

CPSC 340: Machine Learning and Data Mining

Boosting

Andreas Lehrmann and Mark Schmidt

University of British Columbia, Fall 2022

<https://www.students.cs.ubc.ca/~cs-340>

Previously: Ensemble Methods

- Ensemble ~~methods~~ ^{classifiers} are classifiers that have classifiers as input.
 - Also called “meta-learning”.
- They have the best names:
 - Averaging.
 - Boosting.
 - Bootstrapping.
 - Bagging.
 - Cascading.
 - Random Forests.
 - Stacking.
- Ensemble methods often have higher accuracy than input classifiers.

Ensemble Methods

- Remember the fundamental trade-off:
 1. E_{train} : How small you can make the training error.
 - vs.
 2. E_{approx} : How well training error approximates the test error.
- Goal of ensemble methods is that meta-classifier:
 - Does much better on one of these than individual classifiers.
 - Doesn't do too much worse on the other.
- This suggests two types of ensemble methods:
 1. **Averaging**: improves approximation error of classifiers with high E_{approx} .
 2. **Boosting**: improves training error of classifiers with high E_{train} .

AdaBoost: Classic Boosting Algorithm

- A classic boosting algorithm for binary classification is **AdaBoost**.
- AdaBoost assumes we have a “**base**” **binary classifier** that:
 - Is **simple** enough that it doesn't overfit much.
 - Can obtain **>50% weighted accuracy** on any dataset.

$$\sum_{i=1}^n v_i I[\hat{y}_i = y_i]$$

↑ weights (sum to 1)

is example i classified correctly

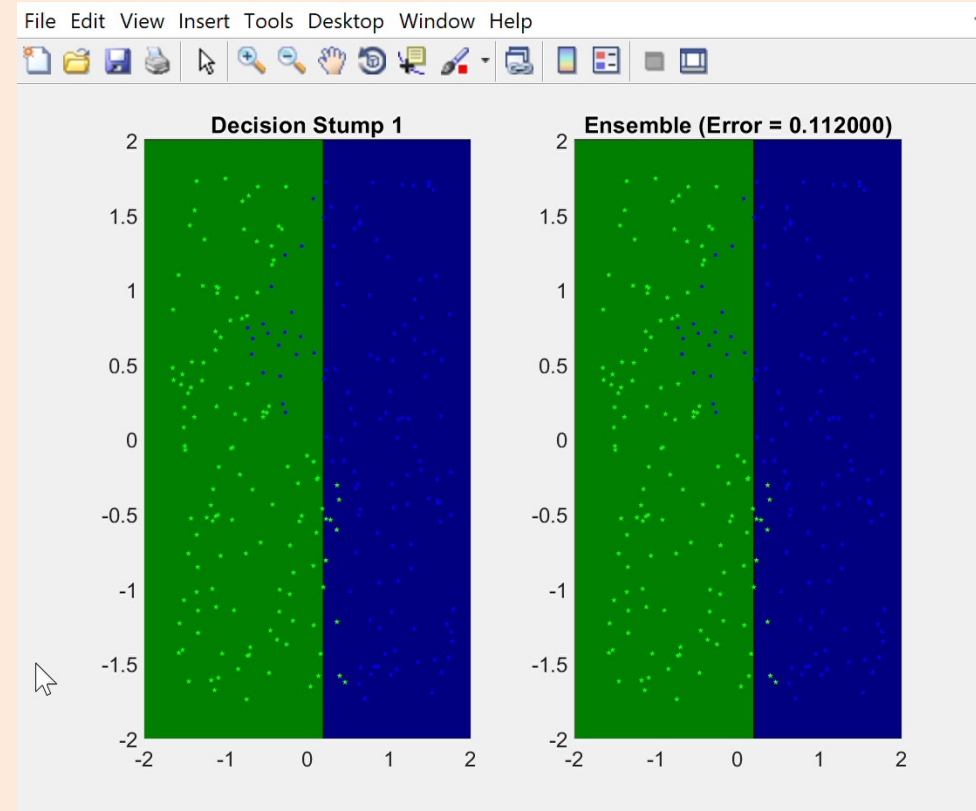
- Example: **decision stumps or low-depth decision trees**.
 - Easy to modify stumps/trees to use weighted accuracy as score.

AdaBoost: Classic Boosting Algorithm

- Overview of AdaBoost:
 1. Fit a classifier on the training data.
 2. Give a higher weight to examples that the classifier got wrong.
 3. Fit a classifier on the weighted training data.
 4. Go back to 2.
 - Weight gets exponentially larger each time you are wrong.
- Final prediction: weighted vote of individual classifier predictions.
 - Trees with higher (weighted) accuracy get higher weight.
- See [Wikipedia](#) for precise definitions of weights.
 - Comes from “exponential loss” (a convex approximation to 0-1 loss).

