W4249 Project 3: Overview of model assessment and selection

Tian Zheng

March 9, 2016

Part I: Supervised Learning

Supervised Learning

- Y the outcome variable is given.
- Learning task I: using original or derived X to construct a model to predict Y.
- Learning task II: Identify subject-matter knowledge from the learnt model.
- For categorical Y, it is referred to as a *Classification* problem.

Classification

Most classification methods can be discussed using

- Decision boundaries between classes;
  - Linear classifier versus non-linear classifier.
- Discriminant functions: for each class k, define $\hat{f}_k(x)$.
  - Prediction is then: $\underset{k}{\text{argmax}} \hat{f}_k(x)$
  - Most current methods work in this domain.

Linear discriminant analysis

- LDA assumes a Gaussian mixture with common variance-covariance matrix.
- $P(G|X) = \{\Pr(g_k|X = x) \propto \Pr(x|g_k)\pi_k\}$
- It is then sufficient to look at the log-ratio

$$\log \frac{\Pr(g_k|X = x)}{\Pr(g_l|X = x)} = \log \frac{\pi_k}{\pi_l} - \frac{1}{2}(\mu_k + \mu_l)^T \Sigma^{-1}(\mu_k - \mu_l) + x^T \Sigma^{-1}(\mu_k - \mu_l)$$

Linear discriminant analysis

- The *linear discriminant function* is then

$$\delta_k(x) = x^T \Sigma^{-1} \mu_k - \frac{1}{2} \mu_k^T \Sigma^{-1} \mu_k + \log \pi_k$$

  and $\hat{G}(x) = \underset{k}{\text{argmax}} \delta_k(x)$.

- Computational considerations
  - $\hat{\pi}_k = N_k / N$
  - $\hat{\mu}_k = \bar{X}_k$
  - $\hat{\Sigma}$ is the pooled estimate of the variance-covariance matrix.

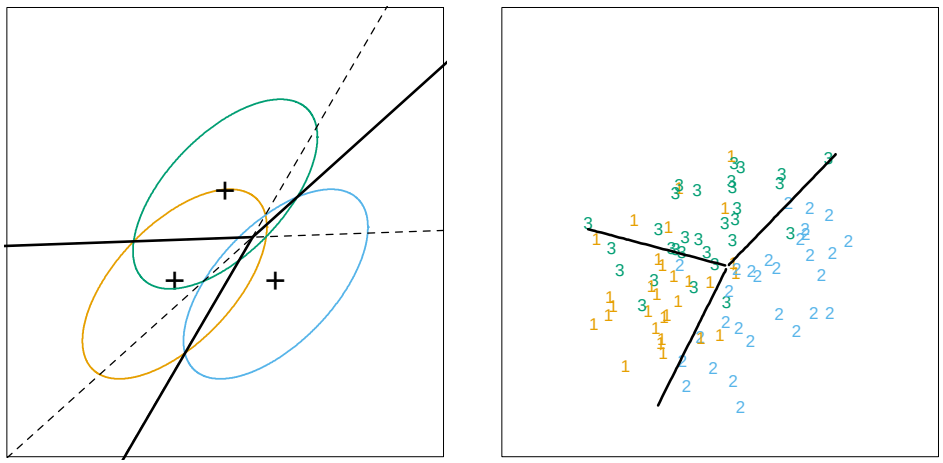


FIGURE 4.5. *The left panel shows three Gaussian distributions, with the same covariance and different means. Included are the contours of constant density enclosing 95% of the probability in each case. The Bayes decision boundaries between each pair of classes are shown (broken straight lines), and the Bayes decision boundaries separating all three classes are the thicker solid lines (a subset of the former). On the right we see a sample of 30 drawn from each Gaussian distribution, and the fitted LDA decision boundaries.*

Quadratic discriminant functions

- If we assume different variance-covariance matrices Σ_k for the classes $k = 1, ..., K$, the discriminant function based on log-likelihood-ratio will be quadratic.
- That is,

$$\delta_k(x) = \log \pi_k - \frac{1}{2} \log |\Sigma_k| - \frac{1}{2}(x - \mu_k)^T \Sigma_k^{-1} (x - \mu_k)$$

- Thus, the discriminant boundary ($\{x : \delta_k(x) = \delta_l(x)\}$) is a quadratic function.

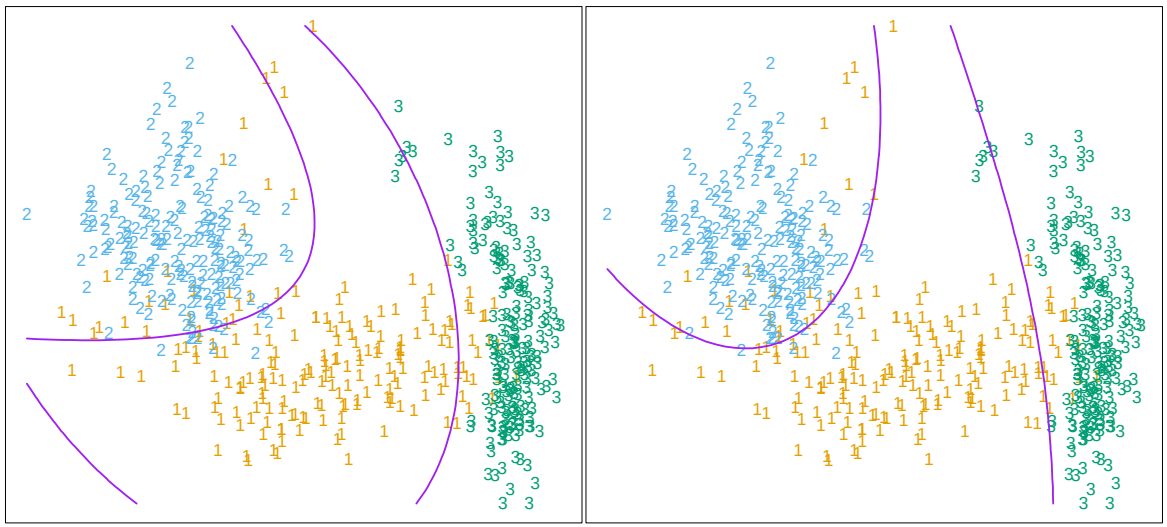


FIGURE 4.6. *Two methods for fitting quadratic boundaries. The left plot shows the quadratic decision boundaries for the data in Figure 4.1 (obtained using LDA in the five-dimensional space $X_1, X_2, X_1X_2, X_1^2, X_2^2$). The right plot shows the quadratic decision boundaries found by QDA. The differences are small, as is usually the case.*

The method of nearest neighbors

- Define $N_k(x)$ as the neighborhood of x, s.t. it contains k nearest points in the training set.
- Here, we implicitly implied a metric of distance is needed for KNN methods.
- The most popular choice is the *Euclidean distance*.
- The prediction is then $\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i$.
- Obviously RSS minimizes at $k = 1$.
- Model fitted using KNN is less rigid than the linear methods such as LDA and generate non-linear prediction functions.

