CSE/STAT 416

Ensemble Methods

Hunter Schafer
University of Washington
July 16, 2019

Roadmap So Far

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

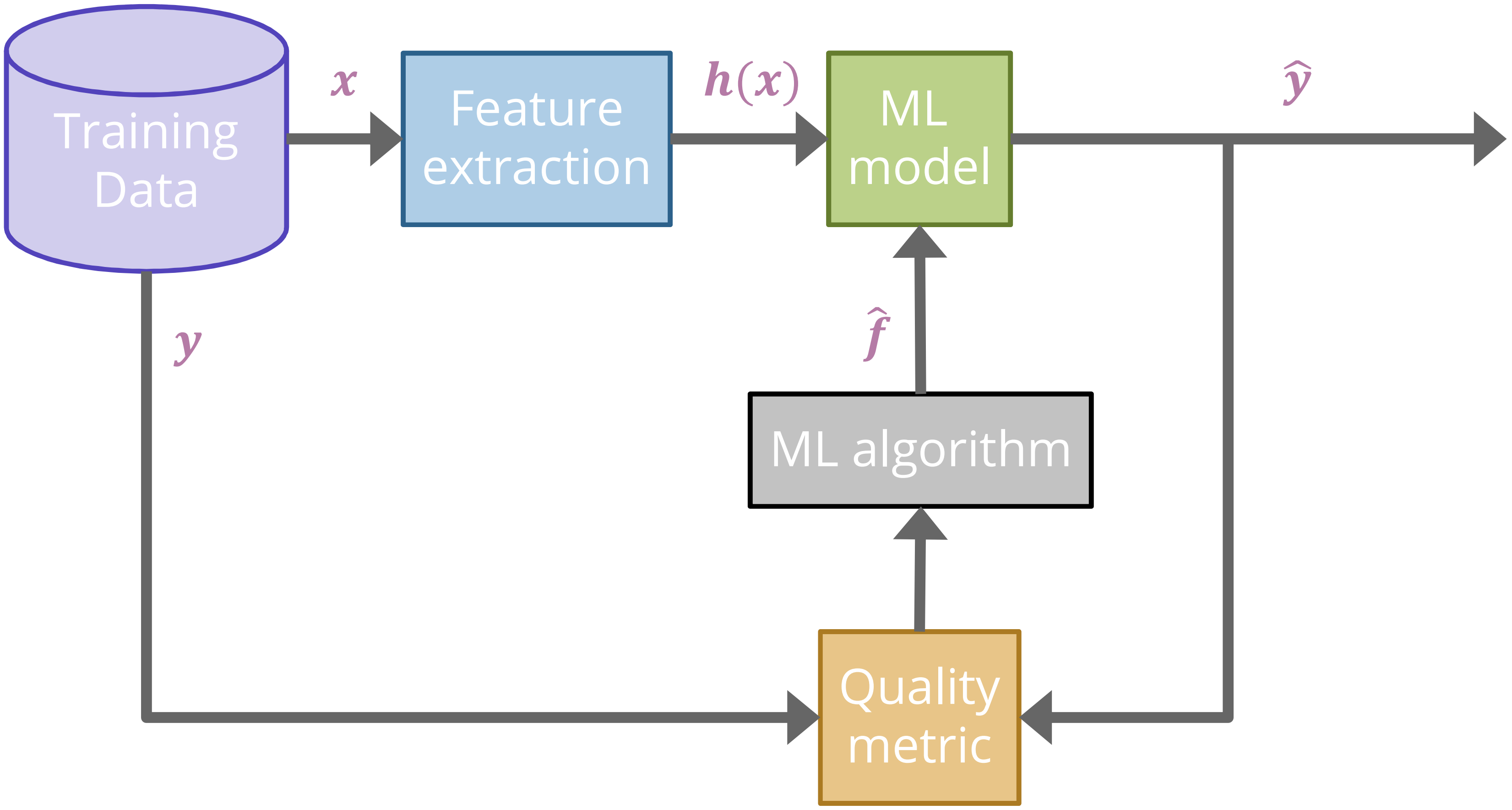
Reflection Notes

Everyone had a good recap of the week and good discussion about how this material relates to previous material

- Forming these connections is incredibly important!
- Reflecting on your own knowledge is one of the most important skills you can learn for mastering a subject. This course is **dense** and requires you to make these connections so you can better understand it

A note about practice / HW

- Concept assignments assess your conceptual knowledge of the topics we have been learning
- Programming assignments assess your ability to apply your conceptual knowledge to solve real world problems.
  - ML is a field where you really aren’t able to do applications without understand the concepts


Training Data
x
Feature extraction
$h(x)$
ML model
$\hat{y}$
y
$\hat{f}$
ML algorithm
Quality metric

Reflection Questions

There were a TON of really good questions that were asked on the reflections. Unfortunately, the paper medium makes it difficult for us to respond to your questions.

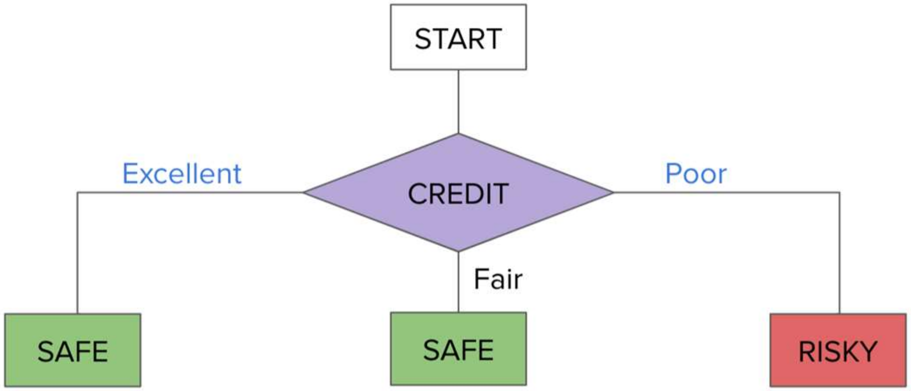
You should use Piazza to ask these questions so staff and your colleagues can respond!