AdaBoost with Decision Stumps in Action

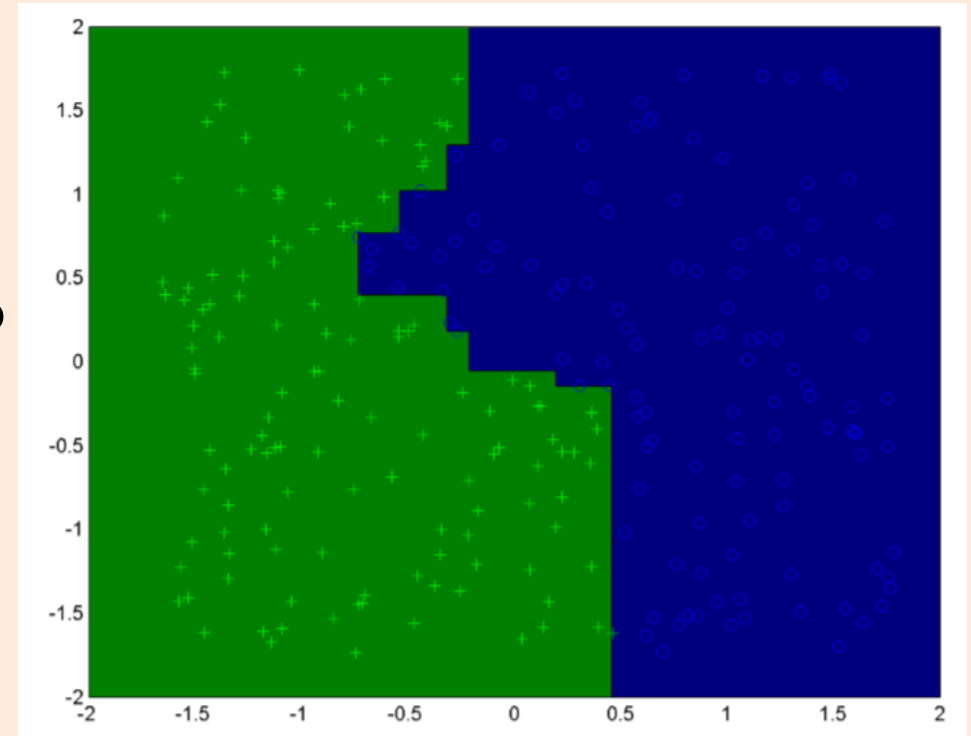
- 2D example of **AdaBoost with decision stumps** (with accuracy score):



- Size of training example on left is proportional to classification weight.

AdaBoost with Decision Stumps

- 2D example of **AdaBoost with decision stumps** (with accuracy score):
 - 100% training accuracy.
 - Ensemble of 50 decision stumps.
 - **Fit sequentially**, not independently.
- Are decision stumps a good base classifier?
 - They tend not to overfit.
 - Easy to get >50% weighted accuracy.
- **Base classifiers that don't work:**
 - Deep decision trees (no errors to “boost”).
 - Decision stumps with infogain (doesn't guarantee >50% weighted accuracy).
 - Weighted logistic regression (doesn't guarantee >50% weighted accuracy).



AdaBoost Discussion

- AdaBoost with shallow decision trees gives **fast/accurate classifiers**.
 - Classically viewed as one of the best “off the shelf” classifiers.
 - Procedure originally came from ideas in learning theory.
- Many attempts to extend theory beyond binary case.
 - Led to “**gradient boosting**”, which is like “gradient descent with trees”.
- Modern boosting methods:
 - Look like AdaBoost, but don’t necessarily have it as a special case.

