The Elements of Statistical Learning - Notes

Matheus Rosso

Contents

1 Supervised learning (Chapter 2, pages 9-38)

Introduction to prediction under the supervised learning framework. Brings up taxonomy, main statistical entities and overall problems involved when estimating a prediction model.

- **Supervised learning task:** from training data $\{(y_i, x_i\}_{i=1}^N$, where $y_i \in \mathbb{R}$ e $x_i \in \mathbb{R}^p$, develop a set of prediction rules so that, given an input vector X, a prediction for Y, $\hat{Y}$, can be calculated. The response variable Y can be either numerical, $Y \in \mathbb{R}$, or categorical (ordered or without order), $Y \in \mathbb{I}$, implying in regression or classification problems. Also X can gather numerical or categorical data, where the later should be transformed into numerical variables. Finally, what differences supervised from unsupervised learning is the existence of a well-defined response variable Y, mainly during the training process.

  Two basic learning methods are:

  - Linear models estimated through least squares (pages 11-14).
  - K-Nearest Neighbors (KNN) (pages 14-18).

- **Statistical decision theory:** gives the theoretical basis for defining the main reference of the learning performance, namely, the *error measure*. Mathematically, these are defined by a **loss function** for a given data point (Y, X), $L(Y, f(X))$.

  - Taking the regression problem, the main loss function is the *squared error loss*:

$$L(Y, f(X)) = (Y - f(X))^2 \tag{1}$$

    Given this loss function, the following entity is central for regression problem:

$$EPE(f) = E(L(Y, f(X))) = E((Y - f(X))^2) \tag{2}$$

    Where EPE accounts for *expected prediction error*. By conditioning (2) on X, minimizing the resulting expression leads to the regression function.

  - Taking the classification problem the main loss function is the *0-1 error loss*:

$$L(Y, f(X)) = I(Y \neq f(X)) \tag{3}$$

$$EPE(f) = \sum_{j=1}^{k} l(Y_k, f(X)).P(Y_k|x) \tag{4}$$

If $Y \in \{1, 2, ..., K\}$, then $L(Y, f(X))$ is a KxK matrix with diagonal elements $l(Y_i, Y_i) = 0$ and with non-diagonal elements $l(Y_i, Y_j) = 1$ giving the unity loss value for classifying a class i observation as belonging to class j. By conditioning (2) on X, minimizing the resulting expression leads to the Bayes classifier.

  - For both problems, the KNN method directly attempts to generate the populational solution to the minimization problems of (2) and (4).

- **Bias-variance decomposition:** given the regression problem and the squared error loss, one can demonstrate that:

$$EPE(x_0) = E_{y_0|x_0}(E_{\mathcal{T}}(y_0 - \hat{y}_0)) = Var_{\mathcal{T}}(\hat{y}_0) + bias^2(\hat{y}_0) + Var(y_0|x_0) \tag{5}$$

  Where $\mathcal{T}$ denotes the training data, $\hat{y}_0 = \hat{f}(x_0)$ and $bias(\hat{y}_0)$ is the bias when estimating $f(x_0)$, $E(f(x_0) - \hat{y}_0)$. Besides, $Var(y_0|x_0) = \sigma^2$ is the irreducible variance when estimating $f(x)$ that follows from the stochastic error ϵ in $y = f(x) + \epsilon$.

  - Check manual notes for demonstration of (5) - file "bias_variance_decomposition.pdf".

- **Flexibility/complexity of models:** the more flexible a mode is, or equivalently, the more complex it is, the more likely is the occurrence of overfitting, given the high level of adherence of the estimated model to the training data. Therefore, it may be advisable to impose restrictions into the estimation procedure, which would constraint the extent to which the training data is used when setting model parameters. Those restrictions may be related with the smoothness of the fitted plane into the data points, with the assumption of prior beliefs over the model parameters, with the size of the neighborhood of data points, or even with the selection of features that will make part of the model.

- **Model selection and the bias-variance trade-off:** generally, the more complex a model is, the smaller the bias of predicting test data, but higher the variance. Therefore, since bias and variance consist on the share of expected prediction error that is under control, the model complexity should be chosen so as to optimally balance between bias reduction and variance increase. Appropriate methods for estimating expected prediction error can allow the definition of the model complexity that minimizes test error for a given method and training data.

