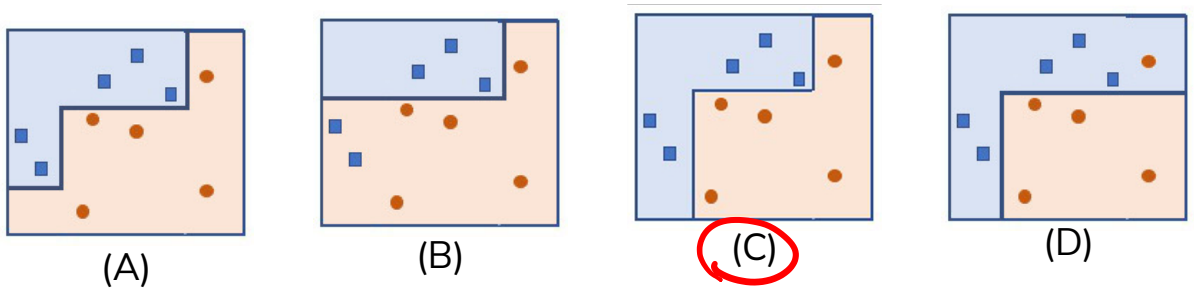
Overlap Decision Boundries

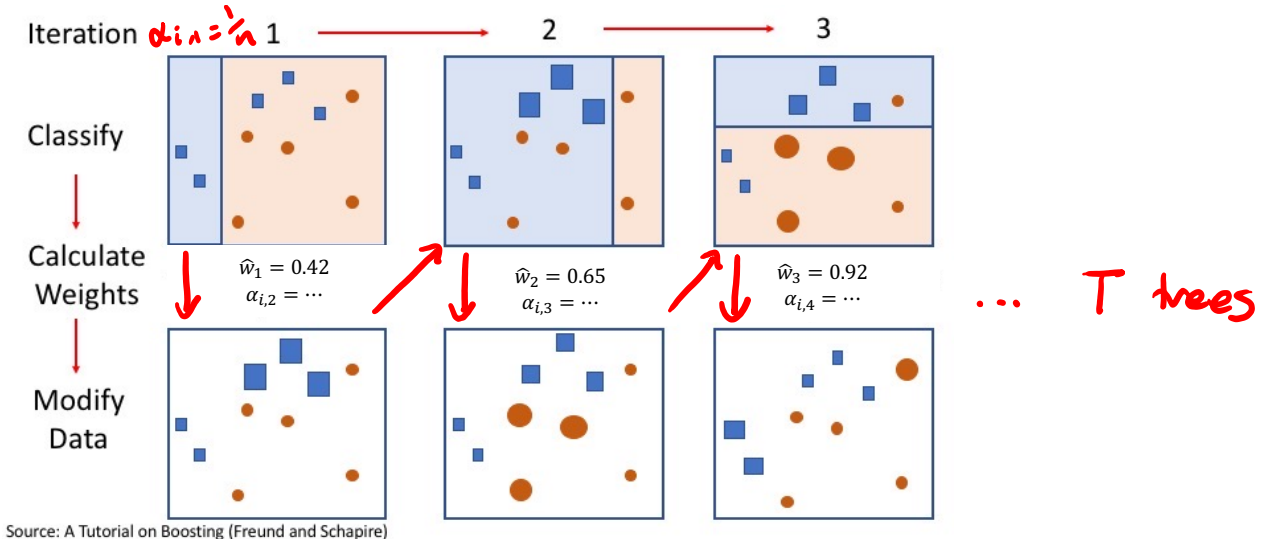


Consider the following ensemble and weights from the AdaBoost example we've been working through.

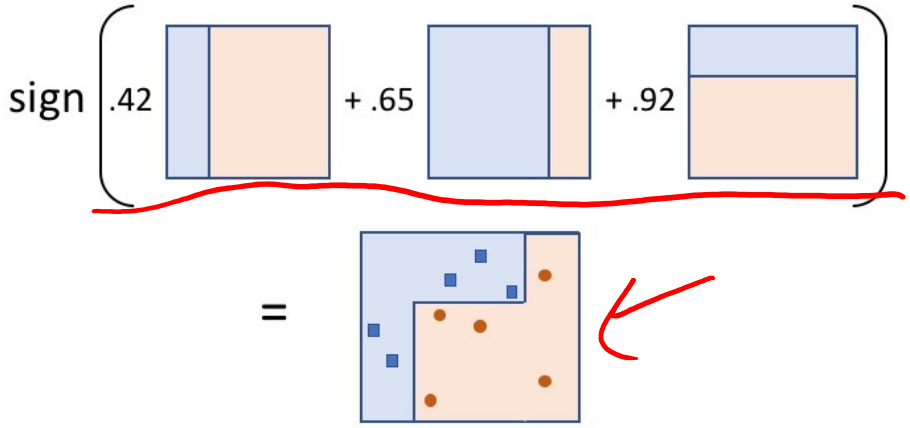


Which of the following is the final decision boundary?