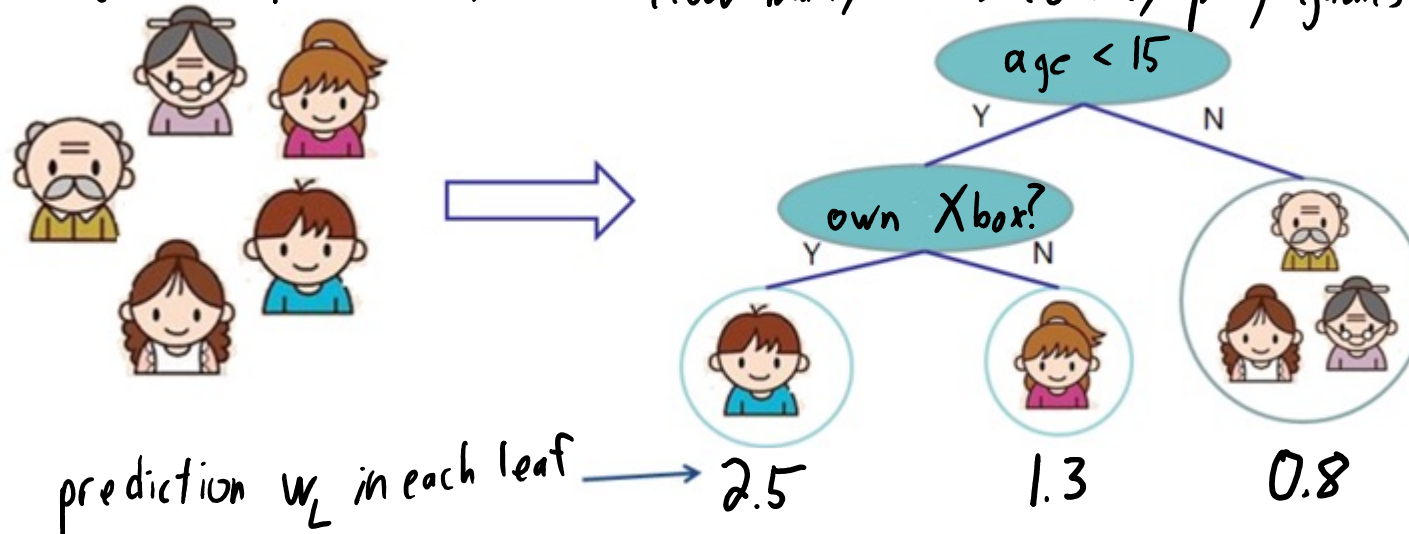
XGBoost: Modern Boosting Algorithm

- Boosting has seen a recent resurgence, partially due to **XGBoost**:
 - A boosting implementation that **allows huge datasets**.
 - Has been part of many recent **winners of Kaggle competitions**.
- As base classifier, XGBoost uses **regularized regression trees**.

Regression Trees

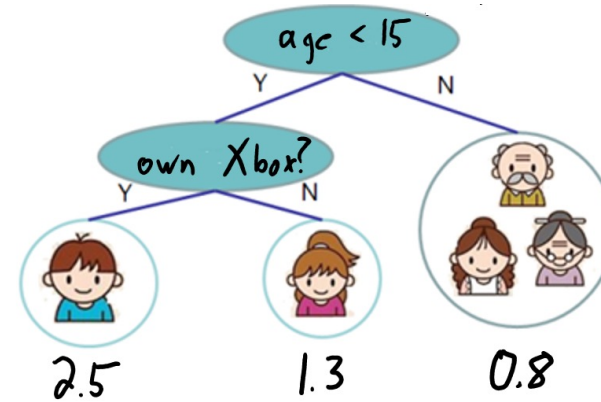
- Regression trees used in XGBoost:
 - Each split is based on 1 feature.
 - Each leaf gives a real-valued prediction.

Input: age, occupation, city, ... How many hours do they play games per day?



- Above, we would predict “2.5 hours” for a 14-year-old who owns an Xbox.

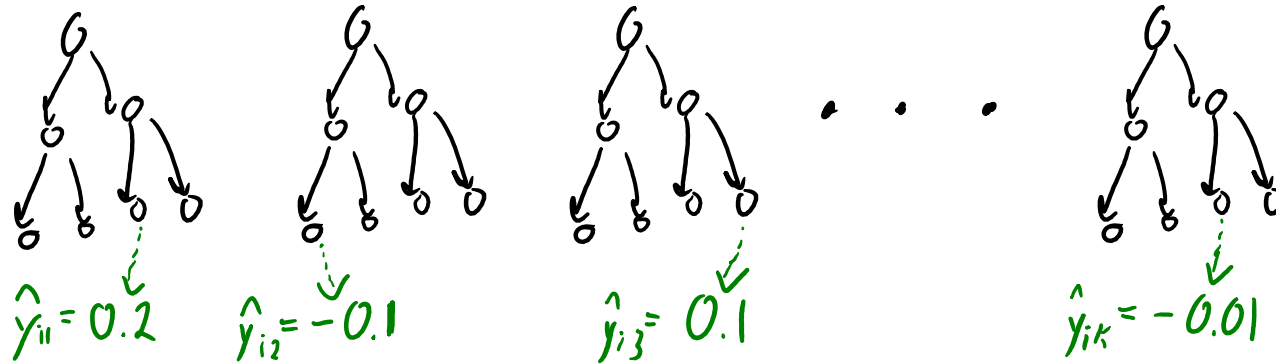
Regression Trees



- How can we fit a regression tree?
- Simple approach:
 - Predict: at each leaf, predict **mean of the training y_i** assigned to the leaf.
 - **Weight w_L at leaf 'L'** is set to $\text{mean}(y_i)$ among y_i at the leaf node.
 - Train: set the w_L values by minimizing the squared error,
$$f(w_1, w_2, \dots) = \sum_{i=1}^n \underbrace{(w_{L_i} - y_i)}_{y_i}^2$$
 - **Same speed as fitting decision trees** from Week 2.
 - Use mean instead of mode, and use squared error instead of accuracy/info gain.
 - Use **greedy strategy for growing tree**, as in Part 1.