2 Linear methods for regression (Chapter 3, pages 43-93)

The least squares estimation of linear models for regression is standard in statistical modeling. Besides developing this basic approach, the chapter discusses methods for features selection, shrinkage and dimension reduction.

- The linear model estimated through least squares can be enhanced with **subset selection**, which defines a collection of features to be part of the final model estimation. This brings two main benefits: prediction accuracy and interpretability.
  - *Best subset selection:* defines the best model for each model size, M_0 (only intercept), M_1, ..., M_p (performance over training data – RSS, R^2). Then, the best model M^* is chosen (performance based on validation/cross-validation, or C_p, AIC, BIC, adjusted-R^2).
  - *Forward-stepwise selection:* starts with the null model M_0 and sequentially increases the number of predictors, M_1, M_2, ..., M_p, by the inclusion of the regressor with the highest decrease in RSS (or, the highest increase in R^2). Then, the best model M^* is chosen from validation/cross-validation procedures, or from C_p, AIC, BIC, adjusted-R^2.
    - Advantage towards best subset selection: computationally more feasible.
    - Disadvantage towards best subset selection: higher variance, since its search is constrained to only a subset of all possible combinations of features.
  - *Backward-stepwise selection:* starts with the full model M_p, sequentially decreasing the number of predictors M_{p-1}, ..., M_1, M_0 by the exclusion of the regressor with the highest increase in RSS (or, the highest decrease in R^2). Then, the best model M^* is chosen from validation/cross-validation procedures, or from C_p, AIC, BIC, adjusted-R^2.
    - Share common characteristics with forward-stepwise regression, however, can only be implemented if $N > p$.
  - A *hybrid approach* also starts with the null model M_0 and sequentially increases the number of predictors, as forward-stepwise regression, but may also remove regressors that does not contribute anymore to the fitting, as backward-stepwise regression.
  - At each step of the *forward-stagewise regression*, it is selected the variable with higher correlation with the current residual, and the coefficient of this simple regression (current residual on

regressor) is added to the current coefficient of the variable. This process can continue until no variable has correlation with the current residual.

  * This approach is particularly promising in high dimensional problems, as the forward-stagewise linear regression shows in the context of ensemble learning.

- **Shrinkage methods** may perform better than subset selection, because of their capacity of reducing even further the variance of least squares estimation, which follows from the more continuous nature of the selection of variables under shrinkage as compared to the discrete selection of subset approaches.

  – *Ridge regression* adds a penalty function equal to the sum of L_2 norm of coefficients vector β, thus being named as *L2 regularization*:

$$\hat{\beta}^{ridge} = \underset{\beta}{\operatorname{argmin}} \Big(\sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \Big) \tag{6}$$

$$\hat{\beta}^{ridge} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 \tag{7}$$

$$\text{s.t.} \sum_{j=1}^{p} \beta_j^2 \leq t$$

  * Variables x_{ij} should be standardized prior to the estimation of $\hat{\beta}^{ridge}$.

  * It is convenient to have a look into the matrix form of the solution to (6):

$$\hat{\beta}^{ridge} = (X^T X + \lambda I)^{-1} X^T y \tag{8}$$

  * *Effective degrees of freedom:* given the singular value decomposition of X, $X = UDV^T$, where U (Nxp) and V (pxp) are orthogonal ($U^T = U^{-1}$, $V^T = V^{-1}$) and D (pxp) is diagonal with non-negative entries d_i, it holds that:

$$df(\lambda) = \sum_{j=1}^{p} \frac{d_j^2}{d_j^2 + \lambda} \tag{9}$$

  Where $d(\lambda)$ is the effective degrees of freedom of ridge regression. The higher the degrees of freedom, the more flexible a model is and the smaller the penalty parameter λ.

