Boosting

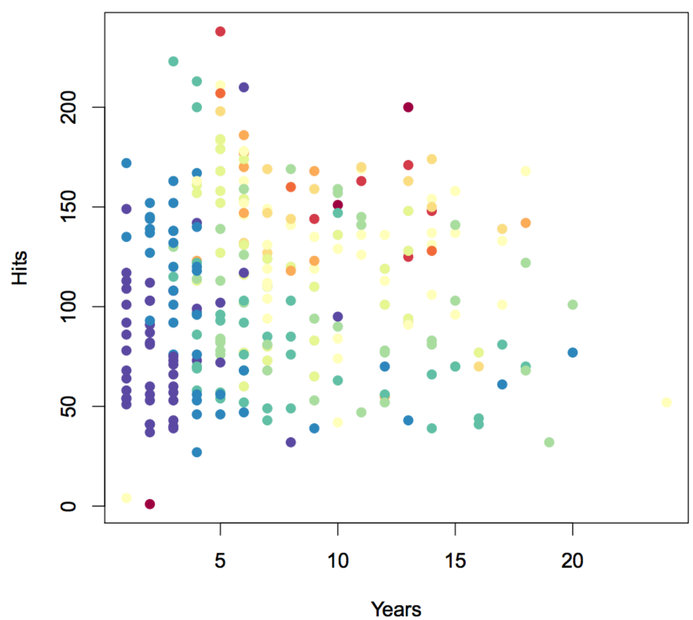
Overview

- Review of Decision Tree – Regression
- What is boosting?
- Boosting
  - AdaBoost
  - Gradient Boosted Regression Trees
- Gradient Boosted Regression Trees in sklearn
  - How to tune

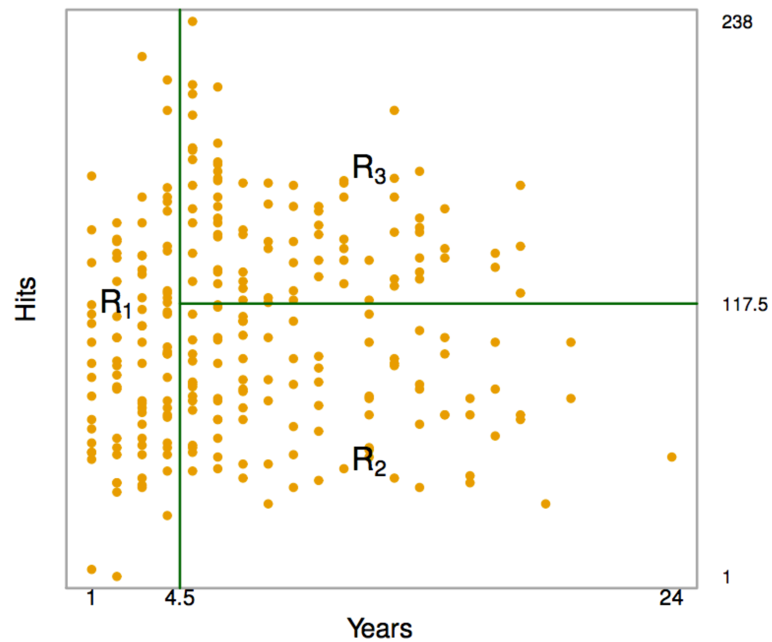
- Discrete AdaBoost

Decision Trees - Regression

Salary is color-coded from low (blue, green) to high (yellow,red)



Decision Trees – Regression



we consider a sequence of trees indexed by a nonnegative tuning parameter α. For each value of α there corresponds a subtree $T \subset T_0$ such that

$$\sum_{m=1}^{|T|} \sum_{i:\, x_i \in R_m} (y_i - \hat{y}_{R_m})^2 + \alpha |T|$$

What is Boosting?

