

Boosting

Objectives

- List features & benefits of boosting
- Compare & contrast boosting vs. other ensemble methods
- Learn, tune, and score a model using scikit-learn's boosting classes
- Implement the AdaBoost algorithm

Agenda

Review & motivation

- decision trees
- bias-variance tradeoff
- ensemble methods

Boosting

- Key concepts
- Ada Boost
- Gradient Boosted Models (GBM)
- Setting tuning parameters (via grid search)

Citations

Several figures and/or text have been taken from:

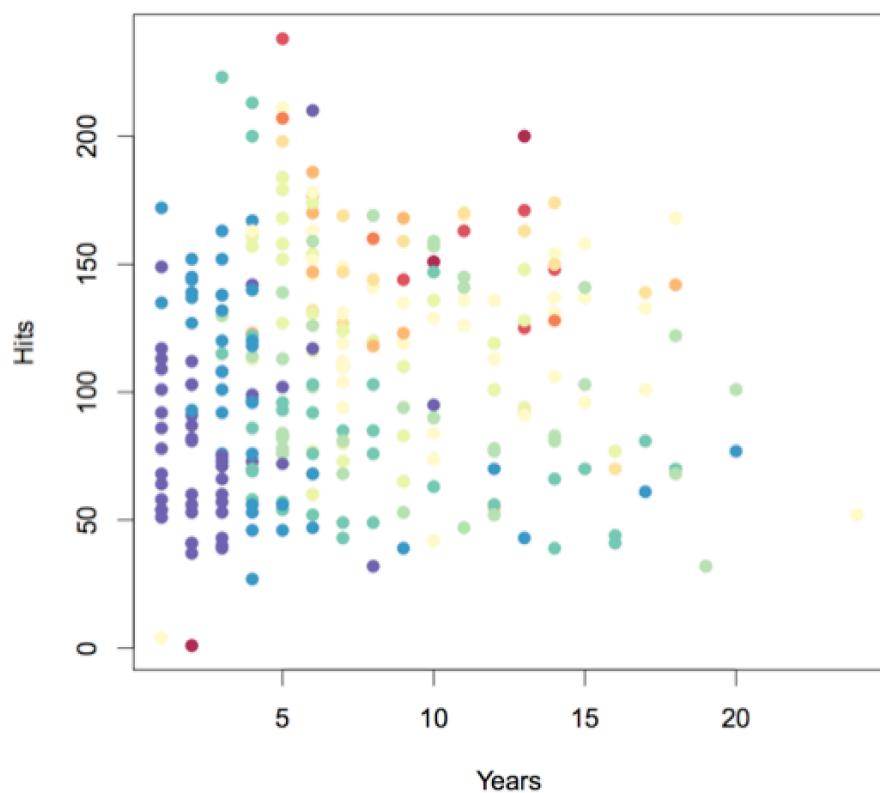
- Peter Prettenhofer's talk, “Gradient Boosted Regression Trees in scikit-learn”
- Hastie,Tibshirani. and Friedman, *The Elements of Statistical Learning*. Springer, Berlin: Springer series in statistics, 2001

Review

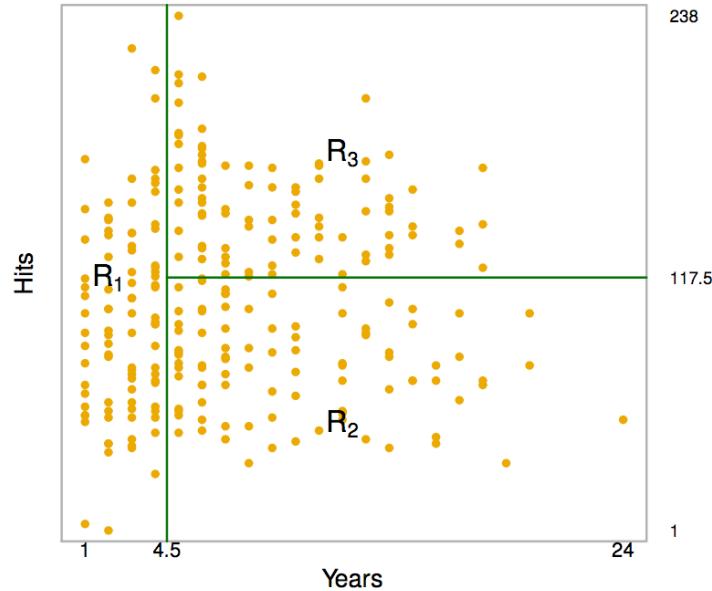
Decision Trees - Regression

Salaries color-coded from low to high

- Low salaries in blue, green
- High salaries in orange, red



Decision Trees – Regression



Consider sequence of trees indexed by tuning parameter α .

For each α , there is a corresponding subtree, T , such that the following is minimized:

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

where $|T|$ is the number of terminal nodes

Bias-Variance Tradeoff

Q: What types of error can we decompose MSE into?

Q: How can we address these types of errors when building a model?

Q: What tools & techniques should we use?

Ensemble Methods

Ensemble methods minimize variance to improve quality of predictions:

- “averaging over a committee of experts”
- Individual tree may have high variance

Two examples:

- Bagging (random split on data not features):
 - Fit many trees on bootstrap datasets
 - Trees are independent
 - Average to decrease variance
- Random forest (random split on features not data):
 - Draw random subset from set of all possible features
 - Train trees on each subset of features
 - Average trees to decrease variance

Boosting

Boosting is an ensemble method

- Boosting combines a sequence “weak” learners to form strong learner:
 - “weak” → error rate only slightly better than random guessing
 - Boosting fits an additive model of weak learners
 - Each learner fits error from prior learner
 - Each stage chooses optimal learner and weight!
 - Differs from bagging and random forest which fit multiple independent models

$$f(x) = \sum_{m=1}^M \beta_m b(x; \gamma_m)$$

- Slow learners often provide best in class predictions
 - Many top entrants in competitions use boosting
 - Apply to many problem domains – both classification and regression
 - Can be more expensive to train
 - Fast to predict

