CPSC 340: Machine Learning and Data Mining

Boosting; start of MLE/MAP

Spring 2022 (2021W2)

Admin

- A4 due Mid-night today.
- A5 to be release by end of this week or early next week.

Previously: Ensemble Methods

- Ensemble methods are models that have models 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 models.

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-model:
  - Does much better on one of these errors than individual model.
  - Doesn't do too much worse on the other error.
- This suggests two types of ensemble methods:
  1. Averaging: improves approximation error (due to overfitting less).
  2. Boosting: improves training error.

bonus!

The basic goal of boosting

Combine "weak learners"…

…into one "strong learner"

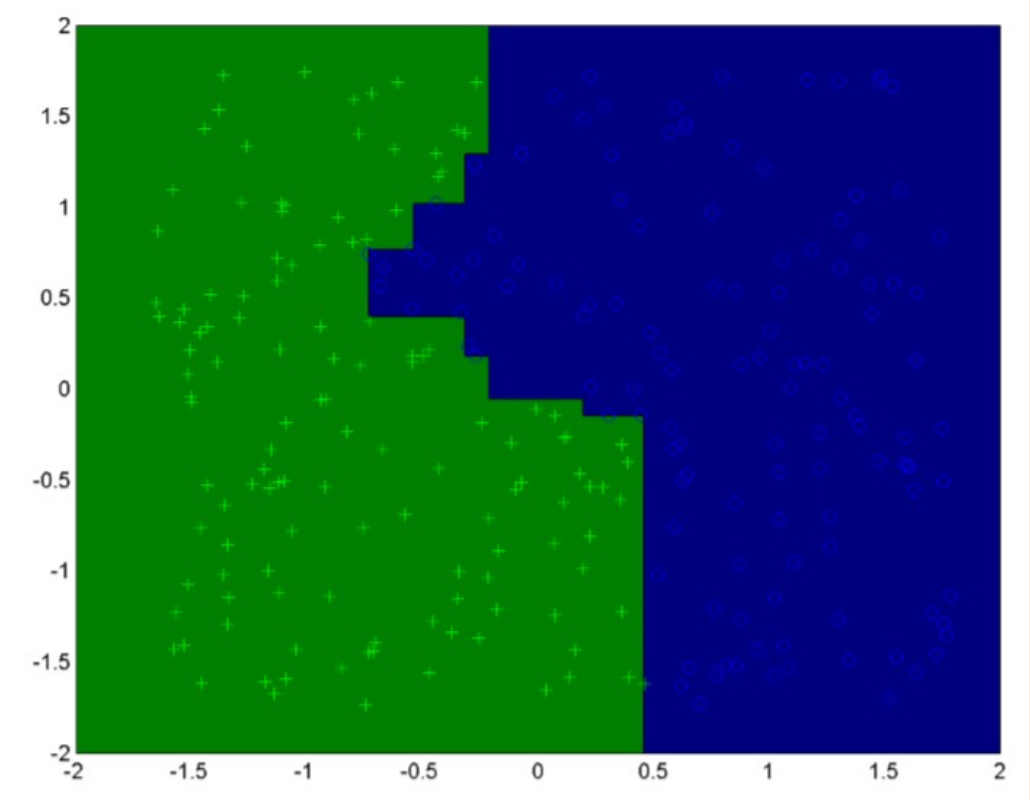
/r/learnML, via Piazza

Nikita Goel

bonus!