- Bagging – Bootstrap many trees, each tree independently grown, in an effort to decrease variance through averaging
- Random Forest – Similar idea, but take random subset of possible features at each split to "decorrelate the trees"

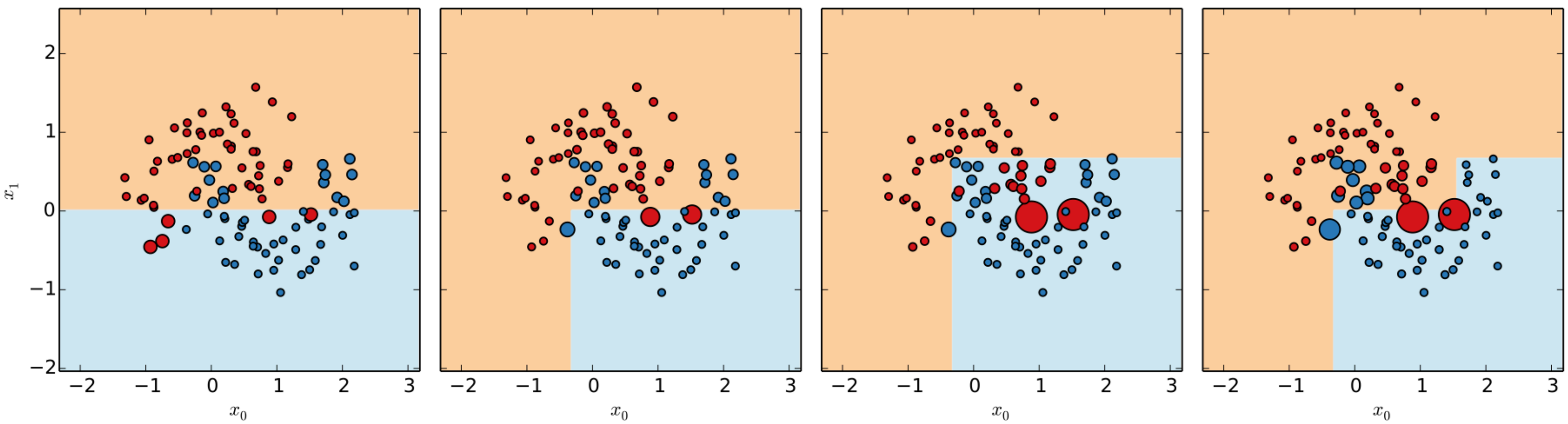
<u>What is Boosting? Not at all like Bagging or Random Forest!</u>

- Idea: Combine set of "weak" learners to form strong learner
  - "weak" in that error rate only slightly better than random guessing
- How: Sequentially apply weak classification algorithm to modified versions of the data → sequence of weak classifiers
  - Each tree is grown using information from last tree

Boosting

AdaBoost

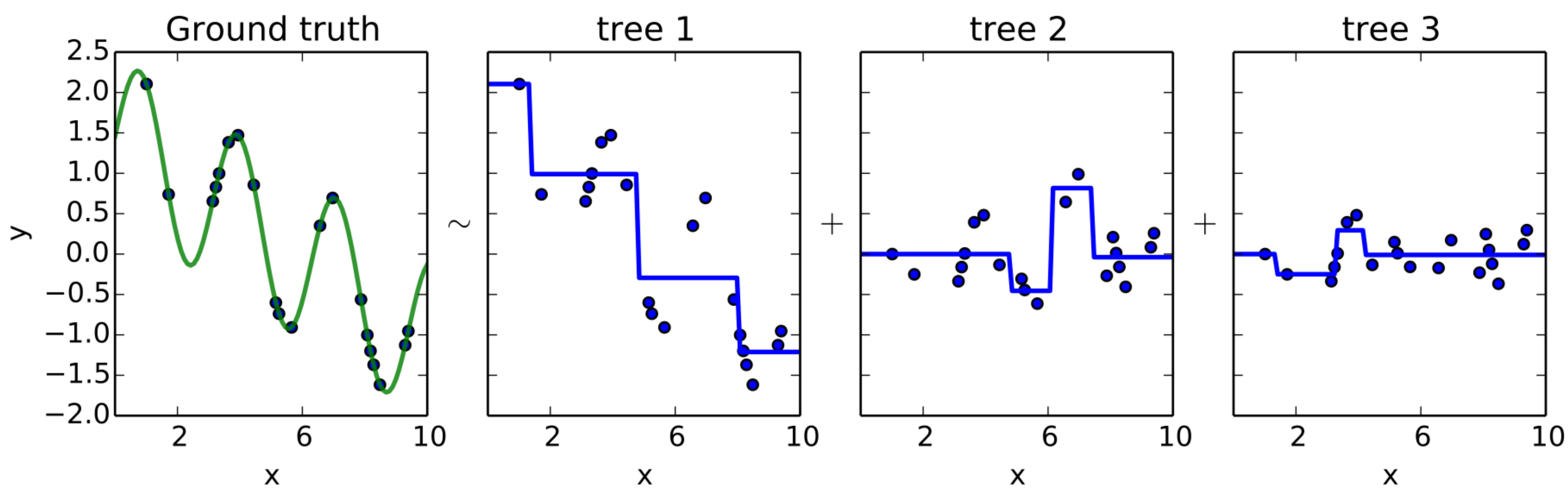
- Each tree is expert on attacking errors of predecessor
- Iteratively re-weights observations based on errors



