CPSC 340:
Machine Learning and Data Mining

Boosting

Fall 2019

Previously: Ensemble Methods

- Ensemble methods 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.

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

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.

AdaBoost: Classic Boosting Algorithm

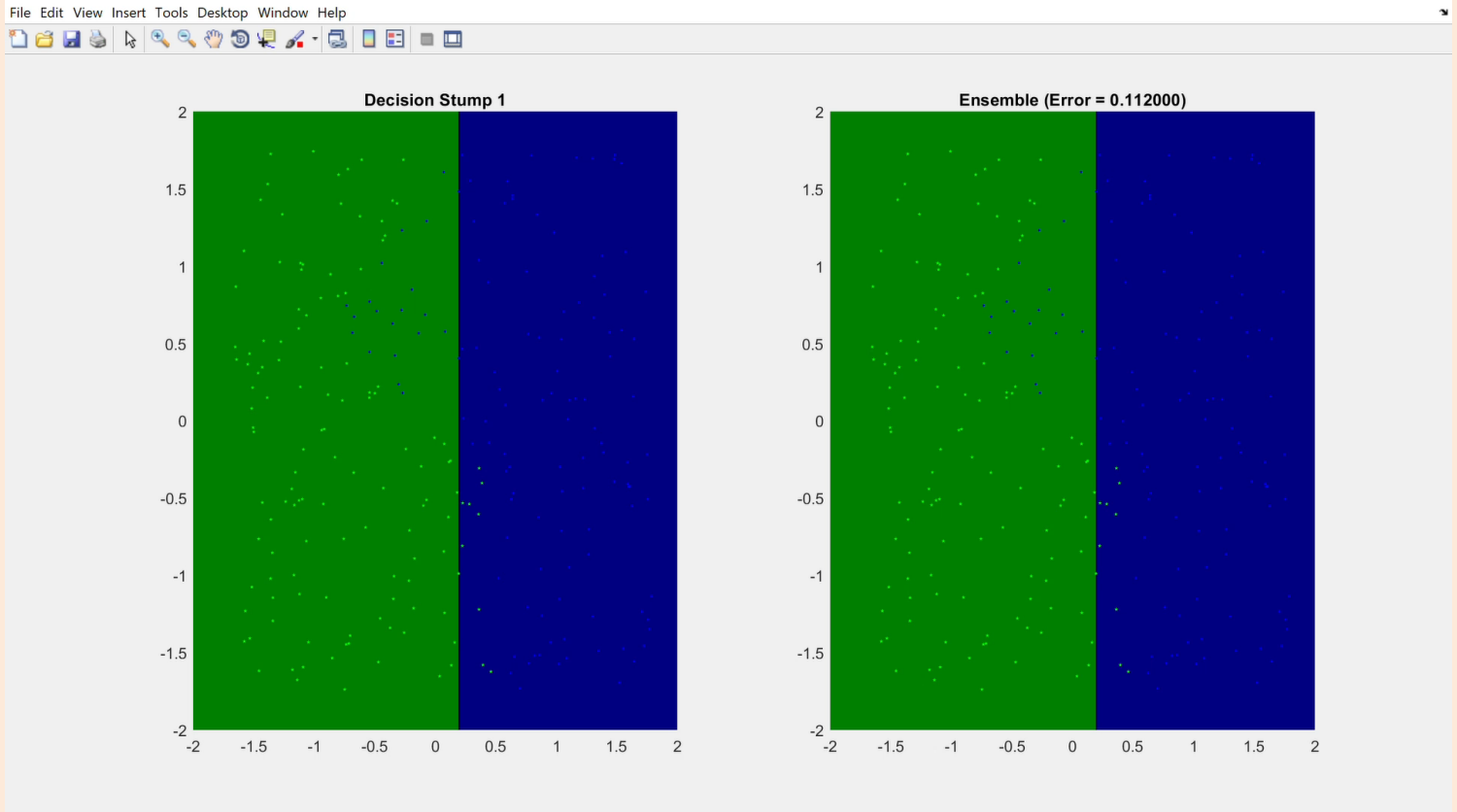
- Overview of AdaBoost:
  1. Fit a classifier on the training data.
  2. Give a higher weight to examples that the classifier got wrong.
  3. Fit a classifier on the weighted training data.
  4. Go back to 2.
     - Weight gets exponentially larger each time you are wrong.