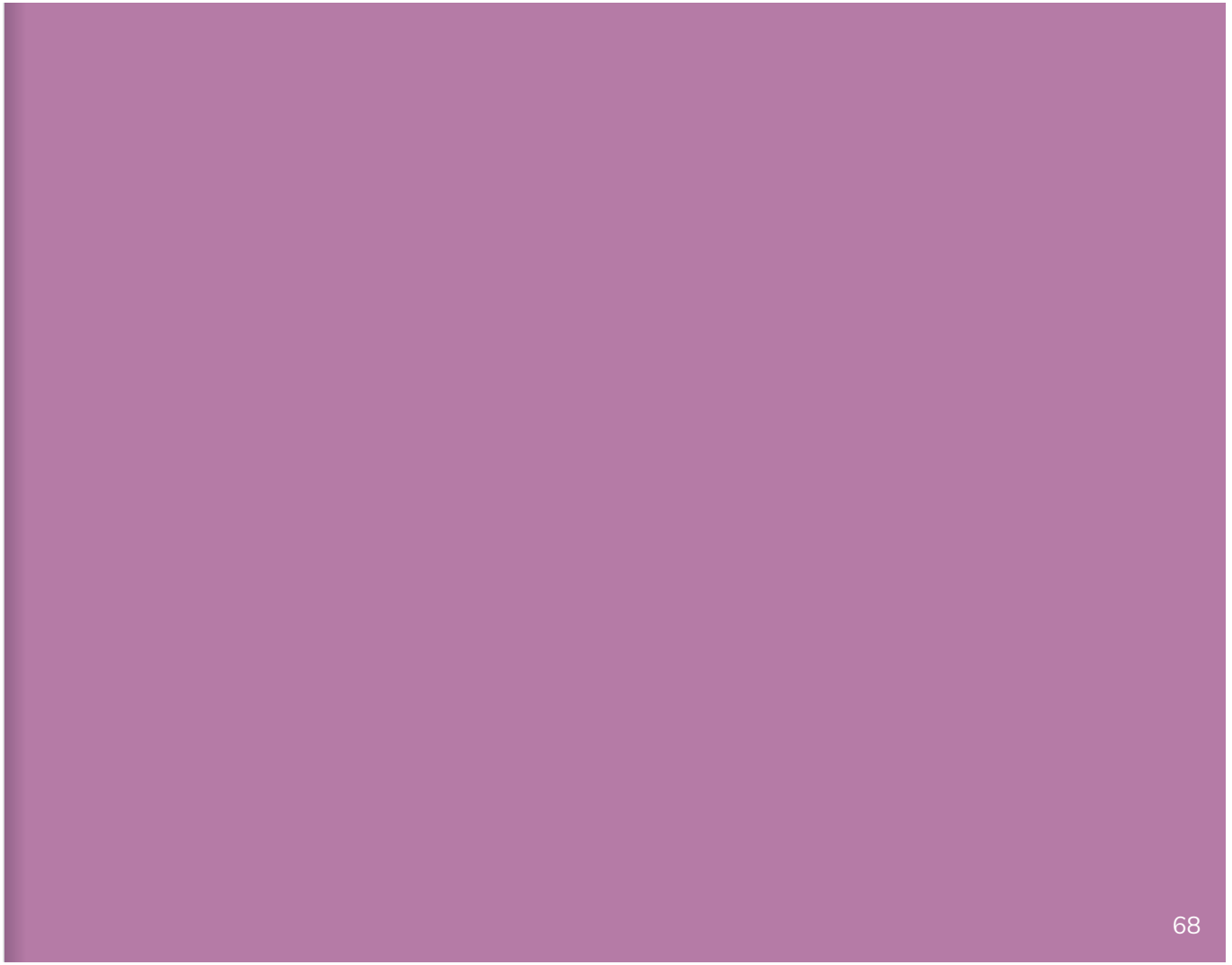
Source: A Tutorial on Boosting (Freund and Schapire)



Source: A Tutorial on Boosting (Freund and Schapire)

Brain Break

AdaBoost Overfitting



AdaBoost
Ada Glance

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

Train

for t in $[1, 2, \ldots, T]$:

- Learn $\hat{f}_t(x)$ based on data weights $\alpha_{i,t}$
- Compute model weight $\widehat{w}_t$
- Compute data weights $\alpha_{i,t+1}$

$$\alpha_{i,t+1} \leftarrow \begin{cases} \alpha_{i,t} e^{-\widehat{w}_t}, & \text{if } \hat{f}_t(x_i) = y_i \\ \alpha_{i,t} e^{\widehat{w}_t}, & \text{if } \hat{f}_t(x_i) \neq y_i \end{cases}$$

$$\alpha_{i,t+!} \leftarrow \frac{\alpha_{i,t+1}}{\sum_{j=1}^{n} \alpha_{j,t+1}}$$

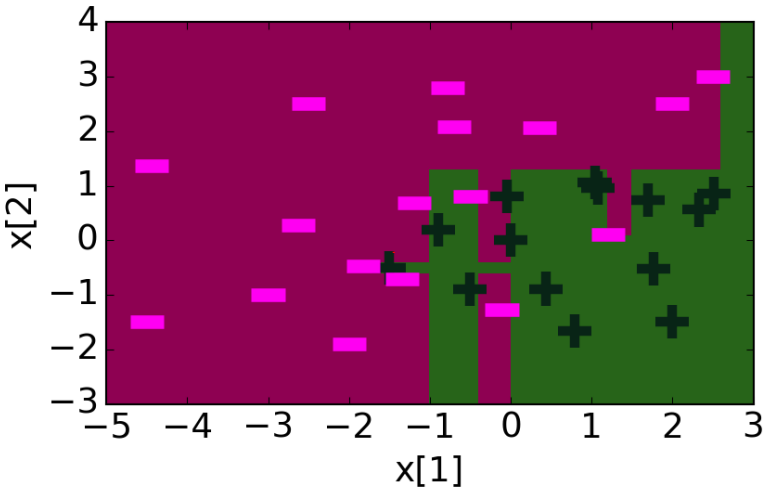
Predict

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

AdaBoost when $t = 30$

Can eventually get 0 training error with a set of weak learners!

This is most likely overfit

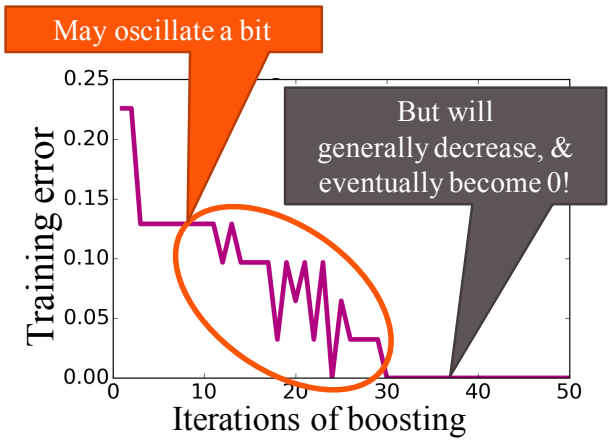


training_error = 0

AdaBoost Theorem

Under some technical conditions…

Training error of boosted classifier → 0 as T→∞

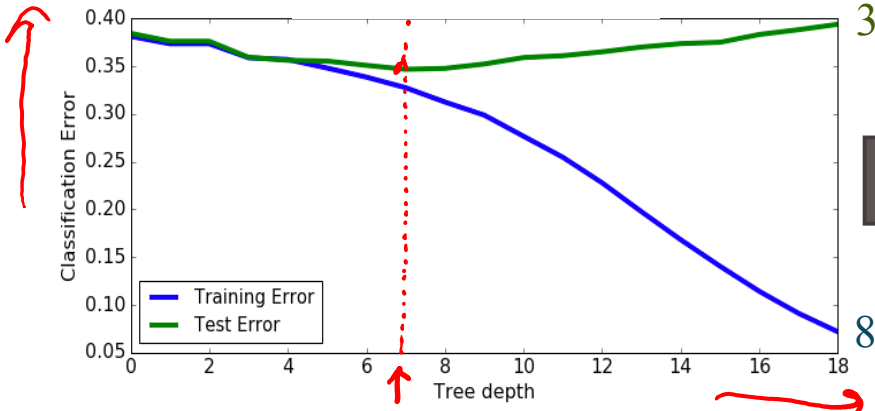


Technical condition: The weak learner can do at least slightly better than complete random guessing

