Boosting

Fabio G. Cozman - fgcozman@usp.br

October 30, 2019

Basics

- Consider a classification problem with a binary class variable Y with values 1 and -1.
- Use X to refer to all features $X_1, \ldots, X_n$.

Basics

- Consider a classification problem with a binary class variable Y with values 1 and -1.
- Use X to refer to all features $X_1, \ldots, X_n$.
- Suppose you have *weak classifiers* $g_k(X)$ for $k = 1, \ldots, M$.
  - These classifiers are "weak" in the sense that they can be only a bit better than random guessing, but they may be accurate ones if possible.

The basic idea

- The idea is to somehow combine these classifiers to produce a "boosted" classifier.
  - Apply each weak classifier to a modified version of the training dataset.
- While g_1 is learned on the original training data, each g_k is learned on some weighted version of the training dataset.

Final result:

- The boosted classifier is

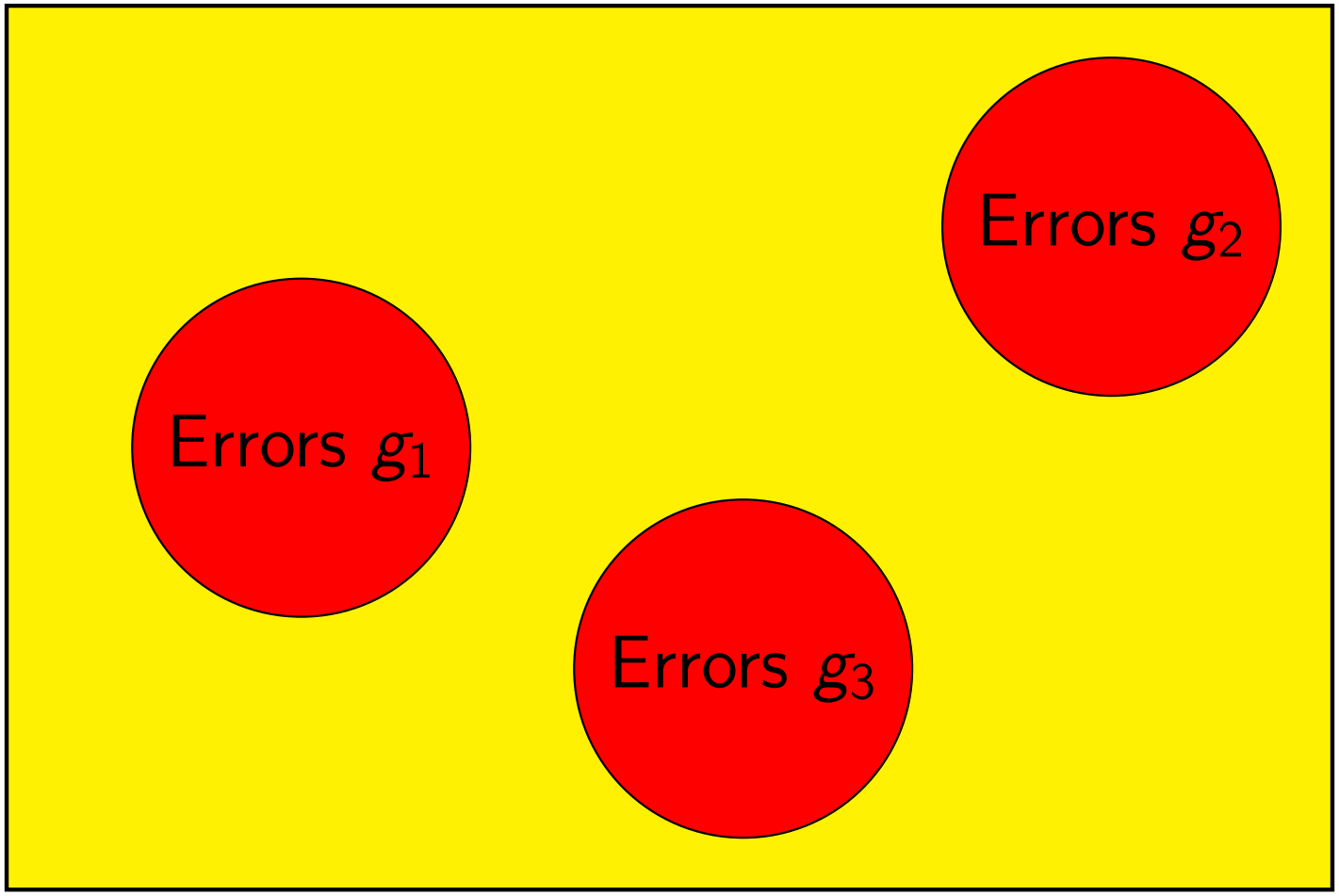
$$g(x) = \text{sign}\left(\sum_{k=1}^{M} \alpha_k g_k(x)\right),$$

for some α_k.