- Try to avoid asking questions that are just asking for answers to concept questions on the HW.
- If it's about a topic on the concept homework, try to make the question about a specific thing you weren't clear about from lecture rather than asking for the answer to the question.

You can always ask questions privately if you're unsure!

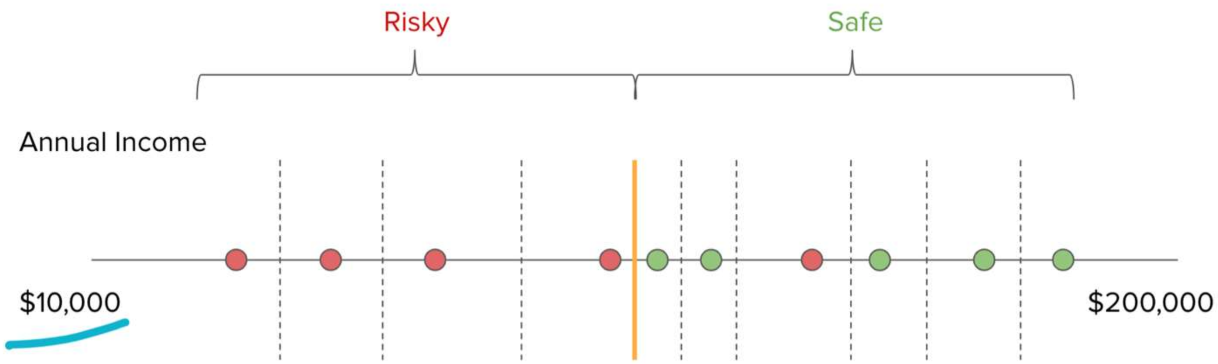
Recap

Decision Tree



- Internal Node: A node that tests a feature
- Branch: Splits input data based on the value of a feature
- Leaf: Assigns a class to data (i.e. SAFE, RISKY)

Real Valued Features



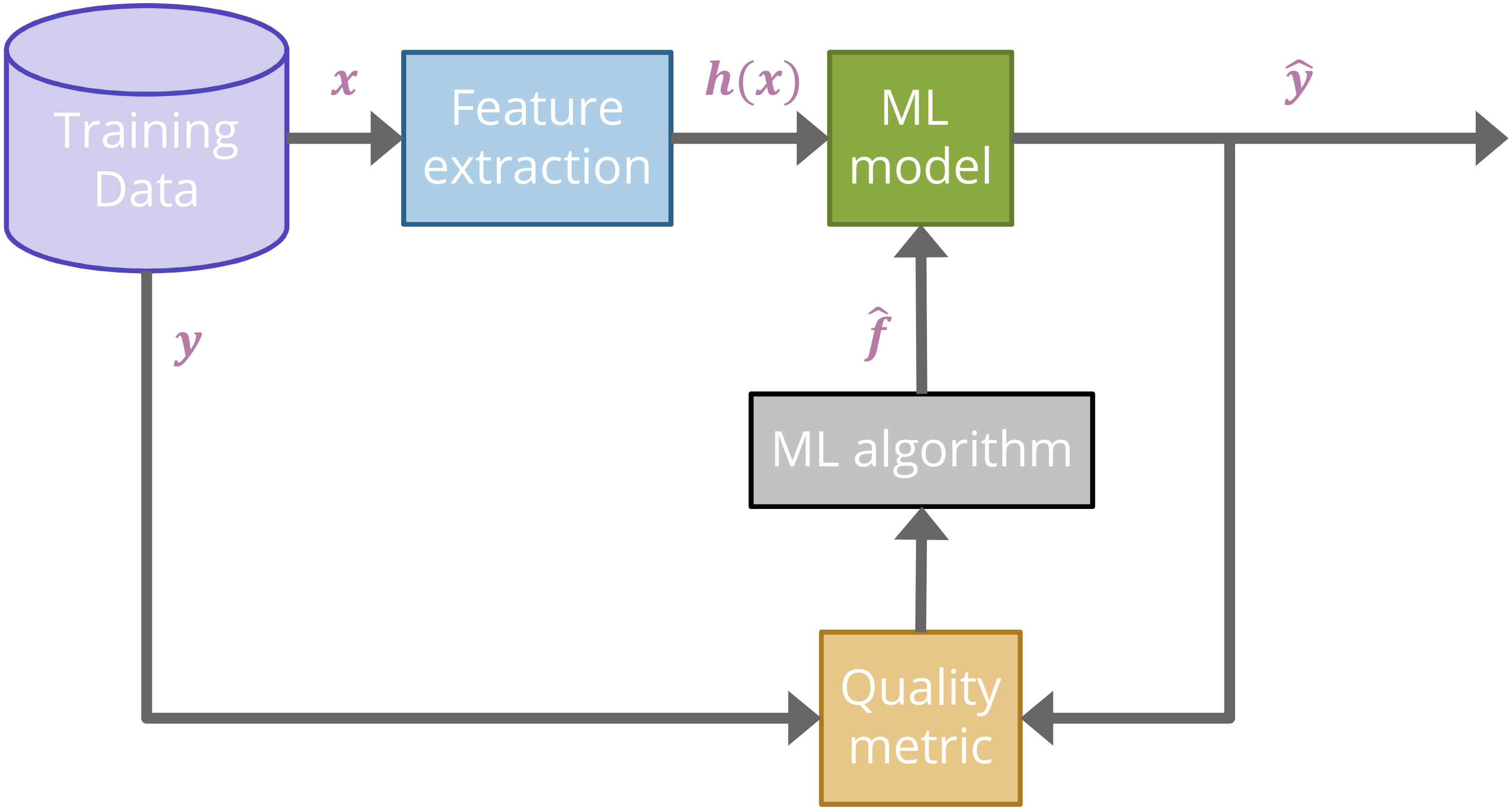
Which split is best? Pick the one that maximizes accuracy.

