Boosting

Slowly Learning from a Lot of Smart Mistakes

Mark Llorente, Much of the Content from Various Instructors

galvanize

Morning: Intro to Boosting

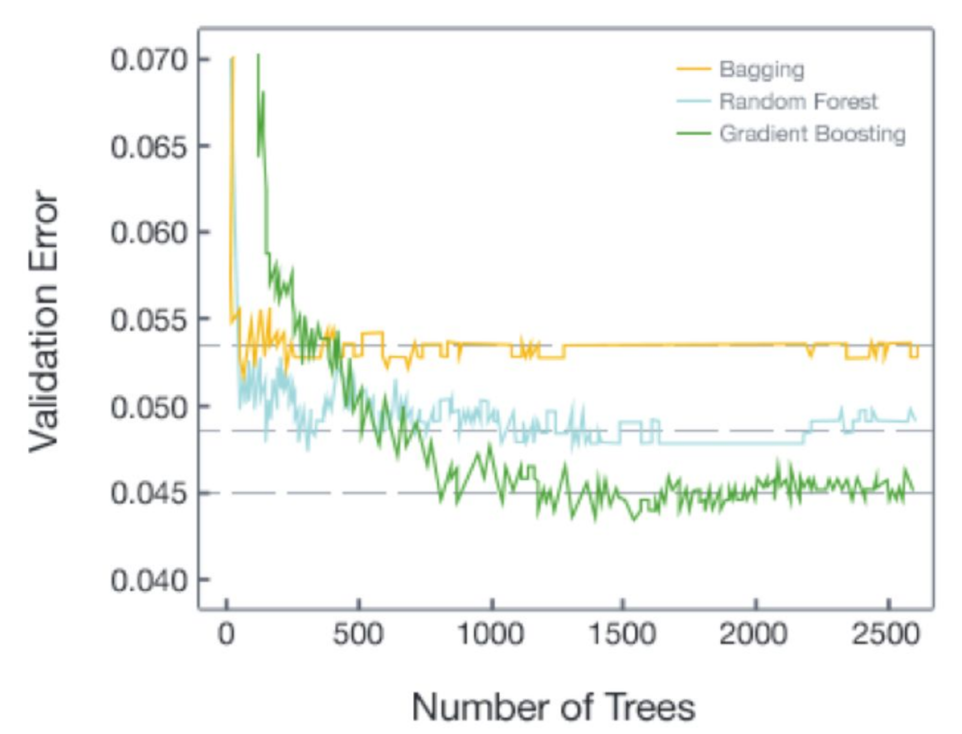
Afternoon: Adaboost vs Gradient Boost, Walking through the Adaboost Algorithm

Morning Objectives

- Contrast Boosting with Random Forests
  - Bias vs Variance
- Be Familiar With the Principles of Boosting
  - Variations on Boosting: Gradient Boosting, Adaptive Boosting (Adaboost), XGBoost
    - Why is it called Gradient Boosting?
  - Why do we use “shallow trees?”
- Know How to Tune Hyperparameters
  - Learning Rate, Number of Trees, Subsampling
    - Depth of trees (Keep Them Short!), and the other standard tree pruning hyperparameters

First Off: Why Boosting?

- Squeeze out the most predictive power from models
    - Boosting can be applied to non-tree models but is most often used with trees
- Resistant to overfitting
- “Smoother” predictions than RF
    - Ensemble of high bias/low-variance predictors
- You prefer to learn from previous mistakes rather than rely on the wisdom of the crowd
    - You’re a rebel who prefers to work alone…



What *Isn't* Boosting?

TOP DEFINITION

Boosting

To steal **retail** items and resell them **on the black market**.

*Women/Men selling **knockoff** purses in **the street** are **boosting**.*

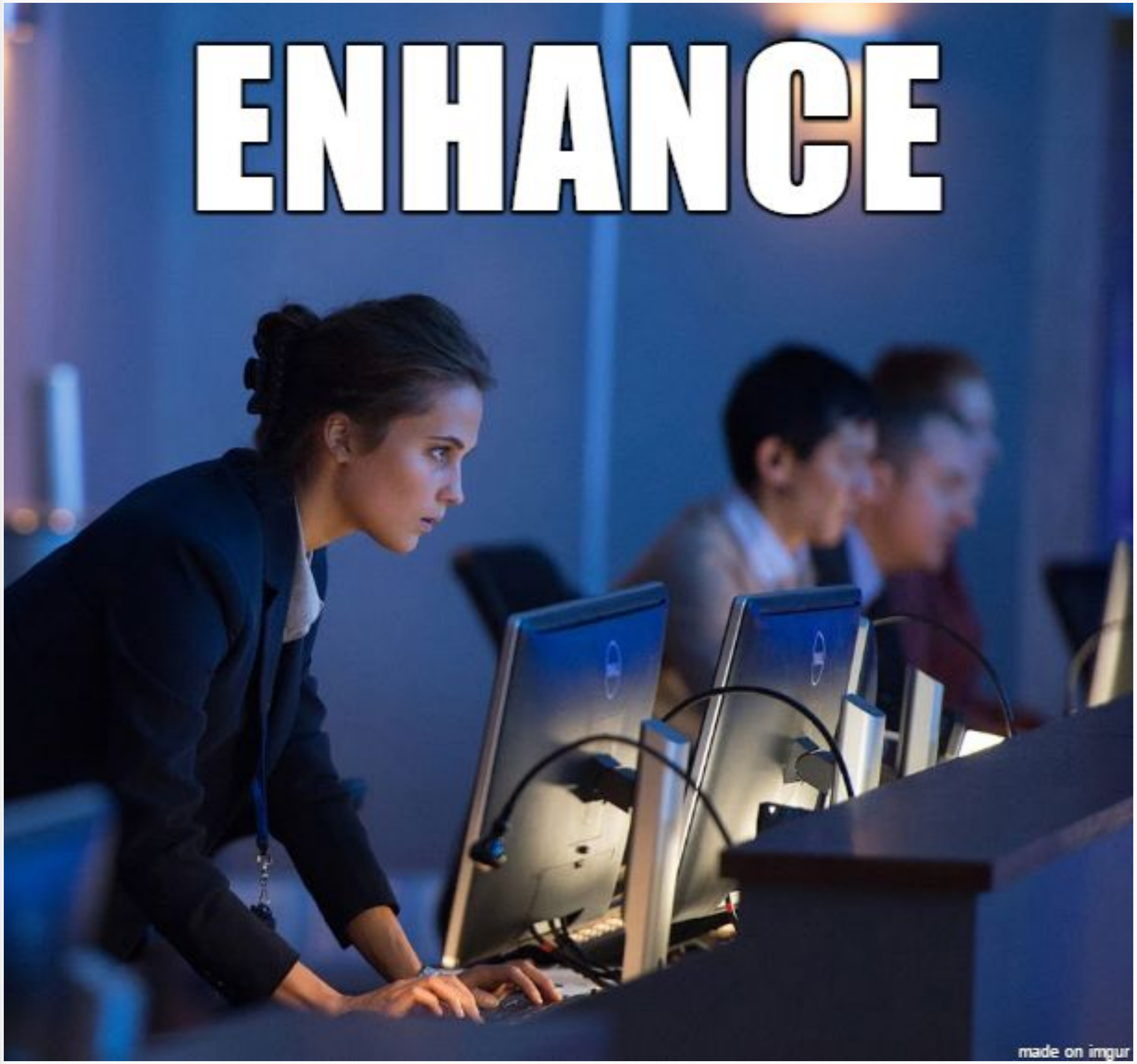
#thief #steal #stolen #kleptomaniac #knockoff

by Goe September 13, 2015

43 7

What *Is* Boosting?