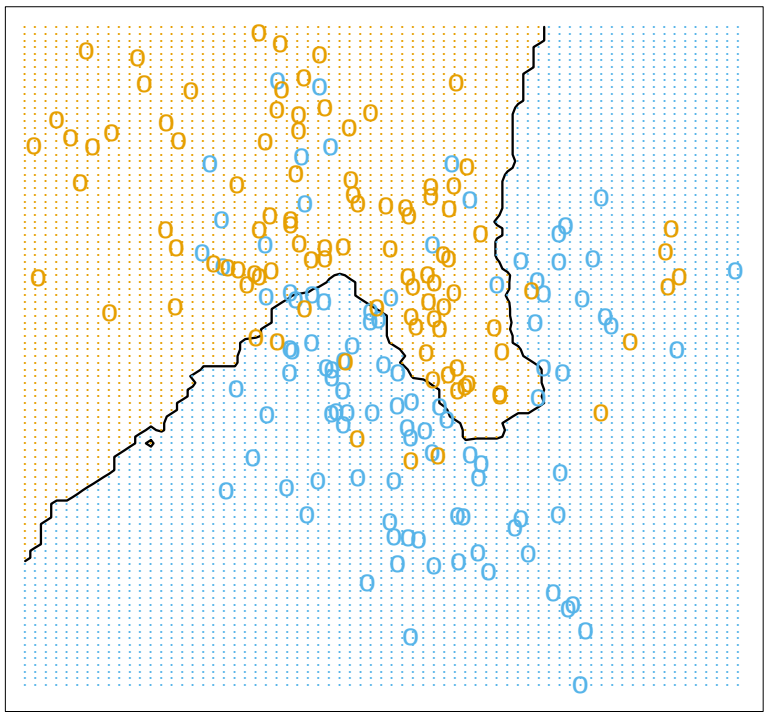


FIGURE 2.2. *The same classification example in two dimensions as in Figure 2.1. The classes are coded as a binary variable* (BLUE $= 0$, ORANGE $= 1$) *and then fit by* 15*-nearest-neighbor averaging as in (2.8). The predicted class is hence chosen by majority vote amongst the* 15*-nearest neighbors.*

Properties

Linear methods: low variance, high bias

- Unbiased under the linear model. Biases if the data follow a nonlinear model.
- Stable, smooth.
- Nice analytical results

Nearest Neighbor methods: high variance, low bias

- No assumption on the model.
- High variance in the predictions since each $\hat{Y}$ is calculated only by a few observations.

"Each method has its own situations for which it works best."

Variants of these two strategies

- Kernel methods can be applied to distance metrics.
- Smoother kernel to replace the kNN "kernel".
- Using local weights, or weights vary across dimensions to make linear methods less rigid.
- Linear models fit to a basis expansion of the original inputs: this expands the class of models considered.

Assumptions made in classification

- If you have infinite data points and unlimited computational power and storage, classification is easy.
- For finite size training sample, you don't have *enough* observations everywhere you need to make prediction.
- Assumptions
  - Probability of class labels are continuous over feature values.
  - The distance metric or kernel function used are meaningful for the classification problem.
  - The test sample were drawn from the same distributions as the training sample.

Part II: Curse of dimensionality

High dimensions

- Let p be the dimension of the input space (usually, number of predictors.)
- Assume that the inputs uniformly distributed in a p-dimensional unit cube.
- Consider the K nearest neighborhood. Let $r = K/N$. The average size of the neighborhood can be approximated by a p-dimensional hyper-sphere with radius d, so that its volume πd^p equals r.
- Therefore, $d = \sqrt[p]{r/\pi}$.

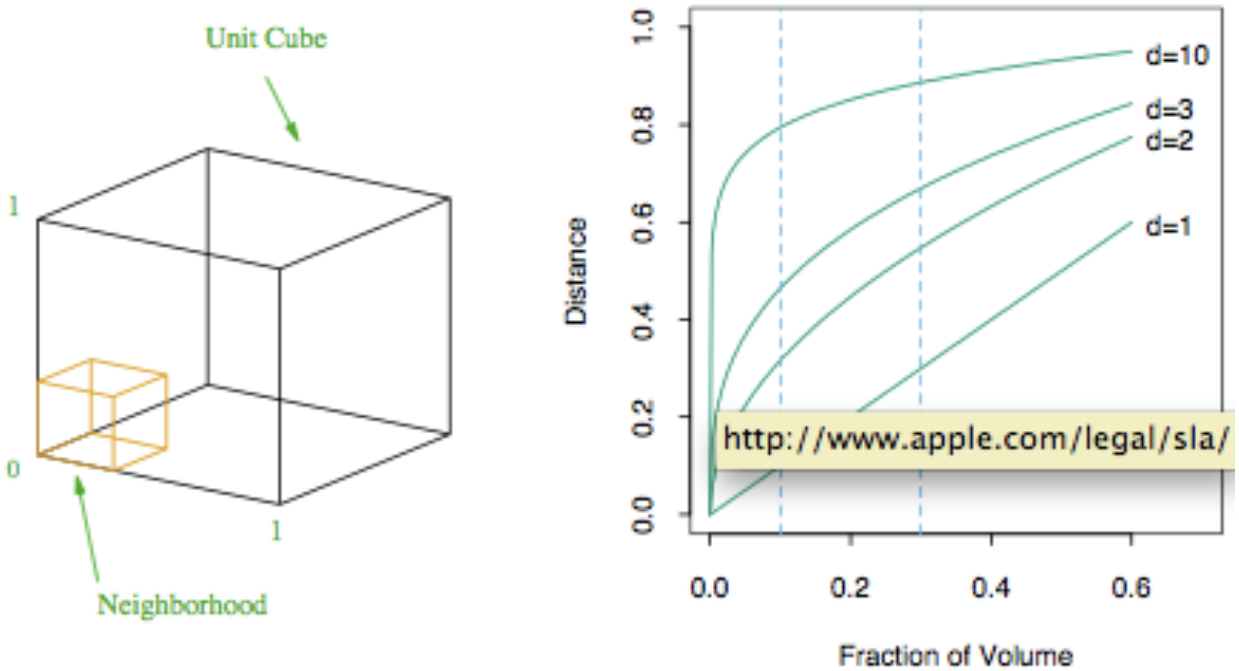


FIGURE 2.6. *The curse of dimensionality is well illustrated by a subcubical neighborhood for uniform data in a unit cube. The figure on the right shows the side-length of the subcube needed to capture a fraction r of the volume of the data, for different dimensions p. In ten dimensions we need to cover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

Curse of dimensionality

- Again, let p be the dimension of the input space (usually, number of predictors.) Assume that the inputs 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 $\text{E} \min(x_i) = \frac{1}{p+1}$.
- This implies that for any point, it is very close to at least one boundary. Inference at the boundary is usually difficult.

