Project 2: Predictive Modelling

STAT GU4243

Applied Data Science

Cynthia Rush
Columbia University

February 19, 2018

Curse of Dimensionality

Assumptions Made in Classification

Assumptions

- Probability of class labels are continuous over feature values.
- The distance metric or kernel function is meaningful for the classification problem.
- Test sample will be drawn from the same distributions as the training sample, $p(x, y)$.

If you have infinite data and unlimited computational power and storage, classification is easy. Often not the case...

- For finite-size training sample, don't have enough observations to make predictions everywhere.
- Bayes rate can tell us about theoretical limits of performance.

High Dimensions

Recall: p is the dimension of the input space.

Assume inputs are uniformly distributed in a p-dimensional unit cube.

- Suppose we construct a hypercubical neighborhood about a target point to capture a fraction r of the observations. This corresponds to a fraction r of the unit volume, so the expected edge lengthis $e_p(r) = r^{1/p}$.
- In ten dimensions $e_{10}(0.01) = 0.63$ and $e_{10}(0.1) = 0.80$, while the entire range is only 1.0. So to capture 1% or 10% of the data, we must cover 63% or 80% of the range of each input.
- Such neighborhoods are no longer "local."

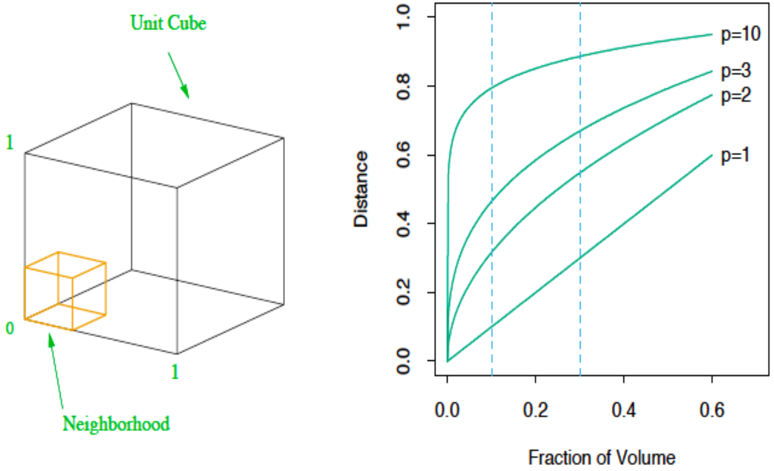

FIGURE 2.6. *The curse of dimensionality is well illustrated by a subcubical neighborhood for uniform data in a unit cube. The figure on the right shows the side-length of the subcube needed to capture a fraction r of the volume of the data, for different dimensions p. In ten dimensions we need to cover 80% of the range of each coordinate to capture 10% of the data.*

Curse of Dimensionality

Recall: p is the dimension of the input space.

Assume inputs are uniformly distributed in a p-dimensional unit cube.

- For a random point $\{x_1, \ldots, x_p\}, x_i \sim \text{Unif}(0,1),$ iid.
- It can be shown that $\mathbb{E}\min(x_i) = \frac{1}{p+1}$.
- This implies that for any point, it is very close to at least one boundary. Inference at the boundary is usually difficult.

Variance-Bias Decomposition

Model

$$Y = f(X) + \epsilon \quad \text{with } \mathbb{E}[\epsilon] = 0 \quad \text{and } \mathrm{Var}[\epsilon] = \sigma^2$$

Let's consider the expected prediction error (EPE) for a model $\hat{f}$ using L_2 loss at a single test point (x_0, y_0):

$$\begin{aligned}
\mathrm{EPE}(\hat{f}) &= \mathbb{E}(y_0 - \hat{f}(x_0))^2 \\
&= \mathbb{E}(f(x_0) + \epsilon - \mathbb{E}(\hat{f}(x_0)) + \mathbb{E}(\hat{f}(x_0)) - \hat{f}(x_0))^2 \\
&= \mathbb{E}(\epsilon^2) + (f(x_0) - \mathbb{E}(\hat{f}(x_0)))^2 + \mathrm{var}(\hat{f}(x_0)) \\
&= \text{Irreducible error} \; + \; \text{Bias}^2 + \mathrm{var}(\hat{f}(x_0))
\end{aligned}$$

where

$$\text{Irreducible error } = \sigma^2,$$

and

$$\text{Bias} = f(x_0) - \mathbb{E}(\hat{f}(x_0)).$$

How Dimensions Affect Estimation

True relationship: $Y = e^{-8\|X\|^2}$ (no measurement error).