Boosting

Gradient Boosted Regression Trees

- Instead of fitting to reweighted training observations, fit residuals to of previous tree



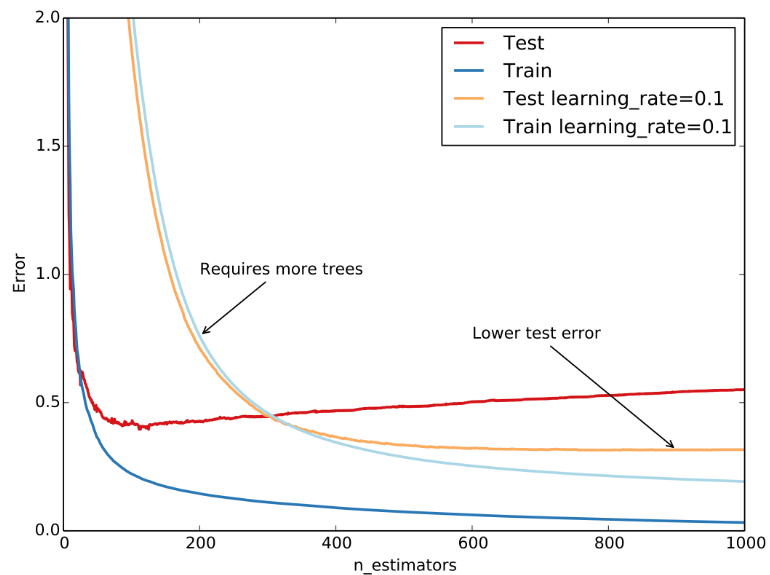
GBRT in sklearn

Tree Structure

- `max_depth`
  - controls degree of interactions
  - Ex. Latitude and Longitude
- `min_samples_per_leaf`
  - may not want terminal nodes with too few leaves

Shrinkage

- `n_estimators`
  - number of trees grown
- `learning_rate`
  - lower learning rate requires higher n_estimators