More like this



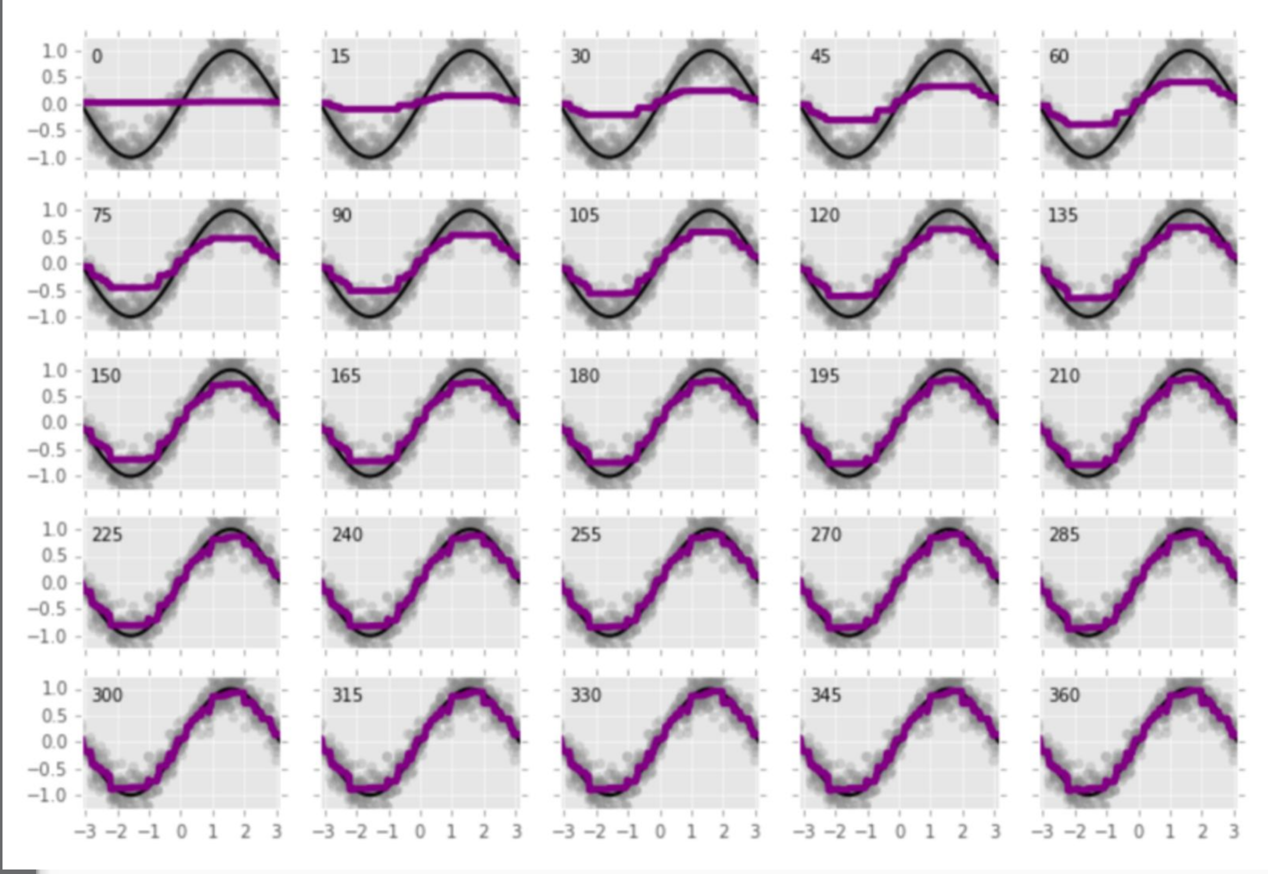
galvanize

What *Is* Boosting?

Many weak approximations added in stages, each stage focusing on what was missed by all previous stages.

“Learning off residuals”

Gradient Boosting Regressor (with small learning rate)

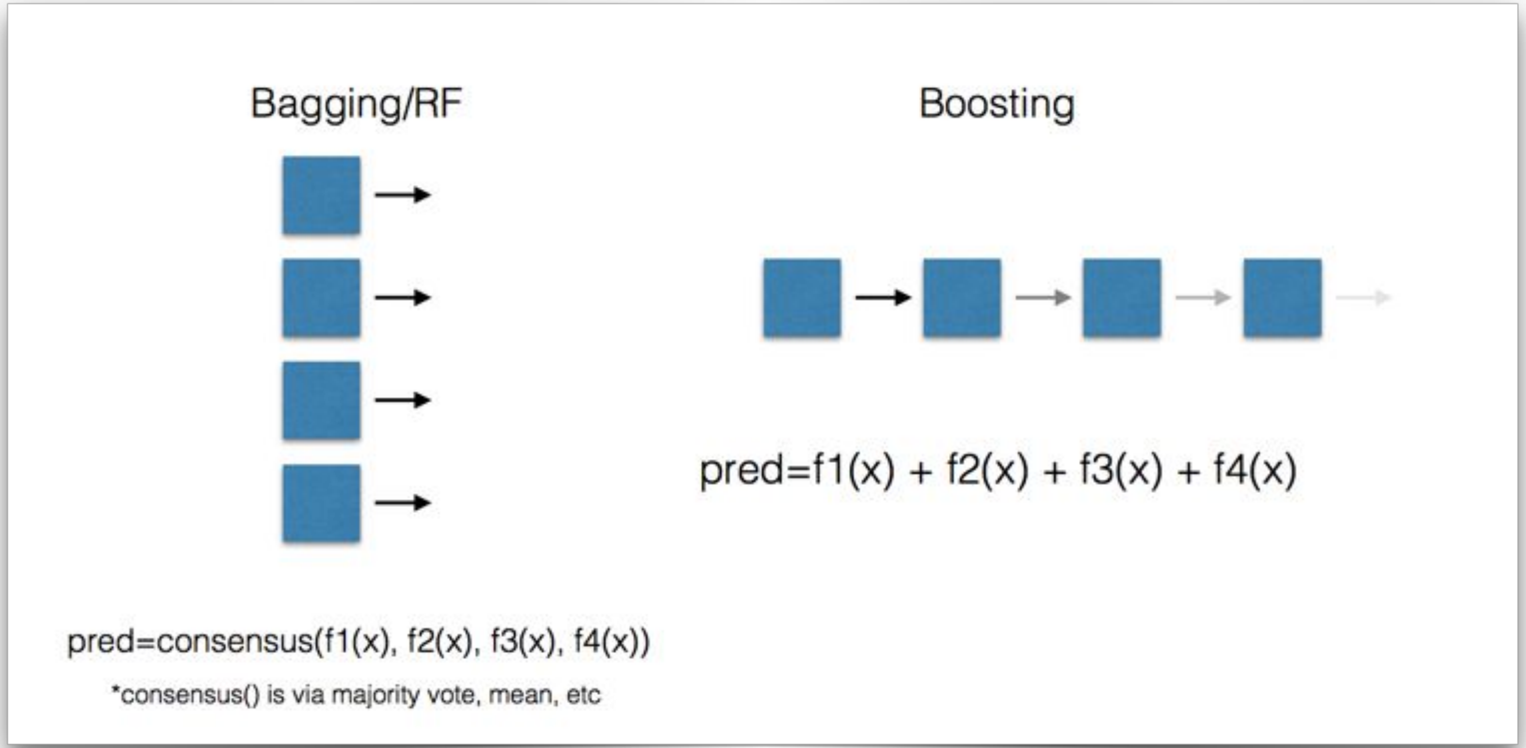


galvanize

What *Is* Boosting?