AdaBoost: Classic Boosting Algorithm

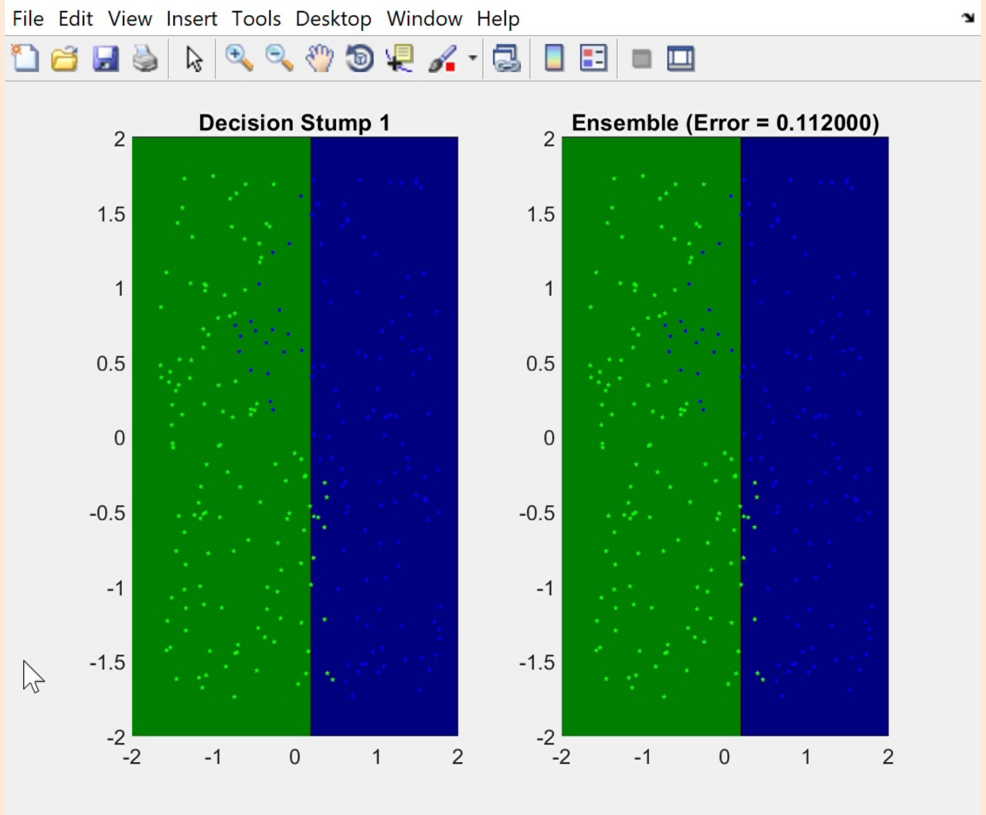
- A classic boosting algorithm for binary classification is AdaBoost.
  - Usually uses decisions stumps as a "weak" classifier.
    - That can get >50% accuracy but may not get high accuracy.
  - Fits one decision stump at a time.
    - Each decision stump gives higher weight to examples that are classified incorrectly.
      - Not fit independently like in random forests.
  - Final prediction based on weighted voting.
    - More details in bonus slides.



bonus!

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.

bonus!

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.

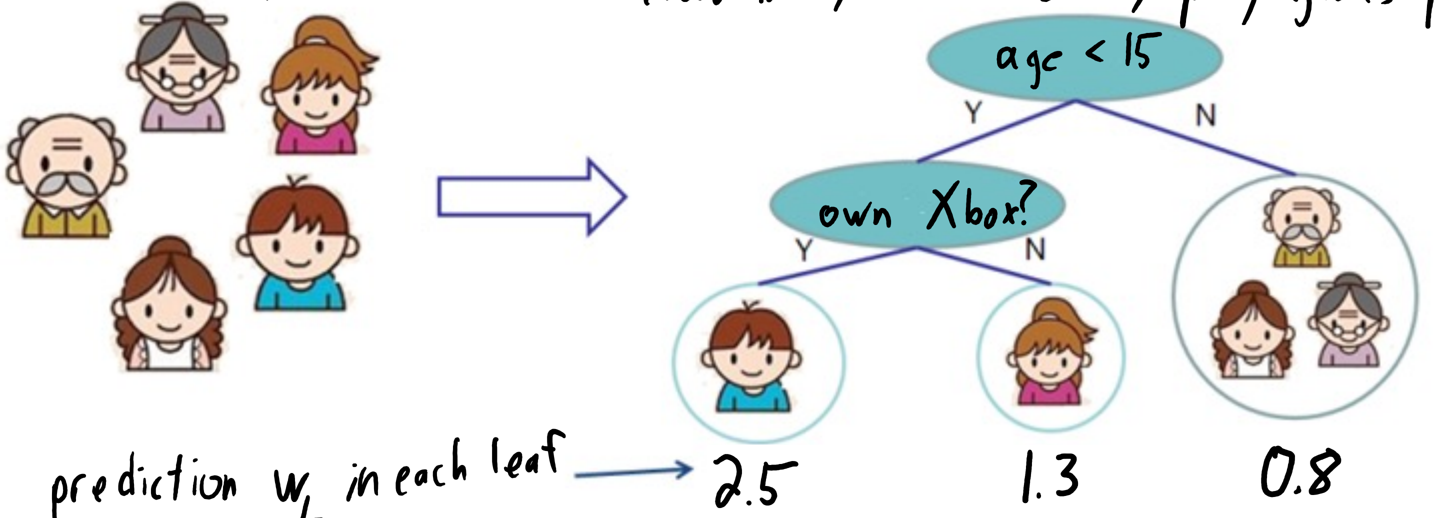
- As base classifier, XGBoost uses regularized regression trees.

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?