– *Lasso regression* assumes a penalty function given by the sum of L_1 norm of coefficients vector β, which leads to the definition of *L1 regularization*:

$$\hat{\beta}^{lasso} = \underset{\beta}{\operatorname{argmin}}\Big(\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}|\beta_j|\Big) \tag{10}$$

$$\hat{\beta}^{lasso} = \underset{\beta}{\operatorname{argmin}} \sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 \tag{11}$$

$$\text{s.t.} \sum_{j=1}^{p}|\beta_j| \leq t$$

  * There is no closed form solution to problems (10) and (11).
  * For λ sufficiently large, or t sufficiently small, some coefficients will be set to zero, then pointing to a (continuous) subset selection nature of L1 regularization.

– *Coefficients profiles:* estimated coefficients against effective degrees of freedom, $df(\lambda)$, or shrinkage factor $s = t/\sum_j |\hat{\beta}_j|$ (lasso).

– Generalizing (6) and (10):

$$\hat{\beta} = \underset{\beta}{\operatorname{argmin}}\Big(\sum_{i=1}^{N}(y_i - \beta_0 - \sum_{j=1}^{p}\beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p}|\beta_j|^q\Big) \tag{12}$$

For $q \geq 0$. $q = 0$ matches the best subset selection problem, while $q = 1$ and $q = 2$ refers to lasso and ridge, respectively.

– Geometrical representation: the solutions for (7) and (11) are found where the iso-RSS elliptical curves first touch the constraint region given by $\sum_j \beta_j^2 \leq t$ or $\sum_j |\beta_j| \leq t$, respectively.

– *Elastic-net penalty* is a compromise between lasso and ridge, since the penalty function is given by:

$$\lambda \sum_{j=1}^{p}\big(\alpha\beta_j^2 + (1-\alpha)|\beta_j|\big) \tag{13}$$

– *Least Angle Regression (LAR):* resembles forward-stepwise regression, as continuously more of a given variable is added into the model. Least Angle Regression (LAR) algorithm:

1. After standardizing all predictors, define the residual $r = y - \overline{y}$ and define $\beta_1 = \beta_2 = ... = \beta_p = 0$.
2. Find the predictor x_j with the highest correlation with r.

3. Move β_j continuously towards the coefficient of the regression of r against x_j until another predictor x_k has as much correlation with the current (updated) residual ($r = y - \hat{\gamma}_0 - \hat{\gamma}_j x_j$) as x_j.
4. Move β_j and β_k towards the coefficients of the regression of the current residual r against x_j and x_k until another predictor x_l has the same amount of correlation with the current residual as x_j and x_k.
5. Continue until all p predictors have entered, in some extent, into the model. After $\min(N-1, p)$ steps, the original least squares solution emerges.

* As for ridge and lasso, also LAR coefficient profiles/paths can be constructed from plotting the current coefficient for a given predictor against the L_1 norm of the vector containing the aggregate changes in coefficients for the current step.
* With an additional step of excluding from the current set of active variables $\mathcal{A}_k$ those whose coefficients of the regression of current residual r against $X_{\mathcal{A}_k}$ ($\beta_{\mathcal{A}_k}$) converges to zero, and then redefining $\mathcal{A}_k$ and re-estimating $\beta_{\mathcal{A}_k}$, the lasso solution can be obtained from the LAR algorithm.
* Thus, LAR algorithm is an effective (and efficient) way for calculating a lasso profile of coefficients, especially useful when $p \gg N$.

– *Effective degrees of freedom:*

$$df(\hat{y}) = \frac{1}{\sigma^2} \sum_{i=1}^{N} Cov(\hat{y}_i, y_i) \tag{14}$$

The greater the fit to the data, the higher the covariance between $\hat{y}_i$ and y_i. For LAR algorithm, in the k-th step the effective degrees of freedom of the fitted vector $\hat{y}$ is equal to k. For the LAR-lasso algorithm, at each step the number of effective degrees of freedom is approximately equal to the number of predictors in the current model.

- **Principal components regression (PCR):** the first M $(z_1, z_2, ..., z_M)$ principal components of a set of inputs $x = (x_1, x_2, ..., x_p)$ are a set of linear combinations of these original inputs:

$$z_m = X v_m = \sum_{j=1}^{p} x_j . v_{mj} \tag{15}$$

Where the weights v_{mj} maximize the variance of z_m subject to the constraint that this vector $z_m = (z_{1m}, z_{2m}, ..., z_{Nm})$ is orthogonal to $(z_1, z_2, ..., z_{m-1})$.