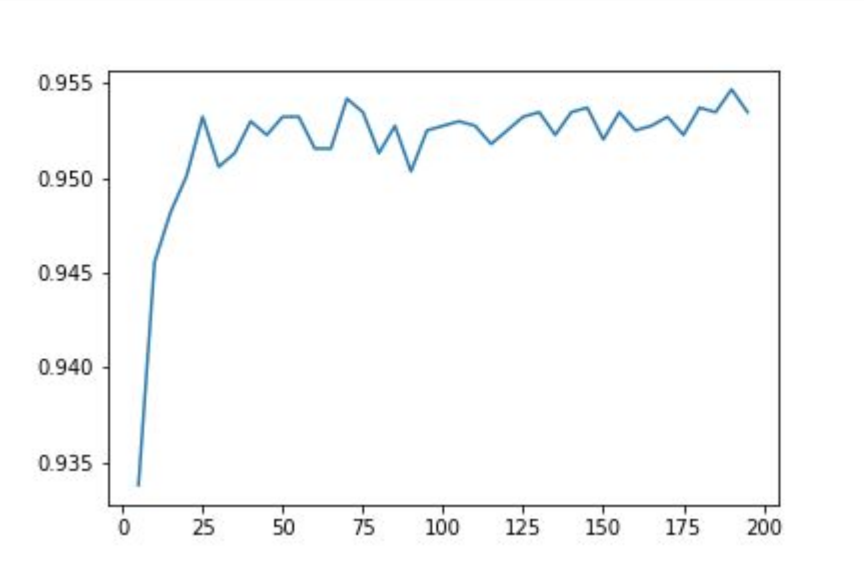
Random Forest (a lot of clever trees)
Can be trained in parallel (multi-core).

Boosting (a lot of not clever trees)
Cannot be trained in parallel!

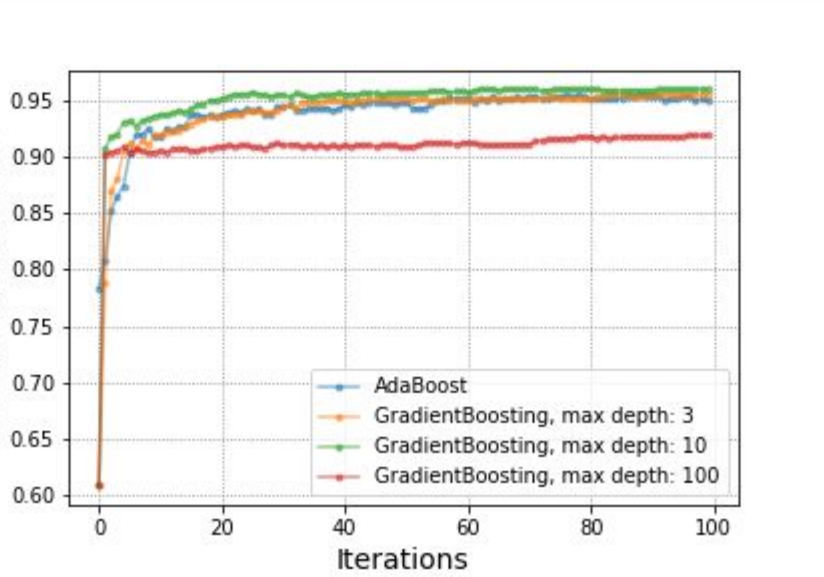


RF and GB Classification Scores with Increasing Number of Trees

Random Forest Test Accuracy



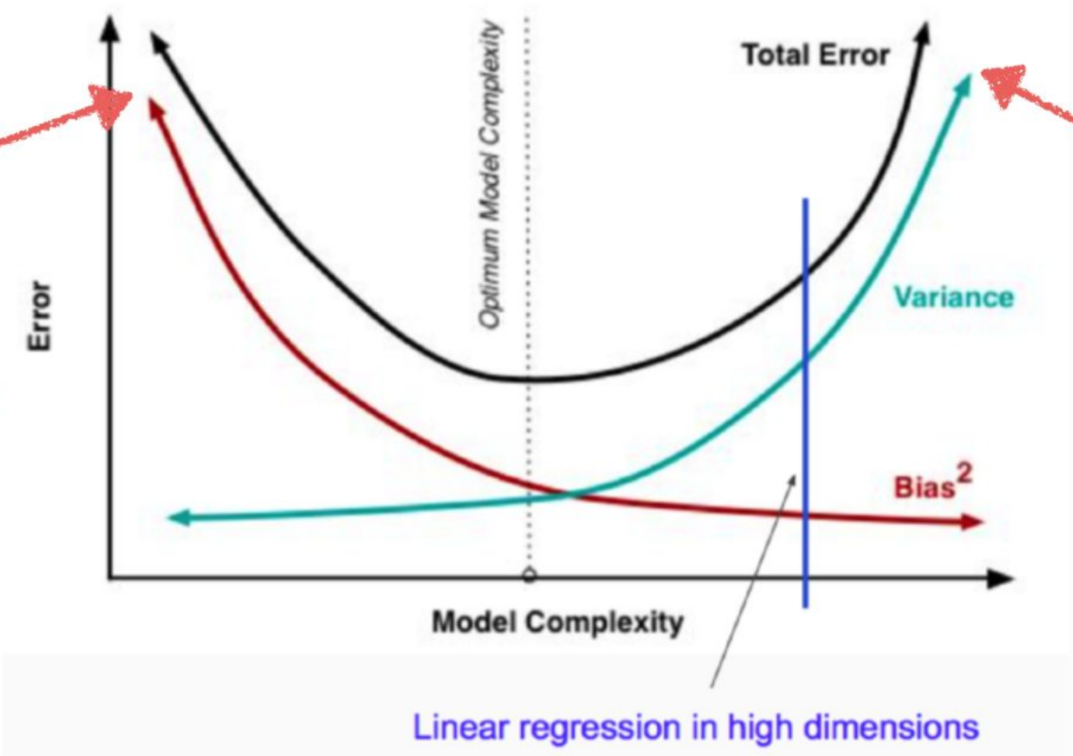
Gradient Boost Test Accuracy



RF vs GB “Starting Points”

Boosting:

start here and move right: start with high Bias low Variance, and chip away at Bias with sequential additions to weakest areas