Boosting Algorithms

Will look at two algorithms:

1. AdaBoost
2. Gradient Boosting Regression Trees

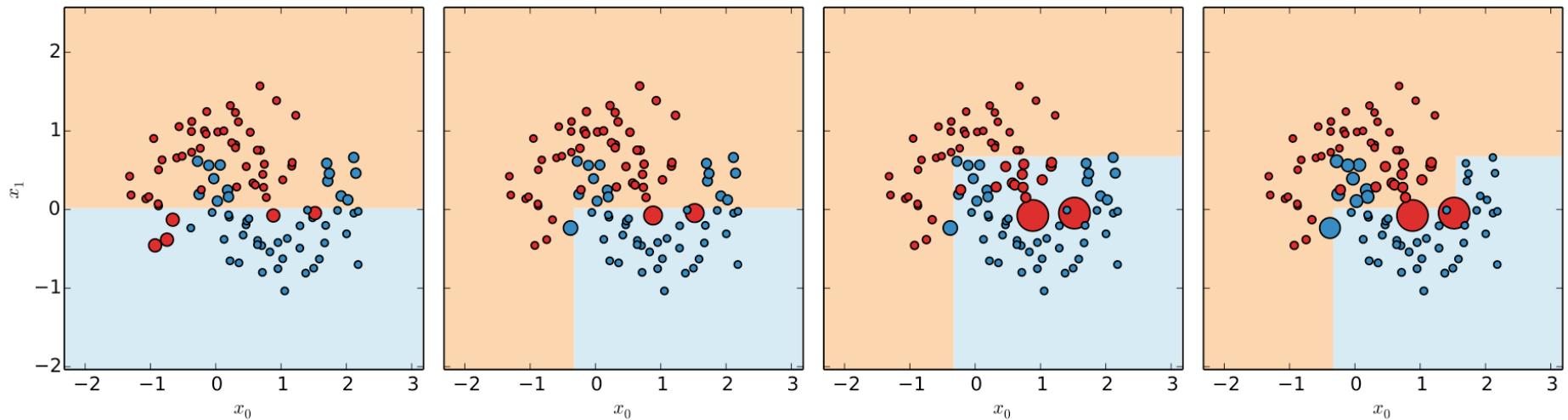
In practice:

1. Choose suitable family of weak learners and loss function for problem and data
2. Fit model using cross validation
3. Use grid search to optimize tuning parameters
4. Evaluate performance

Boosting

AdaBoost

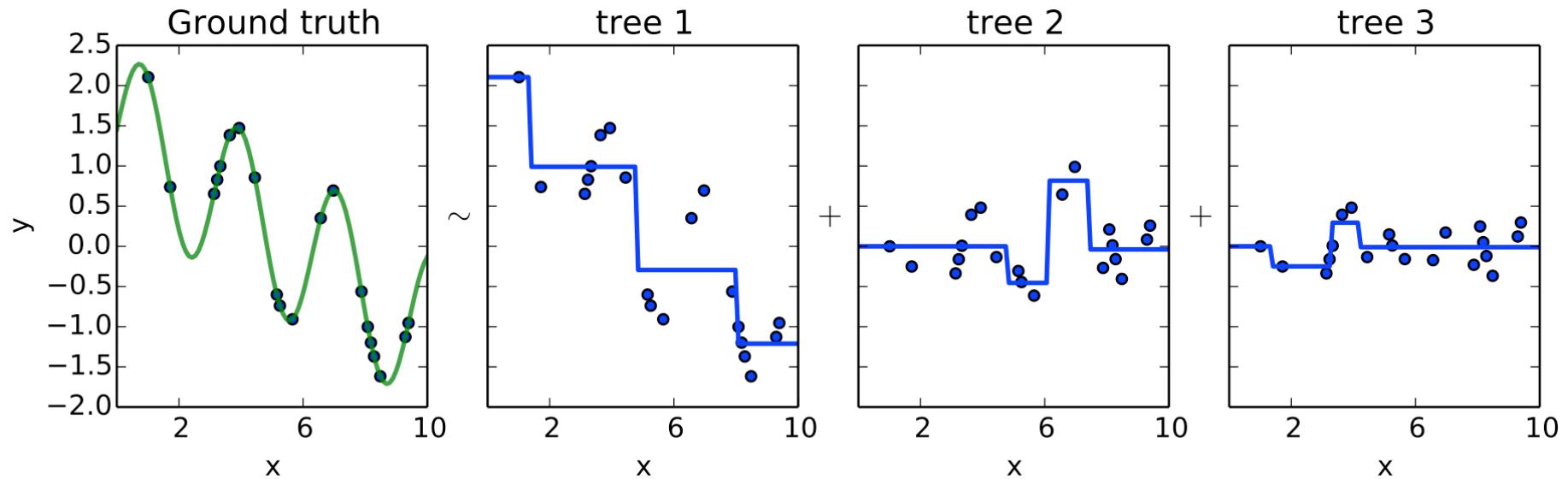
- Each tree is expert on attacking errors of predecessor
- Iteratively re-weights observations based on errors