- Previously of obtaining the weights v_{mj}, one should standardize the inputs x_j.
- Cross-validation is an alternative for selecting the most appropriate M for a given dataset.
- Pages 66-67 discuss that the vector v_m is the m-th eigenvector of $X^T X$.
- These new variables z_m are then used to estimate a linear regression model:

$$\hat{y}^{pcr} = \overline{y} + \sum_{m=1}^{M} \hat{\theta}_m z_m \tag{16}$$

  Where each $\hat{\theta}_m$ follows from the regression of y against z_1, z_2, ..., z_M, which is equivalent to $< y, z_m > / < z_m, z_m >$, given that z_m is orthogonal to z_{-m}.

- Relating β_j with θ_j:

$$\hat{\beta}_j^{pcr} = \sum_{m=1}^{M} \hat{\theta}_m v_{mj} \tag{17}$$

  Which follows from standardizing the inputs and from (16), since $y = \overline{y} + \sum_j \beta_j x$.

- **Partial least squares (PLS) regression:** also uses the response variable when defining the weights of the linear combination of inputs. The PLS algorithm guarantees that each component z_m not only is orthogonal to the previous $(z_1, z_2, ..., z_m)$ but has as much correlation with y as possible.

  1. Standardize each x_j and define $\hat{y}^0 = 1.\overline{y}$ and $x_j^0 = x_j$.
  2. For each $m \in \{1, 2, ..., p\}$:
     (a) $z_m = \sum_j \hat{\phi}_{mj} x_j^{m-1}$, where $\hat{\phi}_{mj} =< x_j^{m-1}, y >$.
     (b) $\hat{\theta}_m =< z_m, y > / < z_m, z_m >$.
     (c) $\hat{y}^m = \hat{y}^{m-1} + \hat{\theta}_m z_m$.
     (d) $x_j^m = x_j^{m-1} - (< z_m, x_j^{m-1} > / < z_m, z_m >) z_m$.

  At each m, step $2a$ defines directions $\hat{\phi}_{mj}$ that have high correlation with the response and calculates a component z_m; step $2b$ defines the least squares estimate of the coefficient for the regression of y against the component z_m; step $2c$ updates the fitted values for the response; step $2d$ feeds the definition of directions $\hat{\phi}_{m+1j}$ and the calculation of components z_{m+1} for the next iteration $m+1$, and does this in such a way that each subsequent component is orthogonal to the others (by construction, x_j^m and z_m are orthogonal to each other).

– As for PCR, inputs should be standardized for the PLS regression, as step 1 shows.

- Page 82 points that ridge regression tends to perform better than best subset selection and similarly to PCR and PLS. Even so, ridge regression should be preferred to PCR and PLS as a consequence of its smoother shrinkage pattern[1]. Lasso, in its turn, shares some characteristics of both ridge and best subet selection.

3 Linear methods for classification (Chapter 4, pages 101-135)

Classification methods that produce linear decision boundaries are developed, mainly, linear discriminant analysis, logistic regression and perceptron learning algorithm. Additionally, the chapter discusses taxonomy related with classification problems.

- The classification problem seeks to split the features space $X_1xX_2x...xX_p$ into regions such that each of them is associated with a single class of the categorical response variable $G \in \{1, 2, ..., K\}$. The split of the features space considers discriminant functions $\delta_k(x)$, $k \in \{1, 2, ..., K\}$ that generate *decision boundaries* given by the points x with $\delta_k(x) = \delta_l(x)$, or $P(G = k|x) = P(G = l|x)$. A data point x is then classified to the class k with largest value for its discriminant function $\delta_k(x)$.

- Linear methods for classification are based on linear decision boundaries, which requires that some monotone transformation of $\delta_k(x)$ or $P(G = k|x)$ to be linear. Given the binary case and the traditional logistic function for the posterior probability $P(G = 1|x) = 1/(1 + \exp(-x\beta))$, the *logit transformation* produces a linear log-odds ratio:

$$\log \frac{P(G = 1|x)}{P(G = 0|x)} = x\beta \tag{18}$$

  Here, the decision boundary is given by those data points x such that $x\beta = 0$. Two methods that estimate linear logits are linear discriminant analysis (LDA) and logistic regression.