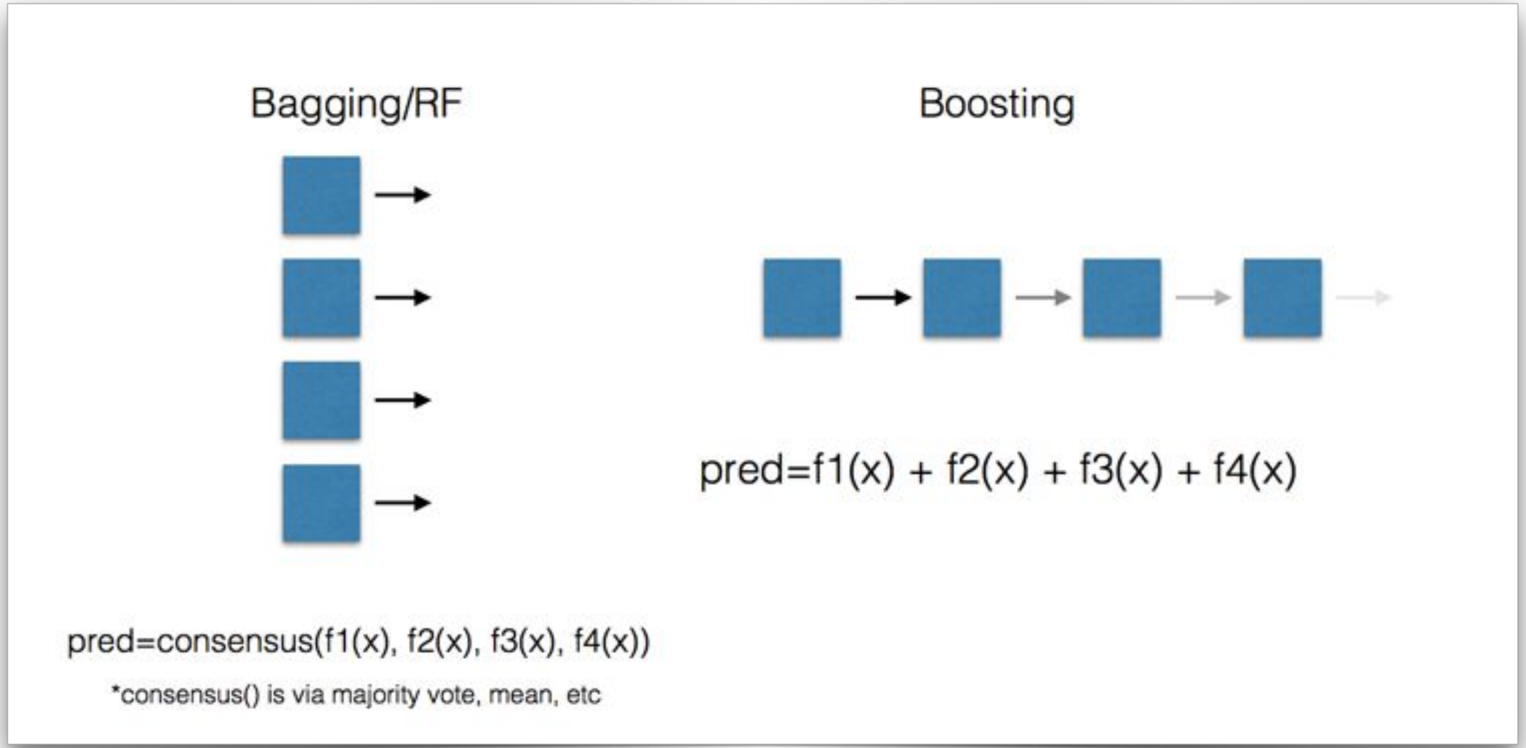


Bagging, Random Forests, and, actually, most models:

start here and move left: start with high Variance low Bias, reign Variance in by averaging. RF: averaging highly correlated quantities doesn’t reduce variance as much as averaging uncorrelated quantities, so de-correlate the trees by restricting predictors available

Random Forest vs Boosting

Random Forest (and Bagging) trees can be trained in parallel. Boosted trees must be trained in series.



In the end, f will be a sum of smaller (often called *weak*) learners

$$f(x) = f_0(x) + f_1(x) + f_2(x) + \cdots + f_{max}(x)$$

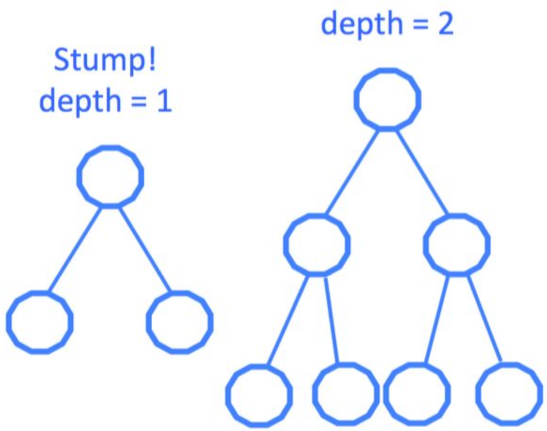
The process of building up the model looks like

$$\begin{aligned}
S_0(x) &= f_0(x) \\
S_1(x) &= f_0(x) + f_1(x) \\
S_2(x) &= f_0(x) + f_1(x) + f_2(x) \\
&\vdots \\
S_{max}(x) &= f_0(x) + f_1(x) + f_2(x) + \cdots + f_{max}(x)
\end{aligned}$$

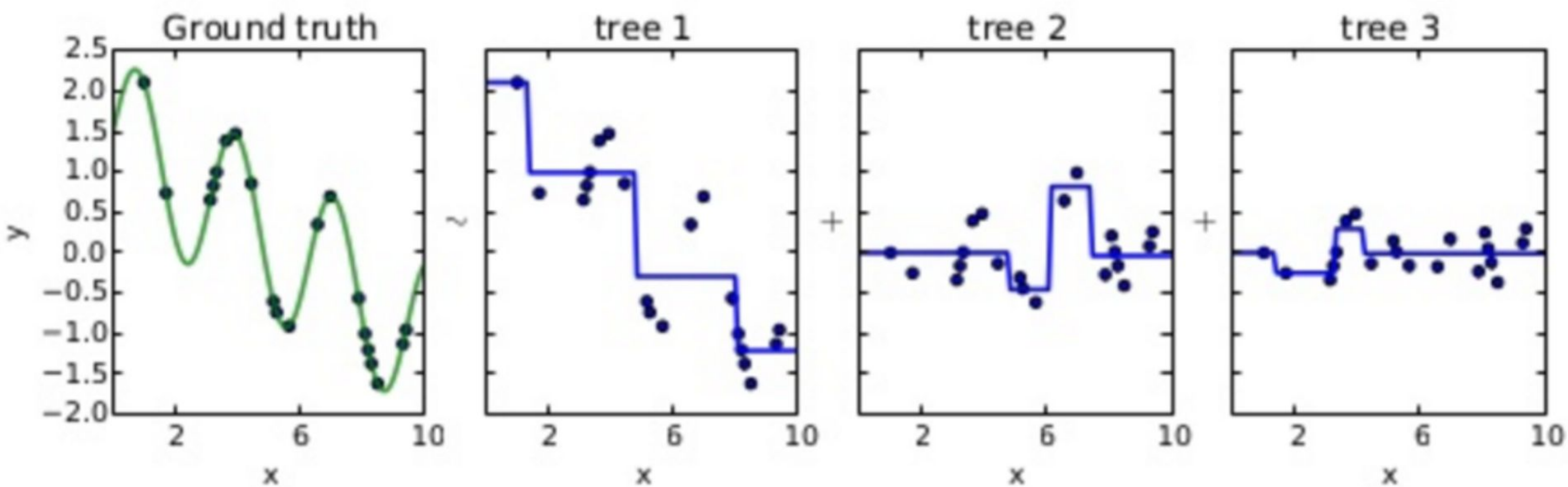
To the Poll(s)!