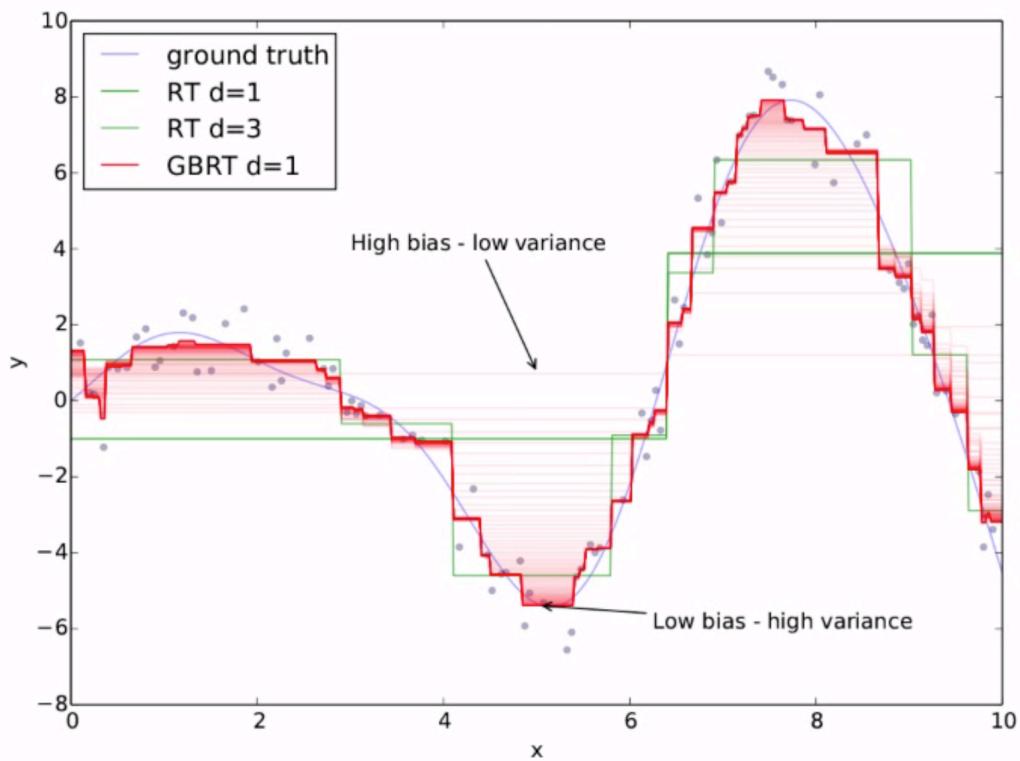
Boosting

Gradient Boosted Regression Trees

- Fit residuals to of previous tree
- Does not reweight training observations



Boosting

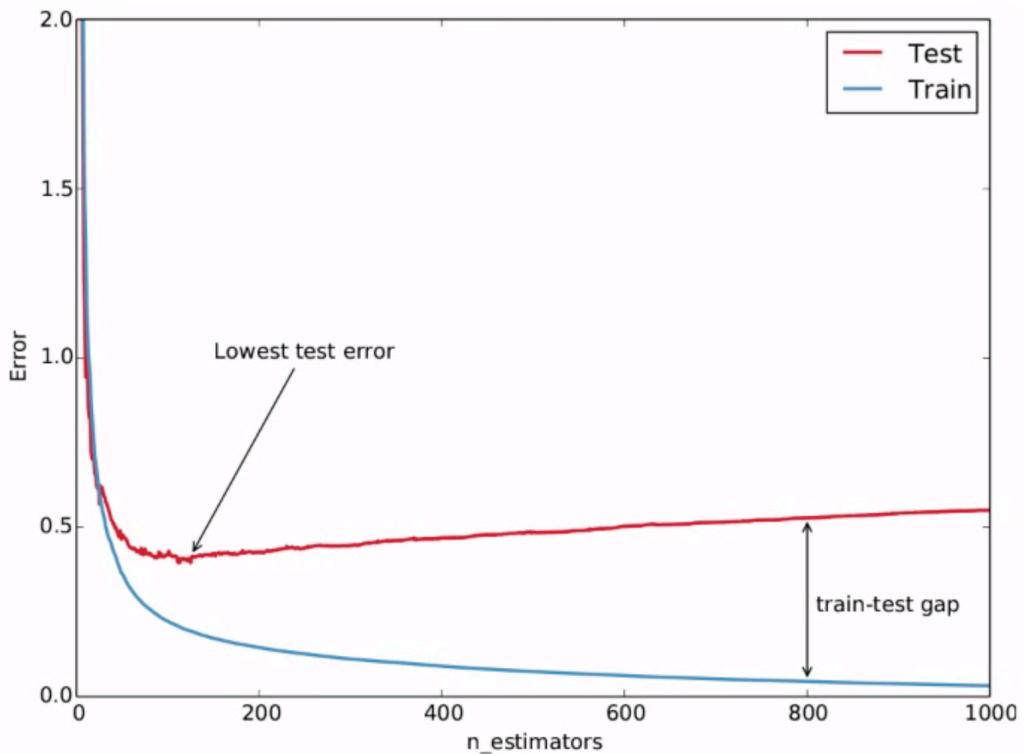
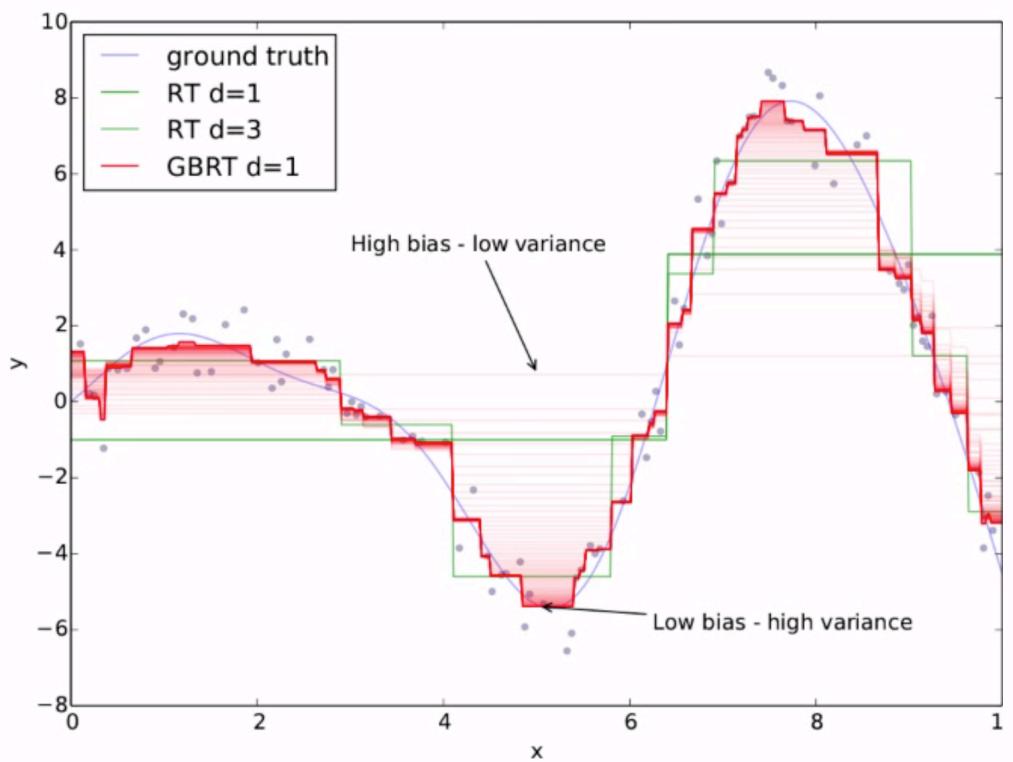