- 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

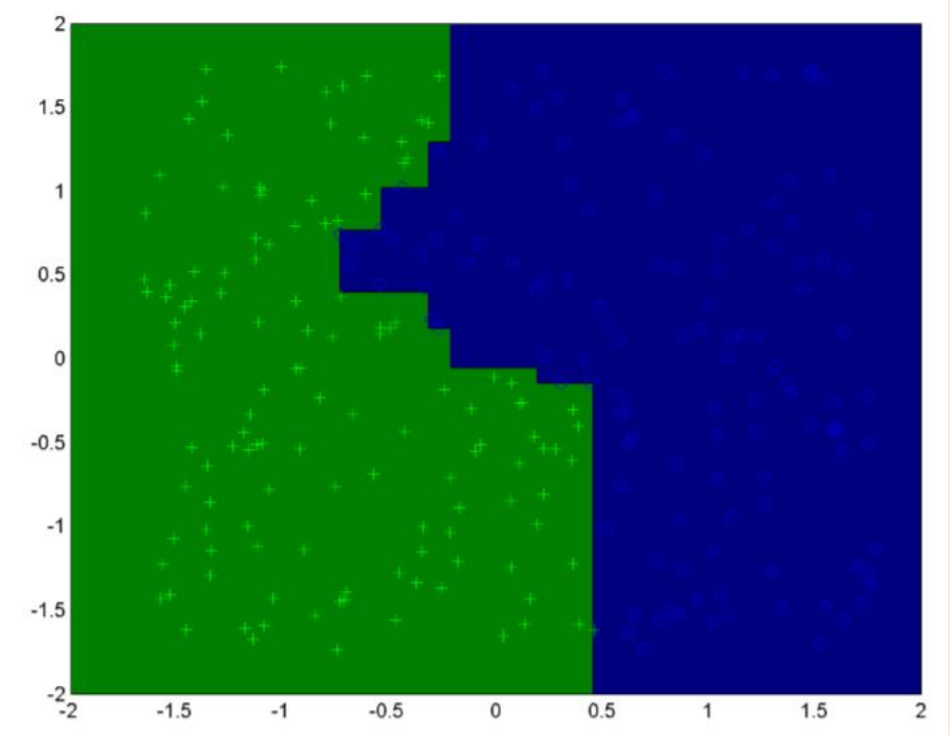
- 2D example of AdaBoost with decision stumps (with accuracy score):



  – 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.

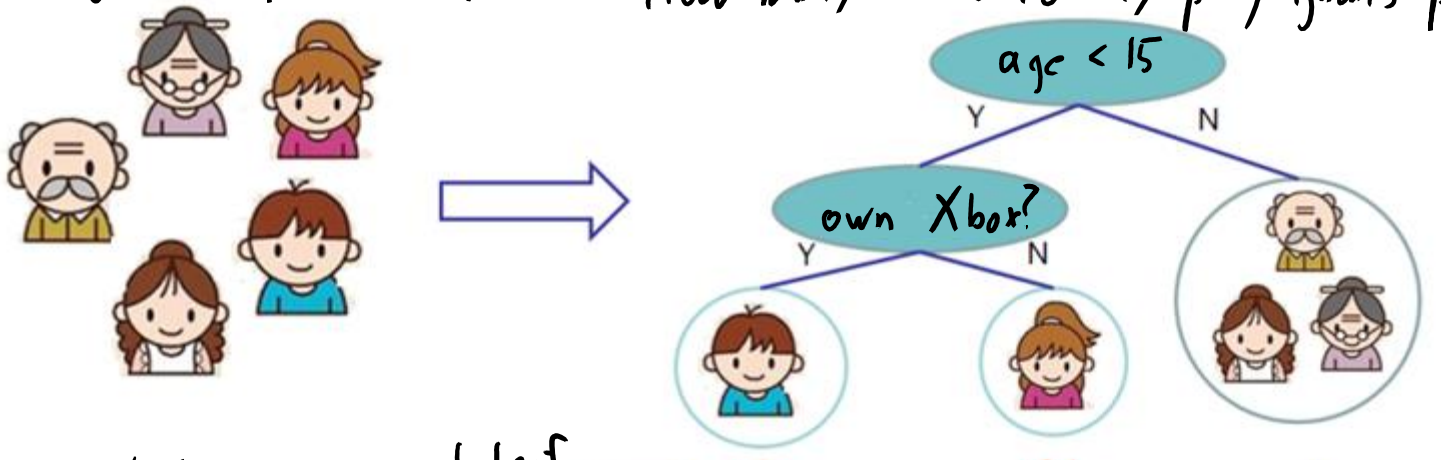
https://xgboost.readthedocs.io/en/latest/tutorials/model.html

Regularized Regression Trees

- Regression trees used in XGBoost:
  - Each split is based on 1 feature.
  - Each leaf 'L' gives a real-valued prediction $\hat{y}_i = w_L$.