Boosted Regression Trees: Prediction

- Consider an **ensemble of regression trees**.
 - For an example 'i', they each make a **continuous prediction**:



- In XGBoost, final **prediction is sum of individual predictions**:

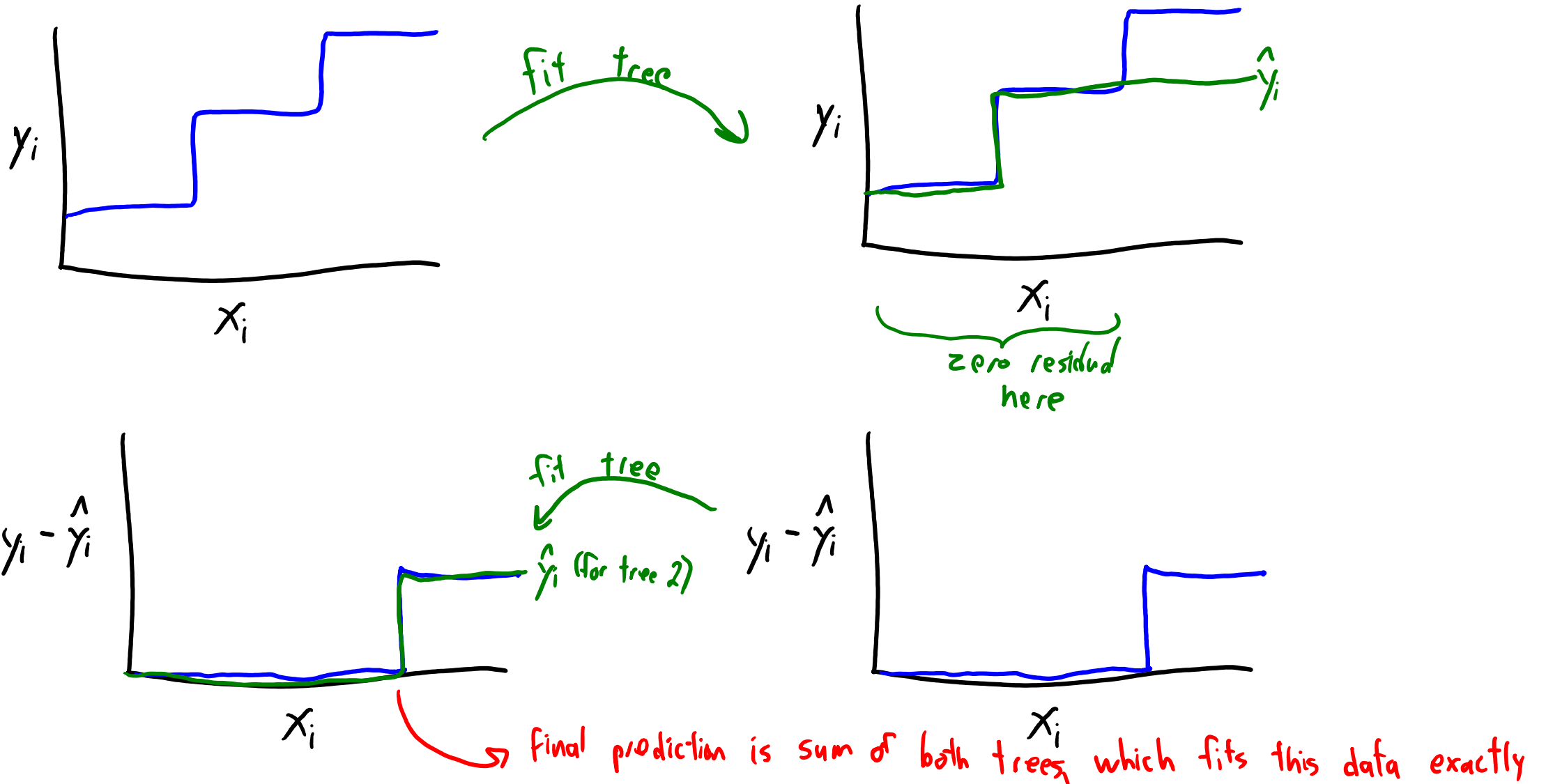
$$\begin{aligned}\hat{y}_i &= \hat{y}_{i1} + \hat{y}_{i2} + \hat{y}_{i3} + \dots + \hat{y}_{ik} \\ &= 0.2 + (-0.1) + 0.1 + \dots + (-0.01)\end{aligned}$$

- Notice we **aren't using the mean** as we would with random forests.
 - In boosting, each **tree is not individually trying to predict the true y_i value** (we assume they underfit).
 - Instead, **each new tree tries to "fix" the prediction made by the old trees**, so that sum is y_i .