A single (boosted) tree,
→ High bias, Low variance

Many many (boosted) trees...
→ Low bias, High variance

Boosting

As number of trees grows....



Practical Issues

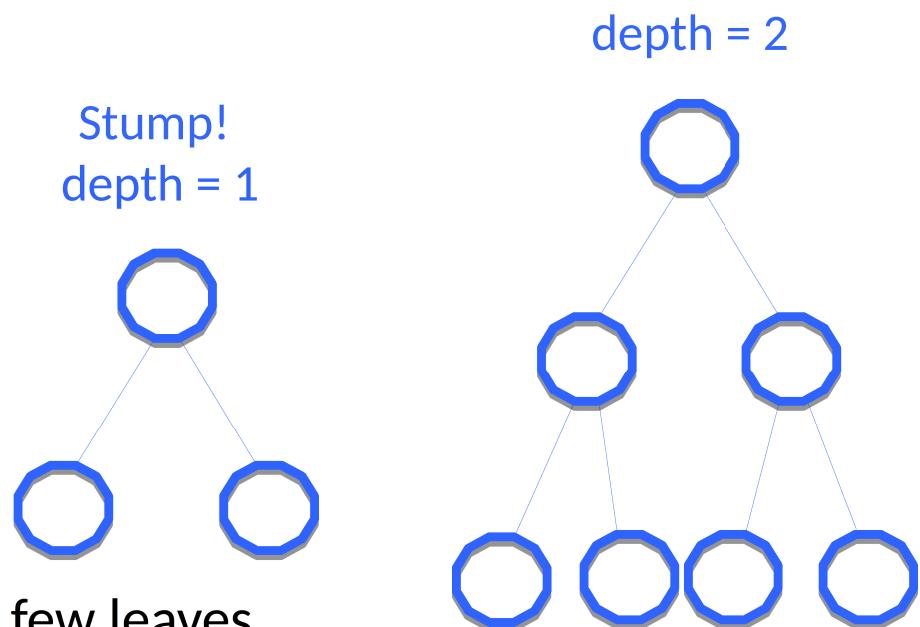
- Parameter tuning
- Using scikit-learn
- Diagnostics

Parameter tuning

- Must set tuning parameters correctly to obtain optimal performance
- Parameters control:
 - Tree structure
 - Shrinkage
 - Stochastic gradient boosting

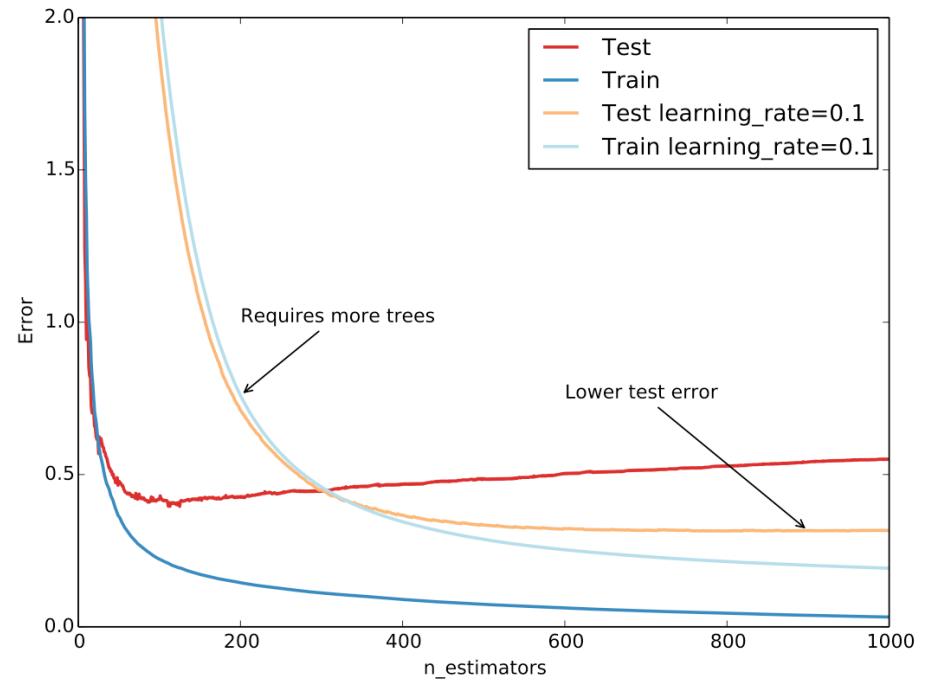
Tree structure

- max_depth
 - controls degree of interactions
 - Ex. Latitude and Longitude
 - not often larger than 4 or 6
- 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



As the **learning rate** goes down, the **number of trees** needed goes up!

Learning rate is a very important tuning parameter.
Number of trees also needs to be tuned.