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


Training Data
x
Feature extraction
$h(x)$
ML model
$\hat{y}$
y
$\hat{f}$
ML 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

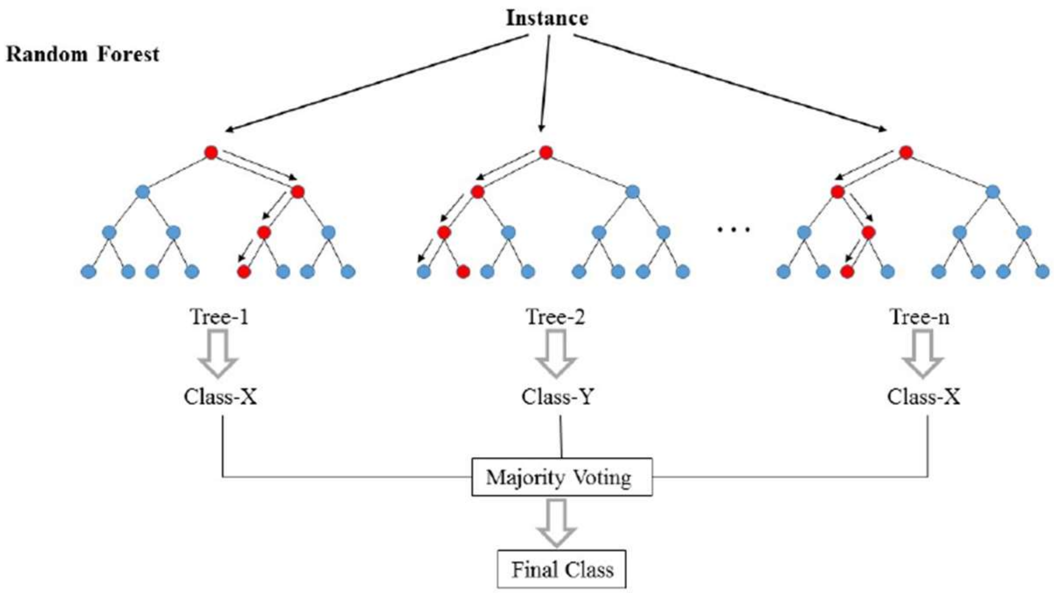
- Random Forest (Bagging)
- AdaBoost (Boosting)

Random Forest

Bagging

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.

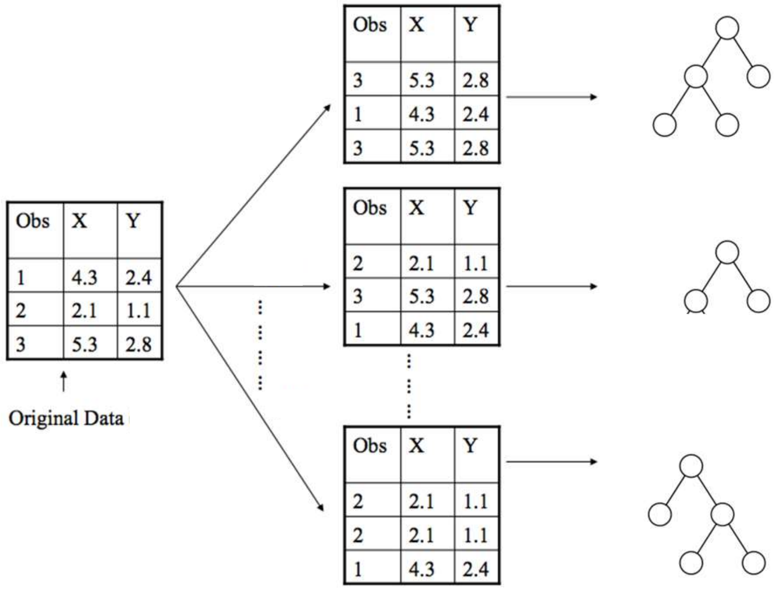


Class - X

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*.



Original Data

Obs	X	Y
1	4.3	2.4
2	2.1	1.1
3	5.3	2.8

Obs	X	Y
3	5.3	2.8
1	4.3	2.4
3	5.3	2.8

Obs	X	Y
2	2.1	1.1
3	5.3	2.8
1	4.3	2.4

Obs	X	Y
2	2.1	1.1
2	2.1	1.1
1	4.3	2.4

Technically, you also randomly select features too!