galvanize

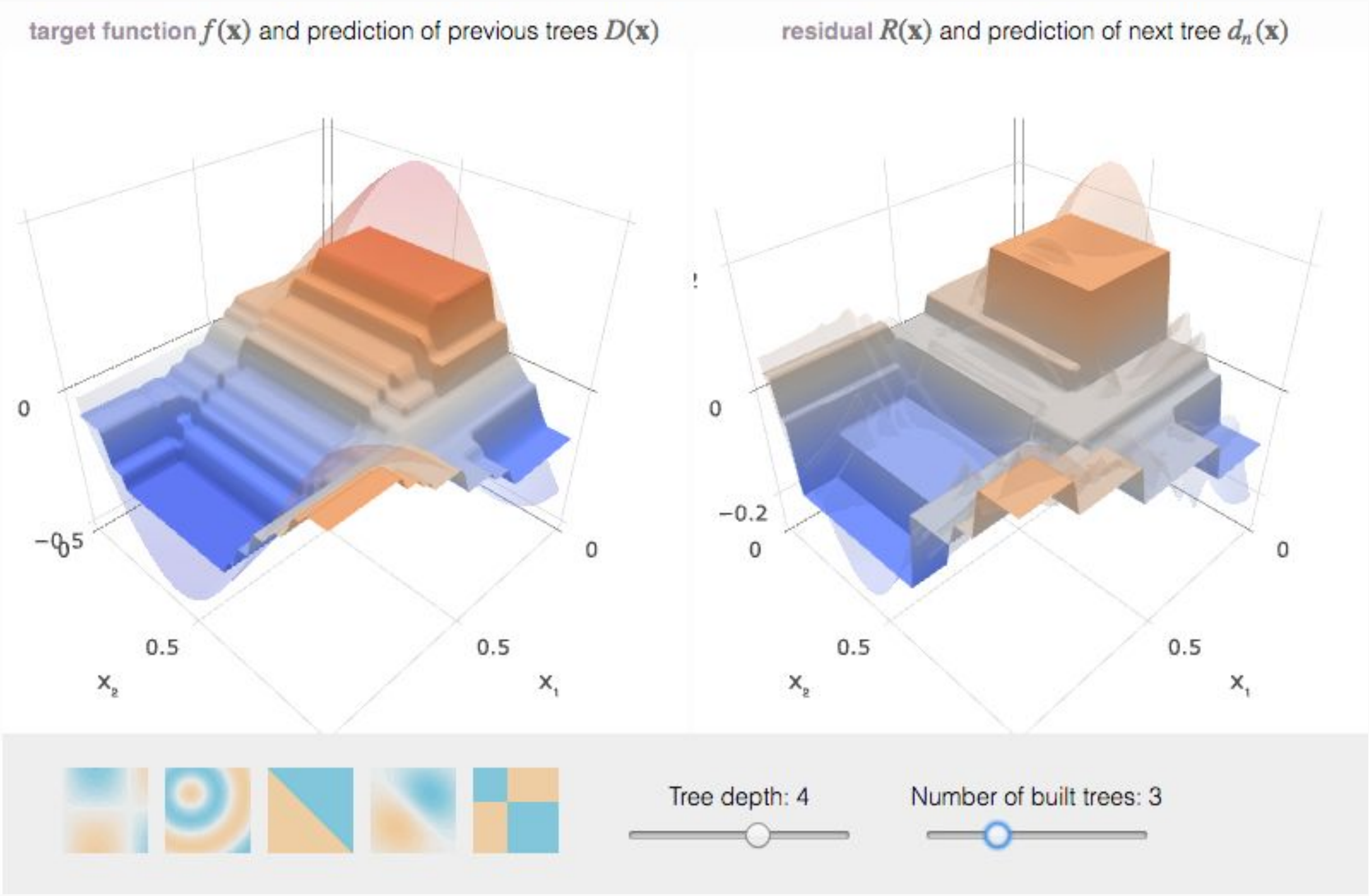
Fitting regressions on sequential residuals



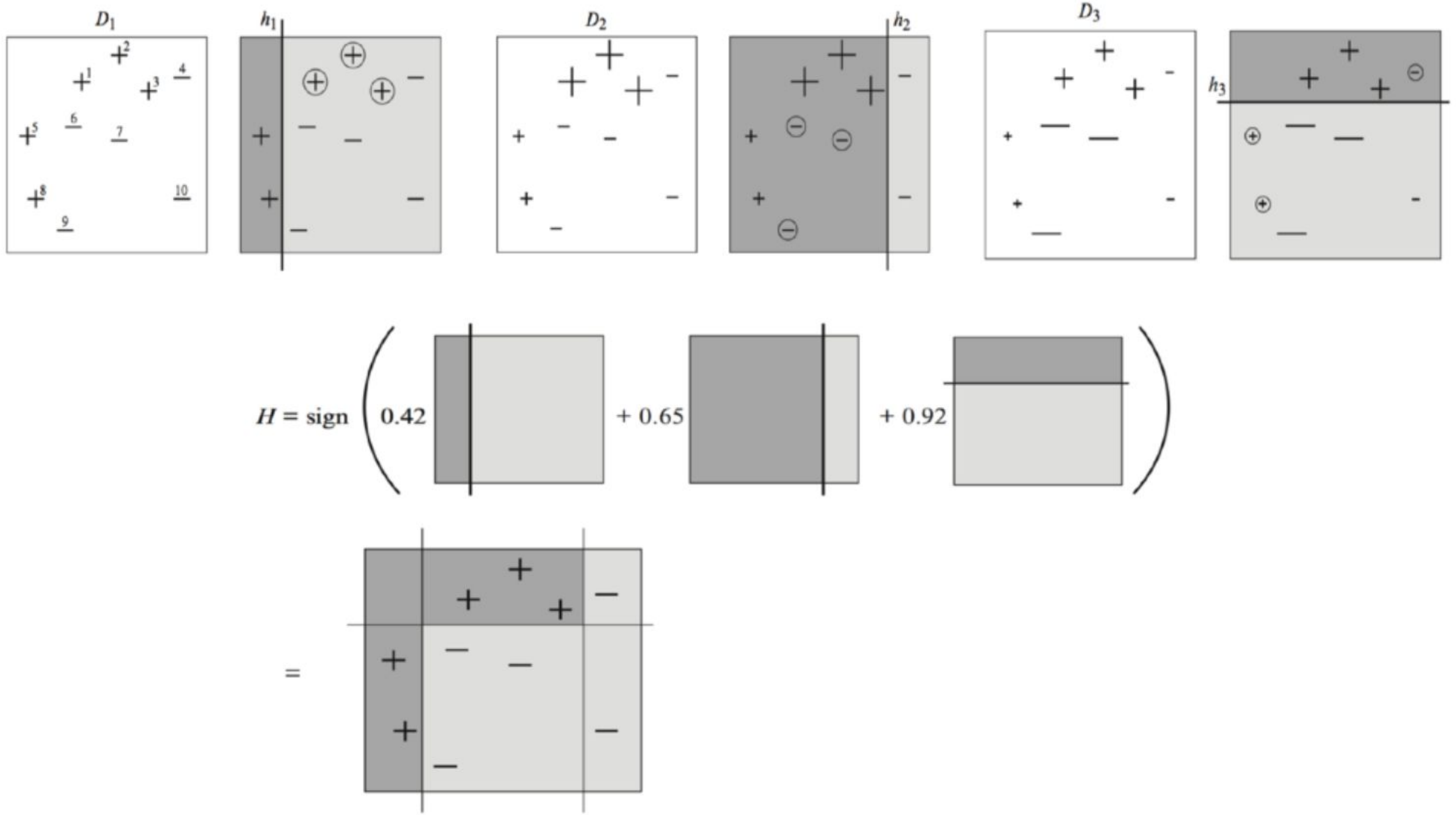
1. Set $\hat{f}^{(0)}(\boldsymbol{x}_i) = 0$ and $\hat{\epsilon}_i^{(1)} = Y_i$
2. For $k = 1, \cdots m$
   - 2.1 Fit a tree $\hat{f}^{(k)}$ to $\hat{\epsilon}^{(k)}$ using features $\boldsymbol{x}$
   - 2.2 Update the estimator $\hat{f}^{(k+1)} = f^{(k)} + \alpha_k f^{(k-1)}$
   - 2.3 Update the residuals $\hat{\epsilon}_i^{(k+1)} = \hat{\epsilon}_i^{(k)} - \alpha_k f^{(k)}(\boldsymbol{x}_i)$
3. Return the boosted model $\hat{f}(\boldsymbol{x}_i) = \sum_{k=1}^{m} \alpha_k f^{(k)}(\boldsymbol{x}_i)$