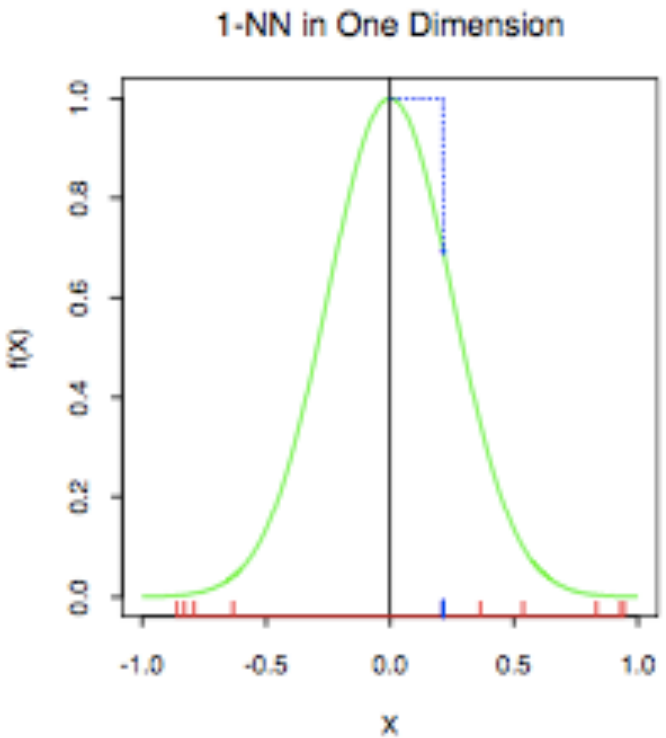
Variance-Bias decomposition

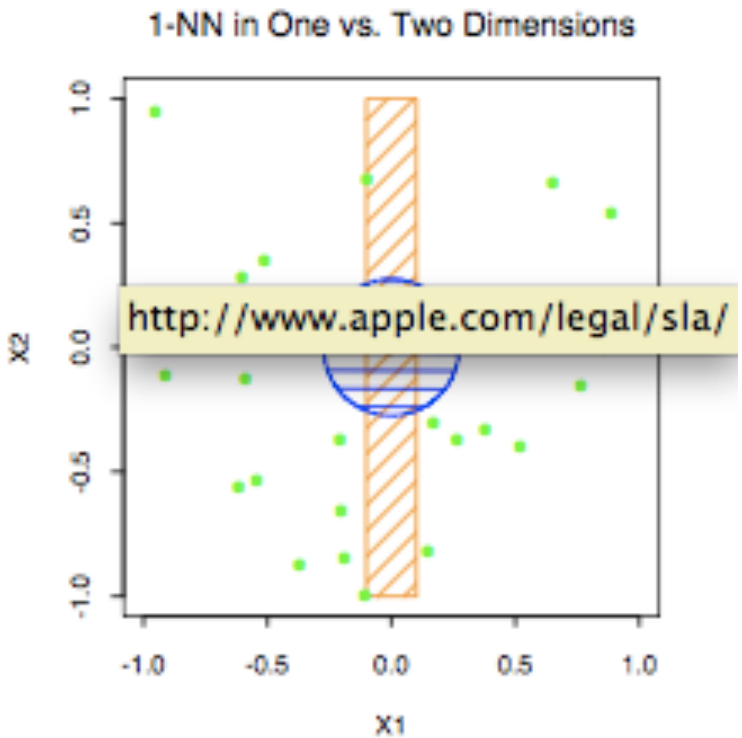
Take the expected prediction error (EPE) for L_2 loss

$$\begin{aligned} \mathrm{E}_{y|x}(y-\hat{y})^2 &= \mathrm{E}_{y|x}(y - E(y|x) + E(y|x) - E(\hat{y}|x) + E(\hat{y}|x) - \hat{y})^2 \\ &= \mathrm{var}(y|x) + (E(y|x) - E(\hat{y}|x))^2 + \mathrm{var}(\hat{y}|x) \\ &= \text{Irreducible error} + \text{Bias}^2 + \mathrm{var}(\hat{y}|x) \end{aligned}$$

How dimensions affect estimation

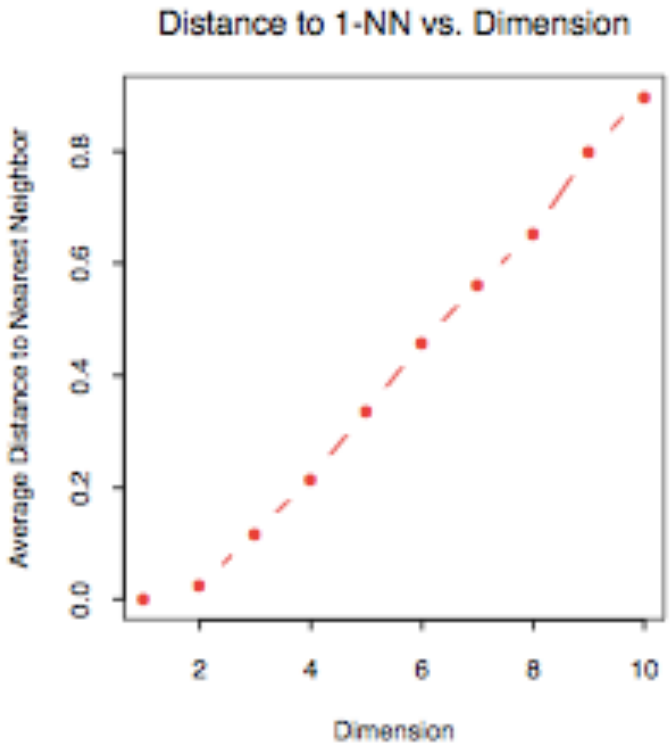
- Consider X in p dimensional space.
- $Y = e^{-8||X||^2}$.
- We are using nearest-neighbor method to estimate Y at $X = 0$.

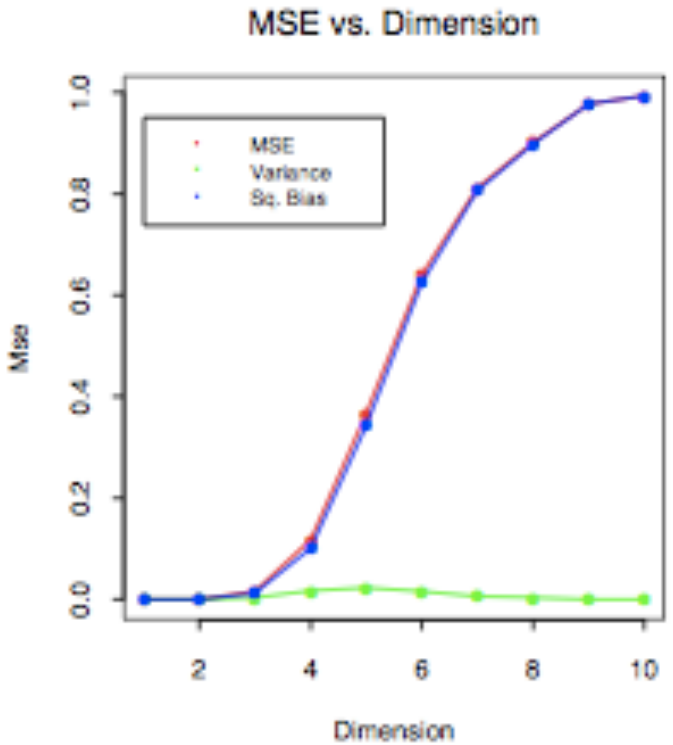




How dimensions affect estimation

- 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.


Distance to 1-NN vs. Dimension
Average Distance to Nearest Neighbor
0.0
0.2
0.4
0.6
0.8
2
4
6
8
10
Dimension


MSE vs. Dimension
Mse
0.0
0.2
0.4
0.6
0.8
1.0
MSE
Variance
Sq. Bias
2
4
6
8
10
Dimension

How dimensions affect estimation

- Now consider another case where $Y = \frac{1}{2}(X_1 + 1)^3$.
- The function depends on only one dimension.
- i.e., the other dimensions are irrelevant for the learning of this function.
- This function does not peak at 0. The bias is then not as prominent.
- The variablity of the estimate is decided on the distance to NN along X_1, which increases as the number of irrelevant dimensions increases.