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 crazy at first, but we are trying to take advantage of what it means to have a high variance model (low bias).
- Remember tnat 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

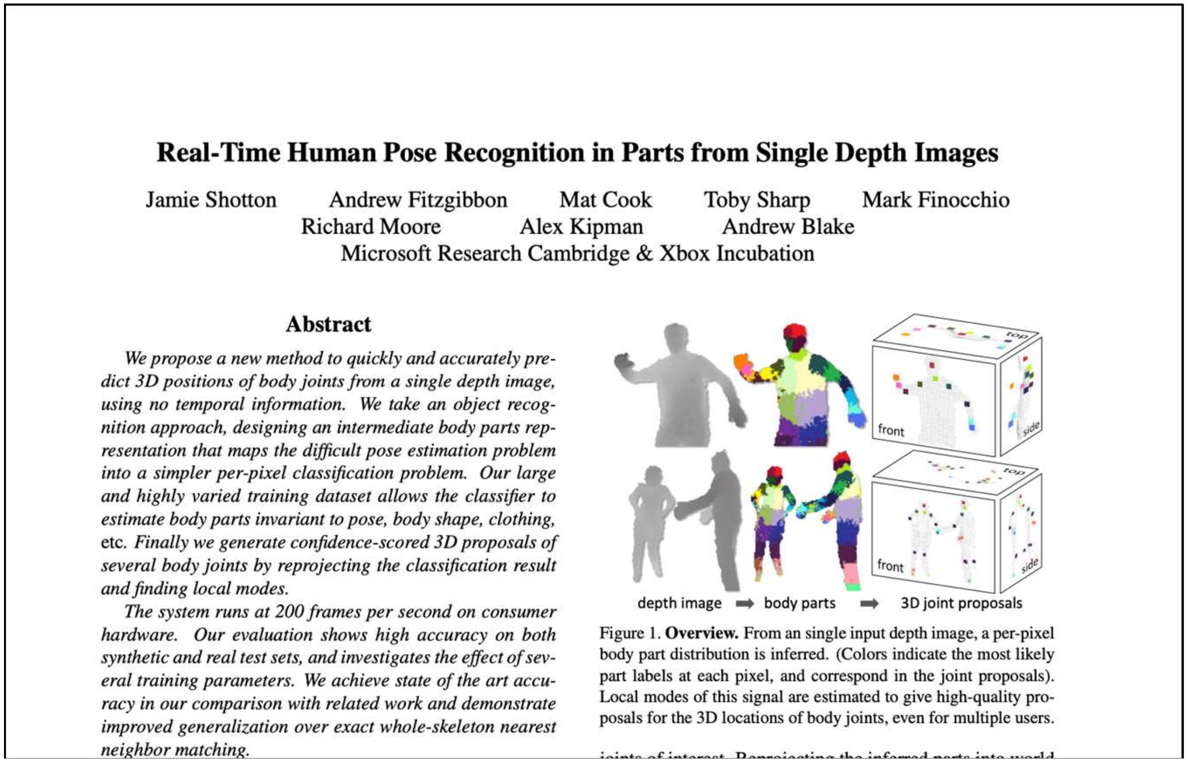
- Make T random samples of the training data that are the same size as the training data but are sampled with replacement
- Train a really tall tree on each sampled dataset (overfit)

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.

joints of interest. Reprojecting the inferred parts into world

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!

Adding trees reduces variance of the ensemble

Brain Break

10:25

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!”

Overview

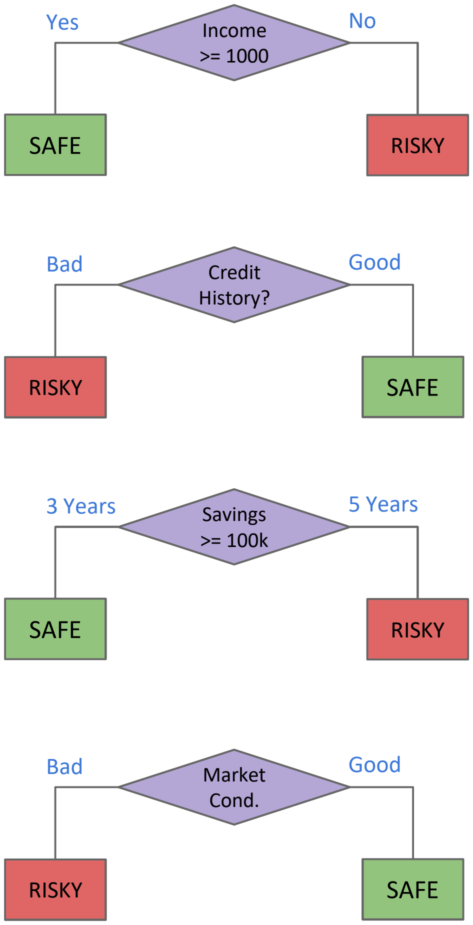
AdaBoost is a model similar to Random Forest (an ensemble of decision trees) with two notable differences that impact how we train it quite severely.

- Instead of using high depth trees that will overfit, we limit ourselves to **decision stumps**.
- Each model in the ensemble gets a weight associated to it, and we take a weighted majority vote

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

$y \in \{-1,1\}$

Example



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

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

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

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

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

$$\hat{y} = \text{sign}\left(\sum_{t=1}^{4} \hat{w}_t \hat{f}_t(x)\right)$$

$$= \text{sign}\Big((2)(+1) + (-1)(-1) + (1.5)(-1) + (0)(1) \Big) = \text{sign}(1.5) = +1$$

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 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

AdaBoost
Ada Glance

Train

```
for $t$ in $[1, 2, \ldots, T]$:
    -   Learn $\hat{f}_t(x)$ based on weights $\alpha_i$
    -   Compute model weight $\widehat{w}_t$
    -   Recompute weights $\alpha_i$
```

Predict

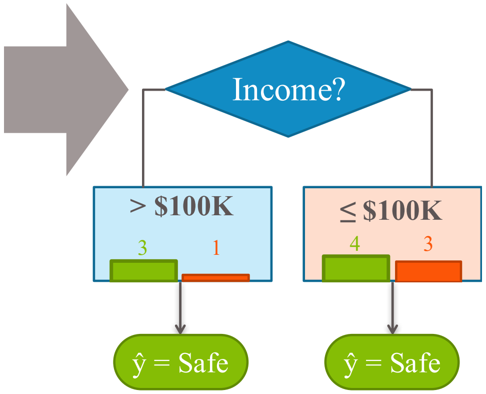
$$\hat{y} = \hat{F}(x) = sign\left(\sum_{t=1}^{T} \widehat{w}_t \hat{f}_t(x) \right)$$

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.