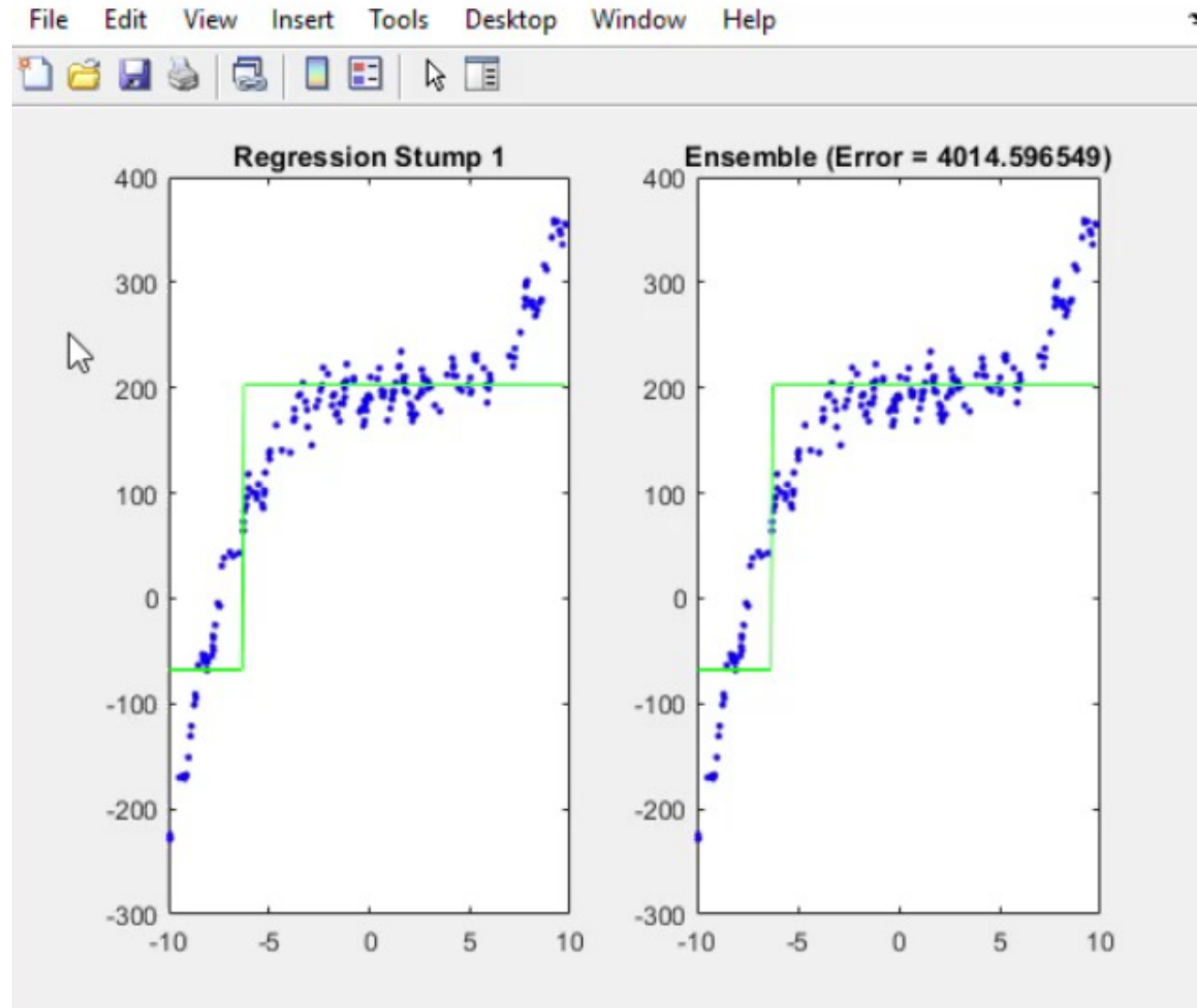
Boosted Regression Trees: Training

- Consider the following “gradient tree boosting” procedure:
 - $\text{Tree}[1] = \text{fit}(X, y)$.
 - $\hat{y} = \text{Tree}[1].\text{predict}(X)$.
 - $\text{Tree}[2] = \text{fit}(X, y - \hat{y})$.
 - $\hat{y} = \hat{y} + \text{Tree}[2].\text{predict}(X)$.
 - $\text{Tree}[3] = \text{fit}(X, y - \hat{y})$.
 - $\hat{y} = \hat{y} + \text{Tree}[3].\text{predict}(X)$.
 - $\text{Tree}[4] = \text{fit}(X, y - \hat{y})$.
 - $\hat{y} = \hat{y} + \text{Tree}[4].\text{predict}(X)$.
 - ...
- Each tree is trying to predict residuals ($\hat{y}_i - y_i$) of current prediction.
 - “True label is 0.9, old prediction is 0.8, so I can improve \hat{y}_i by predicting 0.1.”

Gradient Tree Boosting in Action



Gradient Tree Boosting in Action



Regularized Regression Trees

- Procedure monotonically decreases the training error.
 - As long as not all $w_L=0$, each tree decreases training error.
- It can **overfit** if trees are too deep or you have too many trees.
 - To restrict depth, add **L0-regularization** (stop splitting if $w_L = 0$).

$$f(w_1, w_2, \dots) = \sum_{i=1}^n (w_{L_i} - r_i)^2 + \lambda_0 \|w\|_0$$

- “Only split if you decrease squared error by λ_0 .”
 - To further fight overfitting, XGBoost also adds **L2-regularization** of ‘w’.

$$f(w_1, w_2, \dots) = \sum_{i=1}^n (w_{L_i} - r_i)^2 + \lambda_0 \|w\|_0 + \lambda_2 \|w\|^2$$

XGBoost Discussion

- Instead of pruning trees if score doesn't improve, grows full trees.
 - And then **prunes parts that don't improve score with L0-regularizer added.**
- Cost of fitting trees in XGBoost is **same as usual decision tree cost.**
 - XGBoost includes a lot of tricks to make this efficient.
 - But can't be done in parallel like random forest (since fitting sequentially).
- In XGBoost, it's the **residuals that act like the "weights"** in AdaBoost.
 - Focuses on decreasing error in examples with large residuals.
- How do you maintain efficiency if not using squared error?
 - For non-quadratic losses like logistic, there is **no closed-form solution.**
 - Approximates non-quadratic losses with **second-order Taylor expansion.**
