Boosting

CS 534: Machine Learning

Slides adapted from Lee Cooper, Ryan Tibshirani, Cheng Li, David Sontag, Luke Zettlermoyer, Carlos Guestrin, Andrew Moore, and Yubin Park

Review: Random Forests

- Bagged classifier using decision trees
  - Each split only considers a random group of features
  - Tree is grown to maximum size without pruning
  - Final predictions obtained by aggregating over the B trees

$$\hat{f}_{\mathrm{rf}}^{B}(\mathbf{x}) = \frac{1}{B}\sum_{b} T(\mathbf{x};\theta_b)$$

Boosting

- Weak learner: Model whose error rate is only slightly better than random guessing
- Idea: Combine output of many weak classifiers to produce powerful committee
- Method: Sequentially fit weak learners with later models compensating the shortcomings of the existing learners

Tree with Observation Weights

- Each observation has a weight
  - Higher value means higher importance of correctly classifying this observation
- Tree can be easily adapted to use weights
  - Predictive probability uses weighted class

$$\hat{p}_g(R_j) = \frac{\sum_{\mathbf{x}_i \in R_j} w_i \mathbb{1}_{\{y_i = g\}}}{\sum_{\mathbf{x}_i \in R_j} w_i}$$

AdaBoost

- Popular boosting algorithm developed by Freund & Schapire (1997)
- Consider two-class problem with the output variable is coded as {+1, -1}
- Each weak classifier $G_m(X)$ produces prediction taking one of the two values
- Combine a weighted sum of M different classifiers

AdaBoost: Algorithm

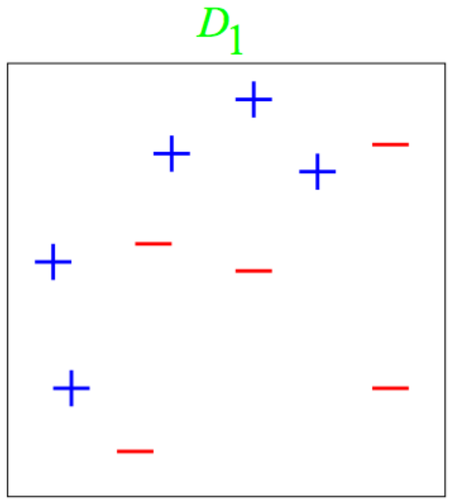
Algorithm 10.1 *AdaBoost.M1.*

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
   $$\text{err}_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}.$$
   (c) Compute $\alpha_m = \log((1 - \text{err}_m)/\text{err}_m)$.
   (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 $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

higher influence to more accurate classifiers

Algorithm 10.1 (Hastie et al.)

Example: Toy Data

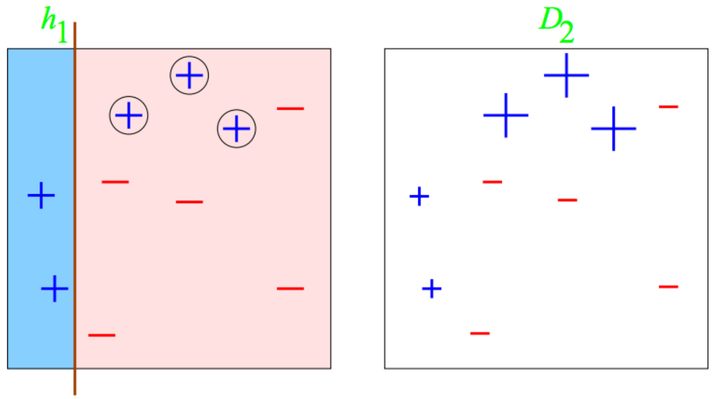


weak classifier: single
horizontal or vertical half-plane

http://media.nips.cc/Conferences/2007/Tutorials/Slides/schapire-NIPS-07-tutorial.pdf