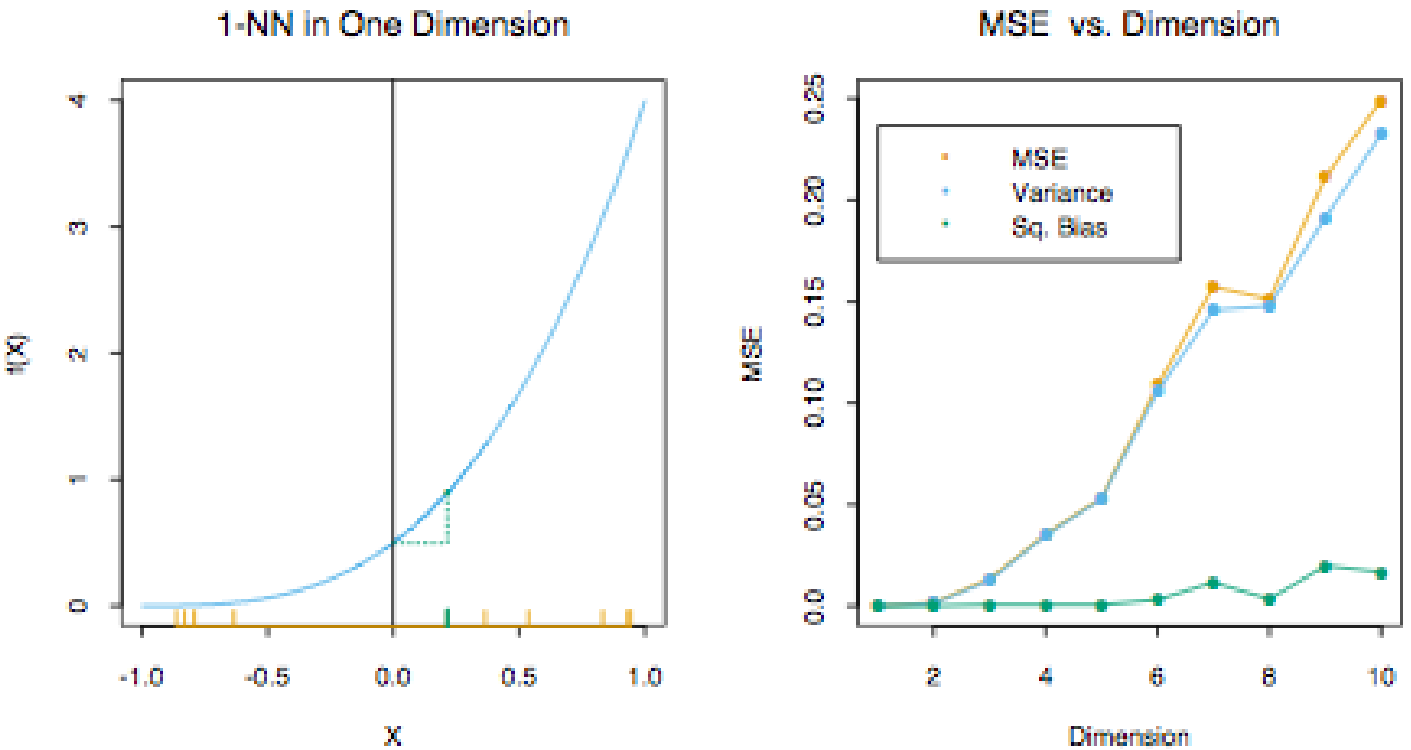


FIGURE 2.8. *A simulation example with the same setup as in Figure 2.7. Here the function is constant in all but one dimension:* $F(X) = \frac{1}{2}(X_1 + 1)^3$. *The variance dominates.*

Part III: Statistical models: an overview

The predictive relation

- The predictive relation between Y and X depends on the definition of "goodness of fit", sometime in the form of a loss function.
- L_2 loss: $f(x) = \mathrm{E}(Y|X = x)$.
- L_1 loss: $f(x) = \mathrm{median}(Y|X = x)$.
- We have shown the nearest-neighbor methods can be viewed as local direct estimates of $f(x)$.
- We have also shown that the nearest-neighbor methods will run into trouble when the dimension of the input space becomes higher.
- It is also intuitive that if the relation between Y and X is known to be more structured, KNN methods are not optimal.

The predictive relation

- For functions f and g that satisfy

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

  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.*
- By constraining to one model family, the hope is the interception between the model class and those equivalent models is unique.
- When the interception is not unique, we will have the *identifiability issue.*

The predictive relation

- The general belief is that the more complicated a model $f(x)$ is, the more likely for $f(x)$ to give prediction further away from $\mathrm{E}(y|x)$ at x that is not close to x_i's observed— the *principle of parsimony* or the *Occam's razor* .
- Therefore, constraints are usually *complexity* constraints.
- Usually, the modeling can be carried out using *structured model families*, basis expansions and *kernel/local regression*.
- The complexity of models are controlled accordingly.

Part IV: Model selection and selection

Model Assessment and Selection

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

Test Error versus Training Error

- **Loss function** $L(Y, \hat{f}(x))$ defines prediction errors (e.g., squared error–quadratic loss; absolute error–L_1 loss;);
- **Test error** is the expected prediction error over *an independent test sample*:

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

- Note: this expectation takes into account the sampling variability in the training data that are used to fit $\hat{f}$ (including effects from the sample size).

Test Error versus Training Error

- The *test error* is also called *generalization error.*
- *Training error* is the average loss of the training set:

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

- Training error is not a good estimate of the test error.

Test error and training error versus model complexity

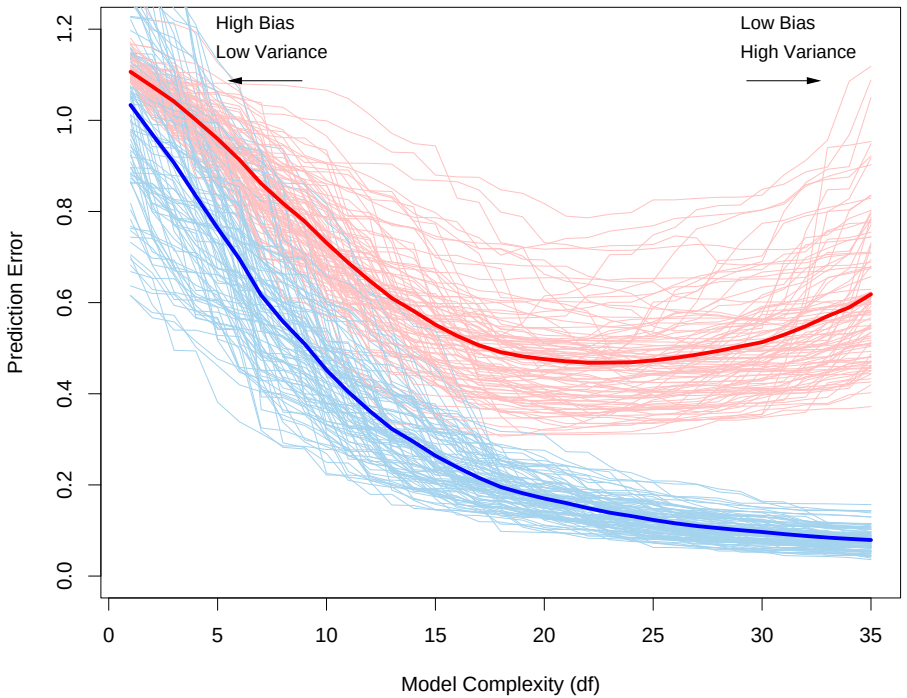
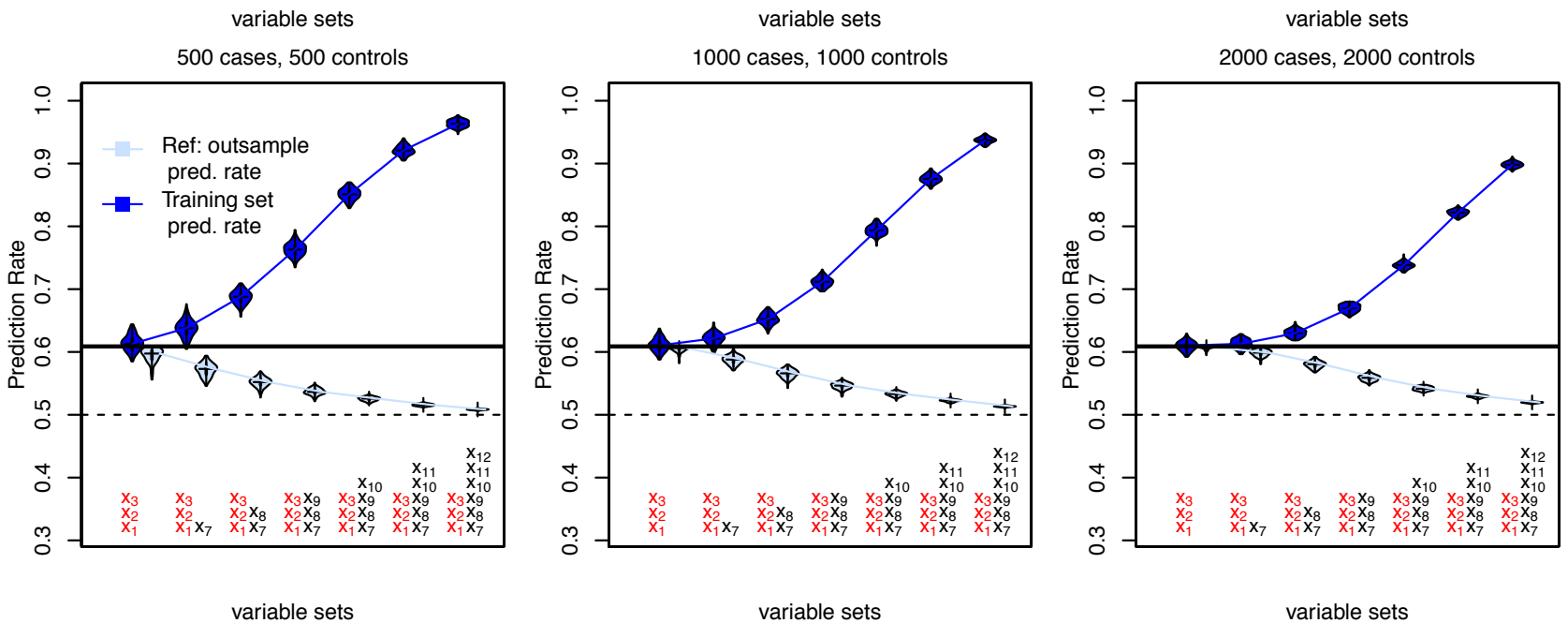