GBRT in sklearn

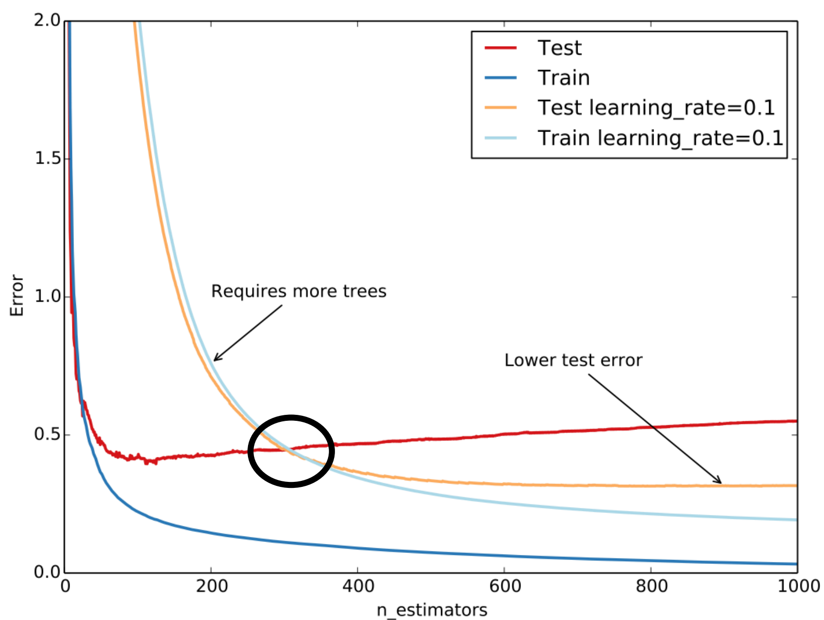
Stochastic Gradient Boosting

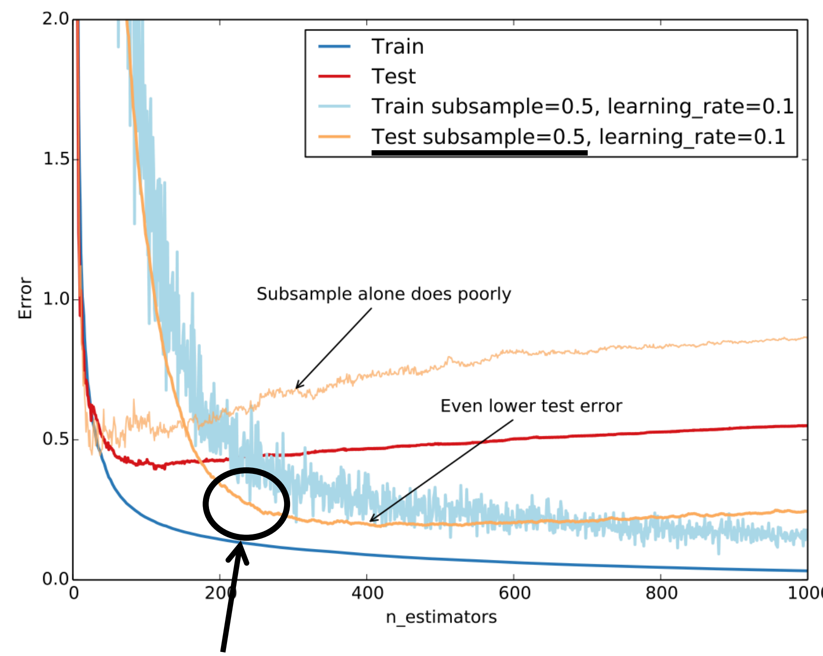
- `max_features`
  - random subsample of features
  - Especially good when you have lots of features

- `sub_sample`
  - random subset of training set

GBRT in sklearn

Stochastic Gradient Boosting can improve accuracy and reduce runtime!





Lower test error!
Fewer trees to get there!

GBRT in sklearn

Tuning – a good starting setup

(1) Set `n_estimators` high as possible

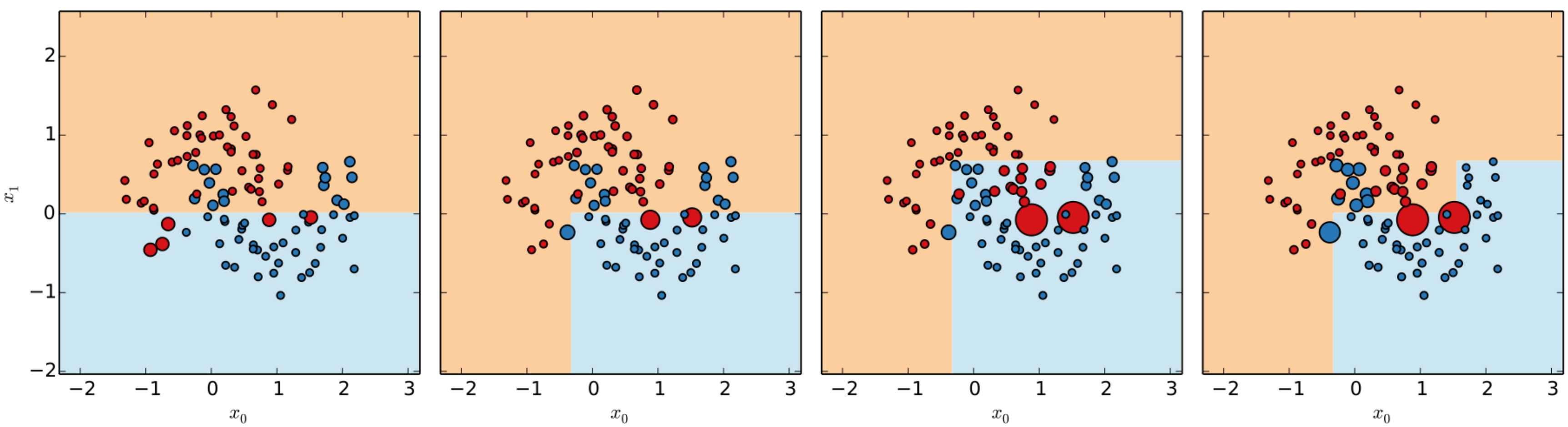
(2) Tune hyperparameters together via grid search

```
from sklearn.grid_search import GridSearchCV
param_grid = {'learning_rate': [0.1, 0.05, 0.02, 0.01],
              'max_depth': [4, 6],
              'min_samples_leaf': [3, 5, 9, 17],
              'max_features': [1.0, 0.3, 0.1]}
est = GradientBoostingRegressor(n_estimators=3000)
gs_cv = GridSearchCV(est, param_grid).fit(X, y)
# best hyperparameter setting
gs_cv.best_params_
```