Stochastic gradient boosting



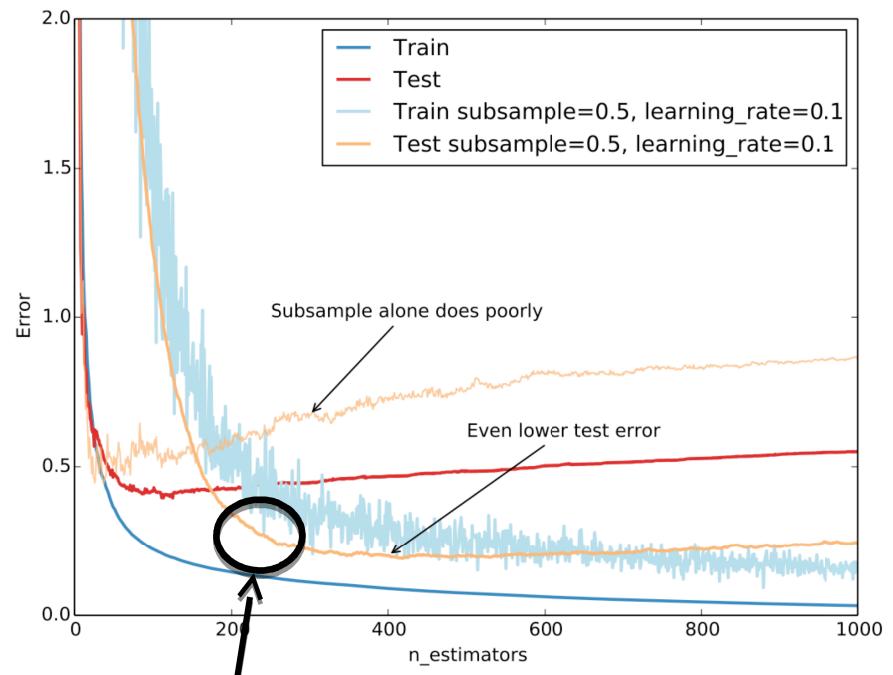
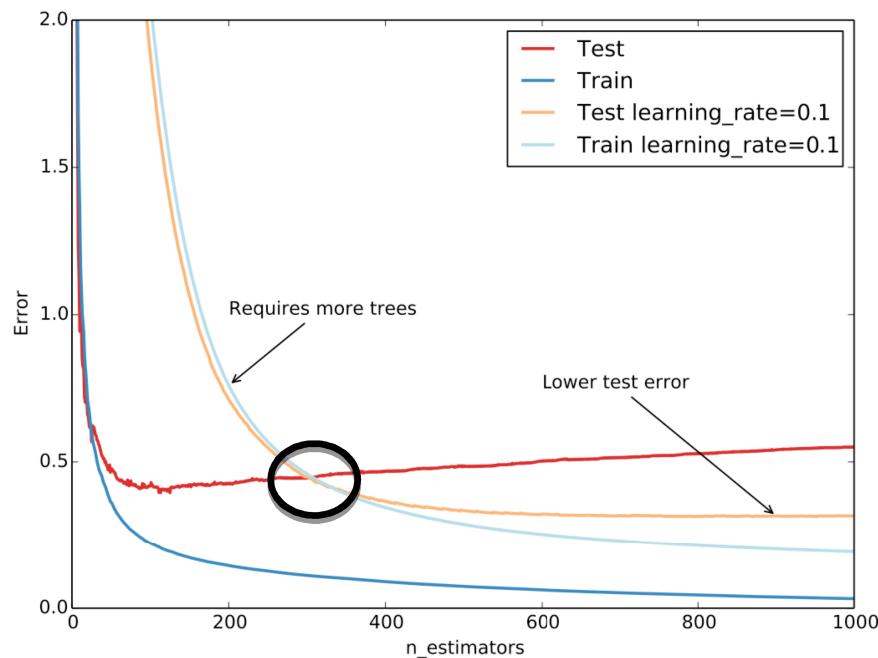
- max_features
 - random subsample of features
 - Especially good when you have lots of features
- sub_sample
 - random subset of the training set

Both **randomly sampling the features** and **randomly subsetting the training set** can lead to improved accuracy and reduced run-time

Pretty good deal!

Example: stochastic gradient boosting

Can improve accuracy and reduce runtime!



Lower test error!
Fewer trees to get there!

Example: setting parameters

```
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_
```

Best practice

Scikit-learn recommends:

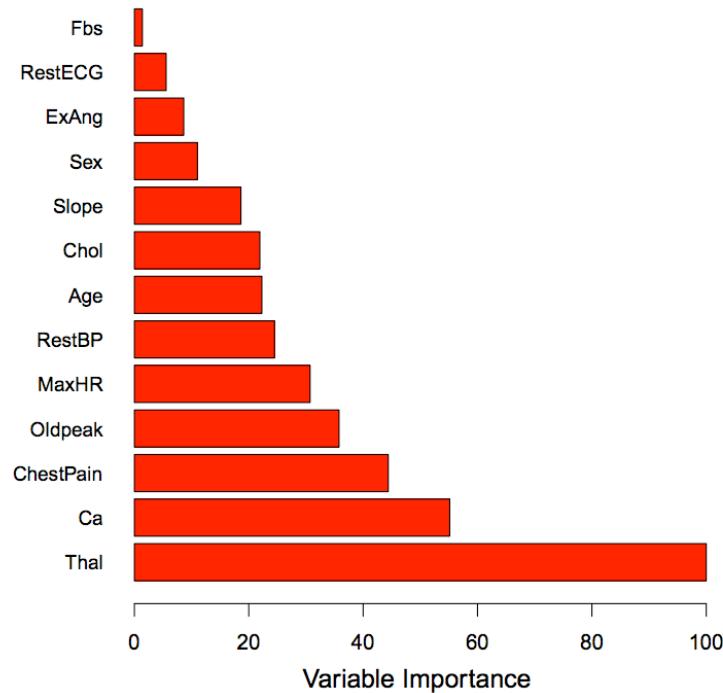
1. Set `n_estimators` high as possible (3000, 5000, etc.)
 2. Tune all hyper-parameters via grid search
 3. Further optimize `n_estimators` and `learning_rate`
- Scikit-learn uses canonical implementation from R's gbm package.
 - gbm's documentation provides additional advice on tuning.
- You must tune the model to get good performance!