FIGURE 7.1. *Behavior of test sample and training sample error as the model complexity is varied. The light blue curves show the training error* $\overline{\text{err}}$*, while the light red curves show the conditional test error* $\text{Err}_{\mathcal{T}}$ *for* 100 *training sets of size* 50 *each,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



Cross-validation

- Cross-validation methods directly estimate the extra-sample prediction error (the generalization error) by using non-overlapping training data and test data.
- K-fold cross-validation Layout

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

⋮

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

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

Cross-validation (cont'd)

- More specifically, prediction error is estimated by

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

- $K = N$ is just leave-one-out validation: the training sets will be too similar.
- Small K, the training set is too small comparing to the total training set. Loss of efficiency.
- We are trying to estimate the prediction performance at the total training set size using K-fold cross-validation.
- K should be chosen (theoretically) so that the learning curve does not have a steep slope at $\frac{k-1}{K} N$.

Cross-validation (cont'd)

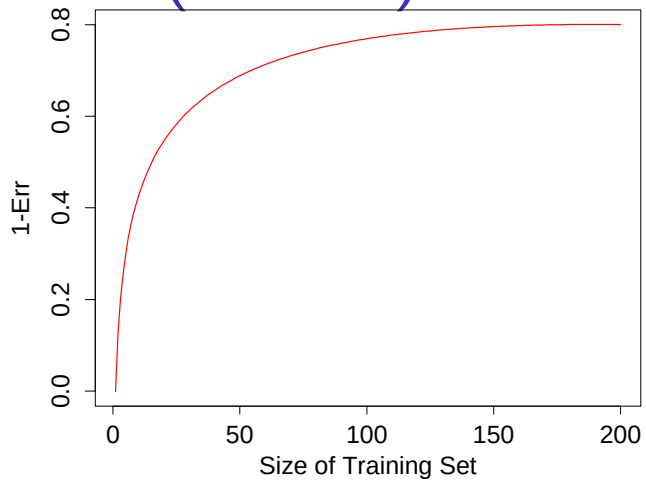


Figure 7.8: *Hypothetical learning curve for a classifier on a given task; a plot of* $1 - \text{Err}$ *versus the size of the training set* N*. With a dataset of* 200 *observations, fivefold cross-validation would use training sets of size* 160*, which would behave much like the full set. However, with a dataset of* 50 *observations fivefold cross-validation would use training sets of size* 40*, and this would result in a considerable overestimate of prediction error.*

Cross-validation (cont'd)

Model selection based on cross-validation

- The model with the best cross-validation prediction error–the "best" model.
- *One-standard-error rule*: since the K-fold cross-validation also allows one to estimate the standard error of prediction errors. We choose **the most parsimonious (simplest) model whose error is no more than one standard error above the error of the "best" model**

Cross-validation (cont'd)

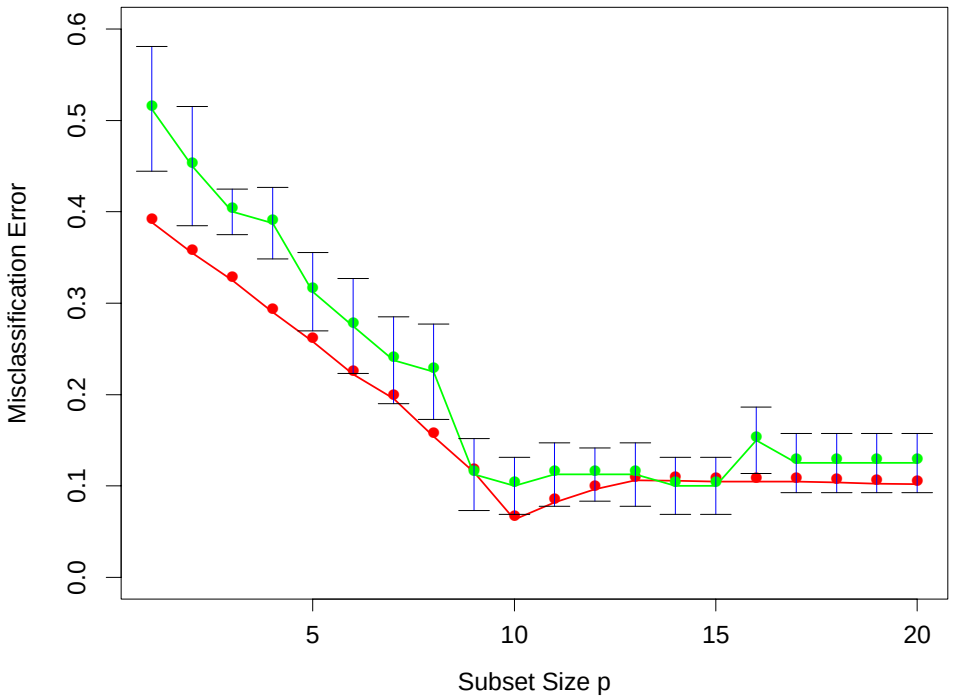


Figure 7.9: *Prediction error (red) and tenfold cross-validation curve (green) estimated from a single training set, from the scenario in the bottom right panel of Figure 7.3.*