Example: Toy Data

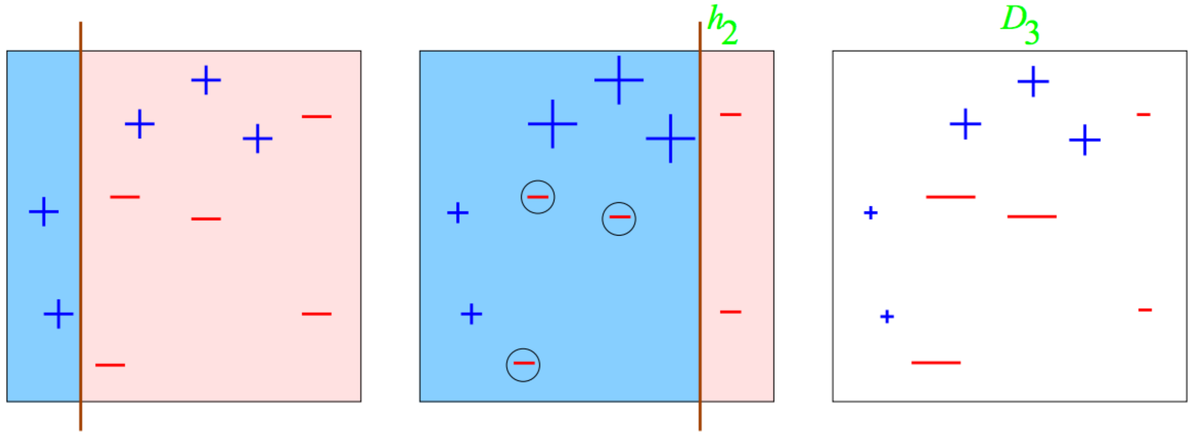
Round 1



$\varepsilon_1 = 0.30$

$\alpha_1 = 0.42$

Round 2

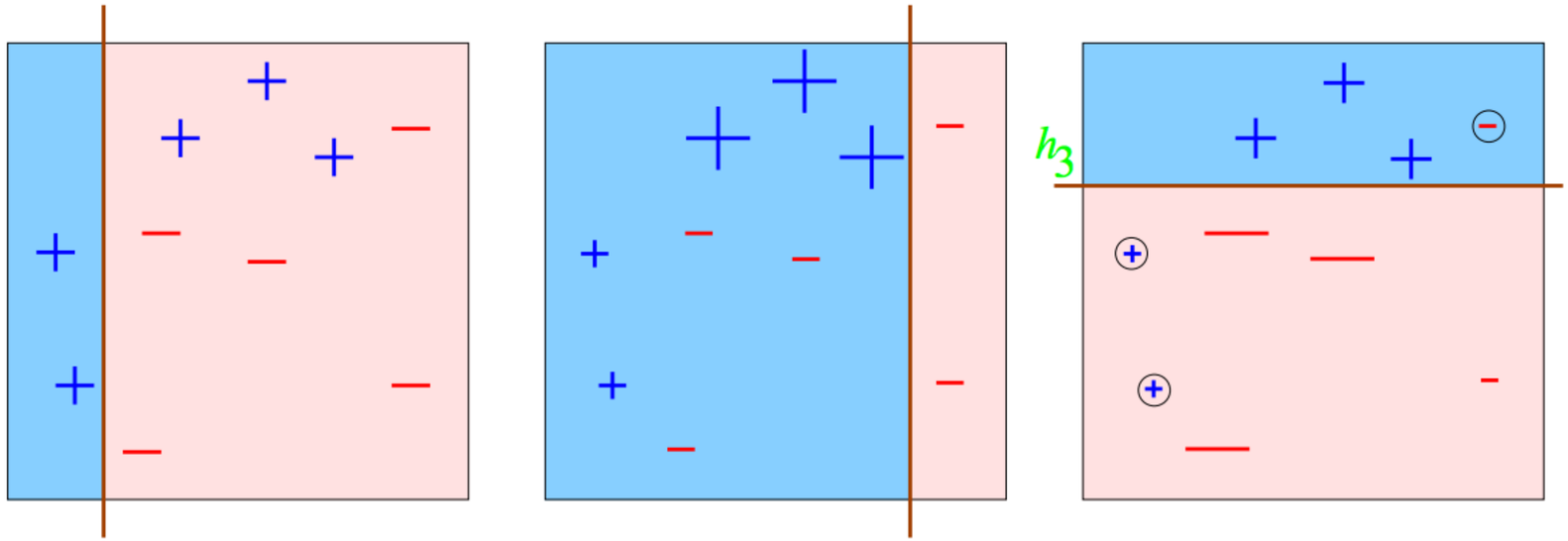


$\varepsilon_2 = 0.21$

$\alpha_2 = 0.65$

Example: Toy Data

Round 3

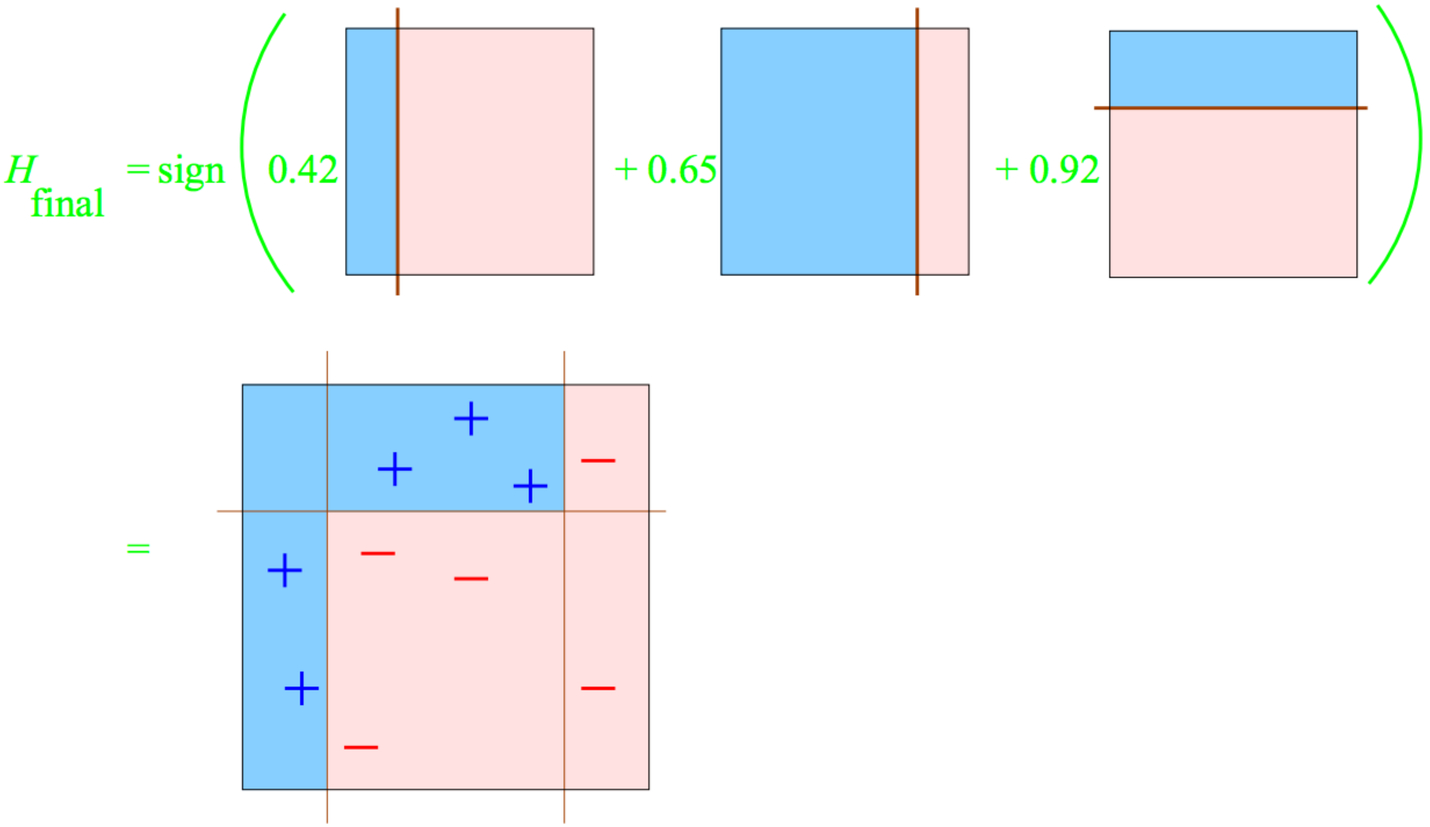


$\varepsilon_3 = 0.14$