 - **Maintains least squares efficiency** for other losses (by approximating with quadratic).

Next Topic: Maximum Likelihood Estimation

Motivation for Learning about MLE and MAP

- Next topic: **maximum likelihood estimation (MLE)** and **MAP estimation**.
 - **Crucial to understanding advanced methods, notation can be difficult** at first.
- Why are we learning about these?
 - Justifies the **naïve Bayes “counting” estimates for probabilities**.
 - Shows the **connection between least squares and the normal distribution**.
 - Makes **connection between “robust regression” and “heavy tailed” probabilities**.
 - Shows that **regularization and Laplace smoothing are doing the same thing**.
 - Justifies **using sigmoid function to get probabilities in logistic regression**.
 - Gives a way to **write complicated ML problems as optimization problems**.
 - How do you define a loss for “number of Facebook likes” or “1-5 star rating”?
 - **Crucial to understanding advanced methods**.

But first: “argmin” and “argmax”

- We’ve repeatedly used the **min** and **max** functions:

$$\min_w \{w^2\} = 0 \qquad \max_w \{\cos(w)\} = 1$$

– Minimum (or maximum) value achieved by a function.

- A related set of functions are the **argmin** and **argmax**:
 - The **set of parameter values** achieving the minimum (or maximum).

$$\min_w \{(w-1)^2\} = 0$$

$$\operatorname{argmin}_w \{(w-1)^2\} = 1$$

'1' is the 'w' value that gives the min.

$$\operatorname{argmin}_w \left\{ \frac{1}{2} \|Xw - y\|^2 + \frac{\lambda}{2} \|w\|^2 \right\} = (X^T X + \lambda I)^{-1} (X^T y)$$

$$\operatorname{argmax}_w \{\cos(w)\} = 0, 2\pi, 4\pi, \dots$$

But first: “argmin” and “argmax”

- The last slide is a little sloppy for the following reason:
 - There **may be multiple values** achieving the min and/or max.
 - So the **argmin and argmax return sets**.

$$\operatorname{argmin}_w \{(w-1)^2\} \equiv \{1\}$$

“set containing the element ‘1’”

“sets are equivalent”

$$\operatorname{argmax}_w \{\cos(w)\} \equiv \{\dots, -4\pi, -2\pi, 0, 2\pi, 4\pi, \dots\}$$

$$\operatorname{argmax}_w \left\{ \frac{1}{2} \|Xw - y\|^2 \right\} \equiv \{w \mid X^T X w = X^T y\}$$

- And we don’t say a variable “is” the argmax, but that it “is in” the argmax.