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 was find the split that minimized classification error.

Now, we want to minimize weighted classification error

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

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!

Poll Everywhere

Think

2 min

pollev.com/cse416

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

Poll Everywhere

Pair

3 min

pollev.com/cse416

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

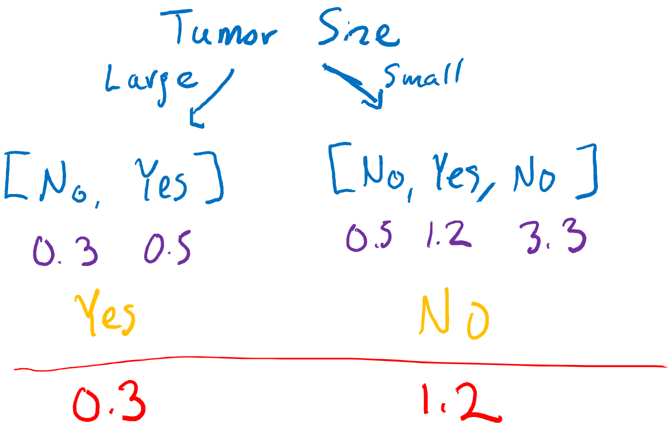
Poll Everywhere

Think

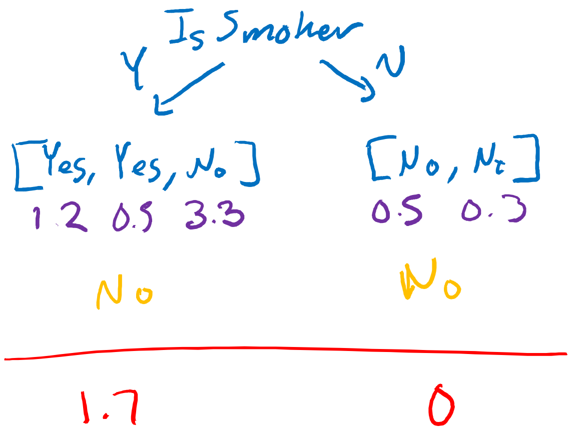
3 min

pollev.com/cse416

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



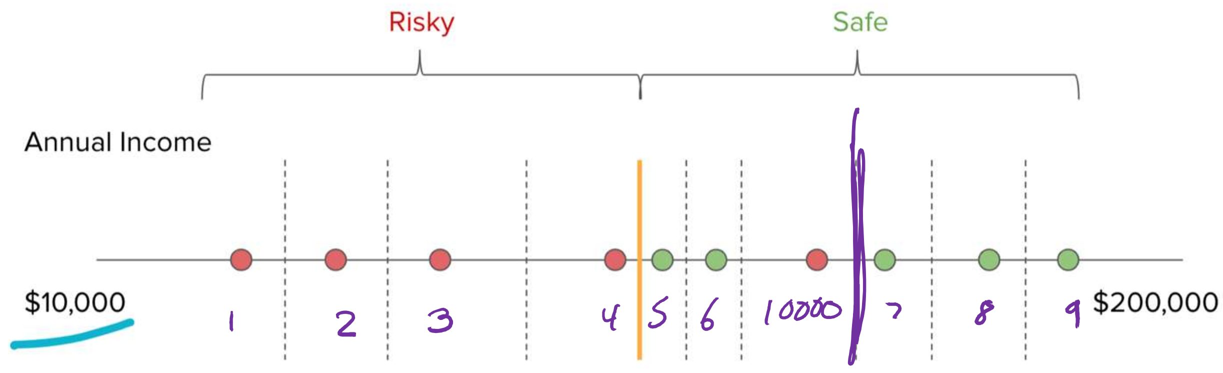
WeightedError= 1.5 ☆



WeightedError = 1.7

Real Valued Features

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



Which split is best? Pick the one that maximizes accuracy.

AdaBoost
Ada Glance

Train

```
for $t$ in $[1, 2, \dots, T]$:
    - Learn $\hat{f}_t(x)$ based on weights $\alpha_i$
    - Compute model weight $\hat{w}_t$
    - Recompute weights $\alpha_i$
```

put more weight on accurate models

put more weight on hard to classify points

Predict

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

Model Weights $\hat{w}_t$

Weighted Error (f_t) 0 and 1

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

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

$WE(\hat{f}_t) = 0.01$

$$\frac{1 - 0.01}{0.01} = 99 \Rightarrow \hat{w}_t = 2.3$$

$WE(\hat{f}_t) = 0.5$

$$\frac{1 - 0.5}{0.5} = 1 \Rightarrow \hat{w}_t = 0$$

$WE(\hat{f}_t) = 0.99$

$$\frac{1 - 0.99}{0.99} = 0.01 \Rightarrow \hat{w}_t = -2.3$$

Updating α_i

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

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

AdaBoost
Ada Glance

Train

$\alpha_i = 1/N$

```
for $t$ in $[1, 2, \ldots, T]$:
    -   Learn $\hat{f}_t(x)$ based on weights $\alpha_i$
    -   Compute model weight $\hat{w}_t$
    -   Recompute weights $\alpha_i$
```

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

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

Predict

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

Normalizing Weights

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 weights so they sum to 1 to prevent them from getting too small or too big.