Bootstrap methods

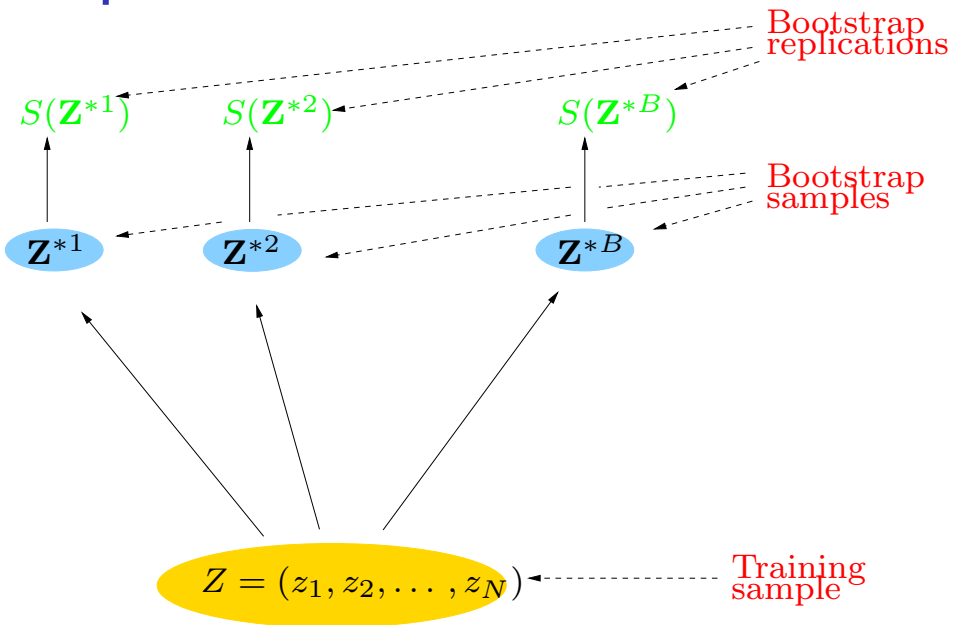


Figure 7.10: *Schematic of the bootstrap process. We wish to assess the statistical accuracy of a quantity $S(\mathbf{Z})$ computed from our dataset. B training sets $\mathbf{Z}^{*b}$, $b = 1, \ldots, B$ each of size N are drawn with replacement from the original dataset. The quantity of interest $S(\mathbf{Z})$ 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 $S(\mathbf{Z})$.*

Basic Bootstrap

- Nonparametric bootstrap: resample (w/o replacements) training data (x_i, y_i).
- Parametric bootstrap: $y_i^* = \hat{f}(x_i) + \varepsilon_i^*$, where ε_i^* is generated from a parametric distribution.

Bootstrap for model assessment

- Prediction error estimated using Bootstrap

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

where

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

- Here, the bootstrap sample are acting as the training samples, while the *original* training sample set is acting as the test sample.
- The problem is that a bootstrap sample and the original sample have a lot of observations in common.

Bootstrap for model validation

- For validation, for each observation (x_i, y_i), we keep track of the bootstrap samples $\mathbf{Z}^b$ that do not contain z_i. We call the set of such b's C^{-i}.
- The probability for a bootstrap sample not containing z_i is $1 - (1 - \frac{1}{N})^N \approx 1 - e^{-1} = 0.632$.
- The leave-one-out bootstrap estimate of prediction error is then

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

Bootstrap for model searching

- Sometime it is hard to optimize the model selection criterion, but you have a similar criterion which is easier to enumerate.
- **Bumping:** uses bootstrap sampling to move randomly through model space.
- The trick is:
  - generate bootstrap samples
  - for each bootstrap sample, choose semi-optimum model according the **easier** criterion
  - evaluate each selected bootstrap model using the hard-to-optimize criterion and select the best one.
- Also useful in situations with multiple local optima.

Part V: Model ensemble

Model averaging: bagging

- Based on bootstrap samples

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

- This is different from model built with bootstrap estimate of the model parameters, when the model is nonlinear.
- For classification problems, the $\hat{f}$ usually takes the form estimated probabilities for classes. And the prediction is usually given as the most probable class.
- The bagged predictor for classification can take two forms:
  - bag the predictions (weighted votes)
  - bag the class probabilities

Model averaging: weighted averages of candidate models

- *Committee methods*: the final model is an unweighted "average" of the fitted individual $\hat{f}_m$'s.

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

- Models M_1, ..., M_M can also be averaged based on $\Pr(M_m|\mathbf{Z})$.

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

- $\Pr(M_m|\mathbf{Z})$ can be well estimated by BIC. Full Bayesian computation can also derive these probabilities but is more computationally intensive, and the computational cost is not justified by performance.

Model averaging: weighted averages (cont'd)

- Models averages from previous slide can be viewed as *weighted average*.

$$\hat{f}(x) = \sum_{m=1}^{M} \omega_m \hat{f}_m(x)$$

- Searching for the best weight, one may consider

$$\hat{\omega} = \underset{\omega}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \sum_{m=1}^{M} \omega_m \hat{f}_m(x_i))$$

- Will this lead to overfitting?

Model averaging: stacking

- Stacking or *stacked generalization*

$$\hat{\omega}^{st} = \underset{\omega}{\operatorname{argmin}} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \omega_m \hat{f}_m^{-i}(x_i)\right)$$

- If we search ω among vectors with elements 0 or 1, and with sum 1, stacking is equivalent to leave-one-out model selection.

Part VI: Boosting.

Boosting

- "Weak" learners: classifiers with error rates slightly better than random guessing.
- Boosting: combines the outputs of many "weak" learners to produce a better prediction.
- The combined prediction is in the form of a *committee vote*, weighted sum of individual votes.
- The key here is that these weak learners should not be highly correlated.
- Boosting procedure sequentially train classifiers (learners) based on current prediction performance and decide the weight of the classifier trained on the fly.
- Each classifier is trained to the training data using weights adaptively updated each step.

The basis of boosting

- A learning algorithm that produces weak learners—*usually easy to compute and interpret.*
- *A mechanism to generate modified versions of data*—so that the classifiers are less correlated.
- A mechanism to assign weights to learner—*so that the performance of the collection of classifiers can be "optimized".*

AdaBoost.M1 algorithm

- Class codes: [-1: class 1; 1: class 2]

Step 0 $w_i = 1/N$, $i = 1, \ldots, N$.

Step m Iterate:

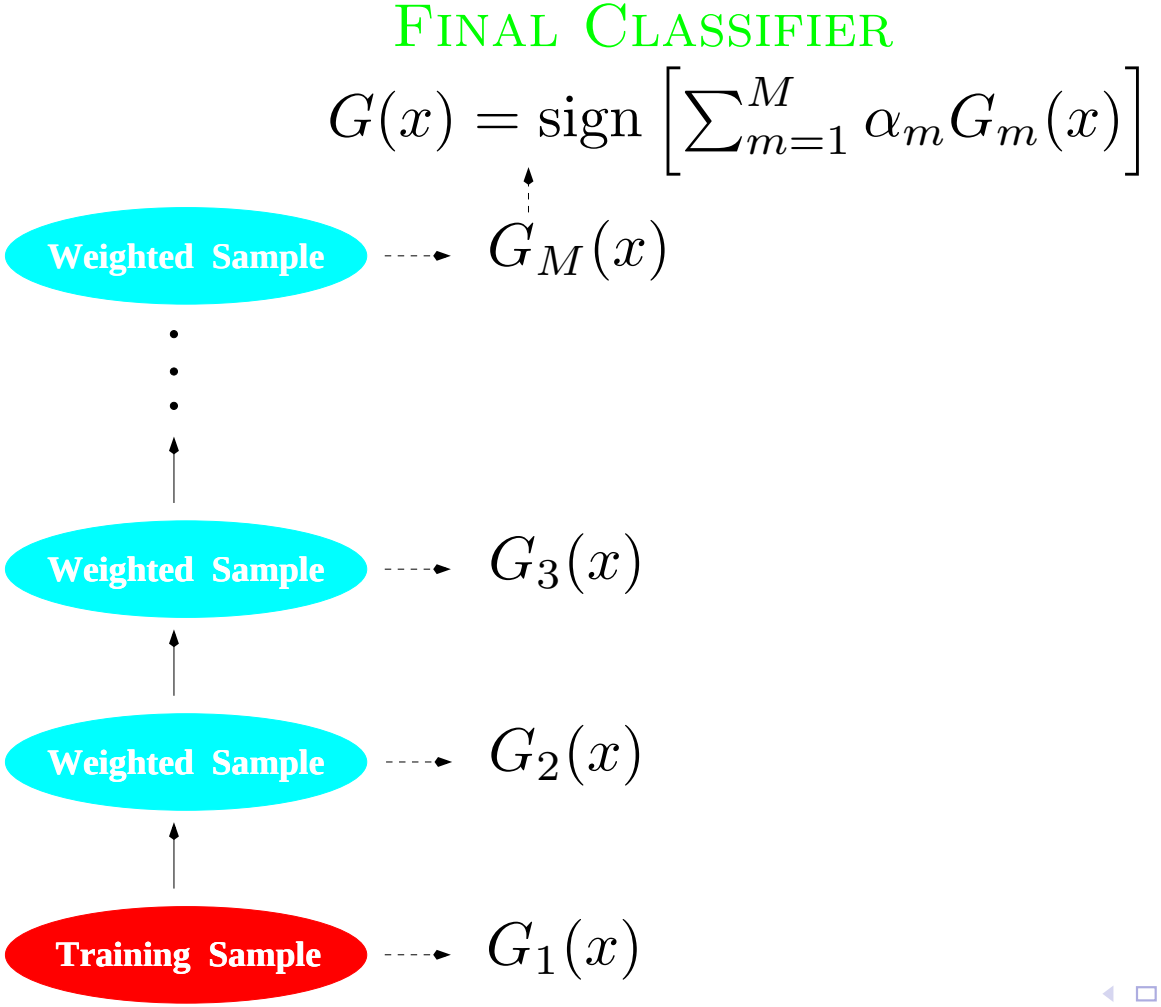
- Fit classifier $G_m(x)$ to the training data with weights w_i.
- Compute