- **Linear discriminant analysis (LDA):** the posterior probability of a given class $G = k$, $P(G = k|x)$, can be derived from the class conditional density of x, $f_k(x)$, and from the prior distribution of G, $\{\pi_k\}_{k=1}^K$, as the Bayes theorem is applied:

$$P(G = k|x) = \frac{f_k(x)\pi_k}{\sum_{l=1}^K f_l(x)\pi_l} \tag{19}$$

[1] All $M = p$ directions enter into the ridge regression, even though more penalty is put to those with smaller variance. For PCR, only $M \leq p$ directions are included, and the rest is discarded.

Given a p-dimensional vector x, if the class density $f_k(x)$ follows from a multivariate normal distribution with common covariance matrix $\Sigma_k = \Sigma$, then the log-odds ratio will be linear in x. Consequently, the decision boundary between classes k and l will be a hyperplane.

- The implementation of classification through LDA can occur with the estimation of the *linear discriminant functions* $\delta_k(x)$, which consists on those terms in the log-odds ratio that depend only on k.
- In the case when $G \in \{1, 2\}$, LDA relates to the estimation of a linear regression model with $Y = 1$, if $G = 2$, and $Y = 0$, if $G = 1$, as the response variable (linear probability model).
  - Even with this resemblance, LDA is more recommended than linear regression model, specially when $G \in \{1, 2, ..., K\}$ for $K > 2$. This because, in multi-class settings, the linear regression model is prone to mask some class, mostly ignoring it during classification.
- Supposing that Σ_k is not necessarily equal for all k, then the discriminant functions depend quadratically on x, thus emerging the *quadratic discriminant analysis (QDA)*. Since $\delta_k(x)$ is a quadratic function of x, the decision boundary $\{x : \delta_k(x) - \delta_l(x)\}$ will also be quadratic.
- For both LDA and QDA, it is central the hypothesis that the data distribution is normal. As discussed on page 111, even so those methods may perform well given the bias-variance trade-off: as Gaussian models are stable, their reduced variance may surpass the bias from the fact that the data may not be normally distributed.
- *Regularized discriminant analysis:* it is a compromise between LDA and QDA, since assumes the following covariance matrix for the class distribution of x: $\Sigma(\alpha) = \alpha\Sigma_k + (1-\alpha)\Sigma$, with $\alpha \in [0, 1]$. Another generalization of the covariance matrix definition is to add a scalar covariance component into Σ: $\Sigma(\gamma) = \gamma\Sigma + (1-\gamma)\sigma^2 I$, with $\gamma \in [0, 1]$.

- **Logistic regression:** given a response variable $G \in \{1, 2, ..., K\}$, the logistic regression model uses the logit transformation over the linear function $x\beta$ in order to produce linear log-odds ratios. This transformation implies in the following format for the posterior probability of class k, $P(G = k|x)$:

$$P(G = k|x) = \frac{\exp(x\beta_k)}{1 + \sum_{l=1}^{K-1} \exp(x\beta_l)} = p_k(x; \theta) \tag{20}$$

Where $\theta = (\beta_1, \beta_2, ..., \beta_{K-1})$ and K is the category of reference.

- The logit transformation ensures that $P(G = k|x)$ is constrained into the interval $[0, 1]$ and that all $P(G = k|x)$ sum to 1 over k.
- The estimation of θ follows from maximizing the log-likelihood of a sample $\{(G_i, x_i)\}_{i=1}^{N}$:

$$L(\theta) = \sum_{i=1}^{N} l_i(\theta) = \sum_{i=1}^{N} \Big(\sum_{k=1}^{K} I(G = k) \log(p_k(x_i; \theta)) \Big) \tag{21}$$