$\alpha_3 = 0.92$

Example: Toy Data

Final Classifier



Example: Boosting Stumps

- Simulated data with 1000 points draw from known model
- Classification tree with one split (two leaves)
- Misclassification rate of 45.8% for single tree

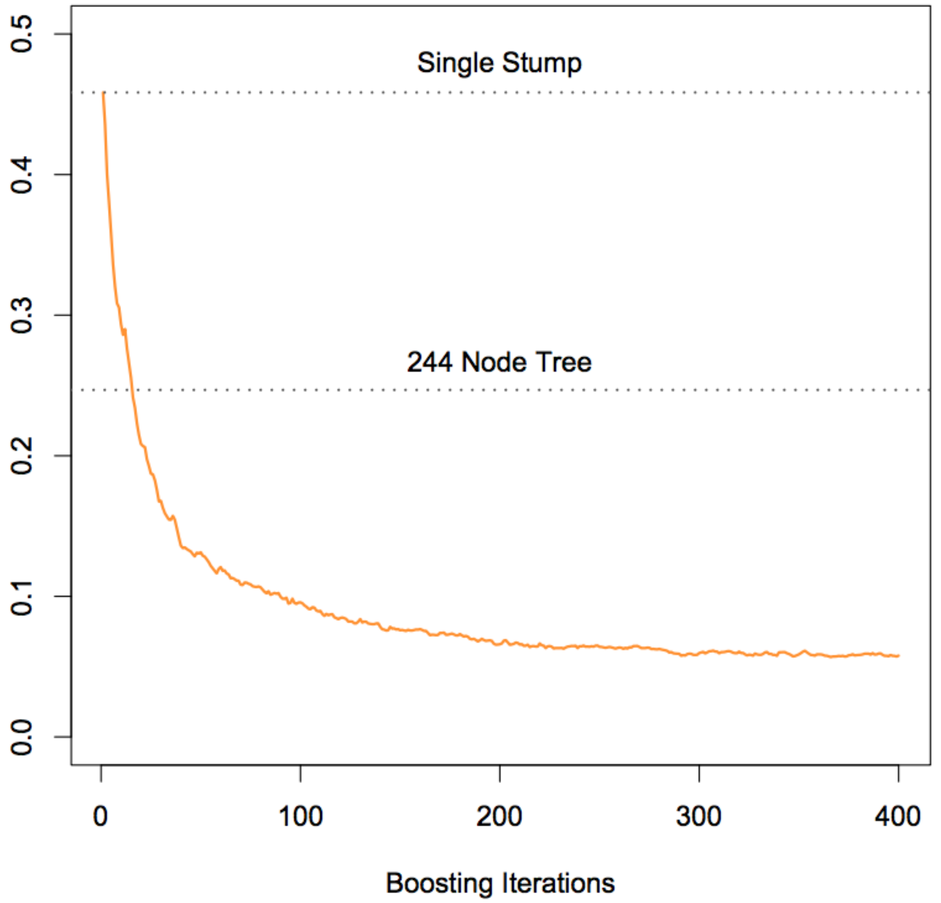


Figure 10.2 (Hastie et al.)