Use nearest-neighbor to estimate Y at $X = 0$.

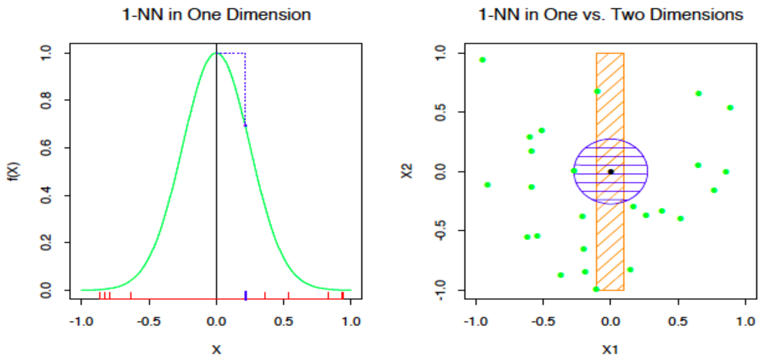

FIGURE 2.7. *A simulation example, demonstrating the curse of dimensionality and its effect on MSE, bias and variance. The input features are uniformly distributed in* $[-1, 1]^p$ *for* $p = 1, \dots, 10$ *The top left panel shows the target function (no noise) in* $\mathbb{R}$: $f(X) = e^{-8||X||^2}$, *and demonstrates the error that* 1*-nearest neighbor makes in estimating* $f(0)$. *The training point is indicated by the blue tick mark. The top right panel illustrates why the radius of the* 1*-nearest neighborhood increases with dimension p. The lower left panel shows the average radius of the* 1*-nearest neighborhoods. The lower-right panel shows the MSE, squared bias and variance curves as a function of dimension p.*

How Dimensions Affect Estimation

- As the number of dimensions increase, the distance of the nearest neighbor to $X = 0$ increases.
- The nearest neighbor estimate is therefore down-biased.
- The function $Y = f(X)$ is symmetric about different dimensions of X.
- Actually, it only depends on the distance between the nearest neighbor at $X = 0$.
- The variability of the estimate is then decided by the variability of the distance to NN.

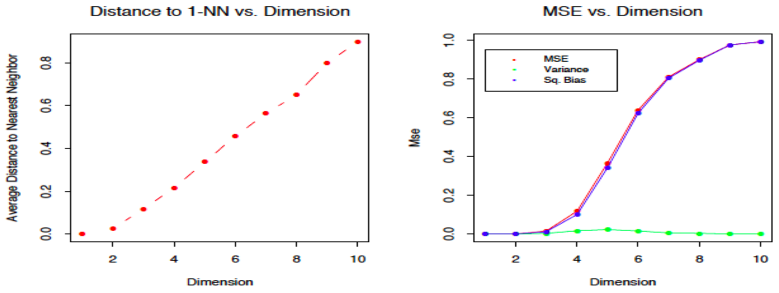

FIGURE 2.7. *A simulation example, demonstrating the curse of dimensionality and its effect on MSE, bias and variance. The input features are uniformly distributed in* $[-1,1]^p$ *for* $p = 1, \dots, 10$ *The top left panel shows the target function (no noise) in* $\mathbb{R}$*:* $f(X) = e^{-8||X||^2}$*, and demonstrates the error that* 1*-nearest neighbor makes in estimating* $f(0)$*. The training point is indicated by the blue tick mark. The top right panel illustrates why the radius of the* 1*-nearest neighborhood increases with dimension* p*. The lower left panel shows the average radius of the* 1*-nearest neighborhoods. The lower-right panel shows the MSE, squared bias and variance curves as a function of dimension* p*.*

How Dimensions Affect Estimation

- Consider another case: $Y = \frac{1}{2}(X_1 + 1)^3$.
- The function depends on only one dimension, i.e., the other dimensions are irrelevant for learning this function.
- This function doesn't peak at 0 and therefore the bias isn't as prominent.
- Variablity of the estimate depnds on distance to NN along X_1, which increases as the number of irrelevant dimensions increases.