Input: age, occupation, city, ...

How many hours do they play games per day?



prediction w_L in each leaf

  - Above, we would predict "2 hours" for a 14-year-old who owns an Xbox.

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

Regularized Regression Trees

- Regression trees used in XGBoost:
  - Fit tree to try to minimize squared error at the leaves:

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

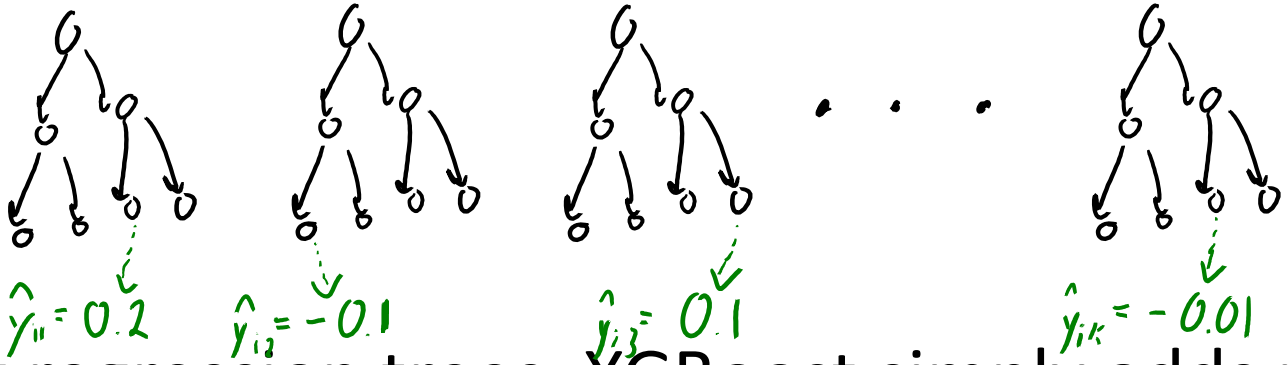
  - Fitting a decision stump with the squared error:
    - Simple closed-form solution at each split: optimal w_L at a leaf is just average of y_i.
    - Same speed as fitting decision trees from Week 2 (use mean instead of mode).
  - Use greedy strategy for growing tree, as in Part 1.
  - To restrict complexity, add L0-regularization (stop splitting if $w_L = 0$).

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

    - "Only split if you decrease squared error by λ_0."
  - To further fight overfitting, XGBoost also adds L2-regularization of 'w'.

XGBoost Prediction

- XGBoost fits a set of regression trees.
  - For an example 'i', they each make a continuous prediction:



- After fitting regression trees, XGBoost simply adds predictions:

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

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

- Unlike random forests, trees are not trained independently.
  - During training, each tree tries to "fix" the previous trees' predictions.

XGBoost Training

- Start with a prediction of "$\hat{y}_i = 0$" for all 'i'.
- Fit the first tree with the true labels:

$$\text{Greedily minimize } f(w_1, w_2, \ldots) = \sum_{i=1}^{n} (\hat{y}_{i1} - y_i)^2 + \lambda_0 \|w\|_0 + \lambda_2 \|w\|^2$$

w_L at leaf

  - Our new prediction is "$\hat{y}_i = \hat{y}_{i1}$".
- Fit the second tree to minimize residual based on old prediction.

$$\text{Greedily minimize } f(w_1, w_2, \ldots) = \sum_{i=1}^{n} (\hat{y}_{i2} + \hat{y}_i - y_i)^2 + \lambda_0 \|w\|_0 + \lambda_2 \|w\|^2$$

w_L at leaf old prediction (now fixed)

  - The w_L in second tree are trying to predict residual ($\hat{y}_i$-y_i) of first prediction.
  - Unless all $w_L = 0$, new prediction "$\hat{y}_i = \hat{y}_{i1} + \hat{y}_{i2}$" has strictly lower training error.