- The first derivative of (21), $\partial L(\theta)/\partial\theta$, named *score vector* (or, gradient vector, since it is evaluated over sample values), should be put equal to zero, so that $\hat{\theta}^{ML}$ is obtained. As the score vector amounts to a non-linear function of θ, optimization algorithms such as Newton-Raphson should be implemented, which requires the calculation of the second derivative of (21), $\partial^2 L(\theta)/\partial\theta\partial\theta^T$, named *hessian matrix*.
- Logistic regression is not only proper to prediction, but is also widely used for inference. For this later purpose, statistical tests are available to assess the significance of each input for explaining the outcome. The Wald test, for example, allows either the test of individual or joint significance.
- Logistic regression estimation can be adapted in order to perform L_1 regularization. One must add to the sample log-likelihood function that should be maximized the following penalty function:

$$-\sum_{j=1}^{p} |\beta_{kj}| \tag{22}$$

- Both LDA and logistic regression yield log-odds ratio that are linear in x, therefore, both classification methods produce linear decision boundaries. The joint density of X and G is given by:

$$P(X, G = k) = P(X).P(G = k|x) \tag{23}$$

  For both LDA and logistic regression, the linearity of the log-odds ratio implies the same functional form for $P(G = k|x)$. LDA, however, supposes that $P(X)$ follows the normal distribution, while logistic regression imposes no restriction over the marginal distribution of X. Consequently, logistic regression is more robust than LDA, as the former is constructed from fewer model assumptions.

- **Separating hyperplanes:** classifiers based on the construction of such hyperplanes explicitly try to separate data points in the inputs space into several regions, where each correspond to a different

class of the categorical response variable. These methods evolve towards support vector classifiers.

- A *separating hyperplane* is given by $\{x \in \mathbb{R}^p : x\beta = 0\}$, and, considering a binary response variable $Y \in \{1, -1\}$, if $x\beta > 0$, then x is assigned to the class corresponding to label $Y = 1$, and if $x\beta < 0$, then x is assigned to the class corresponding to label $Y = 0$.
- An algorithm that computes a linear combination of the input vector x, $x\beta$, and returns a sign from it is named a **perceptron**.
- The *perceptron learning algorithm* minimizes the distance of misclassified data points from the hyperplane that is being constructed, i.e., the decision boundary.
- A data point x is misclassified if $y = 1$ and $x\beta < 0$, or $y = -1$ and $x\beta > 0$. Then, the goal of the perceptron learning algorithm is to minimize:

$$D(\beta) = -\sum_{i \in \mathcal{M}} y_i x_i \beta \tag{24}$$

  Where $\mathcal{M}$ is the set that indexes misclassified data points, and x_i includes a scalar 1 for the intercept β_0 in β.

- The perceptron learning algorithm uses *stochastic gradient descent* to estimate β:
  - The gradient of (24) is given by:

$$\frac{\partial D(\beta)}{\partial \beta} = y_i x_i \tag{25}$$

  - Thus, after listing all misclassified data points, each of them is visited and the estimate of β is updated following:

$$\beta \leftarrow \beta + \rho . y_i x_i \tag{26}$$

    Where ρ is the learning rate.

- There are some problems with the perceptron learning algorithm.
  - First, if classes are perfectly separable, there are many solutions, even though this can be attenuated by imposing restrictions to the separating hyperplane.
  - Second, convergence may take a large time to occur, even though the inclusion of basis function transformations of the original inputs x helps in reducing convergence time.
  - Third, if the data is not separable, then there is no convergence.

4 Model assessment and selection (Chapter 7, pages 219-257)

Discusses taxonomy for prediction error and presents different methods for estimating those metrics. Given the main purpose of prediction for developing statistical learning models, the estimation of test error allows both model selection and assessment.

- Performance metrics evaluated over test data are crucial for selecting a statistical learning method or model and for assessing prediction accuracy.

- Given a response variable Y, an input vector X, an estimated model $\hat{f}(X)$ and a training dataset $\mathcal{T}$, then the **test error** over a test data point is given by:

$$Err_{\mathcal{T}} = E(L(Y, \hat{f}(X))|\mathcal{T}) \tag{27}$$

  Since the training data $\mathcal{T}$ is fixed for (27), it is possible to define the **expected prediction error** (or, **expected test error** taking the average of (27) over all possible training datasets:

$$Err = E[Err_{\mathcal{T}}] = E(L(Y, \hat{f}(X))) \tag{28}$$

  Where the last equality holds from the law of total expectation.

  - Although $Err_{\mathcal{T}}$ is more relevant for a given estimation context, since it gives the test error as a function of the training data used, Err is more likely to be estimated when only a single training dataset is available.