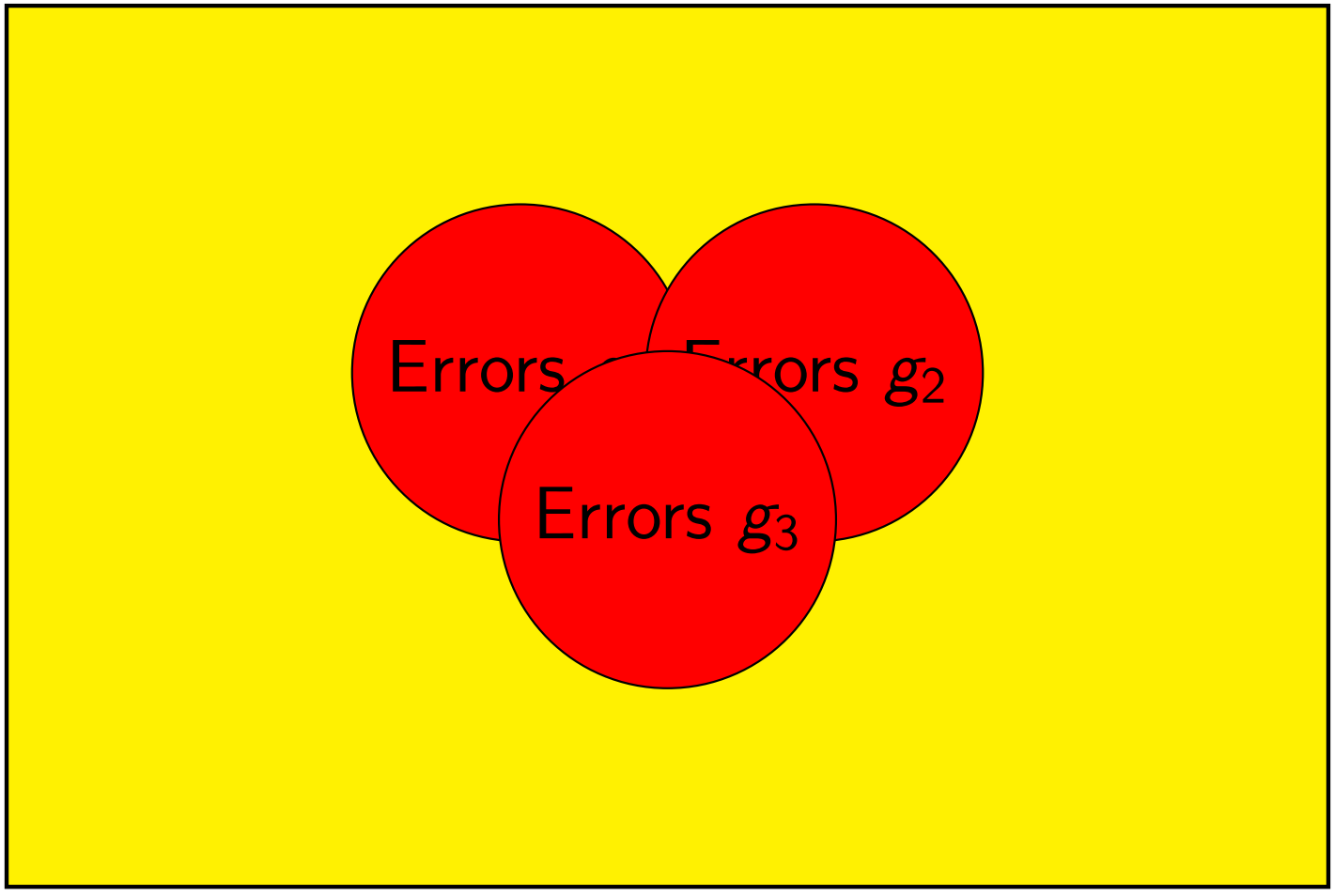
A simple case

- Suppose we have three classifiers and take "majority vote".