(3) Set `n_estimators` even higher while tuning `learning_rate`

AdaBoost

- Each tree is expert on attacking errors of predecessor
- Iteratively re-weights observations based on errors



Discrete AdaBoost

- One of the most popular boosting algorithms
  - also known as AdaBoost.M1,Freund & Schapire (1997)

Y: {-1, 1}

G(X): classifier producing predictions taking two values {-1, 1}

Error rate on the training set:

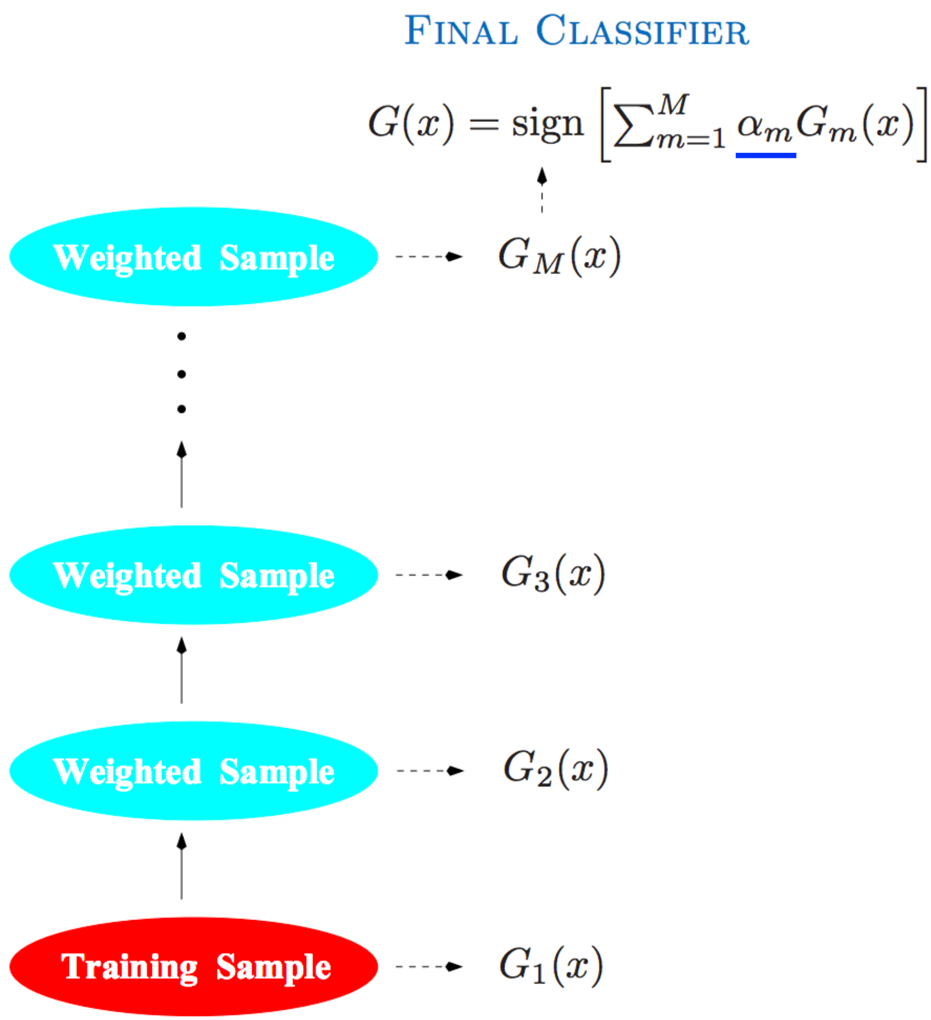
$$\boxed{\overline{\mathrm{err}} = \frac{1}{N}\sum_{i=1}^{N} I(y_i \neq G(x_i))}$$