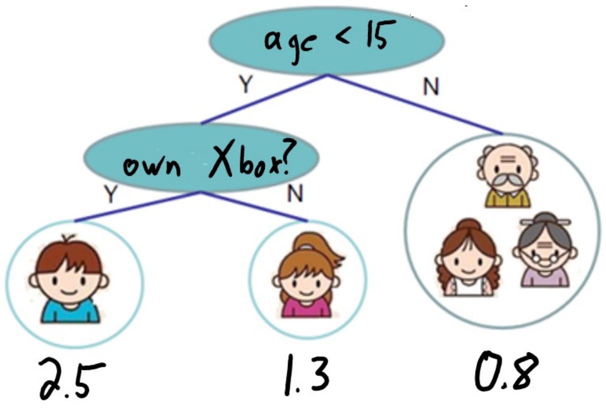


prediction w_L in each leaf

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

https://xgboost.readthedocs.io/en/latest/tutorials/model.html

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 mean(y_i) among y_i at the leaf node.
  - Train: set the w_L values by minimizing the squared error,

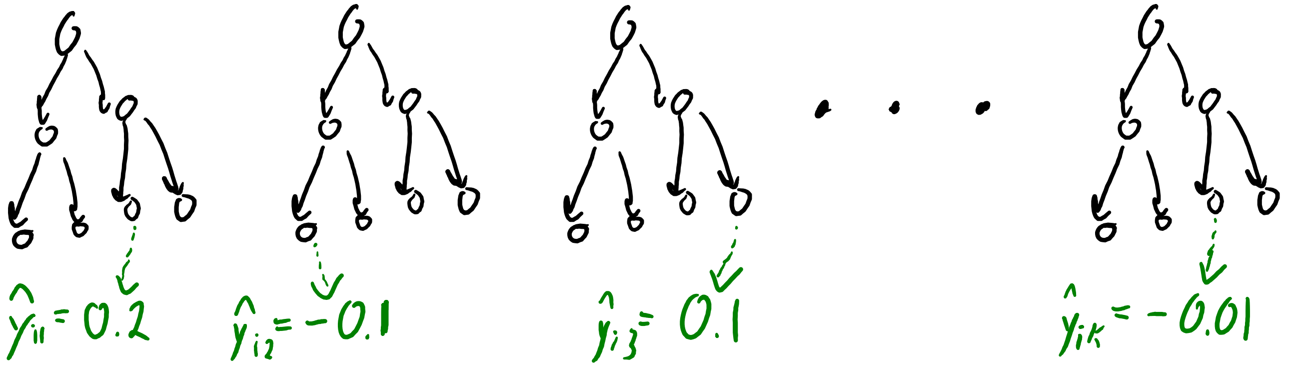
$$f(w_1, w_2, \dots) = \sum_{i=1}^{n} (\underbrace{w_{L_i}}_{\hat{y}_i} - y_i)^2$$

  - Same speed as fitting decision trees from Week 2.
    - Use mean instead of mode, and use squared error instead of accuracy/infogain.
  - Use greedy strategy for growing tree, as in Part 1.

https://xgboost.readthedocs.io/en/latest/tutorials/model.html

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:

$$\hat{y}_i = \hat{y}_{i1} + \hat{y}_{i2} + \hat{y}_{i3} + \cdots + \hat{y}_{ik}$$