Thought Exercise

- We have training data, (x_1, y_1), …, (x_n, y_n), and task to fit model f(x) to minimize square loss
- Friend gives you a model f, with some mistakes
- You are not allowed to remove anything from f or change any parameter in f
- You are allowed to add additional models h to f, so new prediction is f(x) + h(x)

What would you do?

Thought Exercise: Solution

- Simple solution:

$$
\begin{aligned}
f(x_1) + h(x_1) &= y_1 \\
f(x_2) + h(x_2) &= y_2 \\
&\vdots \qquad\qquad \vdots \\
f(x_n) + h(x_n) &= y_n
\end{aligned}
$$

$$
\begin{aligned}
h(x_1) &= y_1 - f(x_1) \\
h(x_2) &= y_2 - f(x_2) \\
&\vdots \qquad\qquad \vdots \\
h(x_n) &= y_n - f(x_n)
\end{aligned}
$$

Can this be done using any regression tree?

Gradient Boosting

- Gradient descent + boosting
- Powerful algorithm that can be used for regression, classification, ranking
- Data mining competition winner most likely uses this algorithm

Gradient Boosting: Regression

- Fit a regression tree to new data, $(x_1, y_1\text{-}f(x_1)), \ldots, (x_n, y_n\text{-}f(x_n))$
- $y_i\text{-}f(x_i)$ are residuals — parts that existing model f cannot do well
- If new model f + h is still not satisfactory, reiterate again

How does this relate to gradient descent?

Gradient Boosting: Regression

- Loss function:

$$L(\mathbf{y}, f(\mathbf{X})) = \sum_i \frac{1}{2}(y_i - f(\mathbf{x}_i))^2$$

- Treat f(x_i) as parameters and take derivatives

$$\frac{\partial L}{\partial f(\mathbf{x_i})} = f(\mathbf{x_i}) - y_i$$

- Interpret residuals as negative gradients

$$f(\mathbf{x_i}) := f(\mathbf{x_i}) + y_i - f(\mathbf{x_i})$$

$$f(\mathbf{x_i}) := f(\mathbf{x_i}) - 1\frac{\partial L}{\partial \mathbf{x}_i}$$

Gradient Boosting: Generic Algorithm

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.

Algorithm:

1. Initialize model with a constant value:

$$F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma).$$

2. For m = 1 to M:
    1. Compute so-called *pseudo-residuals*:

$$r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \ldots, n.$$

    2. Fit a base learner (e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.
    3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$$

    4. Update the model:

$$F_m(x) = F_{m-1}(x) + \gamma_m h_m(x).$$

3. Output $F_M(x)$.

https://en.wikipedia.org/wiki/Gradient_boosting

Gradient Tree Boosting

Algorithm 10.3 *Gradient Tree Boosting Algorithm.*

1. Initialize $f_0(x) = \arg\min_\gamma \sum_{i=1}^{N} L(y_i, \gamma)$.
2. For $m = 1$ to M:
   (a) For $i = 1, 2, \ldots, N$ compute

$$r_{im} = -\left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f=f_{m-1}}.$$

   (b) Fit a regression tree to the targets r_{im} giving terminal regions R_{jm}, $j = 1, 2, \ldots, J_m$.
   (c) For $j = 1, 2, \ldots, J_m$ compute

$$\gamma_{jm} = \arg\min_\gamma \sum_{x_i \in R_{jm}} L\left(y_i, f_{m-1}(x_i) + \gamma\right).$$

   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm} I(x \in R_{jm})$.
3. Output $\hat{f}(x) = f_M(x)$.