Discrete AdaBoost

$G_i(x)$ weak classifiers

$G(x)$ strong learner

Note only $G_1(x)$ fit on training



Discrete AdaBoost

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 $G(x) = \text{sign}\left[\sum_{m=1}^M \alpha_m G_m(x)\right]$.

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 $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$.

- For first weak classifier, $G_1(x)$, just use training data
- For subsequent weak classifier, same classification algorithm but modify weights
  - If previously misclassified, scale by $e^{(\alpha_m)}$
  - else, w_i same
- Final strong classifier $G(x)$ determined by weighted majority votes
  - $\alpha_1, \ldots \alpha_M$ as weight of votes
  - The smaller the error of the weak classifier, the greater the weight

Observe that…

- $\alpha_1, \ldots, \alpha_M$ give higher influence to more accurate classifiers
- At each step m, observations previously misclassified by $G_{m-1}(x)$ have their weights increased

  → Each successive classifier forced to concentrate on training observations previously missed

Questions

- Describe the following tuning parameters for gradient boosting
  - `max_depth`
  - `learning_rate` and `n_estimators`
  - `max_features` and `sub_sample`

- In gradient boosted trees,
  - How do the trees relate to one another?
    - Relate to preceding tree, talk about residuals

Appendix

Boosting Algorithm 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:
   2.1 Fit a tree $\hat{f}^b$ with d splits ($d + 1$ terminal nodes) to the training data (X, r).
   2.2 Update $\hat{f}$ by adding in a shrunken version of the new tree:
   $$\hat{f}(x) \leftarrow \hat{f}(x) + \lambda \hat{f}^b(x).$$
   2.3 Update the residuals,
   $$r_i \leftarrow r_i - \lambda \hat{f}^b(x_i).$$
3. Output the boosted model,
$$\hat{f}(x) = \sum_{b=1}^{B} \lambda \hat{f}^b(x).$$

Simplified version. Easier to understand for building intuition.

Details on GBRT, see page 361
http://web.stanford.edu/~hastie/local.ftp/Springer/OLD/ESLII_print4.pdf

AdaBoost is not black magic

- But there is quite a bit of math and underpinning concepts to go through to really understand it.
  http://web.stanford.edu/~hastie/local.ftp/Springer/OLD/ESLII_print4.pdf - Page 341-346