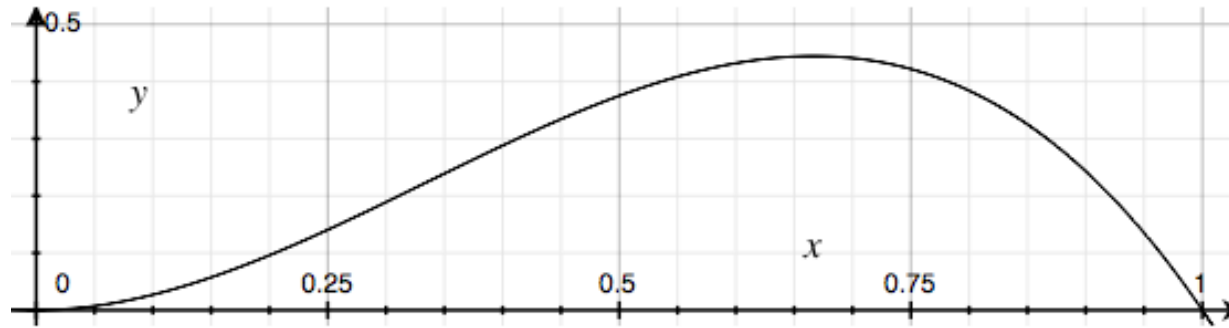
$$2\pi \in \operatorname{argmax}_w \{\cos(w)\} \quad (X^T X + \lambda I)^{-1} (X^T y) \in \operatorname{argmin}_w \left\{ \frac{1}{2} \|Xw - y\|^2 + \frac{\lambda}{2} \|w\|^2 \right\}$$

The Likelihood Function

- Suppose we have a dataset 'D' with parameters 'w'.
- For example:
 - We flip a coin three times and obtain $D = \{\text{"heads"}, \text{"heads"}, \text{"tails"}\}$.
 - The parameter 'w' is the probability that this coin lands "heads".
- We define the likelihood as a probability mass function $p(D | w)$.
 - "Probability of seeing this data, given the parameters".
 - If 'D' is continuous it would be a probability "density" function.
- If this is a "fair" coin (meaning it lands "heads" with probability 0.5):
 - The likelihood is $p(\text{HHT} | w=0.5) = (1/2)(1/2)(1/2) = 0.125$.
 - If $w = 0$ ("always lands tails"), then $p(\text{HHT} | w = 0) = 0$ (data is less likely for this 'w').
 - If $w = 0.75$, then $p(\text{HHT} | w = 0.75) = (3/4)(3/4)(1/4) \approx 0.14$ (data is more likely).

Maximum Likelihood Estimation (MLE)

- We can plot the likelihood $p(\text{HHT} \mid w)$ as a function of 'w':



- Notice:
 - Data has probability 0 if $w=0$ or $w=1$ (since we have 'H' and 'T' in data).
 - Data doesn't have highest probability at 0.5 (we have more 'H' than 'T').
 - This is a **probability distribution over 'D'**, not 'w' (area isn't 1).
- **Maximum likelihood estimation (MLE):**
 - Choose parameters that maximize the likelihood: $\hat{w} \in \arg \max_w \{p(D|w)\}$
 - In this example, MLE is $2/3$.

MLE for Binary Variables (General Case)

- Consider a **binary** feature:

$$X = \begin{bmatrix} 0 \\ \vdots \\ 0 \end{bmatrix}$$

- Using 'w' as "probability of 1", the **maximum likelihood estimate** is:

$$\hat{w} = \frac{\# \text{ of ones}}{\# \text{ of examples}}$$

- This is the **"estimate"** for the probabilities we used in naïve Bayes.
 - The conditional probabilities we used in naïve Bayes are also MLEs.
 - The derivation is tedious, but if you're interested I put it [here](#).