A perfect majority vote



Majority vote, generic case



Simple idea

1. First, learn g_1.
2. Then learn g_2, emphasizing that it must do the right thing whenever g_1 makes a mistake.
3. Then learn g_2, emphasizing that it must do the right thing whenever g_1 and g_2 make a mistake.

Most famous algorithm: AdaBoost

1. Initialize $w_j = 1/N$ for $j \in \{1, \ldots, N\}$.
2. For k from 1 to M:
   1. Fit $g_k(X)$ to training data using weights w_j.
   2. Compute
   $$\beta_k = \frac{\sum_{j=1}^{N} w_j I_{y_j \neq g_k(x^j)}}{\sum_{j=1}^{N} w_j}.$$
   3. Compute $\alpha_k = \ln((1 - \beta_k)/\beta_k)$.
   4. Multiply each weight w_j by $\exp(\alpha_k I_{y_j \neq g_k(x^j)})$.
3. Output $g(x) = \text{sign} \sum_{k=1}^{M} \alpha_k g_k(x)$.

Why??

- AdaBoost was derived from intuitive ideas, using probability bounds to "justify" the results.
- No really general theory.

Popular scheme: AdaBoost with stumps

- A *stump* is a tree with a single split.
- A stump is usually a weak classifier.
- Boosted classifiers based on a large number of stumps work very well.

Boosting as approximate optimization

- Suppose we want to find

$$\min_{g} \sum_{j=1}^{N} L\left(y_j, g(x^j)\right),$$

where $g(x) = \sum_{k=1}^{M} \alpha_k g_k(x)$, with g_k using some parameters γ_k.

Boosting as approximate optimization

- Suppose we want to find

$$\min_g \sum_{j=1}^{N} L\left(y_j, g(x^j)\right),$$

where $g(x) = \sum_{k=1}^{M} \alpha_k g_k(x)$, with g_k using some parameters γ_k.

- Suppose it is easy to find, for each k,

$$\min_{\alpha, g_k} \sum_{j=1}^{N} L(y_j, f(x^j) + \alpha g_k(x^j))$$

for some given f.

- Then there is special optimization method...

Forward stagewise additive minimization

1. Start with $f_0(x) = 0$.
2. For k from 1 to M:
    1. Find α and g_k that minimize

$$\sum_{j=1}^{M} L(y_j, f_{k-1}(x^j) + \alpha g_k(x^j)).$$

    2. Set $f_k = f_{k-1} + \alpha g_k$.
3. Output $g(x) = f_M(x)$.

Relation to AdaBoost

- If $L(y, g(x)) = \exp(-yg(x))$, then f.s.a.m. produces "AdaBoost with output $\sum_k \alpha_k g_k(x)$".

- Note: without the sign in the output, so it is not exactly AdaBoost but it is close enough.

Quadratic loss

- If $L(y, g(x)) = (y - g(x))^2$, then

$$L(y_j, f_{k-1}(x^j) + \alpha g_k(x^j)) = (y_j - f_{k-1}(x^j) - \alpha g_k(x^j))^2.$$

- In this case, as $y_j - f_{k-1}(x^j)$ is the residual from the previous iteration, we are basically regressing on the "errors from the previous iteration".

Boosting in regression trees

1. Set $r_j = y_j$ for all observations in the training dataset.
2. For $k = 1, \ldots, M$, repeat:
   1. Fit a tree $\hat{f}^k$ with d splits to data (X, r).
   2. Update: $r_j \leftarrow r_j - \lambda \hat{f}^k(x_j)$.
3. Output $\hat{f}(x) = \sum_{k=1}^{M} \lambda \hat{f}^k(x)$.

Boosting in regression trees

1. Set $r_j = y_j$ for all observations in the training dataset.
2. For $k = 1, \ldots, M$, repeat:
   1. Fit a tree $\hat{f}^k$ with d splits to data (X, r).
   2. Update: $r_j \leftarrow r_j - \lambda \hat{f}^k(x_j)$.
3. Output $\hat{f}(x) = \sum_{k=1}^{M} \lambda \hat{f}^k(x)$.