XGBoost Training

- To fit tree 'm' we use:

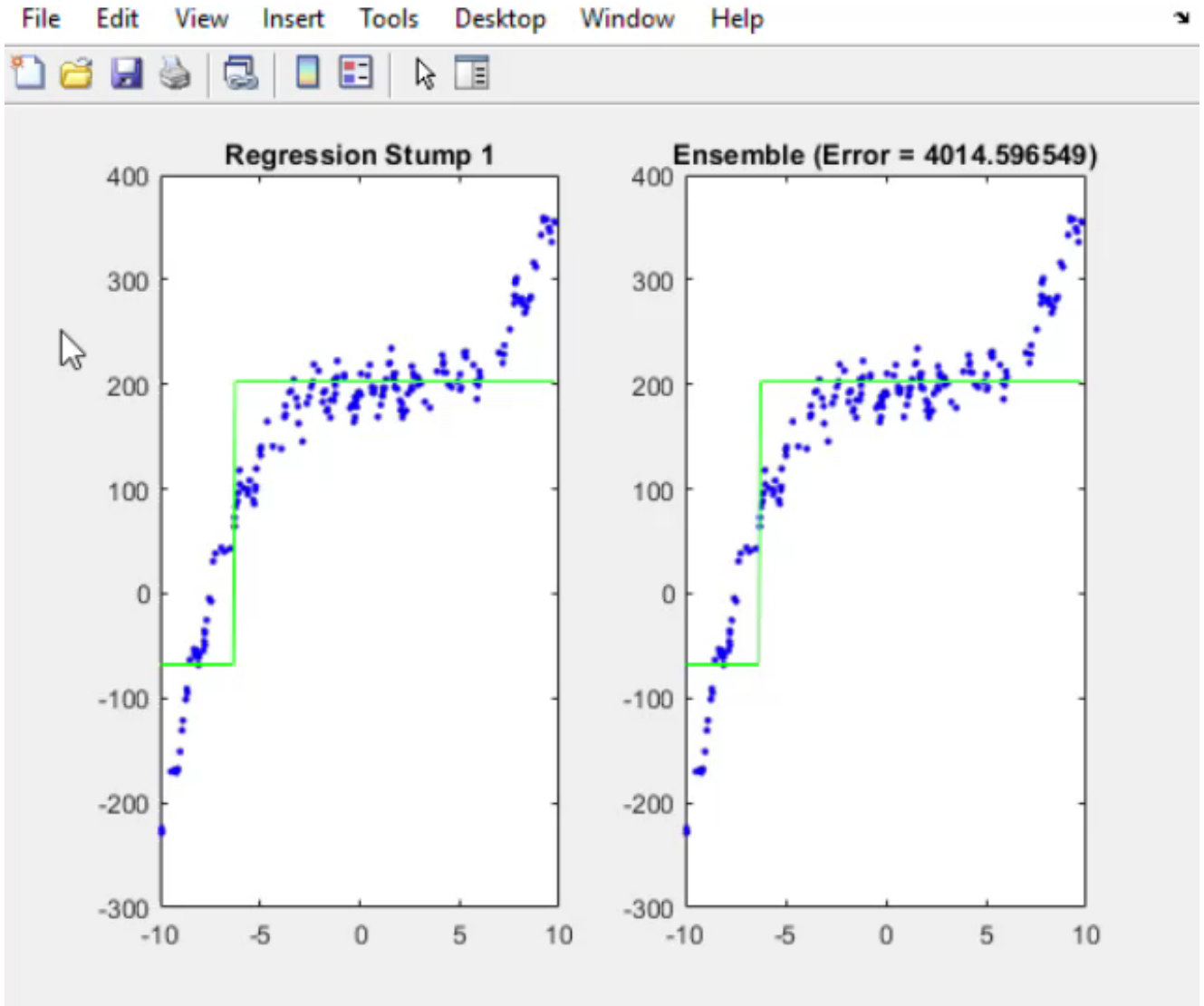
Greedily minimize $f(w_1, w_2, \ldots) = \sum_{i=1}^{n} (\hat{y}_{im} + \hat{y}_i - y_i)^2 + \lambda_0 \|w\|_0 + \lambda_2 \|w\|^2$

w_L at leaf old prediction (now fixed)

  - Here, "$\hat{y}_i = \hat{y}_{i1} + \hat{y}_{i2} + \hat{y}_{i3} + \ldots + \hat{y}_{i(m-1)}$".
  - It's trying to fix the predictions of the first (m-1) trees.
    - "Old prediction is 0.8, true label is 0.9, I'm going to predict 0.1."
    - Training error monotonically decreases with each tree.