$$\text{err}_m = \frac{\sum_{i=1}^{N} w_i \mathbf{1}(y_i \neq G_m(x_i))}{\sum_{i=1}^{N} w_i}$$

- weight for this classifier: $\alpha_m = \log \frac{1-\text{err}_m}{\text{err}_m}$.
- $w_i = w_i \cdot \exp(\alpha_m \mathbf{1}(y_i \neq G_m(x_i)))$. (For weak learner, err_m might be close to 0.5 but should less than 0.5.)

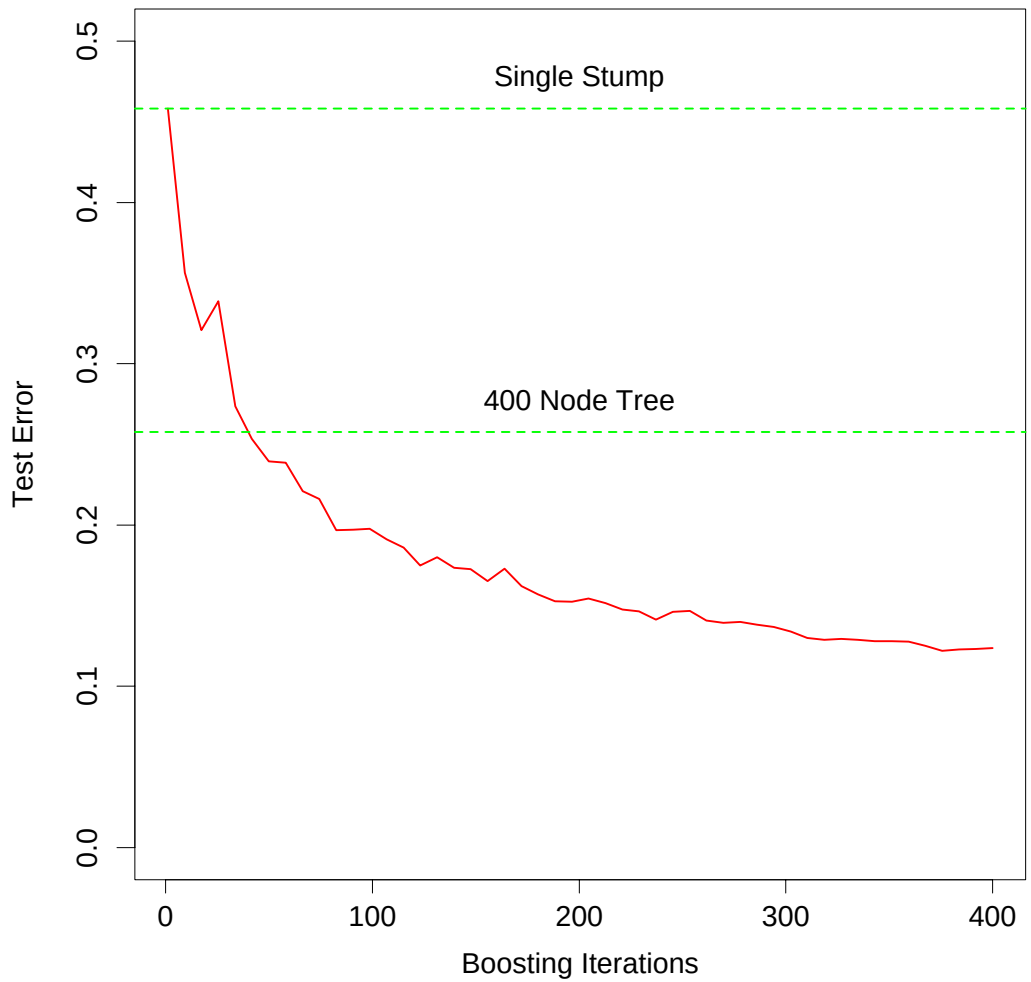
Final classifier: $G(x) = \text{sign}[\sum_{m=1}^{M} \alpha_m G_m(x)]$

AdaBoost.M1 algorithm



Weak learner used: stumps

AdaBoost.M1 algorithm



AdaBoost.M1 algorithm

- The AdaBoost.M1 is know as "discrete AdaBoost" since its classifier gives a categorical prediction.
- It can be modified for regression problems.

Discussion on Boosting

- Boosting fits an additive model. If each $G_m(x)$ is an additive model, $G(x)$ will be an additive model.
- By controlling the complexity of $G_m(x)$, one can prevent boosting from overfitting.
- (section 10.4) it is shown that AdaBoost.M1 is equivalent to forward stagewise additive modeling using the exponential loss function.

$$(\beta_m, G_m(x)) = \underset{\beta, G}{\operatorname{argmin}} \sum_{i=1}^{N} \exp[-y_i(f_{m-1}(x_i) + \beta G(x))]$$

Why exponential loss?

- It leads to simple modular reweighting AdaBoost Algorithm.
- It is also shown that

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

- Binomial log-likelihood loss

$$l(Y, f(x)) = Y' \log p(x) + (1 - Y') \log(1 - p(x))$$

where $Y' = (1 + Y)/2$.