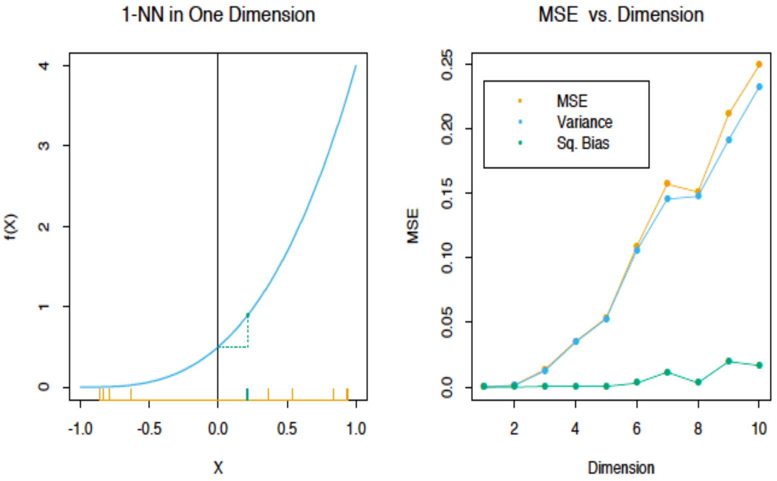

FIGURE 2.8. *A simulation example with the same setup as in Figure 2.7. Here the function is constant in all but one dimension:* $F(X) = \frac{1}{2}(X_1 + 1)^3$*. The variance dominates.*

Statistical Models

Predictive Relation

How We Build Predictors

Predictive relation between Y and X depends on the definition of "goodness of fit", usually determined a loss function.

Examples:

- **$\mathbf{L_2}$ Loss**: Best Estimate $f(x) = \mathbb{E}(Y|X = x)$.
- **$\mathbf{L_1}$ Loss**: Best Estimate $f(x) = \text{median}(Y|X = x)$.

kNN

- Nearest neighbor can be viewed as local direct estimates of $f(x)$.
- BUT, nearest neighbor methods run into trouble when the dimension of the input space becomes large.
- Moreover, if the relation between Y and X is known to be more structured, kNN methods aren't optimal.

Predictive Relation

For functions f and g that satisfy

$$f(x_i) = g(x_i), \quad i = 1, \ldots, n,$$

their fit to the observed data $(x_i, y_i); i = 1, \ldots, n$ is the same.

The above fact leads to the definition of some kinds of equivalent models.

Identifiability

- By constraining the model family, the hope is that within the model class, there are no equivalent models].
- When this is not the case, can have identifiability issues. Which model – f or g in the above definition – do we use?

Predictive Relation

Occam's Razor

General belief: the more complicated a model, the more likely it is to give predictions far away from the truth at points not close to observed x_i's.

- Complexity of models usually controlled with constraints.
- Generally require that the estimated model exhibit some kind of regular behavior in small neighborhoods of the input space.
- Modeling usually carried out using structured model families, basis expansions, and kernel/local regression.

How do we set the model parameters determining the smoothness or complexity?
(multiplier of penalty term, width of kernel, number of basis functions, ...)

Model Assessment and Selection

Why is this a Challenge?

Training Error

For training data $(y_1, x_1), (y_2, x_2), \ldots, (y_n, x_n)$, the RSS of a classification function f is

$$\bar{\text{err}}(f) = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}\{y_i \neq f(x_i)\}.$$

The above counts the misclassifications of the classifer on the training data.

- Can't use RSS on training data to determine model parameters, since we would always pick the parameter giving interpolating fits and zero residuals.
- E.g. for kNN, we would choose $\mathsf{k} = 1$.
- Such a model is unlikely to predict future data well at all.

Generalization

The *generalization* performance of a learning method relates to its prediction capability on independent test data.

Test Error vs. Training Error

Loss function: $L(Y, \hat{f}(x))$ defines prediction errors (e.g., squared error, absolute error, 0-1 classification error).