Diagnostics

- Need to understand strengths & weaknesses of model
- Which features matter?
- Which interactions matter?
- Does this make sense based on your domain knowledge?
- Diagnostic tools:
 - Variable importance
 - Partial dependence plots

Variable Importance

- For bagged/RF regression trees, we record the total amount that the RSS is decreased due to splits over a given predictor, averaged over all B trees. A large value indicates an important predictor.
- Similarly, for bagged/RF classification trees, we add up the total amount that the Gini index is decreased by splits over a given predictor, averaged over all B trees.



Variable importance plot
for the **Heart** data

Partial Dependence Plots

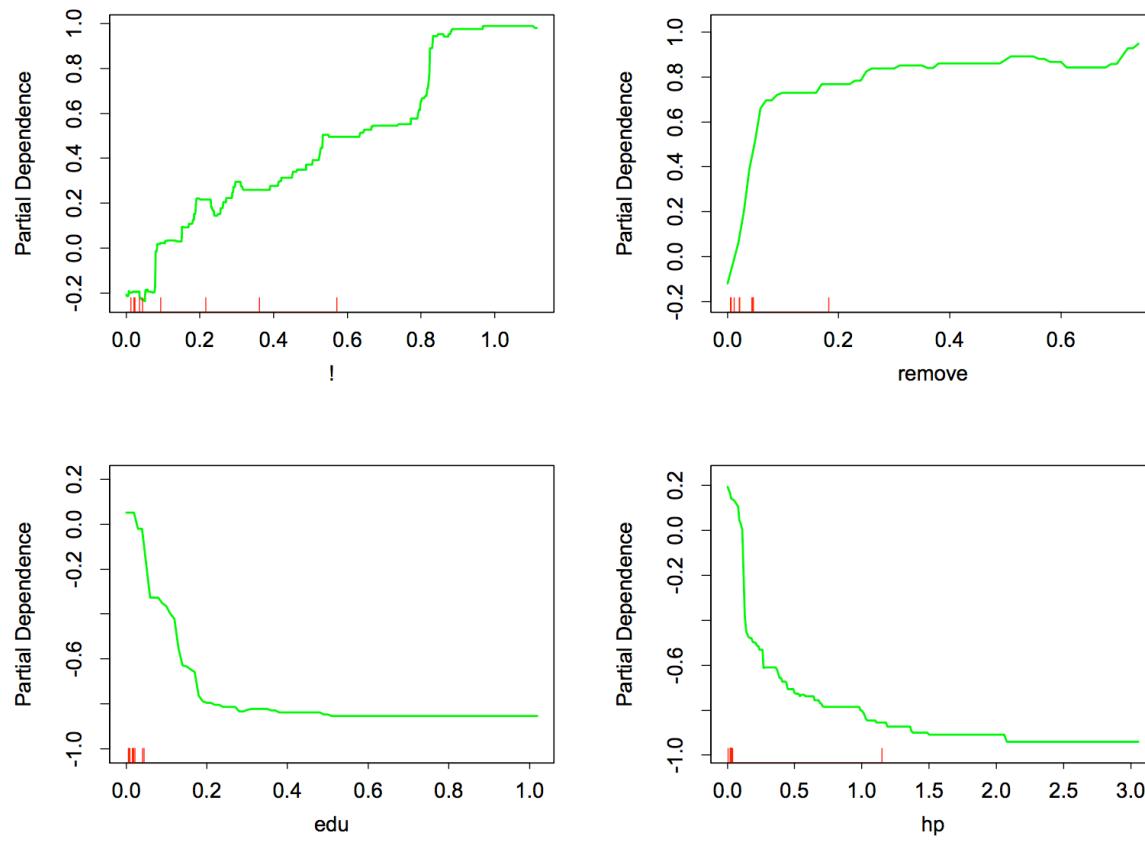


FIGURE 10.7. Partial dependence of log-odds of `spam` on four important predictors. The red ticks at the base of the plots are deciles of the input variable.

Show dependence between target function (in this case, proportion of spam as represented by the log-odds), marginalizing over the values of all other features.

Partial Dependence Plots (2 var)