<u>Very very roughly,</u>

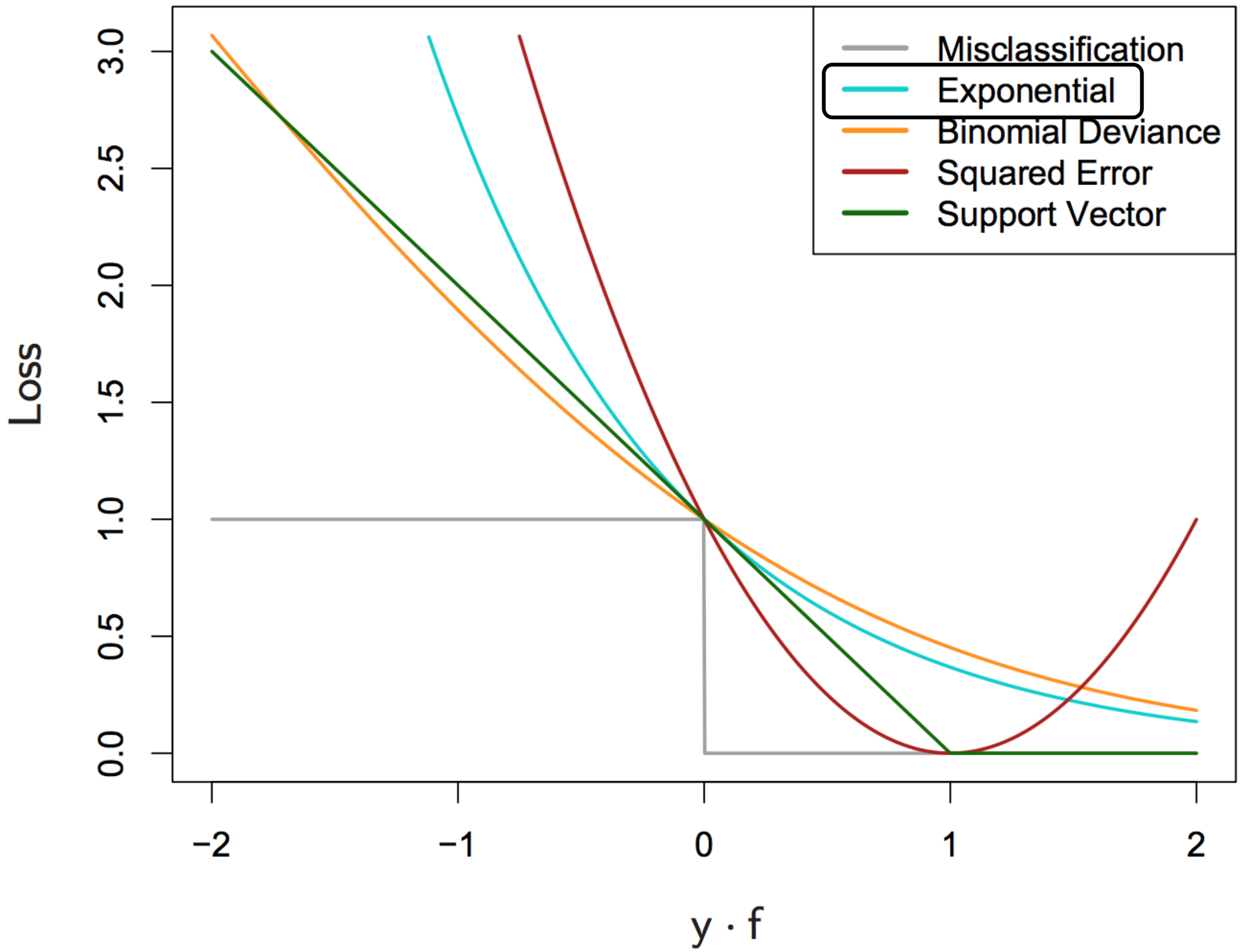
- It is a version of Forward Stagewise Additive Modeling, which adds new basis functions without adjusting previous parameters and coefficients.
  - This contrasts with gradient boosting, see pg 342 vs. pg 361 in link above
  - AdaBoost uses Exponential Loss $L(y, f(x)) = \exp(-y\, f(x))$.
    - It can be shown that to minimize this loss, at each iteration, we can reweight our observations $w_i^{(m+1)} = w_i^{(m)} \cdot e^{\alpha_m I(y_i \neq G_m(x_i))} \cdot e^{-\beta_m}$
  - Use exponential loss because of computational advantage; could consider others.
- Can be shown that the additive expansion in AdaBoost is estimating

  $$f^*(x) = \arg\min_{f(x)} \mathrm{E}_{Y|x}(e^{-Yf(x)}) = \frac{1}{2}\log\frac{\Pr(Y=1|x)}{\Pr(Y=-1|x)}$$

  which justifies taking the sign as classification rule for final classifier

  $$G(x) = \mathrm{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$$

Comparison of Loss Functions for Classification



Comparison of Loss Functions for Regression