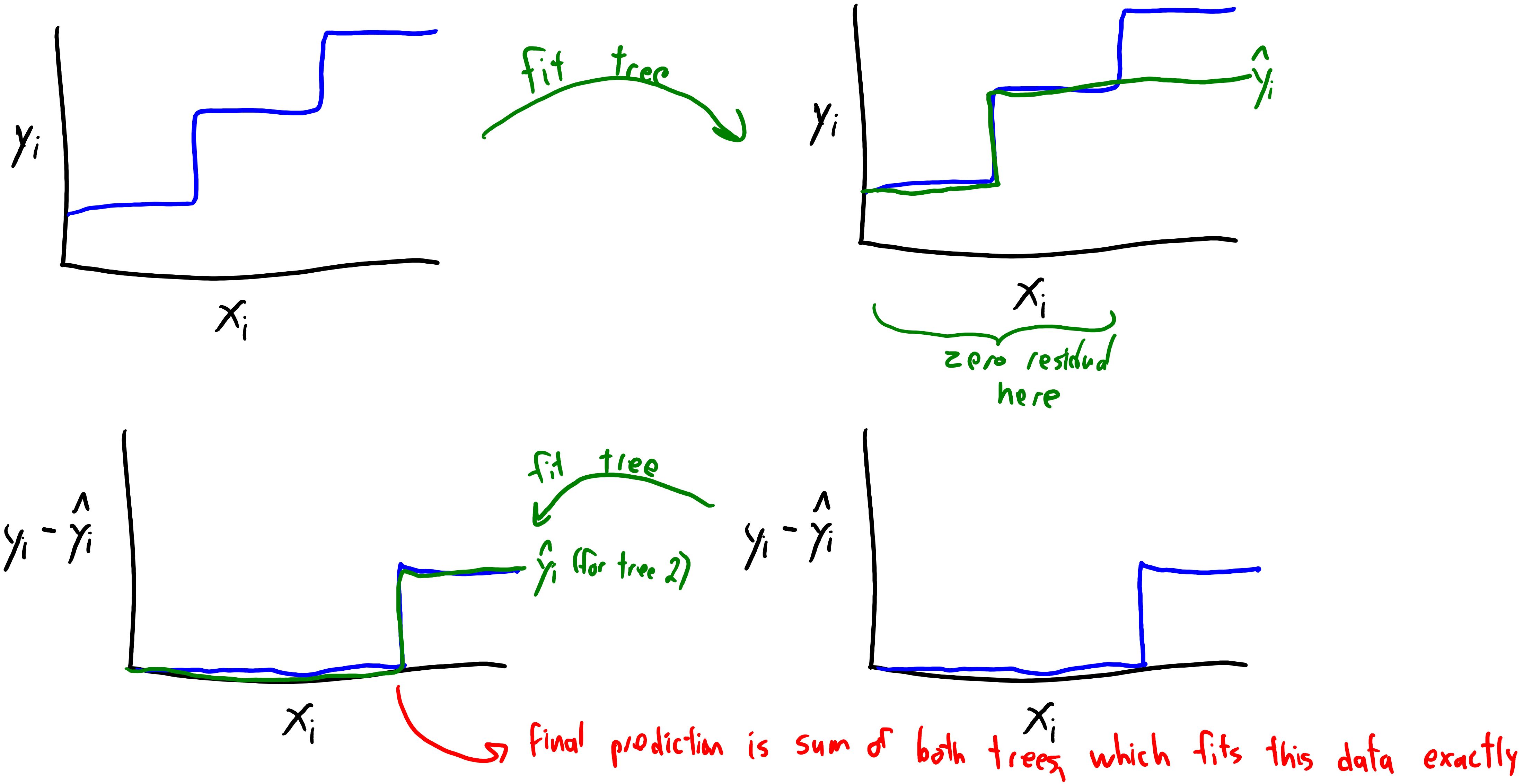
$$= 0.2 + (-0.1) + 0.1 + \cdots + (-0.01)$$

- 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:
  - trees[0] = fit(X, y)
  - $\hat{y}$ = trees[0].predict(X)
  - trees[1] = fit(X, y - $\hat{y}$)
  - $\hat{y}$ = $\hat{y}$ + trees[1].predict(X)
  - trees[2] = fit(X, y - $\hat{y}$)
  - $\hat{y}$ = $\hat{y}$ + trees[2].predict(X)
  - trees[3] = fit(X, y - $\hat{y}$)
  - $\hat{y}$ = $\hat{y}$ + trees[3].predict(X)
  - …
- Each tree is trying to predict residuals ($y_i - \hat{y}_i$) of current prediction.
  - "True target is 0.9, old prediction is 0.8, so I can improve $\hat{y}_i$ by predicting 0.1."

Gradient Tree Boosting in Action



bonus!

Why is it called *gradient* boosting?

- In general, we'd like to greedily add the best possible tree (or whatever):

$$f_m = \operatorname*{argmin}_{\substack{f \in \text{decision trees} \\ \text{(or whatever)}}} \sum_{i=1}^{n} \ell(\hat{y}_{i,m-1} + f(x_i), y_i) \qquad \hat{y}_{i,m} = f_0(x_i) + f_1(x_i) + \dots + f_m(x_i)$$

per-prediction loss

- If $\ell(\hat{y}, y) = \ell(\hat{y}+c, y+c)$ (most regression losses), just fit residuals
- Generally, can do "*functional gradient descent*":

(f

1. $r_{im} = -\left.\frac{\partial \ell(\hat{y}, y_i)}{\partial \hat{y}}\right|_{\hat{y}=\hat{y}_{i,m-1}}$; fit $f_m = \operatorname*{argmin}_f (f(x_i) - r_{im})^2$

   (f_m is fastest direction to reduce loss (locally): the gradient)

   Square loss: $\ell(\hat{y}, y_i) = \frac{1}{2}(\hat{y} - y_i)^2$, $-\frac{\partial \ell(\hat{y}, y_i)}{\partial \hat{y}} = -(\hat{y} - y_i) = y_i - \hat{y}$