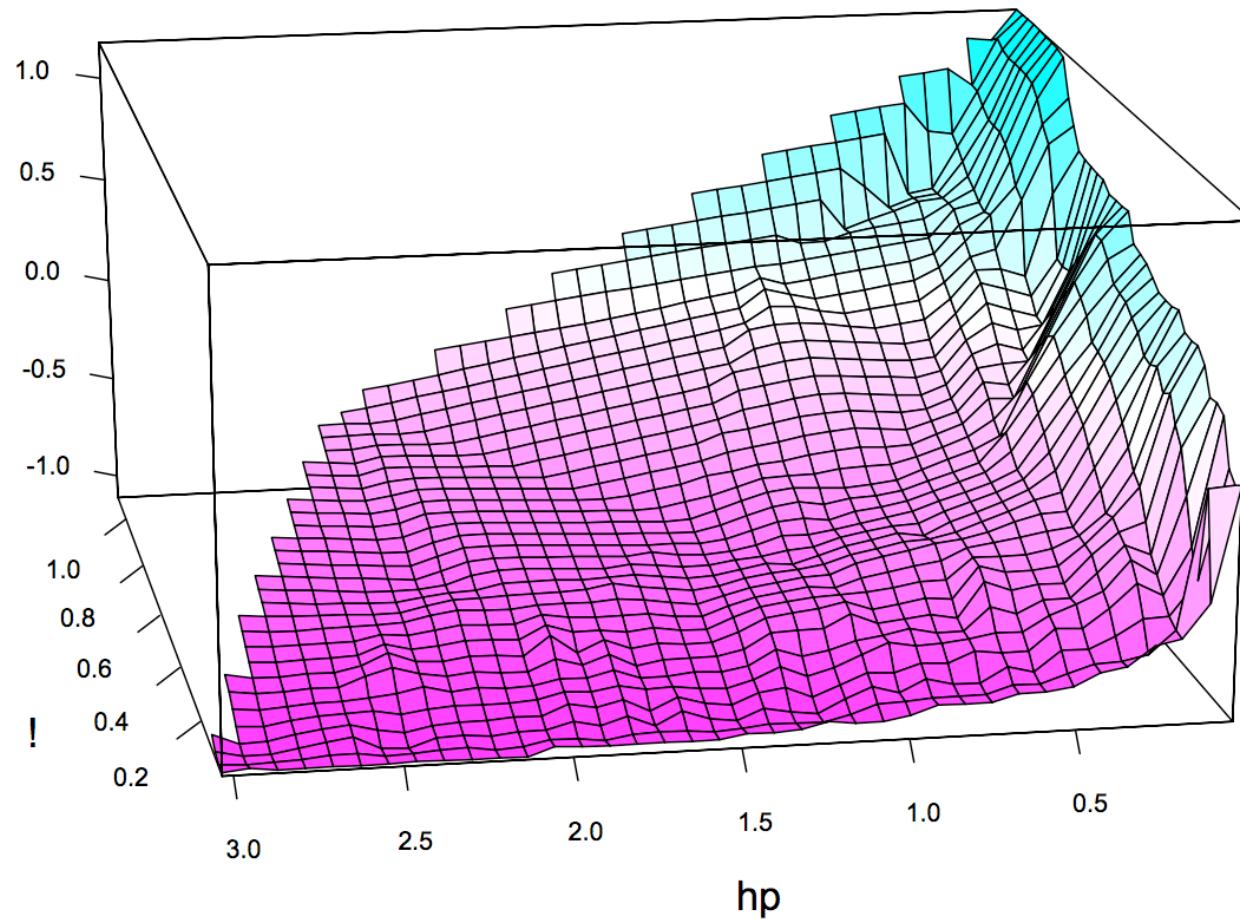


FIGURE 10.8. Partial dependence of the log-odds of `spam` vs. `email` as a function of joint frequencies of `hp` and the character `!`.

Review

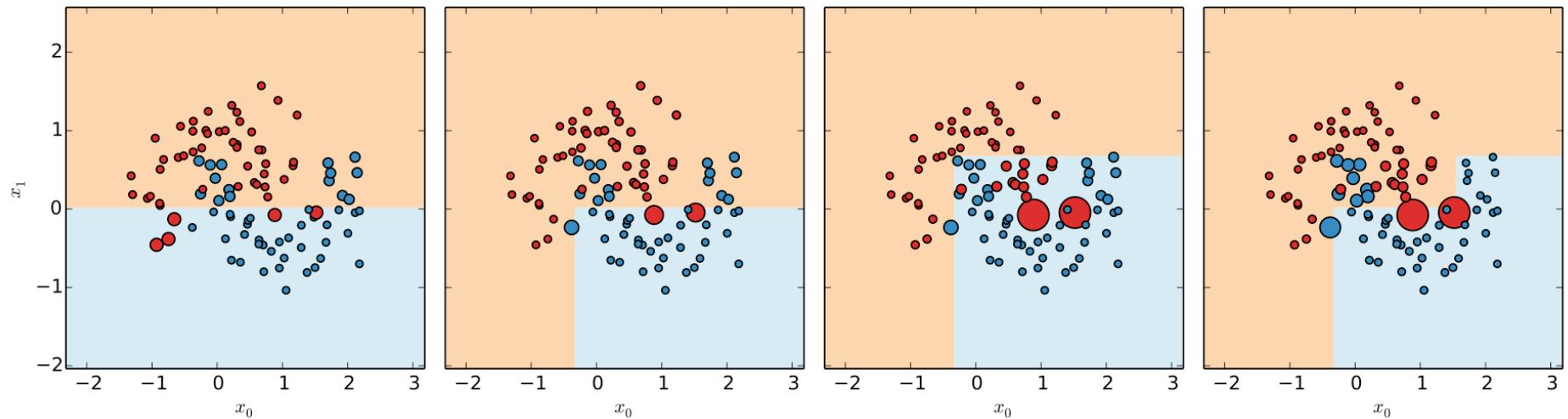
Q: What is another problem we should always check for when building a model with new data?

Q: What tool(s) can we use to check?

AdaBoost Algorithm

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)
- Formulation

Y : $\{-1, 1\}$
 $G(X)$: classifier producing predictions taking two values $\{-1, 1\}$
Error rate on the training set:

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

Discrete AdaBoost

1. Initialize the observation weights $w_i = 1/N, i = 1, 2, \dots, N.$
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 $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))], i = 1, 2, \dots, N.$
3. Output $G(x) = \text{sign} \left[\sum_{m=1}^M \alpha_m G_m(x) \right].$