https://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html



Classification with “Stumps” using Adaboost



- # Boosting Algorithm

Algorithm 8.2 *Boosting for Regression Trees*

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.

2. For $b = 1, 2, \ldots, B$, repeat:

   (a) Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r). The (view of the) 'training data' changes below

   (b) Update $\hat{f}$ by adding in a shrunken version of the new tree:

   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x). \tag{8.10}$$

   (c) Update the residuals,

   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i). \tag{8.11}$$

3. Output the boosted model,

$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x). \tag{8.12}$$

Gradient Boosting as Adapted from Dan Wiesenthal

- Start with a model predicting only 0, whether for classification or regression.
  - That means no trees yet! $f(x) = 0$
- Set our first "residuals" to be our original y values:
  - Why? If we predicted all zeros, we get y back. $y - f(x) = y - 0 = y$
  - We still haven't updated $f(x)$.
- For each of the B trees we want to train:
  - Fit a *new* tree, $f^b(x)$, to the *current* residuals
    - Residuals = y only at the beginning. Residuals should continue to decrease as we fit more trees to them!
    - Note: We limit each tree's depth so we don't overfit
  - Combine previous $f(x)$ with $f^b(x)$ to update it but with a small weight applied to f^b so our "step size" isn't too big:
    - $f(x) \mathrel{+}= \lambda f^b(x) = 0 + \lambda f^1(x) + \lambda f^2(x) + \ldots + \lambda f^b(x)$
    - Update the residuals by using the new 'ensemble' **and repeat**!
      - $r_{current} = y - f_{current}(x)$ or $r \mathrel{-}= \lambda f^b(x)$
- Return $f(x)$, the weighted sum of all B trees

Common Types of Boosting

- Adaptive Boosting, Adaboosting
  - First boosting algorithm circa 1997
  - Binary Classification Only
  - *Adaptively* updates new tree weights based on size of total errors were AND *Adaptively* updates data point weights to focus on where errors were made
  - *Not* the strongest boosting algorithm but still used
- Gradient Boosting
  - A generalized form of Adaboost that can use different loss functions
  - Called gradient boosting because we're implicitly using gradient descent to update our model with trees that help us minimize our loss!
    - Not just a metaphor. More info at end of presentation.
- XGBoost
  - Regularly used in Kaggle Competitions for Big Data
  - Cleverly only selects data to look at if they represent a percentile for a given feature.
  - Avoids needing to evaluate every data point and adds bonus "regularization"
  - Still gradient boosting

Common Types of Boosting

- CatBoosting, Category Boosting
  - First published paper on algorithm in 2017 (one of the newest hot algorithms)
  - Handles categorical variables using statistical relationships between categories and target to do mid-training one-hot encoding on data to maximize signal and minimize creating too many new columns
  - Doesn’t require much tuning as it claims to be self-tuning, but takes an order of magnitude or more time to train compared to other models
  - Uses specially designed “symmetric trees”

	CatBoost		LightGBM		XGBoost		H2O	
	Tuned	Default	Tuned	Default	Tuned	Default	Tuned	Default
Adult	**0.26974**	0.27298 +1.21%	0.27602 +2.33%	0.28716 +6.46%	0.27542 +2.11%	0.28009 +3.84%	0.27510 +1.99%	0.27607 +2.35%
Amazon	**0.13772**	0.13811 +0.29%	0.16360 +18.80%	0.16716 +21.38%	0.16327 +18.56%	0.16536 +20.07%	0.16264 +18.10%	0.16950 +23.08%
Click prediction	**0.39090**	0.39112 +0.06%	0.39633 +1.39%	0.39749 +1.69%	0.39624 +1.37%	0.39764 +1.73%	0.39759 +1.72%	0.39785 +1.78%
KDD appetency	0.07151	**0.07138** −0.19%	0.07179 +0.40%	0.07482 +4.63%	0.07176 +0.35%	0.07466 +4.41%	0.07246 +1.33%	0.07355 +2.86%
KDD churn	**0.23129**	0.23193 +0.28%	0.23205 +0.33%	0.23565 +1.89%	0.23312 +0.80%	0.23369 +1.04%	0.23275 +0.64%	0.23287 +0.69%
KDD internet	**0.20875**	0.22021 +5.49%	0.22315 +6.90%	0.23627 +13.19%	0.22532 +7.94%	0.23468 +12.43%	0.22209 +6.40%	0.24023 +15.09%

More polls!

galvanize

Tuning Hyperparameters of Standard Gradient Boosting

Learning rate: Just like step-size in Gradient Descent. The smaller, the smoother the final forest will predict and the more accurate it will be BUT at the cost of needing way more trees.

Number of trees: Use a lot of trees and you can stop when you're happy with the plateau or you start seeing a trend toward overfitting (happens easily with high tree depth or high learning rates)

Depth: Keep it simple! Increasing depth may tease out feature interactions but too deep and you lose the benefit of a clumsy stubby tree. Stumps are more than okay!

Subsample: Somewhere between 0.5-0.9 is fine. Not necessary but may help depending on your dataset.

Tuning Hyperparameters

In Practice:

Start with relatively high learning rates (~0.1) and grid search over other parameters, primarily to find a nice happy tree depth and a good value for subsampling.

Change to a MUCH smaller learning rate, 0.001 for example. Train with so MANY trees. Your computer may need to run overnight to finish.

Wake up to a very powerful model (and hopefully not an error warning sign).

Pros and Cons

Pros: Powerful and relatively easy to interpret. As with most decision tree based methods, can handle data from mixed sources and different scales. Only a few hyperparameters, primarily learning rate, which can dramatically improve your model output. Adaboost easy to use right out of the box. Variations of GB are used in Kaggle competitions.

Cons: Can't be trained in parallel like Random Forest models. Can eventually cause overfitting if you use too many trees, but you can always choose to get rid of those offending trees.

Individual Sprint

Compare performances for a few different forest models and explore what happens as you adjust hyperparameters!