Least Squares is Gaussian MLE

- It turns out that **most objectives have an MLE interpretation**:
 - For example, consider **minimizing the squared error**:

$$f(w) = \frac{1}{2} \|Xw - y\|^2$$

- This gives **MLE of a linear model with IID noise from a normal distribution**:

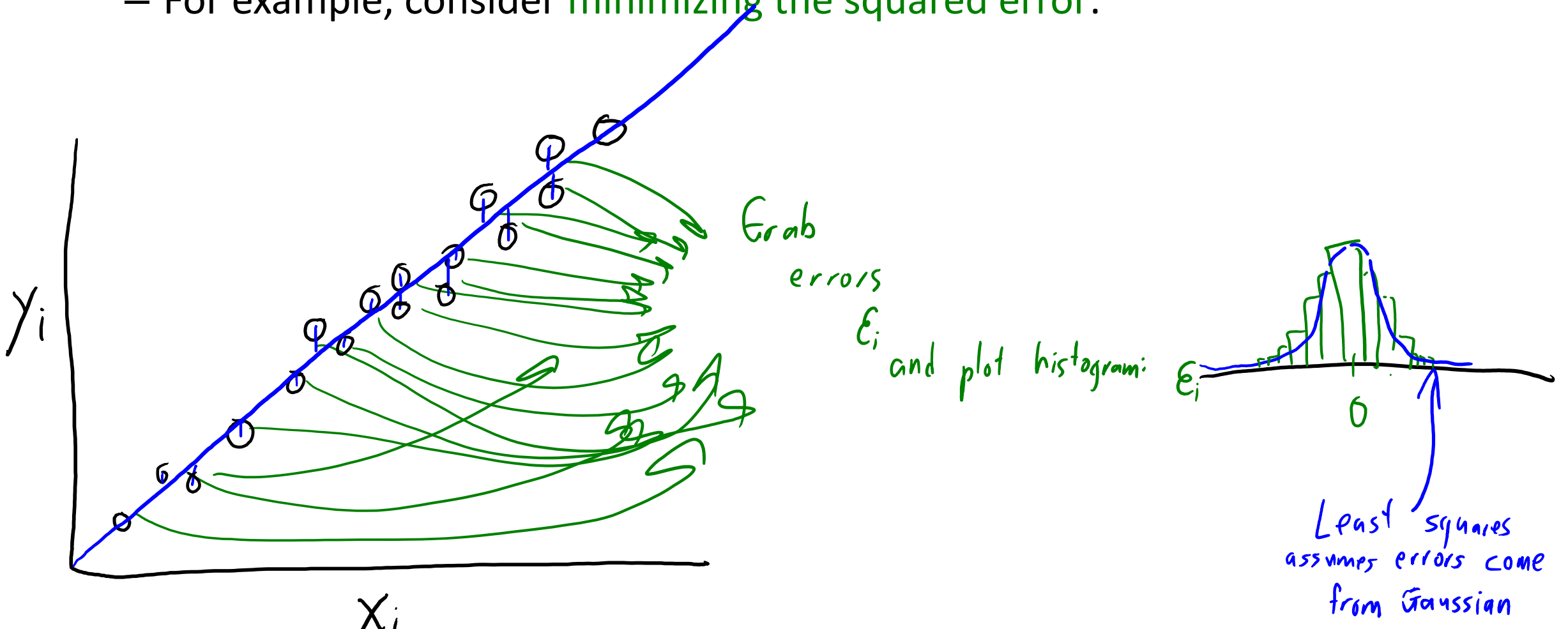
$$y_i = w^T x_i + \epsilon_i$$

where each ϵ_i is sampled independently from standard normal

- **“Gaussian” is another name for the “normal” distribution.**
- Remember that least squares solution is called the **“normal equations”**.

Least Squares is Gaussian MLE

- It turns out that **most objectives have an MLE interpretation:**
 - For example, consider **minimizing the squared error:**



Minimizing the Negative Log-Likelihood (NLL)

- To compute maximum likelihood estimate (MLE), usually we equivalently minimize the **negative “log-likelihood” (NLL)**:
 - “Log-likelihood” is short for “logarithm of the likelihood”.

$$\hat{w} \in \operatorname{argmax}_w \{ p(D|w) \} \equiv \operatorname{argmin}_w \{ -\log(p(D|w)) \}$$

- Why are these **equivalent**? *“equivalent”*
 - Logarithm is strictly monotonic: if $\alpha > \beta$, then $\log(\alpha) > \log(\beta)$.
 - So **location of maximum doesn’t change** if we take logarithm.
 - Changing sign flips max to min.
- See [Max and Argmax](#) notes if this seems strange.

Summary

- **Boosting**: ensemble methods that improve training error.
- **XGBoost**: modern boosting method based on regression trees.
 - Each tree modifies the prediction made by the previous trees.
 - L0- and L2-regularization used to reduce overfitting.
- **Maximum likelihood estimate**:
 - Maximizing likelihood $p(D | w)$ of data 'D' given parameters 'w'.
- **Next time**:
 - How does regularization and Laplace smoothing fit it?