Regression: Loss Functions

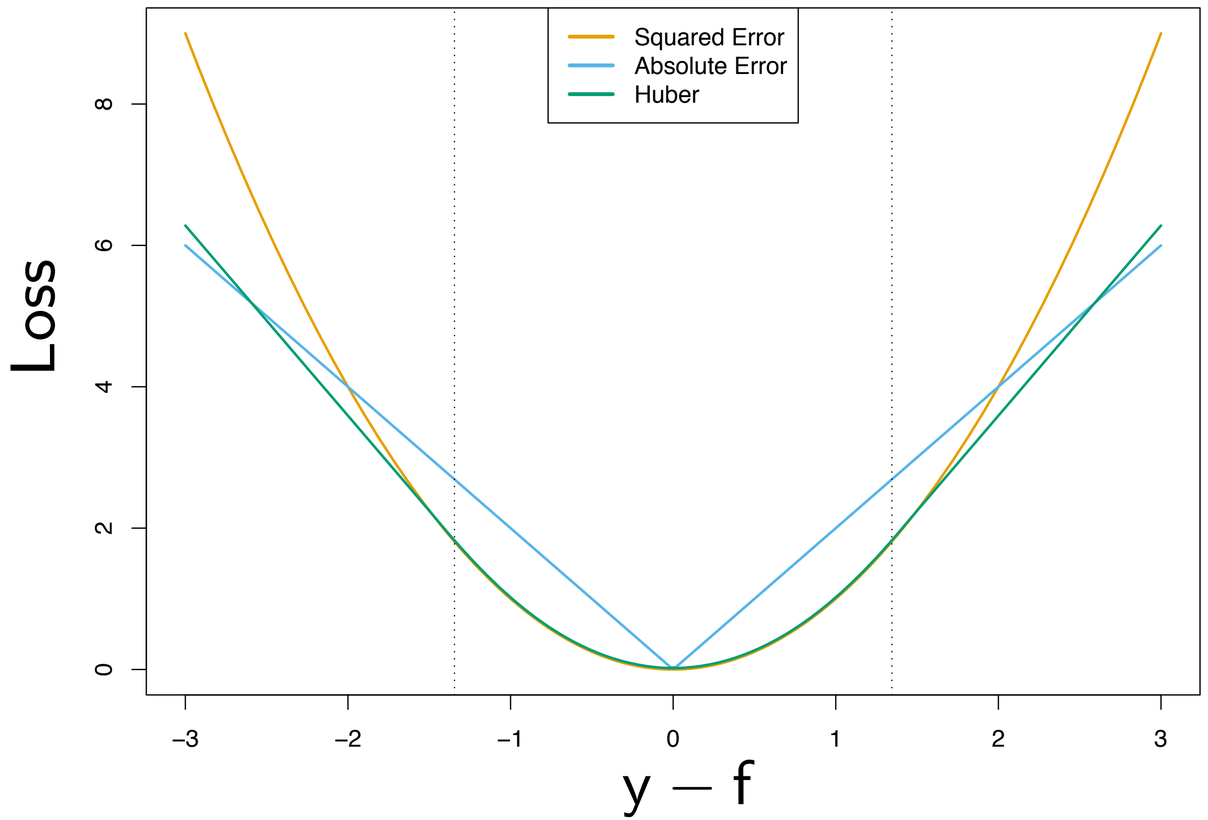


Figure 10.5 (Hastie et al.)

Regression: Robust Loss

- Squared error puts more emphasis on observations with large deviation than absolute loss
- Squared error is less robust and performance degrades for long-tailed error distributions and mislabelings
- Huber loss has strong resistance to gross outliers with efficiency of least squares for Gaussian errors

Gradient Boosting: Regression

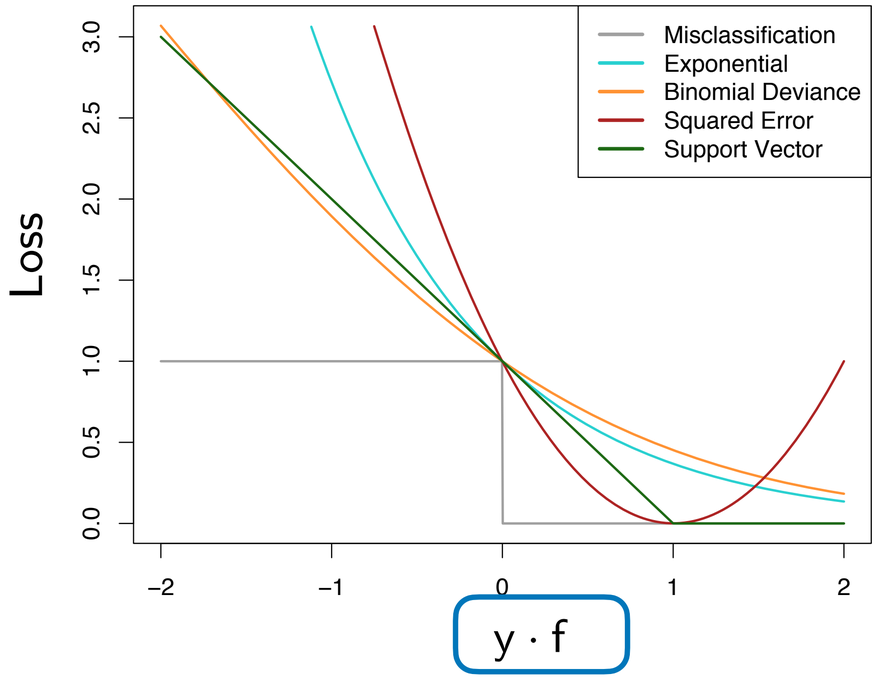
	Loss function	Negative gradient
Squared Loss	$\frac{1}{2}(y_i - f(\mathbf{x_i}))^2$	$y_i - f(\mathbf{x}_i)$
Absolute Loss	$\lvert y - f(\mathbf{x})\rvert$	$\text{sign}(y_i - f(\mathbf{x}_i))$
Huber Loss	$\begin{cases} \frac{1}{2}(y_i - f(\mathbf{x_i}))^2 & \lvert y - f(\mathbf{x})\rvert \leq \delta \\ \delta(\lvert y - f(\mathbf{x})\rvert - \delta/2) & \lvert y - f(\mathbf{x})\rvert > \delta \end{cases}$	$\begin{cases} y_i - f(\mathbf{x}_i) & \lvert y - f(\mathbf{x})\rvert \leq \delta \\ \delta\text{sign}(y_i - f(\mathbf{x}_i)) & \lvert y - f(\mathbf{x})\rvert > \delta \end{cases}$