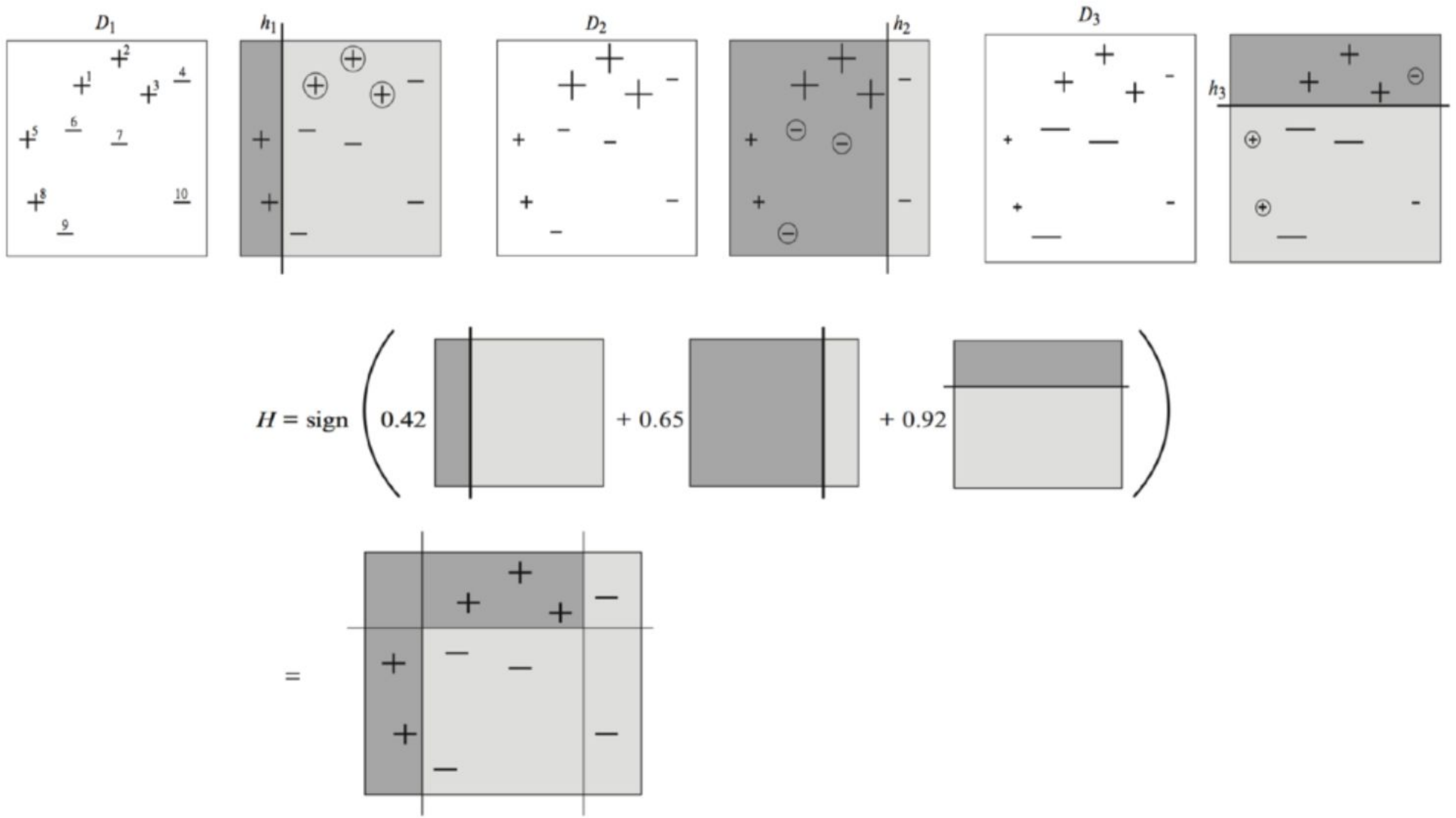
Afternoon Objectives

- Interpret our new friend for the sprint, the Adaboost (Adaptive Boosting) algorithm
- That's it. It looks complicated but it's just a few tweaks from a normal classifier problem with some math in the way.

First: More Polls!

galvanize

Classification with "Stumps" using Adaboost



Algorithm 10.1 *AdaBoost.M1.*

1. Initialize the observation weights $w_i = 1/N,\ i = 1, 2, \ldots, N$.
2. For $m = 1$ to M:
   (a) Fit a classifier $G_m(x)$ to the training data using weights w_i.
   (b) Compute
   $$\mathrm{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \mathrm{err}_m)/\mathrm{err}_m)$.
   (d) Set $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))],\ i = 1, 2, \ldots, N$.
3. Output $G(x) = \mathrm{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

galvanize

Parts of the Algorithm

$G_m(x)$ = Prediction from Decision Tree number m

> **Important Note: The algorithm requires y to be either -1 or +1 instead of 0 or +1 for the two classes! Your sklearn decision tree uses 0 and 1.**

w_i=weight for datapoint i

> Start with uniform weights 1/N and update weight values using formula and use in next tree.
> You can pass newly updated sample weights to a DecisionTreeClassifier using the sample_weight keyword argument of the .fit() method.

$I(g_k(x_i) \neq y_i)$ Indicator function. This one indicates when an error is found! Returns 1 when inside is **True** (i.e. when label and prediction *don't* match!) and 0 when inside is **False** (i.e. *correct* prediction).

Parts of the Algorithm

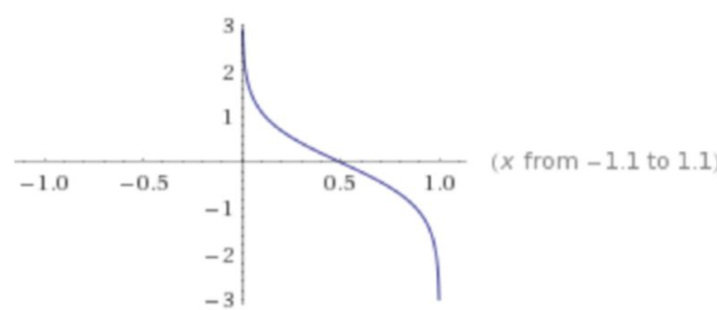
α_k= the tree weight, how much weight we give the *prediction* from a particular tree to our total prediction. Each tree gets one to say how “important” it is to the prediction.

$$\alpha_t = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_t}{\varepsilon_t} \right)$$

Input:

$$0.5 \log\left(\frac{1-x}{x}\right)$$

Plots:



Tips for Afternoon Sprint

- Reminder: We're using sklearn's standard decisions tree classifiers as the base Adaboost classifier. It uses 0 and 1 for the two classes but the algorithm needs -1 and 1 to calculate and update the weights.
    - If y = either 0 or 1, (2*y-1) will give: -1 for y==0, +1 for y==1.
- The final model *also* assumes we're using the sign of the output that can span -1 to 1 as the way to discern between the two classes. You'll need to account for that with a reverse transformation -1 back to 0 to return to the original class labels.
- Using data point weights, repeated from previous slide:
    - Start with uniform weight 1/N and update weight values using formula and use in next tree.
    - You can pass newly updated sample weights to a DecisionTreeClassifier using the sample_weight keyword argument of the .fit() method.

Additional Info: Loss

Gradient Boosting
From: A Gentle Introduction to Gradient Boosting, Cheng Li

Gradient Boosting for Regression

Gradient Descent

Minimize a function by moving in the opposite direction of the gradient.

$$\theta_i := \theta_i - \rho \frac{\partial J}{\partial \theta_i}$$

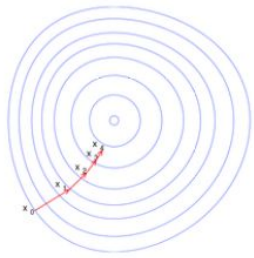


Figure : Gradient Descent. Source: http://en.wikipedia.org/wiki/Gradient_descent

Gradient Boosting for Regression

How is this related to gradient descent?