Discrete AdaBoost

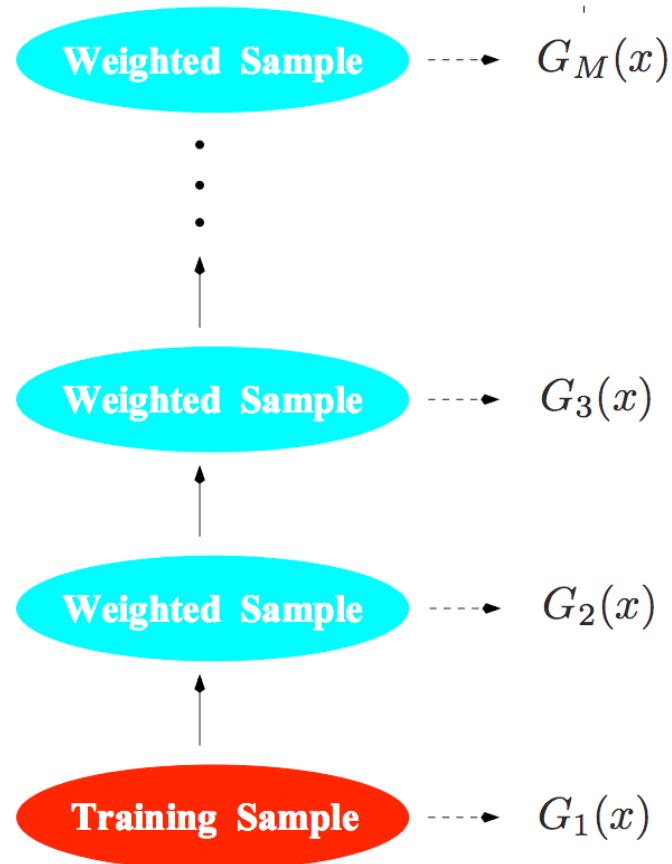
$G_i(x)$ weak classifiers

$G(x)$ strong learner

Note only $G_1(x)$ fit on training

FINAL CLASSIFIER

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



1. Initialize the observation weights $w_i = 1/N$, $i = 1, 2, \dots, N$.
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 $w_i \leftarrow w_i \cdot \exp[\alpha_m \cdot I(y_i \neq G_m(x_i))]$, $i = 1, 2, \dots, N$.
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^{α_m}
 - else, w_i same
- Final strong classifier $G(x)$ determined by weighted majority votes
 - $\alpha_1, \dots, \alpha_M$ as weight of votes
 - The smaller the error of the weak classifier, the greater the weight

Observe that...

- $\alpha_1, \dots, \alpha_M$ 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

Algorithm: GBRT

1. Set $\hat{f}(x) = 0$ and $r_i = y_i$ for all i in the training set.
2. For $b = 1, 2, \dots, B$, 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:

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

- 2.3 Update the residuals,

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

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/I_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

Very very roughly,

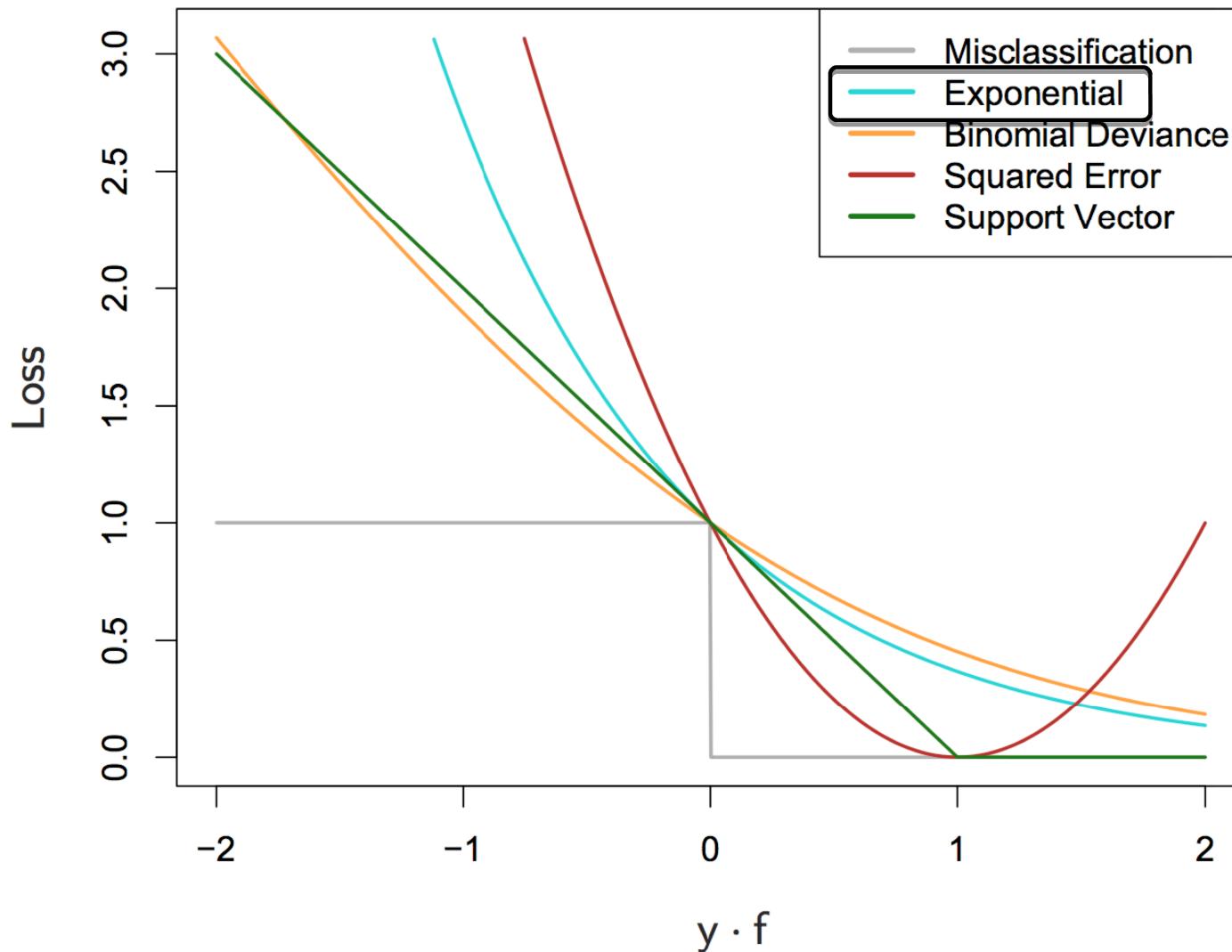
- It is a version of **Forward Stagewise Additive Modeling**, which adds new basis functions without adjusting previous parameters and coefficients.
 - This contrasts with gradient boosting, see pg. 342 vs. pg. 361 in link above
 - AdaBoost uses **Exponential Loss**
 - It can be shown that to minimize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

which justifies taking the sign as classification rule for final classifier

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

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

Comparison of Loss Functions for Classification



Comparison of Loss Functions for Regression