Test Error

Test error or **generalization error** is the expected prediction error over an independent test sample:

$$\mathrm{Err}(\hat{f}) = \mathbb{E}[L(Y, \hat{f}(X))].$$

Expectation takes into account the sampling variability in the training data used to fit $\hat{f}$ (including sample size effects).

Training Error

Training error is the average loss of the training set

$$\overline{\mathrm{err}}(\hat{f}) = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}(x_i)),$$

and, in general, isn't a good estimate of the test error.

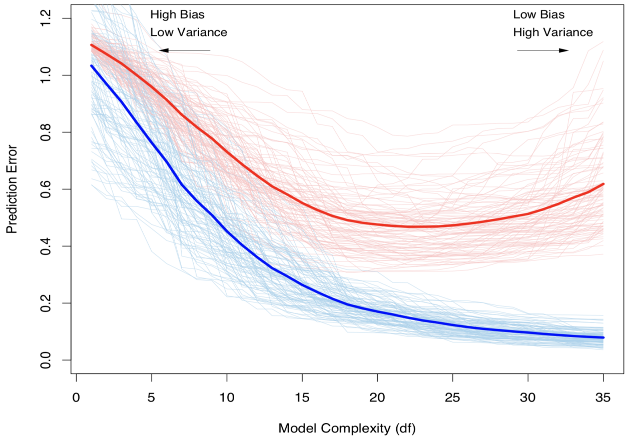

FIGURE 7.1. *Behavior of test sample and training sample error as the model complexity is varied. The light blue curves show the training error* $\overline{\text{err}}$*, while the light red curves show the conditional test error* $\text{Err}_{\mathcal{T}}$ *for* 100 *training sets of size* 50 *each, as the model complexity is increased. The solid curves show the expected test error* Err *and the expected training error* $\text{E}[\overline{\text{err}}]$.

Test error and training error vs. model complexity.

Model Assessment and Selection

We want to study methods for estimating the expected test error for a model.

Possible Goals:

Model Selection: Use training data to choose the (approximately) best model among a group of candidates.

Model Assessment: Use available data to evaluate the performance of the final model.

Data Division

- In data-rich situation: training, validation, test.
- No general rule on how to choose number of observations in each partition.
- Example given by the "statistical learning" book: 50%, 25%, 25%.

Cross-validation (CV)

What do we do when there is insufficient data to split it into three parts?

- CV directly estimates the test error by using non-overlapping training data and test data.
- K-fold CV layout:

1	2	...	K-1	K
Test	Train	...	Train	Train

1	2	...	K-1	K
Train	**Test**	...	Train	Train

...

1	2	...	K-1	K
Train	Train	...	Train	**Test**

K estimates of test errors are combined to estimate the test error of the training set.

Cross-validation

More specifically, test error is estimated by

$$\widehat{\text{ERR}}_{CV} = \frac{1}{N} \sum_{k=1}^{K} \sum_{i \in \text{ block } k} L(y_i, \hat{f}_{-k}(x_i)),$$

where $\hat{f}_{-k}$ is model fit with k^{th} section removed.

Choosing K

- $K = N$ is 'leave-one-out' and training sets will be too similar.
- Small K: training set is too small compared to total training set. Loss of efficiency since want to estimate test error at the total training set size.
- Typical choices are $K = 5$ or $K = 10$.

Cross-validation

Model selection with CV

Choose model with the best CV test error.

One-standard-error Rule

Since K-fold CV also allows estimation of standard error of test errors, choose most parsimonious model whose error is no more than one standard error above error of the "best" model.

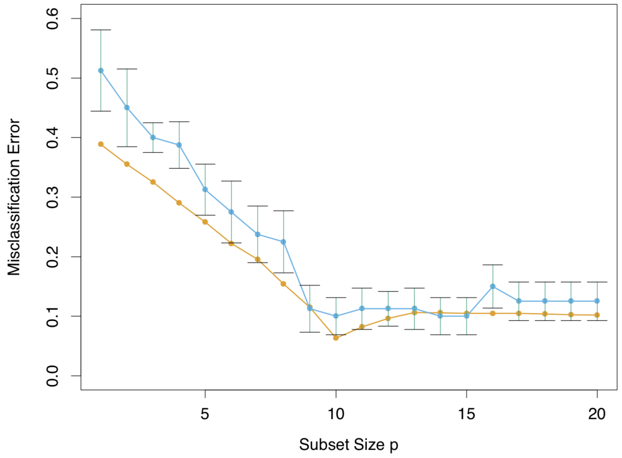

FIGURE 7.9. *Prediction error (orange) and tenfold cross-validation curve (blue) estimated from a single training set, from the scenario in the bottom right panel of Figure 7.3.*

Basic Bootstrap

The bootstrap is a general tool for assessing statistical accuracy.