Gradient Boosting: Classification

- Regression setting can be easily adapted for classification
- Generalization of Adaboost to general classification loss functions

Classification: Loss Functions

	Loss function
Exponential	$\exp(-yf(\mathbf{x}))$
Binomial deviance	$-\log(1+\exp(-2yf(\mathbf{x}))$
Misclassification	$\mathbb{1}_{\{yf(\mathbf{x})<0\}}$
Squared error	$(y-f(\mathbf{x}))^2$
Support vector	$\max(0,1-yf(\mathbf{x}))$



Margin: positive = correct, negative = incorrect

Figure 10.4 (Hastie et al.)

Classification: Robust Loss

- Exponential and deviance are continuous approximations to misclassification loss
  - Binomial deviance penalty increases linearly with negative margin
  - Exponential loss penalty increases exponentially with negative margin
- Binomial criterion far more robust than exponential in noisy settings

Gradient Boosting vs AdaBoost

- Boosting: Fit additive model in a forward stage-wise manner where each stage, new weak learner compensates shortcomings of existing models
- Gradient boosting — “shortcomings” identified by gradient
- Adaboost — “shortcomings” identified by high-weight data points

Size of Trees

- Best method is to grow small trees with no pruning
- Right size will depend on level of interaction between variables
- ~2-8 leaves works well

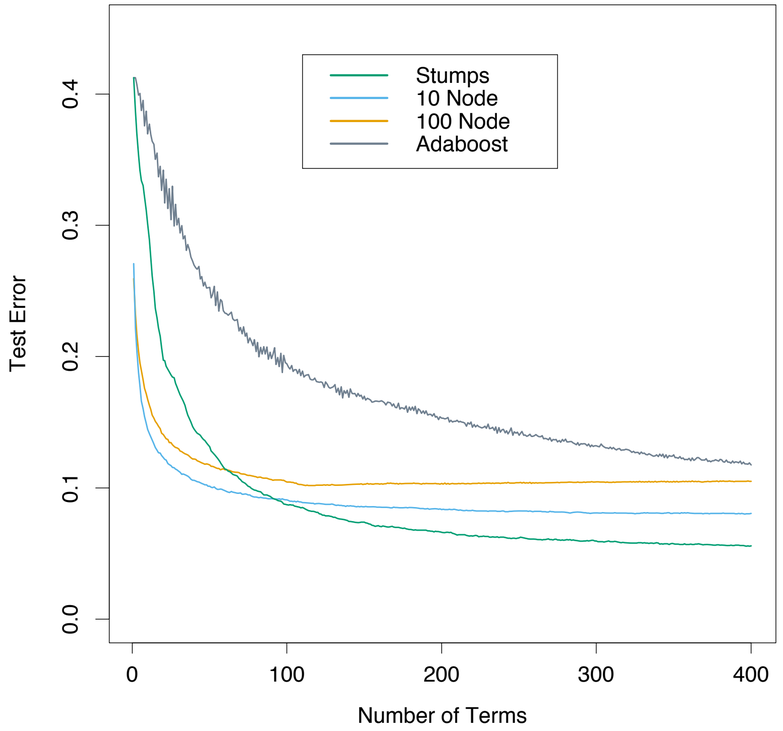


Figure 10.9 (Hastie et al.)

Boosting: Variable Importance

- Average squared importance over all fitted trees

$$\mathrm{Imp}_j^2(\hat{f}^{\mathrm{boost}}) = \frac{1}{M}\sum_{m=1}^{M}\mathrm{Imp}_j^2(\hat{f}_m^{\mathrm{tree}})$$

- Stabilizes variable importances —> more accurate than for single tree

- Relative importance: Scale largest importance to 100 and scale all other variable importances accordingly