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

AdaBoost
Ada Glance

Train

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

```
for $t$ in $[1, 2, \dots, T]$:
    - Learn $\hat{f}_t(x)$ based on weights $\alpha_i$
    - Compute model weight $\hat{w}_t$
    - Recompute weights $\alpha_i$
    - Normalize $\alpha_i$
```

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

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

Predict

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

Brain Break

11:25

Visualizing AdaBoost

$t = 1$ Learn a Classifier

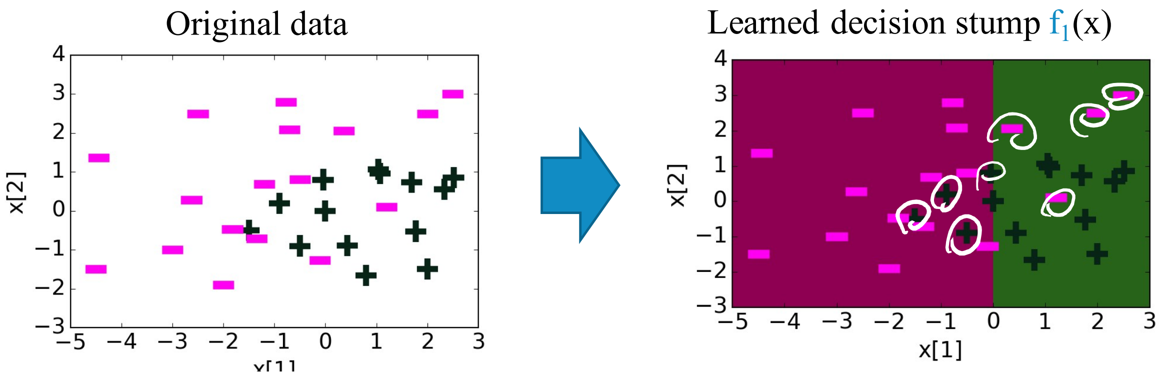
Start with all data having same weight

Learn a decision stump that minimizes weighted error

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

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

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



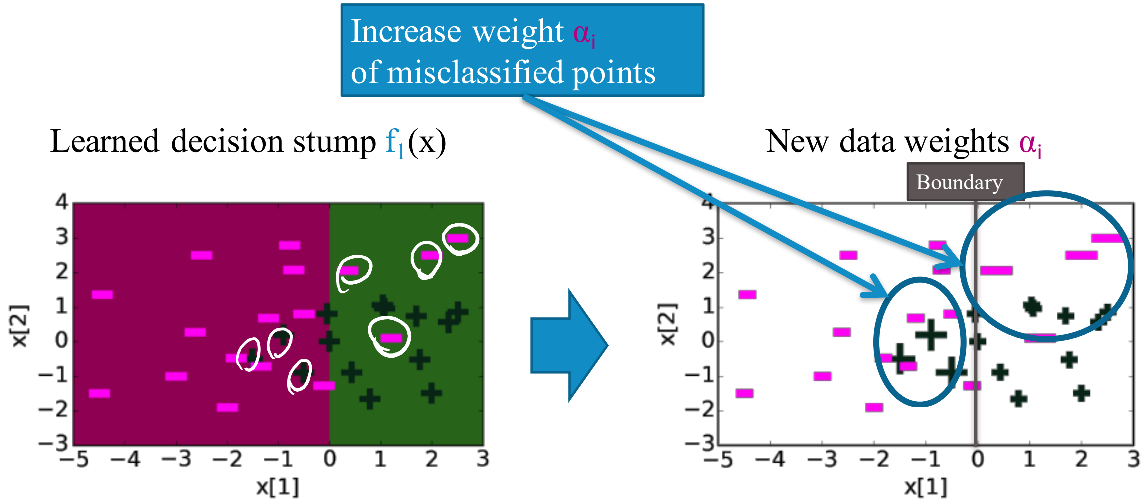
Original data

Learned decision stump $f_1(x)$

$t = 1$ Update Data Weights

Compute new weights α_i based on the errors of $\hat{f}_1$

The points with more weight are drawn larger



Increase weight α_i of misclassified points

Learned decision stump $f_1(x)$

New data weights α_i

Boundary

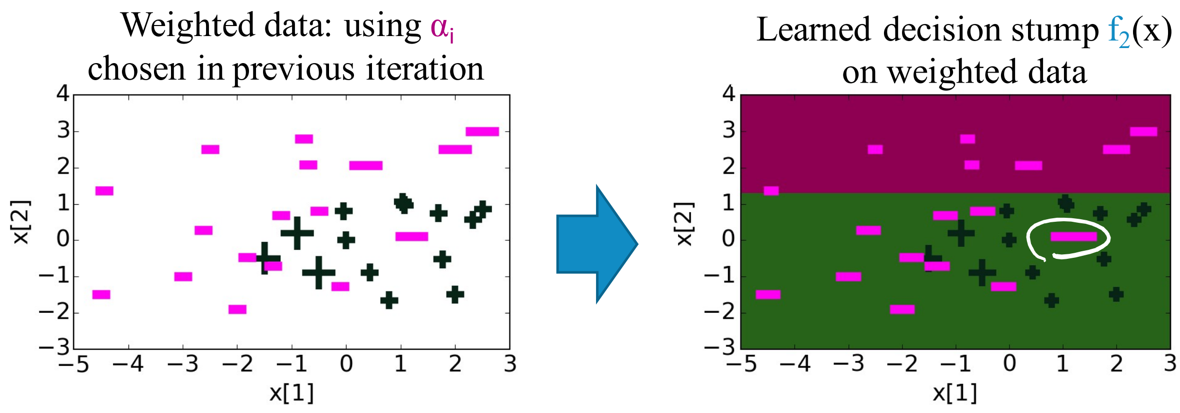
$t = 2$
Learn a Classifier