- 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).

Regression-Tree Boosting in Action



XGBoost Discussion

- In XGBoost, it's the residuals that act like the weights on examples.
  - Decrease error more by focusing on examples with larger errors.

- Instead of pruning trees if score doesn't improve, grows full trees.
  - And then prunes parts that aren't increasing the score.

- 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).

(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.

The Likelihood Function

- Suppose we have a dataset 'D' with parameters 'w'.

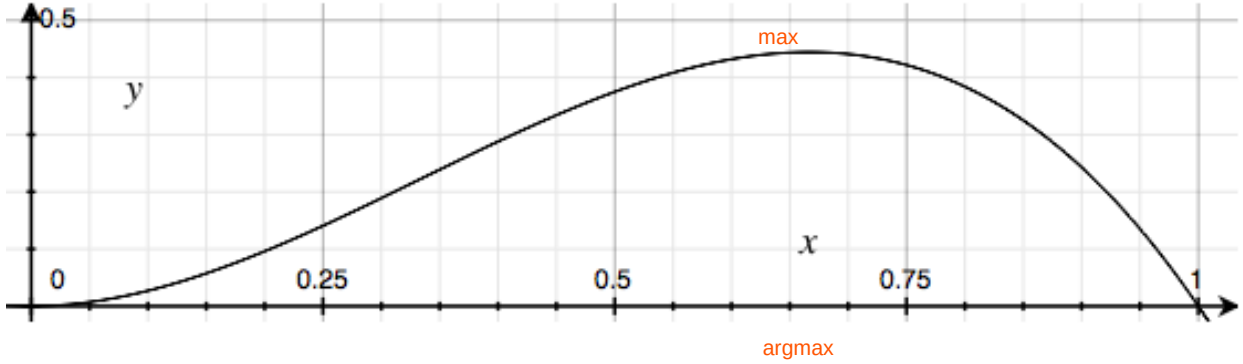
- For example:
  - We flip a coin three times and obtain D={"heads", "heads", "tails"}.
  - The parameter 'w' is the probability that this coin lands "heads".

- We define the likelihood as a probability mass function $p(D \mid w)$.
  - "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 $p(HHT \mid w=0.5) = (1/2)(1/2)(1/2) = 0.125$.
  - If $w = 0$ ("always lands tails"), then $p(HHT \mid w = 0) = 0$ (data is less likely for this 'w').
  - If $w = 0.75$, then $p(HHT \mid w = 0.75) = (3/4)(3/4)(1/4) \approx 0.14$ (data is more likely).

Maximum Likelihood Estimation (MLE)

- We can plot the likelihood p(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: $\hat{w} \in \underset{w}{\operatorname{argmax}} \{p(D \mid w)\}$
    - 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 I put it 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:

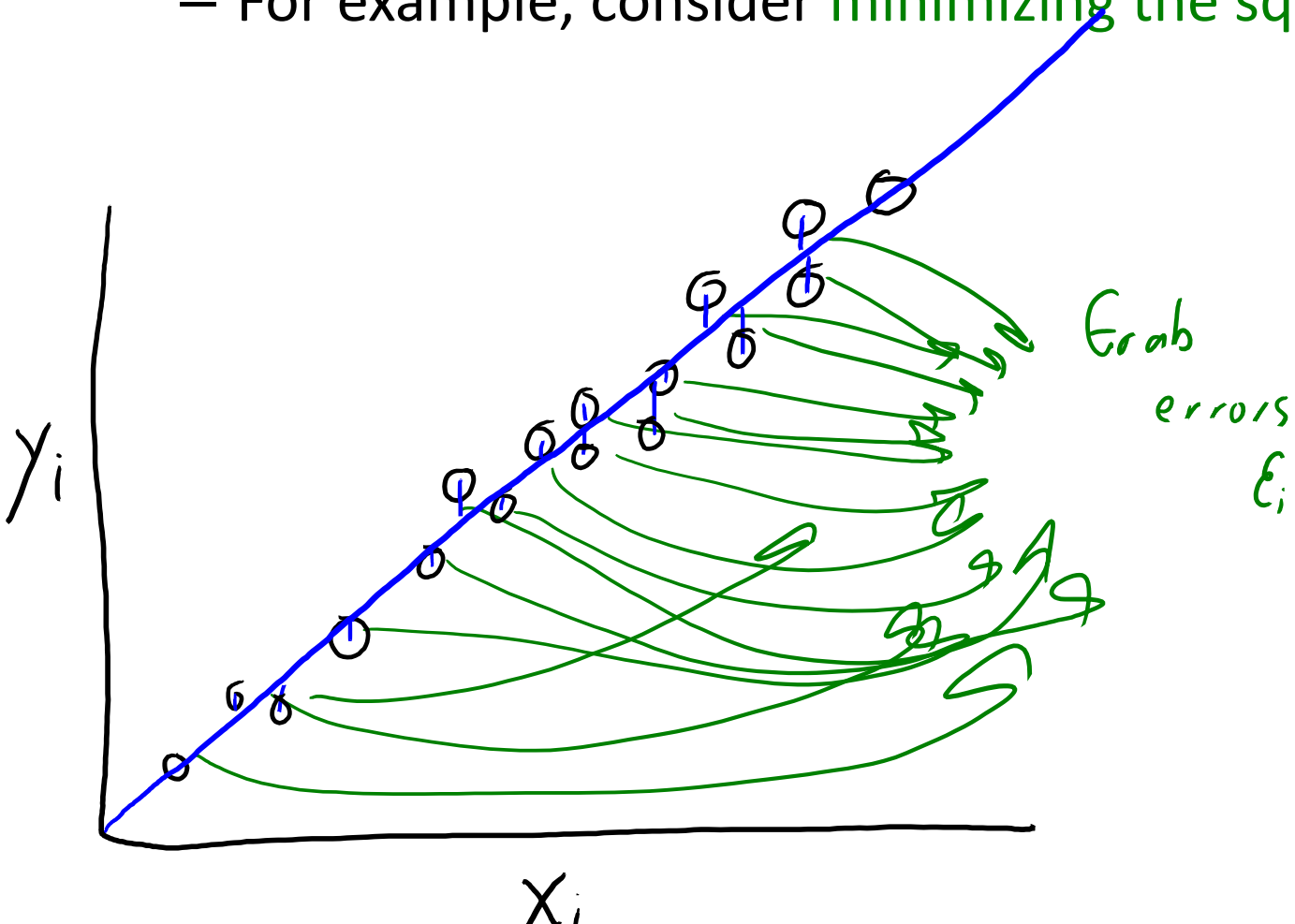
$$y_i = w^T x_i + \varepsilon_i$$

where each ε_i is sampled independently from standard normal

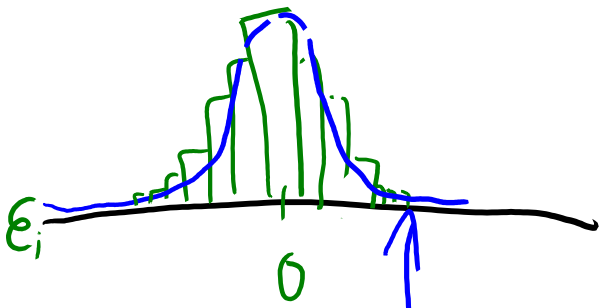
    - "Gaussian" is another name for the "normal" distribution.
  - Remember that least squares solution is called the "normal equations".

Least Squares is Gaussian MLE

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



Grab errors ϵ_i and plot histogram:



Least squares assumes errors come from Gaussian

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.

Minimizing the Negative Log-Likelihood (NLL)

- We use logarithm because it turns multiplication into addition:

$$\log(\alpha\beta) = \log(\alpha) + \log(\beta)$$

- More generally: $\log\left(\prod_{i=1}^{n} a_i\right) = \sum_{i=1}^{n} \log(a_i)$

- If data is 'n' IID samples then $p(D|w) = \prod_{i=1}^{n} \underbrace{p(D_i|w)}_{\text{likelihood of example 'i'}}$

and our MLE is $\hat{w} \in \underset{w}{\operatorname{argmax}} \left\{ \prod_{i=1}^{n} p(D_i|w) \right\} \equiv \underset{w}{\operatorname{argmin}} \left\{ -\sum_{i=1}^{n} \log(p(D_i|w)) \right\}$