- Another error measure is the **training error**, which is based exclusively on the training data:

$$\overline{err} = \frac{1}{N}\sum_{i=1}^{N} L(y_i, \hat{f}(x_i)) \tag{29}$$

  As the model complexity/flexibility increases, necessarily the training error should decrease, once the model becomes more able to capture patterns in training data. However, concerning test data, if this implies a reduction in bias, the variance may increase as some of those patterns do not generalize. Thus, further to some optimal point, the test error (and the expected test error) may increase in response to increasing complexity, due to this overfitting.

- Expressions (27)-(29) apply similarly for a categorical response variable G.

- Since in most learning problems both **model selection** and **model assessment** are required, the available dataset should be split into:
  - *Training data*, used for fitting models.
  - *Validation data*, applied for model selection.
  - *Test data*, dedicated to assess the final model performance.
  - One possibility for split ratios points to 50% of data for training, 25% for validation and 25% for test.
  - As available dataset has limited volume of data, there are several methods to approximate the validation step. Some are based on calculations applied over training error (AIC, BIC, etc.), while others efficiently re-use data (cross-validation, bootstrap).
- **Bias-variance decomposition:** given the additive error model $Y = f(X)+\epsilon$ and the squared-error loss, the expected prediction error decomposes into three factors:

$$Err(x_0) = bias^2(\hat{f}(x_0)) + Var(\hat{f}(x_0)) + \sigma_\epsilon^2 \tag{30}$$

Check manual notes for demonstration of (30) - file "bias_variance_decomposition.pdf". While the first and the second terms may vary in response to changes in the estimated model $\hat{f}(x)$ (flexibility, set of features, etc.), the third component arises inevitably as a consequence of the randomness in the data generating process.

  - For the family of linear models, another decomposition applies for the squared bias, which is given by:

$$bias^2(\hat{f}(x_0)) = [E(\hat{f}(x_0) - f(x_0))]^2 = [E(\hat{f}(x_0)) - f(x_0)]^2 \tag{31}$$

Then, it holds that:

$$bias^2(\hat{f}(x_0)) = Avg(ModelBias)^2 + Avg(EstimationBias)^2 \tag{32}$$

Where $Avg(ModelBias)$ is the average bias due to the divergence between the best linear model and the true function, which in fact may not be linear. In its turn, $Avg(EstimationBias)$ follows from the difference between the average estimated model and the best linear one.

- Under the assumption that the inputs X are uncorrelated with the error term ϵ, the least squares estimation presents $Avg(EstimationBias) = 0$. Linear models estimated with regularization, however, have $Avg(EstimationBias) > 0$. Thus, the regularization should trade-off this increased bias with variance reduction.
- In the linear model framework, model bias can only be mitigated by transformations in the original features.
- The above discussion does not generalize fully when squared-error loss is replaced by 0-1 loss.

- In addition to test error[2] (27), expected prediction error[3] (28) and training error[4] (29), there is the **in-sample error**:

$$Err_{in} = \frac{1}{N}\sum_{i=1}^{N} E_{Y^0}(L(Y_i^0, \hat{f}(x_i))|\mathcal{T}) \tag{33}$$

Where Y_i^0 refers to a new observation for the response variable of training data point x_i.

- The **optimism of training error** is given by:

$$op = Err_{in} - \overline{err} \tag{34}$$

While the average optimism is taken over the response variable values:

$$\omega = E_y(op) \tag{35}$$

  * As usually methods estimate Err instead of $Err_{\mathcal{T}}$, other methods also tend to estimate ω rather than op.

- The expression in (35) can be calculated through:

$$\omega = \frac{2}{N}\sum_{i=1}^{N} Cov(y_i, \hat{y}_i) \tag{36}$$

Using (34)-(36):

$$E_y(Err_{in}) = E_y(\overline{err}) + \frac{2}{N}\sum_{i=1}^{N} Cov(y_i, \hat{y}_i) \tag{37}$$

[2]Which takes as fixed the training data $\mathcal{T}$ and as random the test data point, (X_0, Y_0).

[3]Which takes both training data and the test data point as random.

[4]Which is based on the same data used for fitting the model.