Example: Spam Data

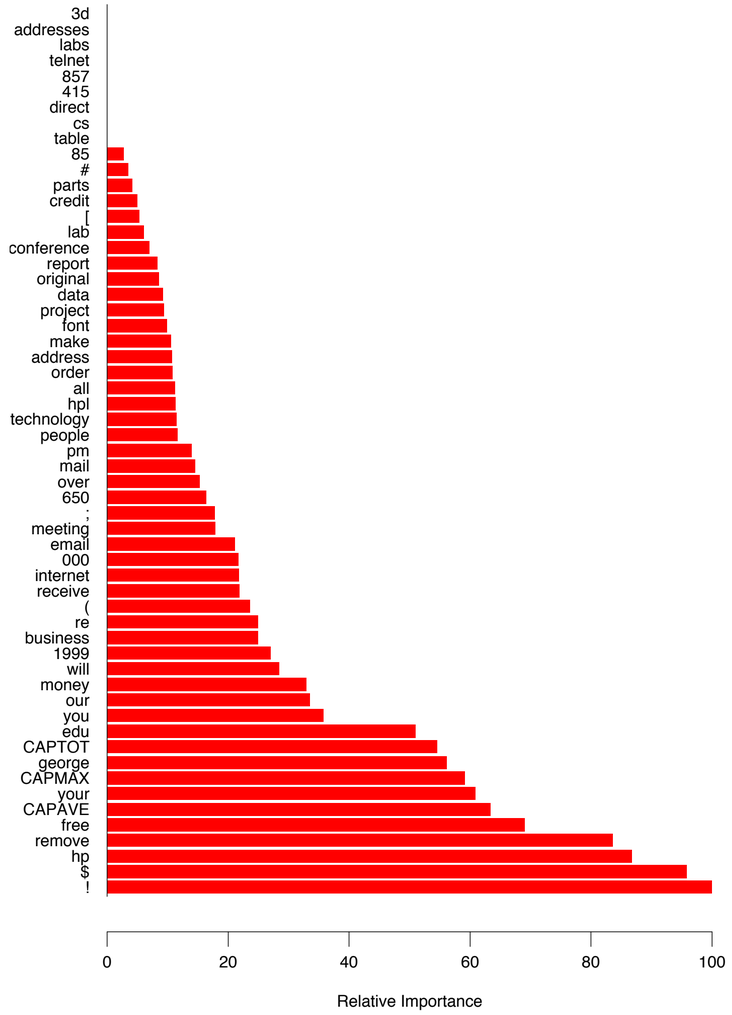


Figure 10.6 (Hastie et al.)

Boosting: Advantages

- Provably effective — reduces both bias and variance
- Easy — limited number of parameters to tune
- Flexible — combine with any learning algorithm
- Versatile — can be used with data that is numeric, discrete, textual

Boosting: Disadvantages

- Lack of interpretability — lose benefit of classification tree
- Serializability — computation can be difficult and hard to perform in parallel
- Performance dependent on weak learner and data — can fail if weak classifiers are too complex or too weak
- Noise susceptibility — empirically seems susceptible to uniform noise

Extreme Gradient Boosting (XGBoost)

Homesite Quote Conversion, Winners' Interview: 3rd place, Team New Model Army | CAD & QuY

Kaggle Team | 02.29.2016

Prudential Life Insurance Assessment, Winner's Interview: 2nd place, Bogdan Zhurakovskyi

Kaggle Team | 03.14.2016

Airbnb New User Bookings, Winner's Interview: 2nd place, Keiichi Kuroyanagi (@Keiku)

Kaggle Team | 03.17.2016

Telstra Network Disruption, Winner's Interview: 1st place, Mario Filho

Kaggle Team | 03.23.2016

What these various data mining competitors have in common: all used XGBoost

“XGBoost scales beyond billions of examples using far fewer resources than existing systems.”