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

$$\hat{f}_{\underset{\nearrow}{2}}(x) = \cdots$$

Calculate $\hat{w}_{\overset{2}{1}} \approx 0.53$

Then update weights based on errors.



Weighted data: using α_i chosen in previous iteration

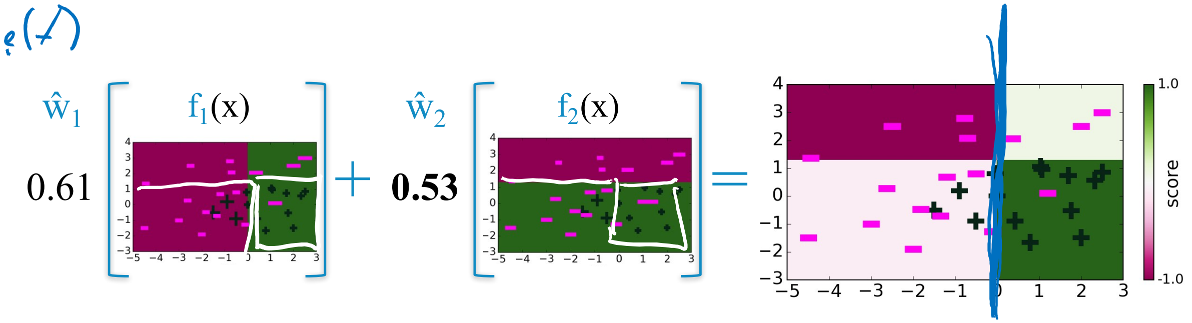
Learned decision stump $f_2(x)$ on weighted data

AdaBoost Ensemble

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

$$\hat{y}_i = \text{sign}\left(\underbrace{\hat{w}_1 \hat{f}_1(x_i) + \hat{w}_2 \hat{f}_2(x_i)}_{Score(x_i)}\right)$$

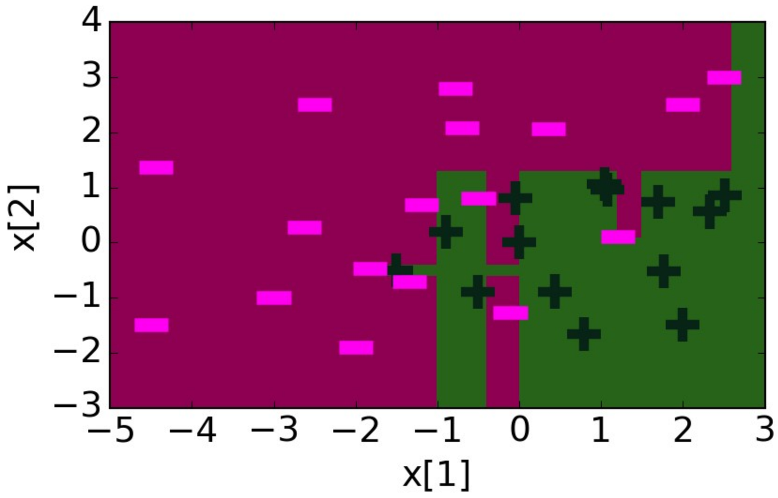
Score(x)



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 Ada Glance

Train

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

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

- Learn $\hat{f}_t(x)$ based on weights α_i
- Compute model weight $\hat{w}_t$
- Recompute weights α_i
- Normalize α_i

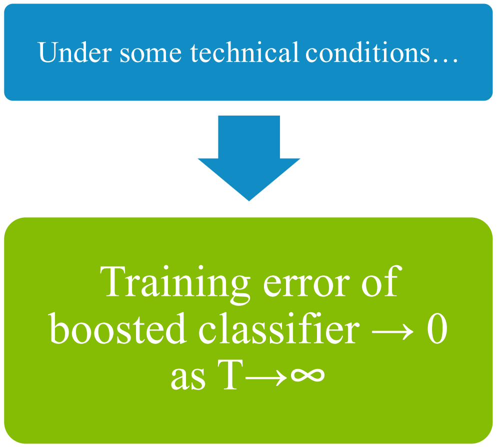
$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t}, & if\ \hat{f}_t(x_i) = y_i \\ \alpha_i e^{\hat{w}_t}, & if\ \hat{f}_t(x_i) \neq y_i \end{cases}$$

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

Predict

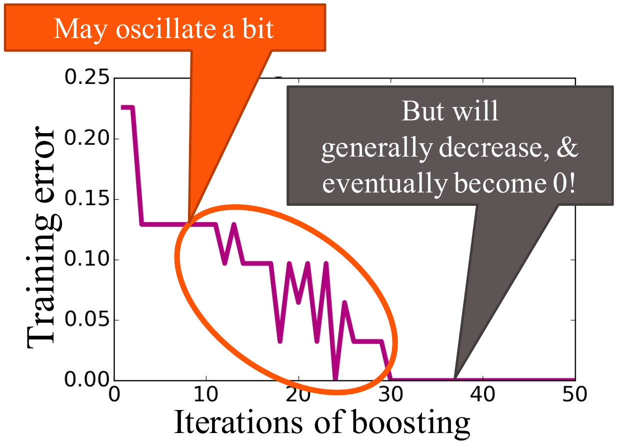
$$\hat{y} = \hat{F}(x) = sign\left(\sum_{t=1}^{T} \hat{w}_t \hat{f}_t(x)\right)$$