2. Scale the outputs optimally:

   (picks best step size with a line search)

   $$\gamma_m = \operatorname*{argmin}_{\gamma \in \mathbb{R}} \sum_{i=1}^{n} \ell(\hat{y}_{i,m-1} + \gamma f_m(x_i), y_i)$$

   $\ell = \frac{1}{2}(\hat{y} - y_i)^2$: $\gamma_m = 1$

   $\ell = \frac{1}{n}(\hat{y} - y_i)^2$: $\gamma_m = \frac{n}{2}$

3. $\hat{y}_{i,m} = \hat{y}_{i,m-1} + \gamma_m f_m(x_i)$

XGBoost slightly different: also uses ℓ'' (Newton-Raphson)

Regularized Regression Trees

- Procedure monotonically decreases the training error.
  - As long as at least one leaf pred $w_L \neq 0$, 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'.

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

bonus!

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 we're fitting sequentially).
- In XGBoost, it's the residuals that act like the "weights" in AdaBoost.
  - Focuses on decreasing error in examples with large residuals.

(pause)

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).

$$\min_w (w-1)^2 = 0 \qquad \underset{w}{\operatorname{argmin}} \frac{1}{2}\|Xw-y\|^2 + \frac{\lambda}{2}\|w\|^2 = (X^TX+\lambda I)^{-1}(X^Ty)$$

$$\underset{w}{\operatorname{argmin}} (w-1)^2 = 1 \qquad \underset{w}{\operatorname{argmax}} \cos(w) = 0, 2\pi, 4\pi, \ldots$$

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

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.

$$\underset{w}{\mathrm{argmin}}\ (w-1)^2 \equiv \{1\}$$

"set containing the element '1'"

"sets are equivalent"

$$\underset{w}{\mathrm{argmax}}\ \cos(w) \equiv \{\ldots, -4\pi, -2\pi, 0, 2\pi, 4\pi, \ldots\}$$

$$\underset{w}{\mathrm{argmax}}\ \frac{1}{2}\|Xw-y\|^2 \equiv \{w \mid X^TXw = X^Ty\}$$

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

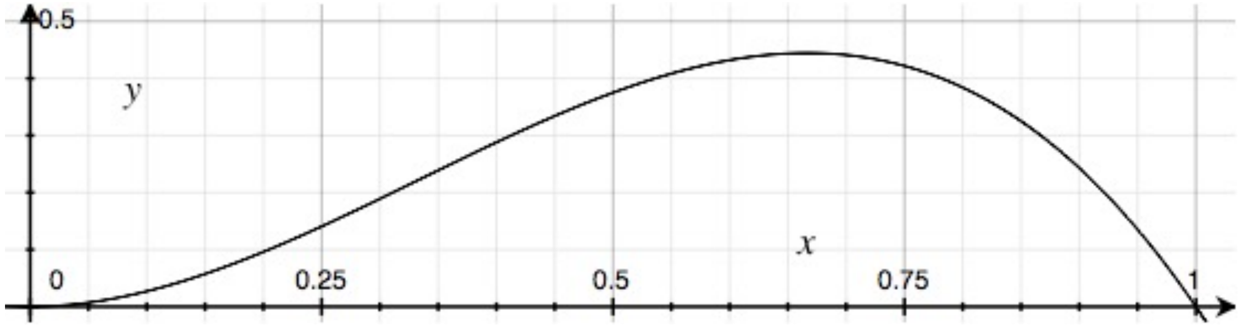
$$2\pi \in \underset{w}{\mathrm{argmax}}\ \cos(w) \qquad (X^TX+\lambda I)^{-1}(X^Ty) \in \underset{w}{\mathrm{argmin}}\ \frac{1}{2}\|Xw-y\|^2 + \frac{\lambda}{2}\|w\|^2$$

The Likelihood Function

- Suppose we have a dataset 'D' with parameters 'w'.
- For example:
  - We flip a coin three times and obtain D = "heads", "heads", "tails" (in that order).
  - 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(HHT | w=0.5) = (1/2)(1/2)(1/2) = 0.125.
  - If w = 0 ("always lands tails"), then p(HHT | w = 0) = 0 (data is less likely for this 'w').
  - If w = 0.75, then p(HHT | w = 0.75) = (3/4)(3/4)(1/4) ≈ 0.14 (data is more likely).

Maximum Likelihood Estimation (MLE)

- We can plot the likelihood p(HHT | 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: $\hat{w} \in \underset{w}{\operatorname{argmax}}\ p(D|w)$
    - In this example, MLE is 2/3.