Loss function $L(y, F(x)) = (y - F(x))^2/2$
We want to minimize $J = \sum_i L(y_i, F(x_i))$ by adjusting $F(x_1), F(x_2), ..., F(x_n)$.
Notice that $F(x_1), F(x_2), ..., F(x_n)$ are just some numbers. We can treat $F(x_i)$ as parameters and take derivatives

$$\frac{\partial J}{\partial F(x_i)} = \frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = F(x_i) - y_i$$

So we can interpret residuals as negative gradients.

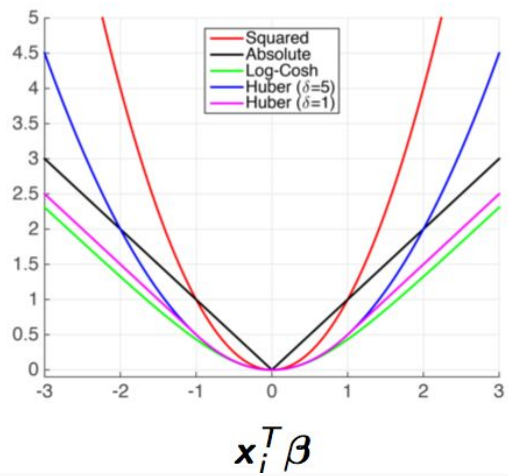
$$y_i - F(x_i) = -\frac{\partial J}{\partial F(x_i)}$$

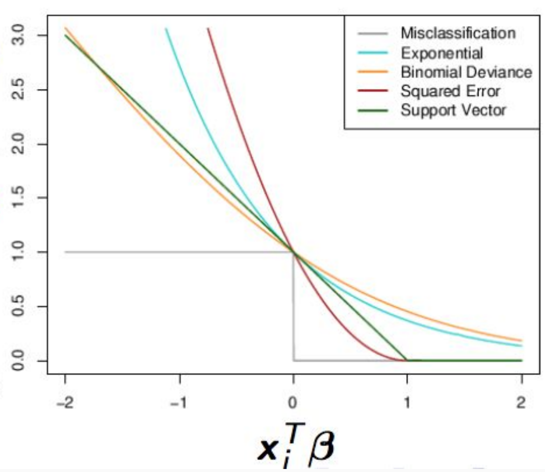
Additional Info: Loss

Some loss functions

Regression	
Squared Loss	$\frac{1}{2}(Y_i - \boldsymbol{x}_i^T\boldsymbol{\beta})^2$
Absolute Loss	$\lvert Y_i - \boldsymbol{x}_i^T\boldsymbol{\beta}\rvert$

Classification {-1, 1}		
Log Loss	$\frac{1}{ln2} ln(1 + e^{-Y_i/\left(1+e^{-\boldsymbol{x}_i^T\boldsymbol{\beta}}\right)})$	(Logistic Reg.)
Exponential Loss	$\exp(-Y_i \cdot \boldsymbol{x}_i^T\boldsymbol{\beta})$	(Ada Boost)





Last poll questions!

galvanize

Partial Dependence Plots (very useful for tree based models but works on others, too!)

We can always pull up feature importances from tree based models to give us an intuition as to which features were split on that produced the most information gain.

Does feature importance say anything about what values the splits were made on? Does it say which splits, left or right, correspond to higher output values?

In linear (and logistic regressions) we used β to tell us about a linear relationship between a feature and our target (or hypothesis function). Here we can't assume we linear relationships between them!

For forest regression models, we want some kind of tool to evaluate how our output varies as a function of a given feature (or two).

Partial Dependence Plots

One of the best tools we have is the PDP

Premise: Try to plot the regression output as a function of a continuous predictor, X_k.

Problem: What does “regression output” mean here? We use data points with explicit values of X_k to decide on splits, not coefficients that act on those values. We’re going to have try a bunch of X_k values and get some kind of “expected prediction” somehow.

Solution: “Hold X_k”, i.e. create a version of your training data where we impute a particular value across all rows for column k. For that value of X_k we predict across all data points using our fitted model and **average across** all predictions. Then we change X_k and repeat.

You won’t actually need to do this yourself. sklearn has a PDP module.

Partial Dependence Plots

Solution: “Hold X_k”, i.e. create a version of your training data where we impute a particular value across all rows for column k. For that value of X_k we predict across all data points using our fitted model and **average across** all predictions. Then we change X_k and repeat.

```python
x_df=pd.DataFrame(x, columns=['x_{}'.format(i) for i in range(5)])
print(x_df.round(3))

x_temp = x_df.copy()
x_temp['x_2']=-.05
print(x_temp)
```

```
      x_0    x_1    x_2    x_3    x_4
0   2.237  2.151 -0.744  3.011  1.367
1   2.857 -1.285  0.030  1.110  0.039
2   3.414  1.650 -0.588  2.078  0.694
3   3.729 -0.308  2.291  0.098 -1.602
4  -0.258 -1.992  0.053  0.044 -1.127
5  -1.675 -0.329 -0.949 -2.007 -0.087
6   3.524 -1.483 -0.050  1.760  0.026
7  -0.679  0.574  0.428 -0.525 -0.020
8   0.757  0.915 -0.262  1.212  0.886
9  -1.247  0.060  0.933 -1.337 -0.341
10  0.529 -1.125 -1.643  0.460 -0.343
11  1.271  0.770  0.385  0.563  0.480
12 -1.054  1.009  2.016 -2.251 -0.478
13 -0.690  0.638 -0.107  0.087  0.459
14 -0.048  0.964  0.705  0.066 -1.118
15  0.023 -0.074  0.835  0.438  0.233
16  0.248  0.347  0.490 -0.630 -0.297
17 -0.481 -0.417 -0.177  0.748  0.823
18  0.193  0.009 -1.762 -0.148  0.147
19  0.176  1.131  2.037  0.758 -1.523
         x_0       x_1   x_2       x_3       x_4
0   2.237484  2.151320 -0.05  3.011355  1.367388
1   2.856829 -1.285465 -0.05  1.110094  0.039212
2   3.414422  1.649781 -0.05  2.078326  0.694224
3   3.728715 -0.308179 -0.05  0.097925 -1.601570
4  -0.258391 -1.992198 -0.05  0.044465 -1.127496
5  -1.674844 -0.329407 -0.05 -2.007032 -0.087153
6   3.523594 -1.482936 -0.05  1.760179  0.025795
7  -0.679330  0.573797 -0.05 -0.525451 -0.020077
8   0.757013  0.914796 -0.05  1.212236  0.886230
9  -1.247149  0.060440 -0.05 -1.337019 -0.340678
10  0.528934 -1.125494 -0.05  0.459632 -0.342575
11  1.270993  0.769636 -0.05  0.562928  0.480113
12 -1.053889  1.009383 -0.05 -2.250512 -0.478132
13 -0.690291  0.638064 -0.05  0.087327  0.459330
14 -0.047997  0.964020 -0.05  0.065826 -1.118222
15  0.022930 -0.073953 -0.05  0.438103  0.233305
16  0.248286  0.346882 -0.05 -0.629815 -0.297241
17 -0.481002 -0.417327 -0.05  0.748211  0.822902
18  0.192629  0.009351 -0.05 -0.148321  0.147117
19  0.175596  1.131393 -0.05  0.757922 -1.523296
```

Partial Dependence Plots