AdaBoost Theorem



Under some technical conditions…

Training error of boosted classifier → 0 as $T\to\infty$



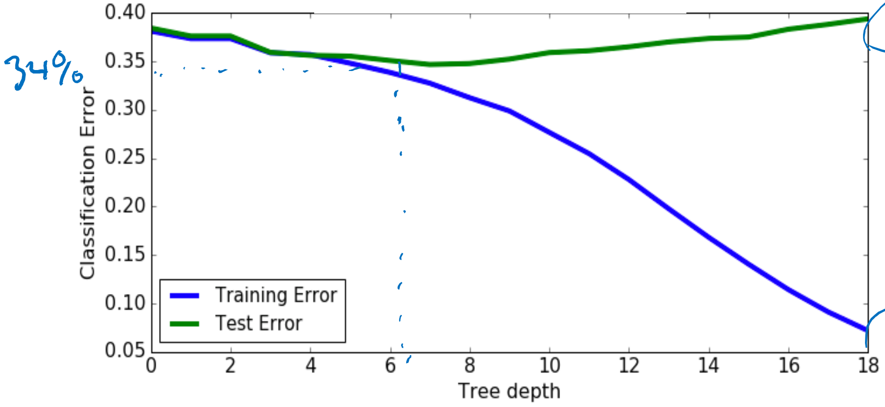
May oscillate a bit

But will generally decrease, & eventually become 0!

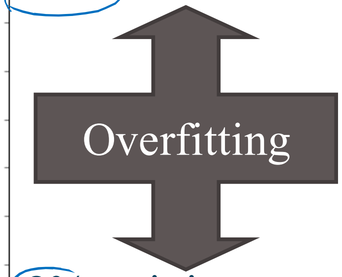
Technical conditions namely that the data can be perfectly classified by some function

Compare

Decision Tree



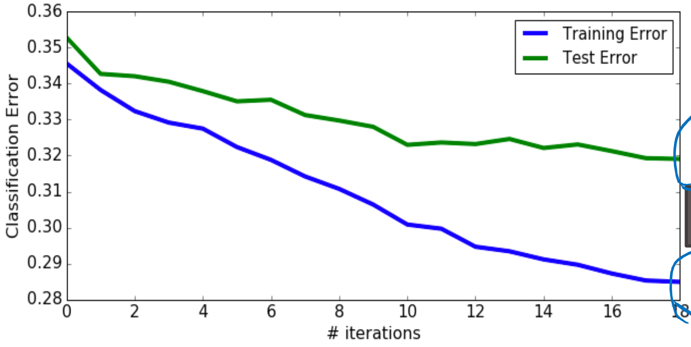
39% test error



Overfitting

8% training error

AdaBoost



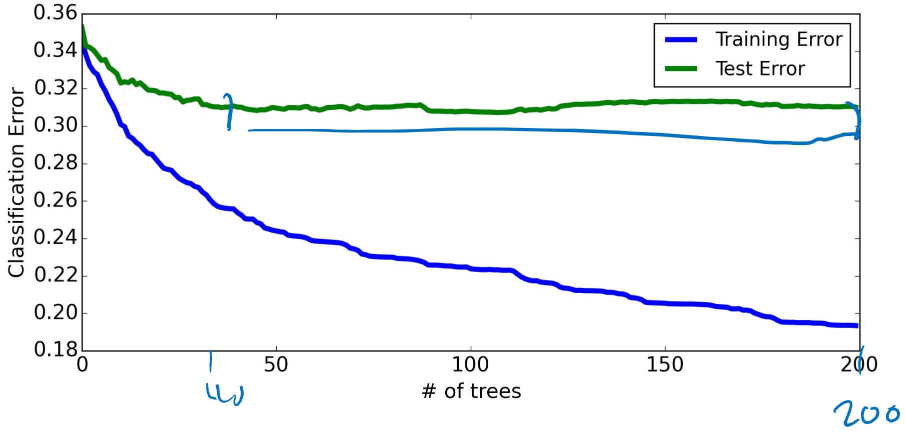
32% test error

Better fit & lower test error

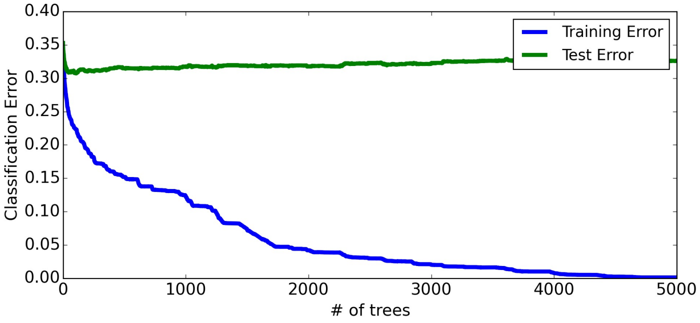
28.5% training error

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 use 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 parameters
  - AdaBoost: Number of trees is technically important, but the model tends to be robust to overfitting in practice.
  - Gradient Boosting: MANY 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 and how those classifiers are learned.
- Compare/contrast bagging and boosting.
- Describe the steps of the AdaBoost algorithm.