Compare

* don't choose based on test

Decision Tree



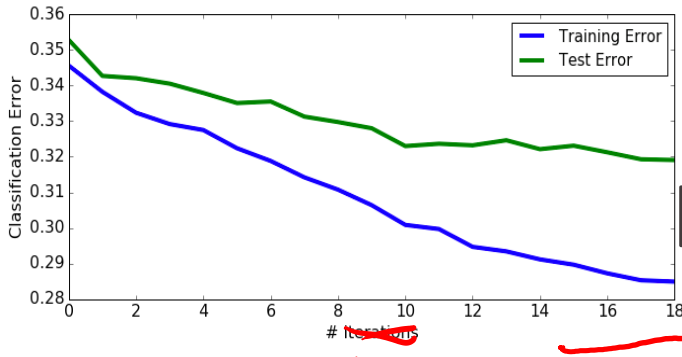
optimal* depth

39% test error

Overfitting

8% training error

AdaBoost



stumps

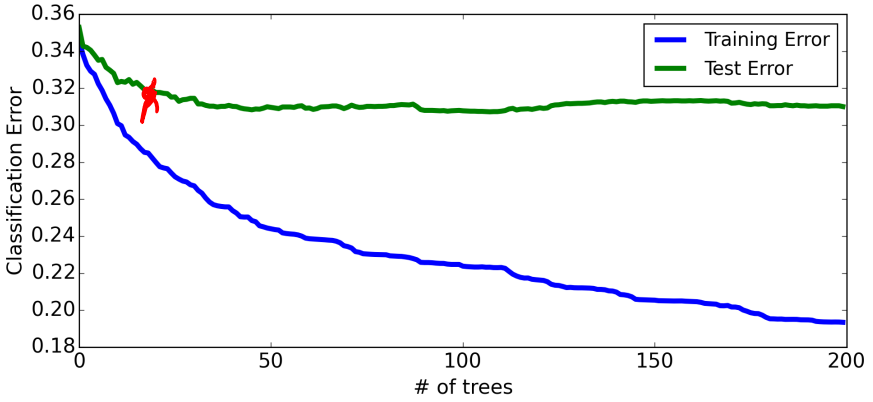
32% test error

Better fit & lower test error

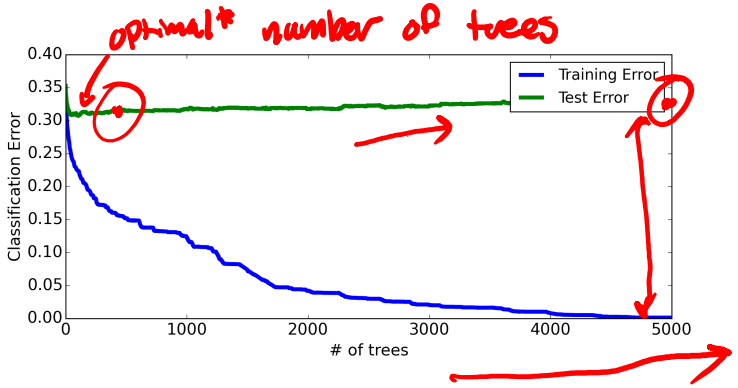
28.5% training error

Overfitting?

Boosting tends to be robust to overfitting



But will eventually overfit



Choose T?

How do you end up choosing the number of trees T for boosting?

Like always

- Find T that minimizes validation error
- Do cross validation

You **can't**

- Find T that minimizes training error
- Find T that minimizes test error

Application

Boosting, AdaBoost and other variants like gradient boosting, are some of the most successful models to date.

They are extremely useful in computer vision

- The standard for face detection

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

Most industry ML systems use a model ensembles

- Some with boosting, some with bagging
- Many times just use 6 different types of models and hand specify their weights.

AdaBoost Overview

Powerful! One of the most powerful set of models for many real world datasets.

- Typically does better than random forest with the same number of trees.

Higher Maintenance: You do have to tune hyper-parameters

- AdaBoost: Number of trees is technically important, but the model tends to be robust to overfitting in practice.
- Gradient Boosting: MANY hyper-parameters (all important)

Expensive: Boosting is inherently sequential which means its slow to learn ensembles with many trees.

- Can be made faster with optimized software like XGBoost (UW)

Recap

Theme: Compare two different ways of making ensembles

Ideas:

- Describe what an ensemble model is
- Explain what a random forest is and why adding trees improves accuracy.
- Formalize how AdaBoost combines weighted votes from simple classifiers (weak learners) and how those classifiers are learned.
- Compare/contrast bagging and boosting.
- Describe the steps of the AdaBoost algorithm.