- Randomly resample datasets (having the same size as the original data) with replacements from the training data (x_i, y_i).
- Produce B 'bootstrap' datasets in the above manner.
- Refit model to each of the bootstrap datasets.

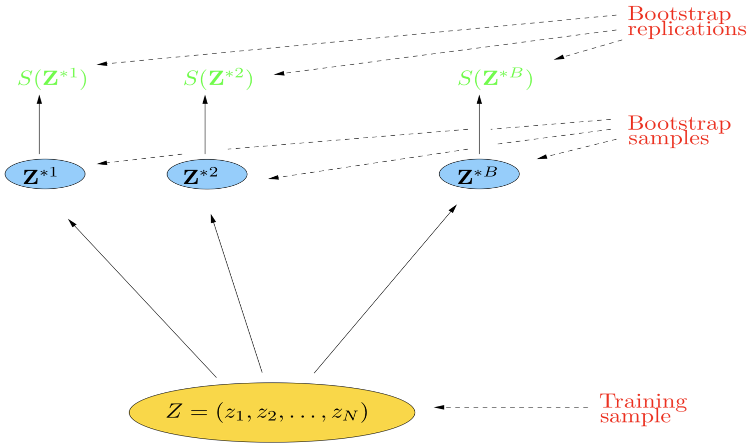

FIGURE 7.12. *Schematic of the bootstrap process. We wish to assess the statistical accuracy of a quantity $S(\mathbf{Z})$ computed from our dataset. B training sets $\mathbf{Z}^{*b}$, $b = 1, \ldots, B$ each of size N are drawn with replacement from the original dataset. The quantity of interest $S(\mathbf{Z})$ is computed from each bootstrap training set, and the values $S(\mathbf{Z}^{*1}), \ldots, S(\mathbf{Z}^{*B})$ are used to assess the statistical accuracy of $S(\mathbf{Z})$.*

Bootstrap for Model Assessment

How do we use the bootstrap to estimate test error?

Basic Set-up

Test error estimated by fitting the model to bootstrap resamples and tracking how well it predicts the original training set:

$$\widehat{\text{Err}}_{\text{boot}} = \frac{1}{B} \sum_{b=1}^{B} \hat{\text{Err}}^{b}$$

where

$$\widehat{\text{Err}}^{b} = \frac{1}{N} \sum_{i=1}^{N} L(y_i, \hat{f}^{b}(x_i))$$

and $\hat{f}^{b}(x_i)$ is predicted value at x_i by model fit on b^{th} bootstrap sample.

- Bootstrap samples act as training sets, while original training sample act as test set.
- Problem: bootstrap sample and original sample have common observations.

Bootstrap for Model Assessment

Mimicking CV, a better bootstrap can be obtained.

Leave-one-out Bootstrap

- For each observation (y_i, x_i), only keep track of predictions from bootstrap samples not containing that observation. Denote C^{-i} as the set of such bootstrap samples, b.
- Probability of a bootstrap sample not containing observation (y_i, x_i) is $1 - (1 - \frac{1}{N})^N \approx 1 - e^{-1} = 0.632$.
- The leave-one-out bootstrap estimate of test error is then

$$\widehat{\text{Err}}^{(1)} = \frac{1}{N} \sum_{i=1}^{N} \frac{1}{|C^{-i}|} \sum_{b \in C^{-i}} L(y_i, \hat{f}^b(x_i)).$$

Model Ensemble

The idea of ensemble learning is to build a prediction model by combining the strengths of a collection of simpler base models.

Model Averaging: Bagging

Idea: Use the bootstrap to improve the estimate or prediction itself.