MLE for Binary Variables (General Case)

- Consider a binary feature:

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

- 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, it's here.

(pause)

Maximum Likelihood Estimation (MLE)

- Maximum likelihood estimation (MLE) for fitting probabilistic models.
  - We have a dataset D.
  - We want to pick parameters 'w'.
  - We define the likelihood as a probability mass/density function p(D | w).
  - We choose the model $\hat{w}$ that maximizes the likelihood:

$$\hat{w} \in \underset{w}{\operatorname{argmax}} \; p(D \mid w)$$

- Appealing "consistency" properties as n goes to infinity (take STAT 4XX).
  - "This is a reasonable thing to do for large data sets".

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:

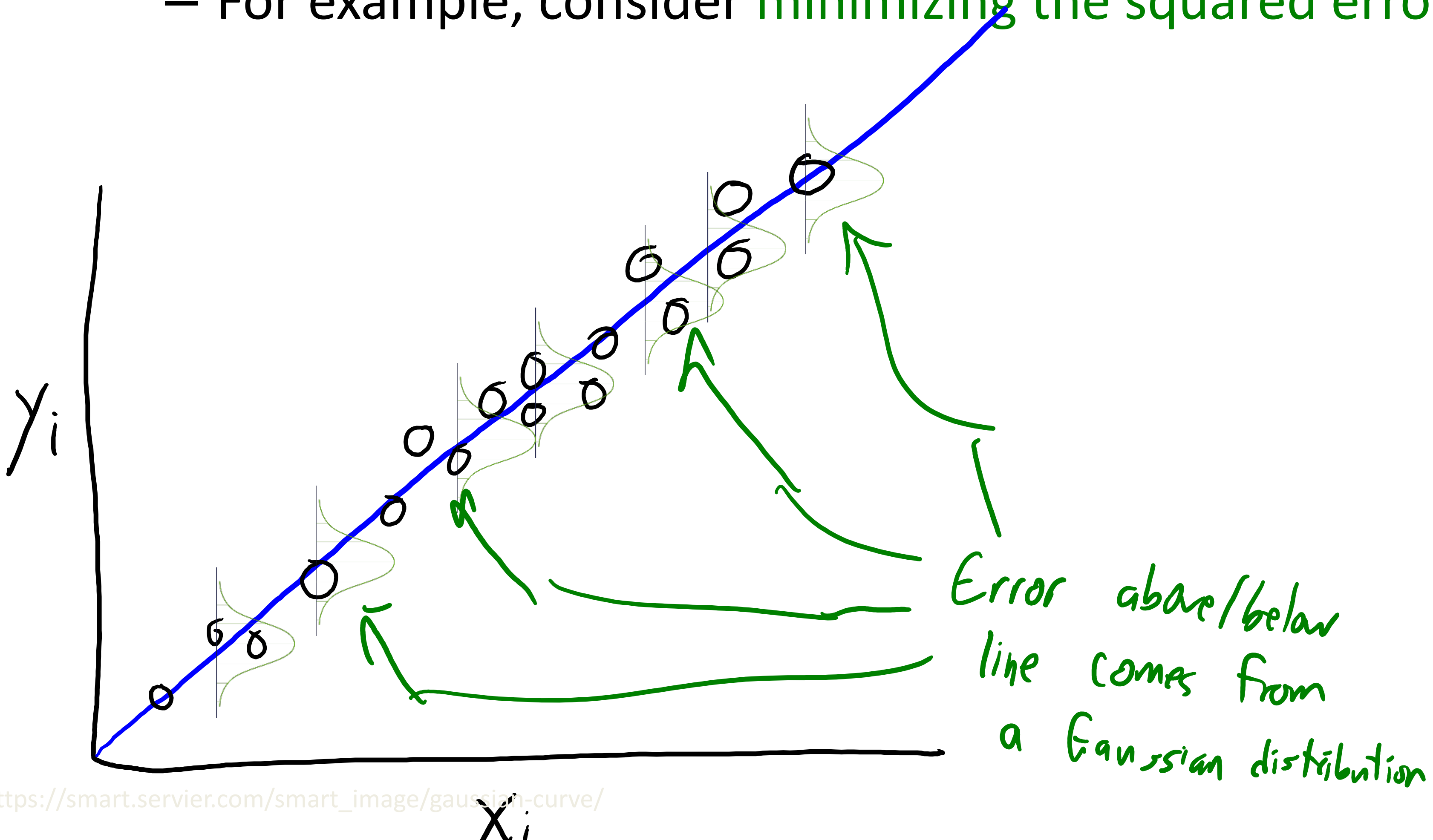
$$y_i = w^T x_i + \epsilon_i$$

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".