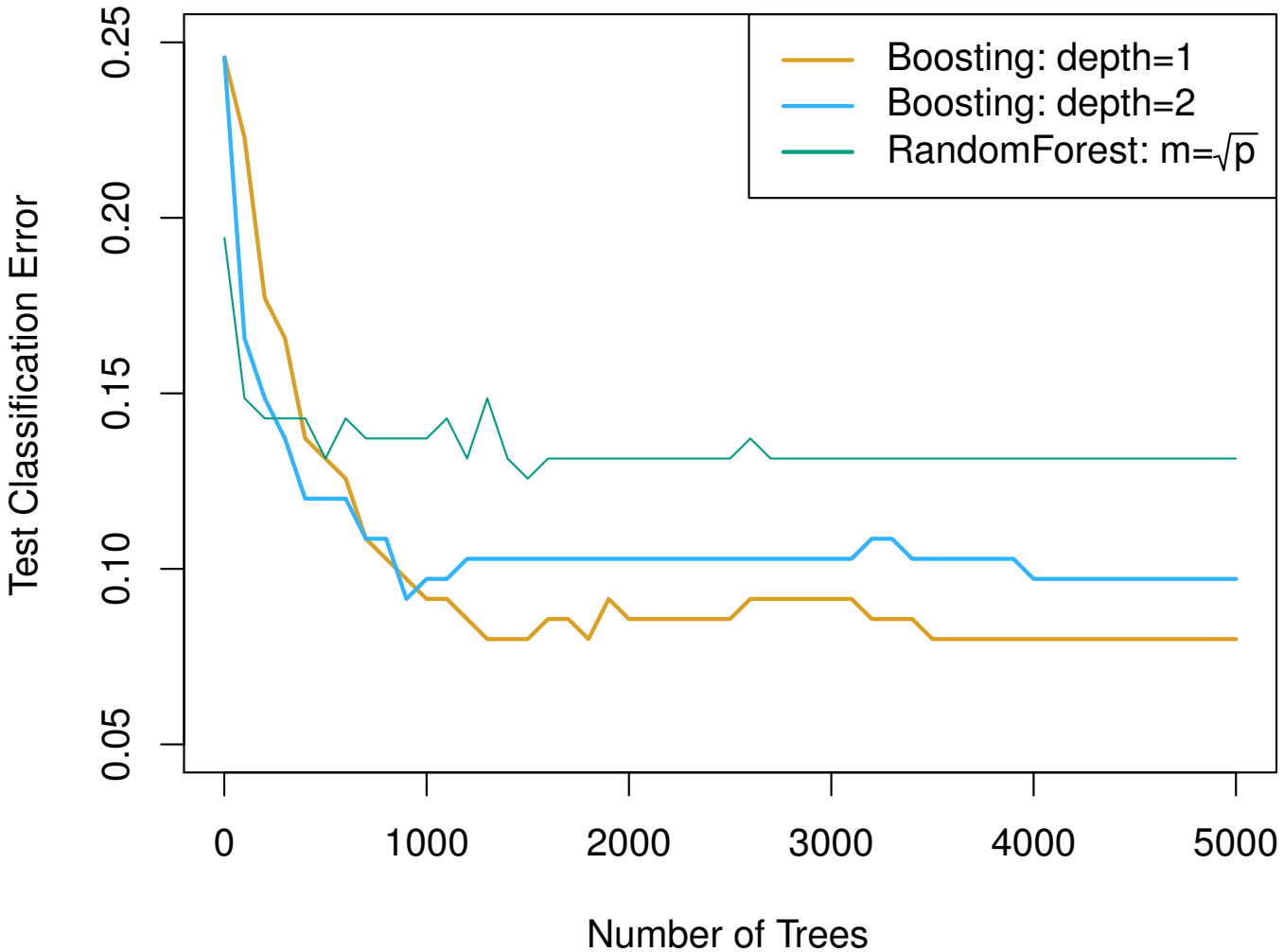
Note:

- d is a parameter, often $d = 1$ (stumps).
- λ is a parameter, usually small (0.01, 0.001, ...).
- M is a parameter, usually large (100, 1000, ...).

Some intuition

- If $\lambda = 1$, then r_j is the "residual".
- Smaller λ "dampens" the residuals.

Gene expression: 2 labels, 500 features



Single tree: error rate about 24%.

A note

Some of the figures in this presentation are taken from *An Introduction to Statistical Learning, with applications in R* (Springer, 2013) with permission from the authors: G. James, D. Witten, T. Hastie and R. Tibshirani.