- Bagging averages predictions over a collection of bootstrap samples, thereby reducing variance. Baggin estimate:

$$\hat{f}_{\text{bag}}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{\,b}(x)$$

- Different from model built with bootstrap estimate of model parameters, when the model is nonlinear.
- For classification, $\hat{f}$ usually estimates probabilities for each class and prediction is most probable class.
- Bagged predictor for classification can take two forms:
  - bag the predictions (weighted votes).
  - bag the class probabilities.

Model Averaging: Weighted Averages of Candidate Models

Candidate models may be:

1. Same type with different parameter values (e.g., subsets in regression),
2. Different models for same task (e.g., neural nets and regression trees).

Committee Methods

Final model is simple unweighted "average" of fitted individual $\hat{f}_m$'s:

$$\hat{f}(x) = \frac{1}{M} \sum_{m=1}^{M} \hat{f}_m(x; \hat{\Theta}_m).$$

Model Averaging

Candidate models $M_1, ..., M_M$ can be averaged based on $\Pr(M_m|\mathbf{Z})$ where $\mathbf{Z}$ is our training data.

$$\hat{f}(x) = \frac{1}{M} \sum_{m=1}^{M} \hat{f}_m(x; \hat{\Theta}_m) \Pr(M_m|\mathbf{Z}),$$

where $\Pr(M_m|\mathbf{Z})$ can be estimated by BIC.

Boosting

Motivation

Combine outputs of many "weak" classifiers to produce a powerful "committee".

Weak Classifiers

Classifiers with error rates slightly better than random guessing. For bagging to work, they shouldn't be highly correlated.

- One of the most powerful learning ideas introduced in the last 20 years.
- Originally designed for classification, also extended to regression.

Boosting: How it Works

How Do We Combine the Weak Learners?

Want to sequentially apply weak classifiers to repeatedly modifed versions of the data.

Produce a sequence of weak classifiers $G_1(\cdot), G_2(\cdot), \ldots, G_M(\cdot)$. Predictions combined through weighted majority vote:

$$f(x) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right),$$

where $\alpha_1, \alpha_2, \ldots, \alpha_M$ are weights computed by the boosting algorithm. Weight effect is to give higher influence to more accurate classifiers in sequence.

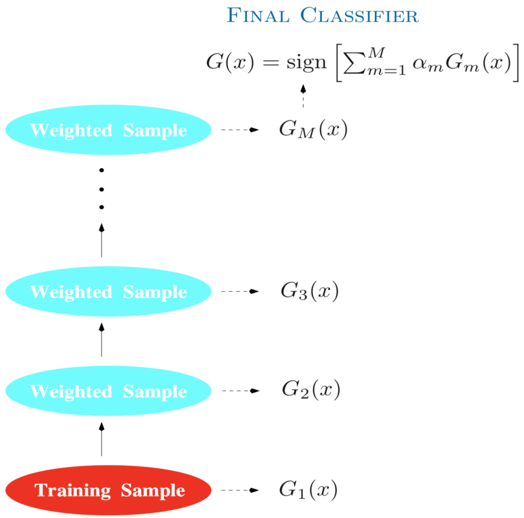

FIGURE 10.1. *Schematic of AdaBoost. Classifiers are trained on weighted versions of the dataset, and then combined to produce a final prediction.*

Boosting

Overview

- Boosting adds weak learners one at a time.
- A weight value is assigned to each training point (y_i, x_i), $i = 1, 2, \ldots, M$.
- At each step, data points which are currently classified correctly are weighted down (i.e. the weight is smaller the more of the weak learners already trained classify the point correctly).
- The next weak learner is trained on the weighted data set: In the training step, the error contributions of misclassified points are multiplied by the weights of the points.
- Roughly speaking, each weak learner tries to get those points right which are currently not classified correctly.