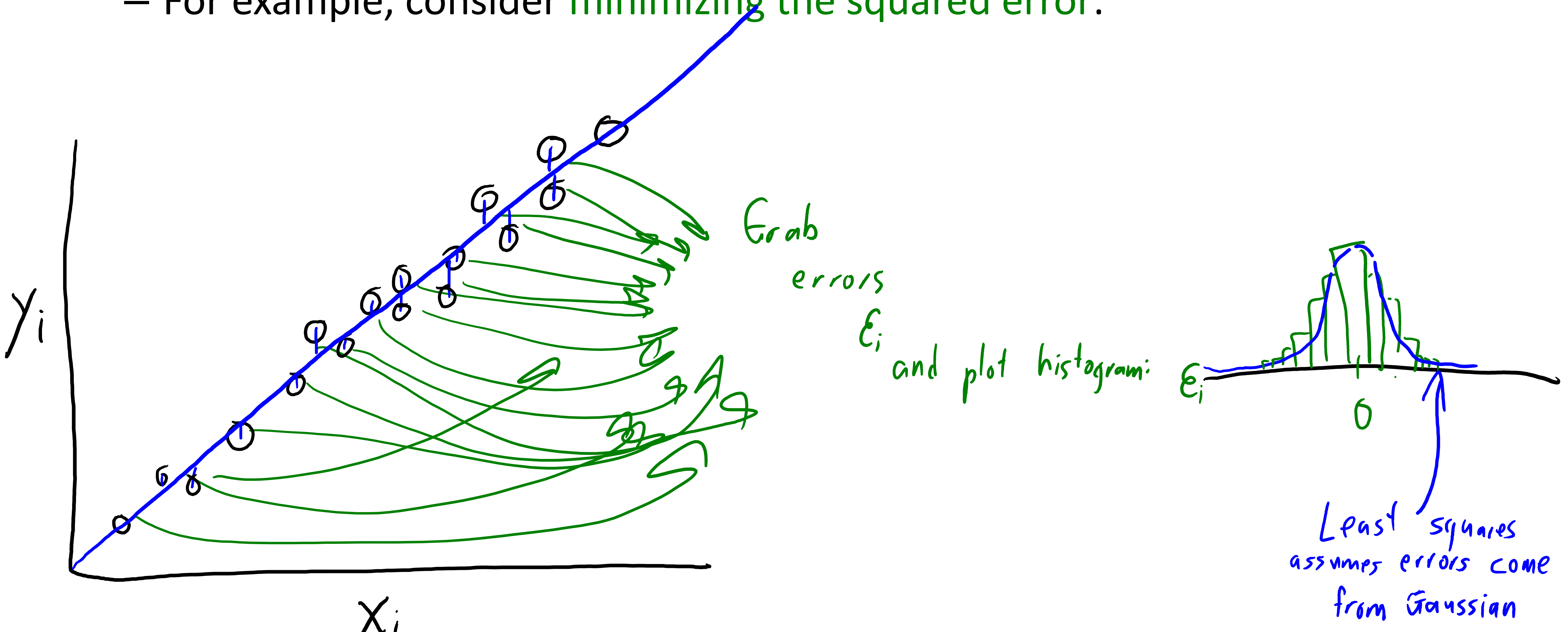
Least Squares is Gaussian MLE

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



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 maximize likelihood estimate (MLE), usually we equivalently minimize the negative "log-likelihood" (NLL):
  - "Log-likelihood" is short for "logarithm of the likelihood".

$$\hat{w} \in \underset{w}{\operatorname{argmax}}\; p(D \mid w) \equiv \underset{w}{\operatorname{argmin}}\; -\log(p(D \mid w))$$

"equivalent"

- Why are these 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 in?

bonus!

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.

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

is example i classified correctly

weights (sum to 1)

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

bonus!

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.