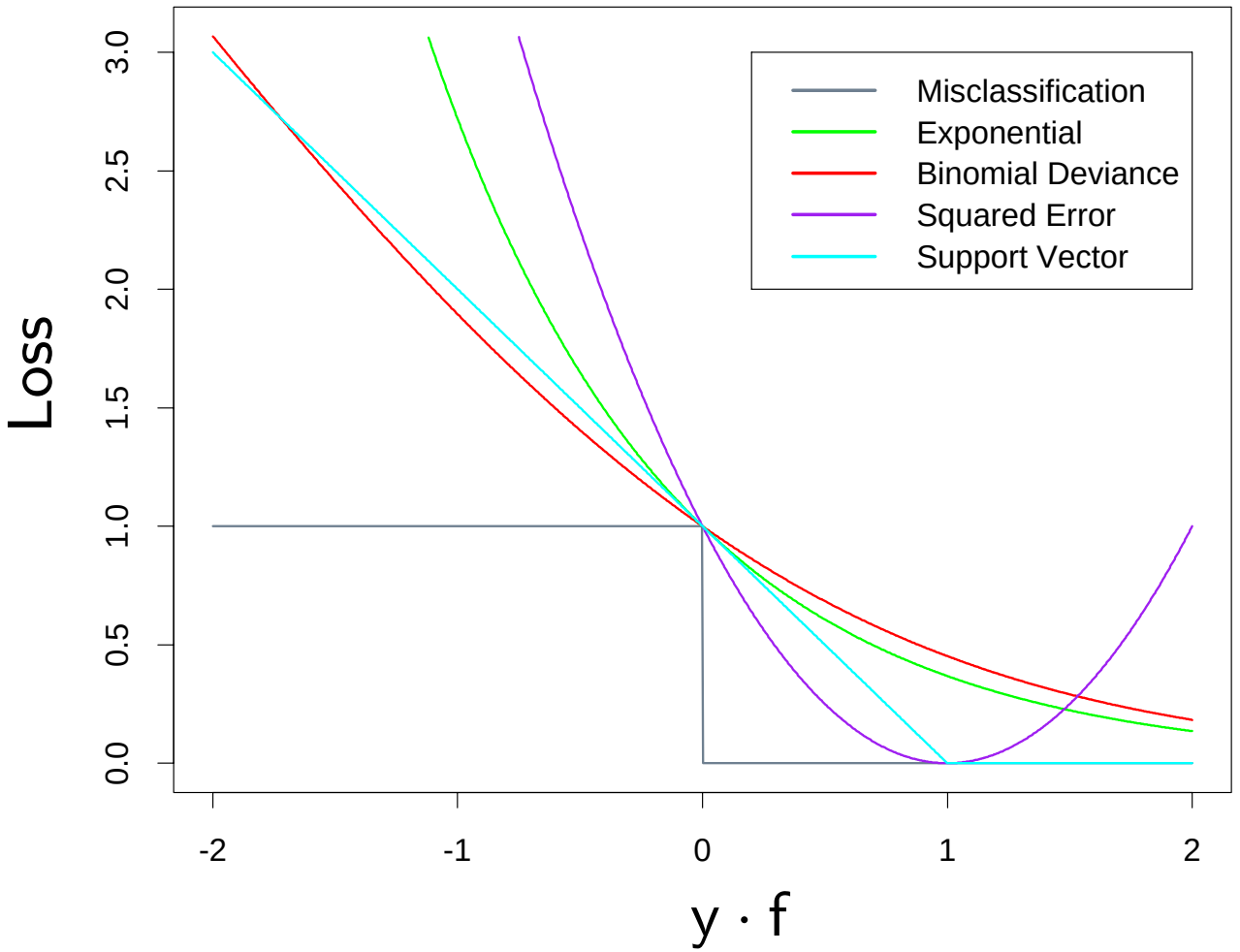
- Here

$$-l(Y, f(x)) = log\left(1 + e^{-2Yf(x)}\right).$$

Discussion on loss functions

- Consider classification problems. Outcome Y takes value 1 or -1.
- Let $f(x)$ be a regression function used for classification. The prediction based on $f(x)$ is then $G(x) = \text{sign}(f(x))$.
- Misclassification when $y_i \neq G(x_i)$.
- "margin": $yf(x)$, where f(x) takes continuous value.
- Large positive value of the "margin" means that the prediction is correct and well within the classification boundary.
- Large negative value means the misclassification is very much cross the boundary (thus less like to go away with a small random changes to the data).

Discussion on loss functions



Discussion on loss functions

- Misclassification rate only penalize misclassifications.
- The exponential loss is a continuous function that approximates the misclassification loss.
- However, the penalty for negative margin increase faster than the "reward" for positive margin.
- The support vector also penalize small positive margins.
- Also from this figure, we see the problem with squared loss in this classification setting.

Discussion on loss functions

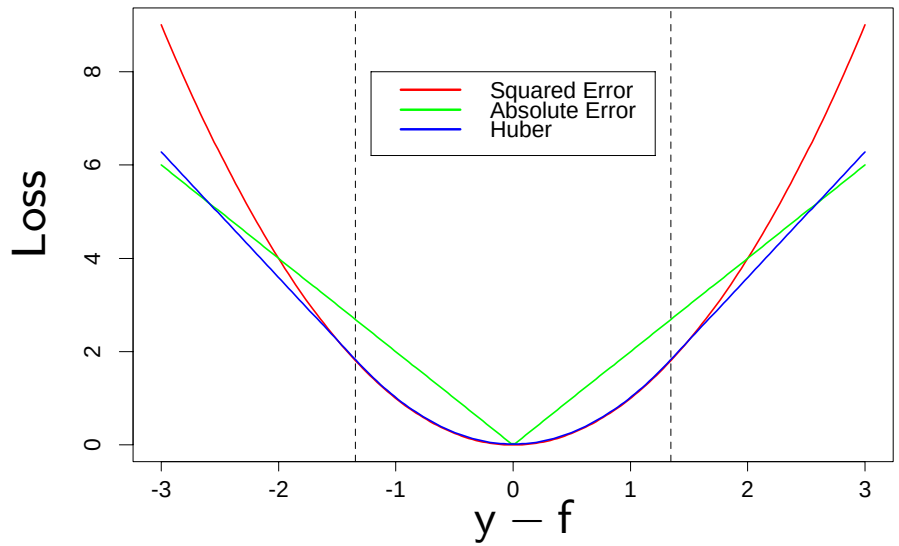


Figure 10.5: *A comparison of three loss functions for regression, plotted as a function of the margin* $y - f$. *The Huber loss function combines the good properties of squared-error loss near zero and absolute error loss when* $|y - f|$ *is large.*

A single tree

- A tree is represented by the regions $R_j$represented by its terminal nodes.

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

- Once R_j are identified, γ_j is easy to estimate.
- Global optimal R_j's are hard to find. CART uses a top-down greedy algorithm.
- Smoother target functions work better than misclassification loss.

The boosted tree

- is a sum of trees

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m)$$

- Here Θ_m is a general notation representing both the tree topology (partition) R_j's and the γ_j's at terminal nodes.
- At each step

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)).$$

- Finding R_j for Θ_m may be more difficult than fitting a single tree.

The boosted tree

- For square-loss, $T(x, \Theta_m)$ can be identified by fitting a single tree to the residuals $y_i - f_{m-1}(x_i)$.
- For exponential loss, the optimal tree is identified by minimizing a weighted misclassificaiton rate.
- For other loss function, we need *gradient boosting*.

Steepest gradient

- Recall

$$\hat{\Theta}_m = \arg\min_{\Theta_m} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)).$$

- $T(x_i; \Theta_m)$ should be updated along the steepest decent of $L(y, f)$ at $f = f_{m-1}$.
- Let $\mathbf{g}_m$ be the gradient with components

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

- Problem: a tree may not be able to move along this gradient exactly.

Gradient boosting

- An approximate step is then introduced.
- A tree is fitted to $-g_{im}$ with squared loss, i.e.,

$$\tilde{\Theta}_m = \arg\min_{\Theta} \sum_{i=1}^{N} (-g_{im} - T(x_i, \Theta))^2.$$

TABLE 10.2. *Gradients for commonly used loss functions.*

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \le \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha$th-quantile$\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

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

Right-sized trees

- To avoid overfitting at each step, it is recommended to use trees of a fixed size J.
- J is then becomes a tuning parameter.

Regularization

- For boosting, regularization means when to stop or how many steps should be used.
- Another strategy is *shrinkage*

$$f_m(x) = f_{m-1}(x) + \nu T(x; \Theta_m)$$

where $0 < \nu < 1$.

- Subsampling can also be used at each step to avoid overfitting $T(x; \Theta_m)$.

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

Relative importance of variables

- Relative importance of each variable X_l is defined as

$$I_l^2 = \frac{1}{M} \_sum_{m=1}^{M} I_l^2(T_m)$$

- where

$$I_l^2(T_m) = \sum_{\text{each internal node } t} \Delta_t^2 I(X_l \text{is used for this split})$$

  and Δ_t is the improvement at internal node t.
- This measure depends on data. We can set the largest to 100 and scale the others accordingly.