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

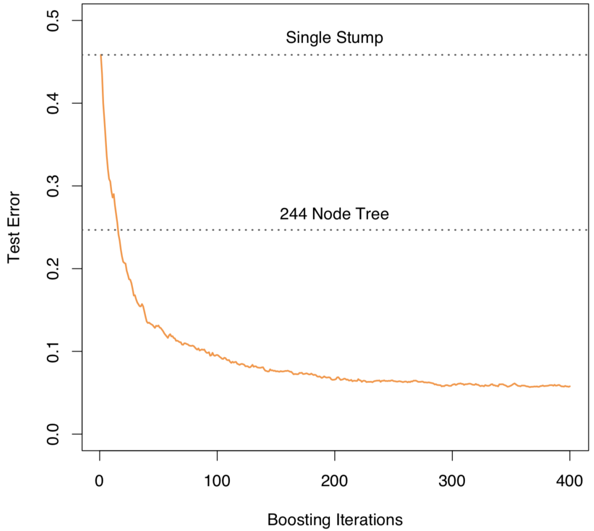

FIGURE 10.2. *Simulated data (10.2): test error rate for boosting with stumps, as a function of the number of iterations. Also shown are the test error rate for a single stump, and a 244-node classification tree.*

Why is Boosting Successful?

Boosting is a way of fitting a 'basis' expansion where the basis functions are individual classifiers $G_1(\cdot), G_2(\cdot), \ldots, G_M(\cdot)$.

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

More generally, basis function expansions take the form

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

where $\beta_1, \beta_2, \ldots, \beta_M$ are expansion coefficients and $b(x, \gamma_m) \in \mathbb{R}$ are simple functions characterized by parameters γ.

Basis expansions like this are at the heart of many learning techniques: neural nets, wavelets, regression splines, etc.

Boosting: Why It Works

What's Actually Happening

Typically these models would be fit by minimizing a loss function

$$\min_{\beta_m \gamma_m} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i, \gamma_m)\right).$$

However, this requires a computationally intensive numerical optimization.

Boosting approximates the above using a Greedy approach (sequentially adding new basis functions without adjusting parameters that have already been estimated) and an exponential loss function .

Functionally, it requires solving the (easier) subproblem of fitting just a single basis function.

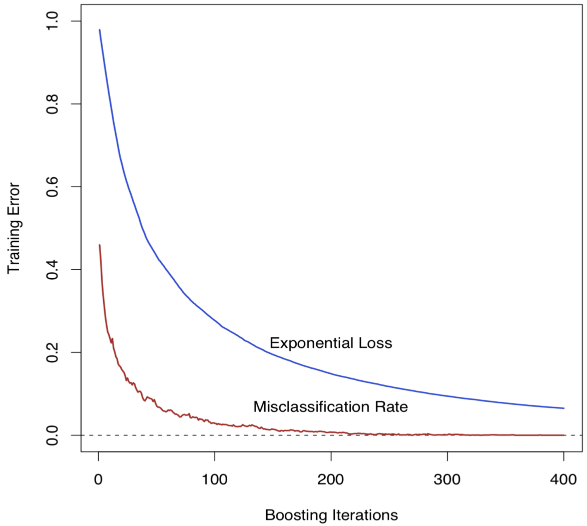

FIGURE 10.3. *Simulated data, boosting with stumps: misclassification error rate on the training set, and average exponential loss:* $(1/N)\sum_{i=1}^{N}\exp(-y_i f(x_i))$*. After about* 250 *iterations, the misclassification error is zero, while the exponential loss continues to decrease.*

Boosting Trees

Decision Trees

Partition the space of all joint predictor variable values into disjoint regions $R_1, R_2, \dots, R_J$ as represented by terminal nodes of the tree.

Classifier

A constant γ_j is assigned to each such region and the predictive rule is

$$x \in R_j \implies f(x) = \gamma_j.$$

Formally,

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j \mathbb{I}\{x \in R_j\},$$

where $\Theta = (R_1, \gamma_1, R_2, \gamma_2, \dots, R_J, \gamma_J)$ is the collection of parameters that are found by minimizing empirical risk.

Boosted Trees

Definition

A sum of tree models

$$f(x) = \sum_{m=1}^{M} T(x, \Theta_m)$$

found in a greedy approach as discussed previously.

Gradient boosting

Algorithm needs to minimize the loss,

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i)),$$

with respect to f where f is constrained to be a sum of trees.

'Gradient boosting' is a greedy algorithm for computing the above minimization.

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 $\hat{f}(x) = f_M(x)$.
```