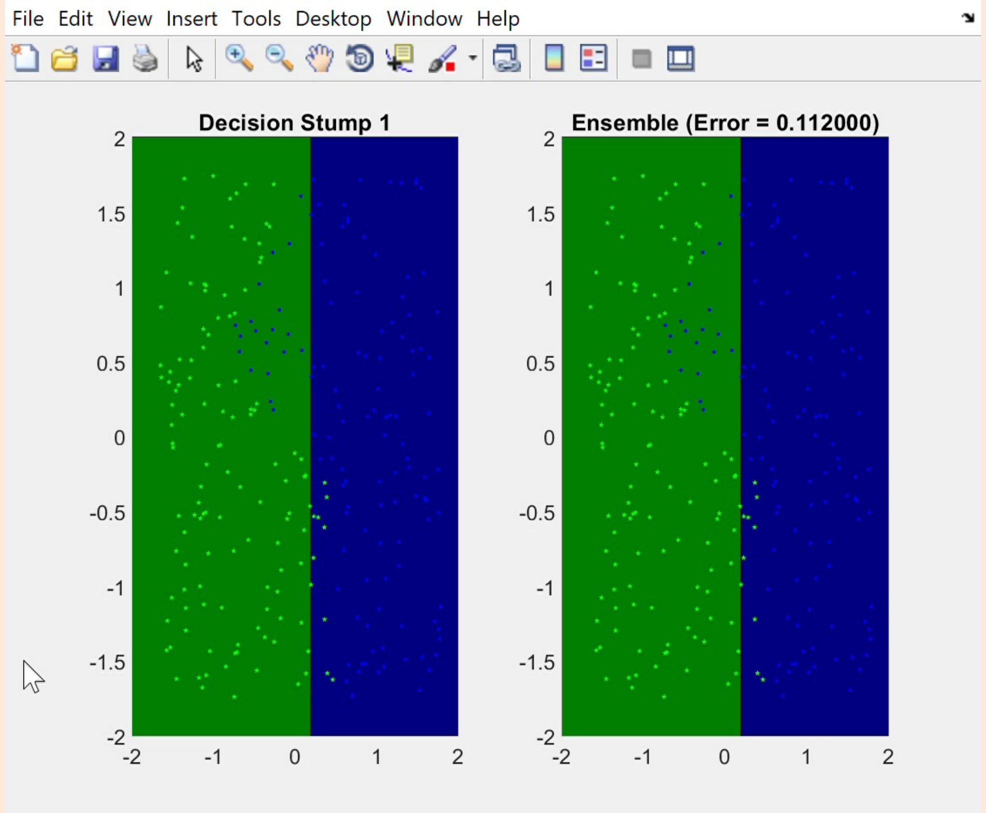
- Final prediction: weighted vote of individual classifier predictions.
  – Trees with higher (weighted) accuracy get higher weight.

- See Wikipedia for precise definitions of weights.
  – Comes from “exponential loss” (a convex approximation to 0-1 loss).

bonus!

AdaBoost with Decision Stumps in Action

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

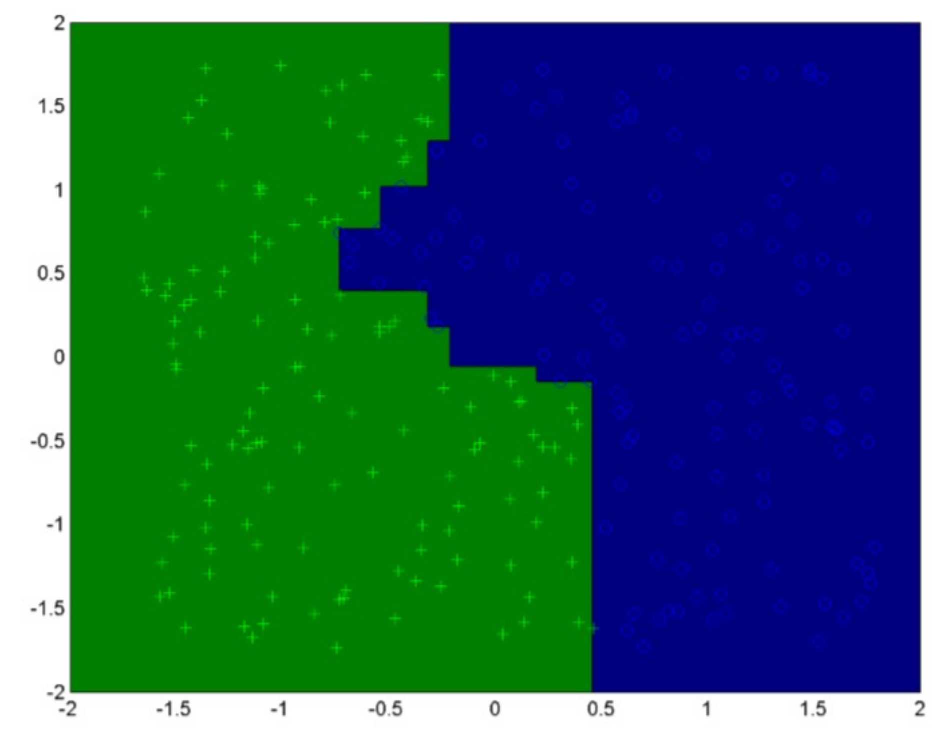


  – Size of training example on left is proportional to classification weight.

bonus!

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

bonus!

- 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.