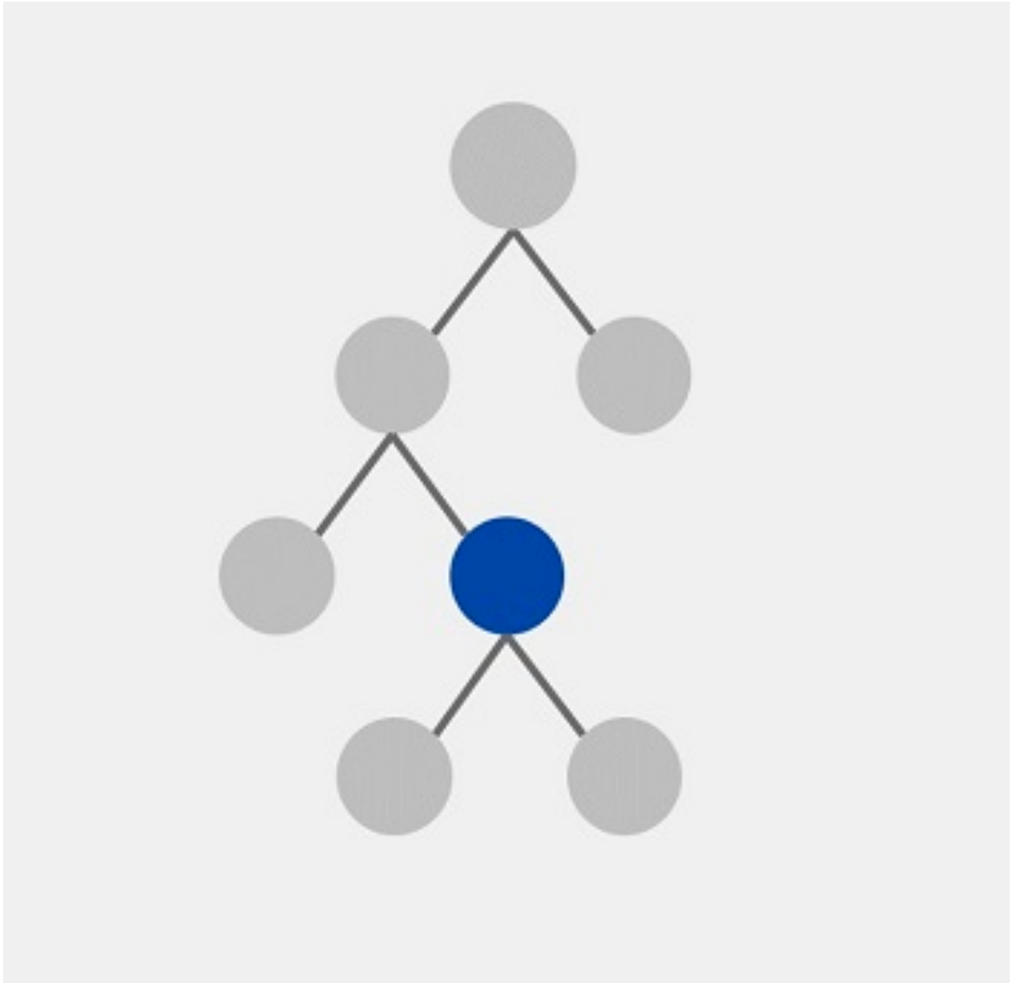
–T. Chen and C. Guestrin

Why XGBoost?

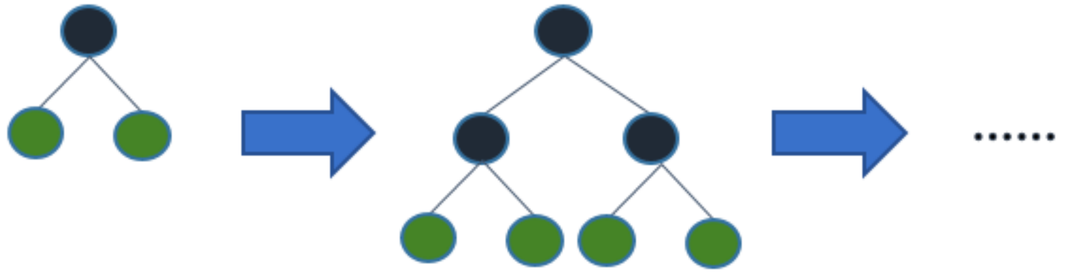
- Implements basic idea of GBM with some tweaks
  - Regularization of base tree
  - Approximate split finding
  - Weighted quantile sketch
  - Sparsity-aware split finding
  - Cache-aware block structure for out of core computation

LightGBM: The New XGBoost

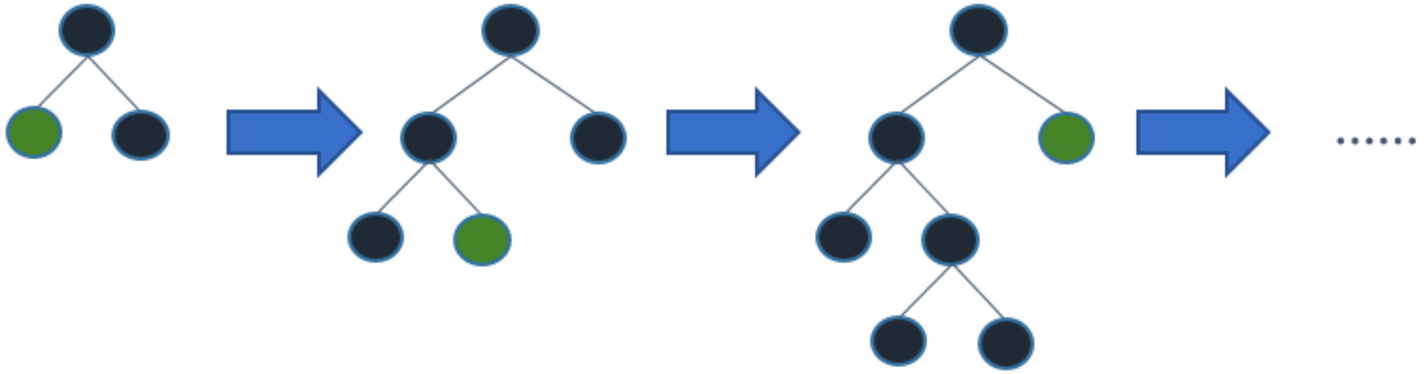
- New gradient boosting framework
  - Several times faster than existing implementations
  - Difference is in how to split the tree — depth or level wise vs leaf-wise



Level-wise vs Leaf-wise



Level-wise tree growth



Leaf-wise tree growth

Light GBM

- Faster training sped and higher efficiency — histogram-based algorithm
- Lower memory usage (see above)
- Better accuracy than any other boosting algorithm*
- Compatibility with large datasets — equal performance with significant reduction in training time compared to XGBOOST

https://www.analyticsvidhya.com/blog/2017/06/which-algorithm-takes-the-crown-light-gbm-vs-xgboost/