Least Squares is Gaussian MLE (Gory Details)

- Let's assume that $y_i = w^T x_i + \varepsilon_i$, with ε_i following standard normal:

$$p(\varepsilon_i) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{\varepsilon_i^2}{2}\right)$$

also known as "Gaussian" distribution

- This leads to a Gaussian likelihood for example 'i' of the form:

$$p(y_i \mid x_i, w) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(w^T x_i - y_i)^2}{2}\right)$$

- Finding MLE (minimizing NLL) is least squares:

$$f(w) = -\sum_{i=1}^{n} \log(p(y_i \mid w, x_i))$$

$$= -\sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(w^T x_i - y_i)^2}{2}\right)\right)$$

$$= -\sum_{i=1}^{n} \left[\underbrace{\log\left(\frac{1}{\sqrt{2\pi}}\right)}_{\text{constant in 'w'}} + \underbrace{\log\left(\exp\left(-\frac{(w^T x_i - y_i)^2}{2}\right)\right)}_{\text{operations cancel}}\right]$$

$$= -\sum_{i=1}^{n} \left[(\text{constant}) - \frac{1}{2}(w^T x_i - y_i)^2\right]$$

$$= (\text{constant}) + \frac{1}{2}\sum_{i=1}^{n} (w^T x_i - y_i)^2$$

$$= (\text{constant}) + \frac{1}{2}\|Xw - y\|^2$$

Loss Functions and Maximum Likelihood Estimation

- So least squares is MLE under Gaussian likelihood.

If $p(y_i \mid x_i, w) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{(w^T x_i - y_i)^2}{2}\right)$

then MLE of 'w' is minimum of $f(w) = \frac{1}{2}\|Xw - y\|^2$

- With a Laplace likelihood you would get absolute error.

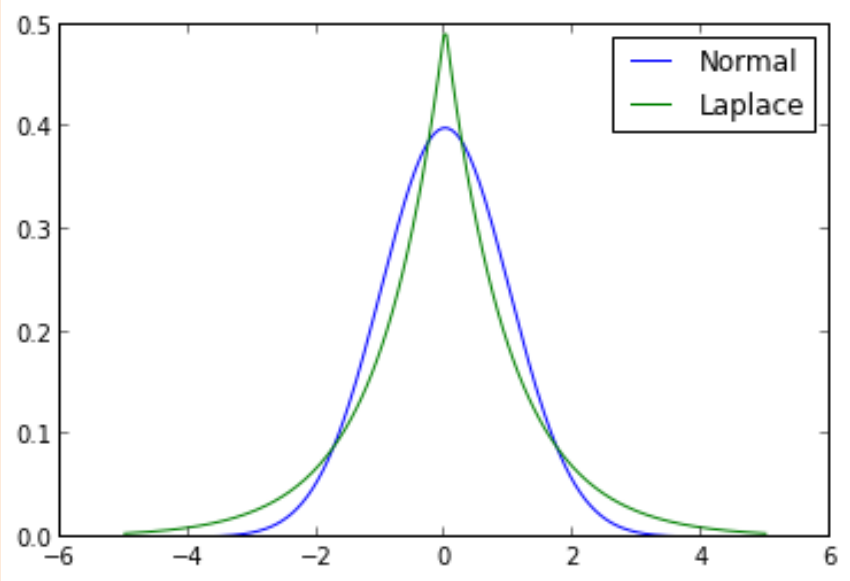
If $p(y_i \mid x_i, w) = \frac{1}{2} \exp(-|w^T x_i - y_i|)$

then MLE is minimum of $f(w) = \|Xw - y\|_1$

- Other likelihoods lead to different errors ("sigmoid" -> logistic loss).

"Heavy" Tails vs. "Light" Tails

- We know that L1-norm is more robust than L2-norm.
  - What does this mean in terms of probabilities?



Here "tail" means "mass of the distribution away from the mean."

  - Gaussian has "light tails": assumes everything is close to mean.
  - Laplace has "heavy tails": assumes some data is far from mean.
  - Student 't' is even more heavy-tailed/robust, but NLL is non-convex.

http://austinrochford.com/posts/2013-09-02-prior-distributions-for-bayesian-regression-using-pymc.html

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 viewpoint of common models.
  - Objective functions are equivalent to maximizing $p(y, X \mid w)$ or $p(y \mid X, w)$.

- Next time:
  - How does regularization fit it?