– Given a linear model with d inputs or basis functions:

$$\sum_{i=1}^{N} Cov(y_i, \hat{y}_i) = d\sigma_\epsilon^2 \tag{38}$$

This expression leads to the *effective number of parameters* in a model, as in equation (14), which simplifies to d in the context of linear models.

* Besides, the following expression also holds for linear models:

$$df(S) = trace(S) \tag{39}$$

Where S is a NxN matrix such that $\hat{y} = Sy$. As with d in equation (38), $df(S)$ is the effective number of parameters involved in the fitting of $\hat{y}$.

- Equations (37) and (38) imply:

$$E_y(Err_{in}) = E_y(\overline{err}) + \frac{2}{N} d\sigma_\epsilon^2 \tag{40}$$

Therefore, the in-sample error increases linearly with the number of inputs or basis functions d and decreases as the sample size N increases.

– The in-sample error can then be estimated by adding the estimation of optimism to the training error.

– Even though Err is the main focus of concern in prediction problems, Err_{in} is useful for model selection purposes.

- The general form of in-sample error estimates is given by:

$$\widehat{Err_{in}} = \overline{err} + \hat{\omega} \tag{41}$$

– The $\mathbf{C_p}$ **statistic** follows the expression in (40):

$$C_p = \overline{err} + \frac{2}{N} d\hat{\sigma}_\epsilon^2 \tag{42}$$

– The **Akaike information criterion (AIC)**, in its turn, requires the log-likelihood of the response variable evaluated at the maximum likelihood estimate of the parameters θ, $P_{\hat{\theta}}(Y)$:

$$loglig = \sum_{i=1}^{N} \log P_{\hat{\theta}}(y_i) \tag{43}$$

For logistic regression with binomial likelihood:

$$AIC = -\frac{2}{N} loglik + \frac{2}{N} d\hat{\sigma}_\epsilon^2 \tag{44}$$

For the gaussian model, $AIC = C_p$.

- Expressions (42) and (44) apply for linear models, for which d is the number of inputs. For non-linear models, defining α as a tuning parameter representing model complexity, then $d(\alpha)$ captures the number of parameters in the model, while $\overline{err}(\alpha)$ denotes the dependence of training error to model complexity:

$$AIC(\alpha) = \overline{err}(\alpha) + \frac{2}{N} d(\alpha) \hat{\sigma}_\epsilon^2 \tag{45}$$

  * Moreover, for non-linear models, expression (38) will not hold because more than d parameters were estimated if the final model has d parameters.
  * Expression (38) holds exactly only for linear models with squared error loss and approximately for linear models with log-likelihood as the objective function of the estimation procedure.
  * The more general expression of the *effective number of parameters* in a model is given by equation (14).

- The **Bayesian information criterion (BIC)** is another estimation of in-sample error. With *loglik* given by (43):

$$BIC = -2 loglik + (\log N) d \tag{46}$$

Supposing a gaussin model, then (46) becomes:

$$BIC = \frac{N}{\sigma_\epsilon^2} \Big[\overline{err} + (\log N) \frac{d}{N} \sigma_\epsilon^2 \Big] \tag{47}$$

  - BIC penalizes model complexity more heavily than AIC and C_p.
  - BIC is asymptotically consistent, which means that the probability of choosing the correct model approaches one as $N \to \infty$.
  - As $N \to \infty$, AIC tends to choose models too complex. For finite samples, BIC favors models that are too simple.
  - As shown in page 234, a model $\mathcal{M}_m$ with minimum BIC also has the maximum posterior probability $P(\mathcal{M}_m | \mathcal{D})$.

- **Cross-validation:** consists on a set of techniques that estimate the expected prediction error, the average error of predicting the response variable for test data.

  - **K-fold CV:** simulates the use of validation set to assess the performance of a model estimated based on training data. This is done by splitting the available data into K folds, followed by K estimation procedures where, at each step k, all folds of data but k are used to estimate a model that will predict the response variable for data points in fold k.

    * This allows the estimation of K prediction errors, or K performance metrics of any kind. The average of such quantities consists on the K-fold CV estimate of (expected) prediction error, or the K-fold CV estimate of the performance metric.
    * When $K = N$, cross-validation following the structure above is named as **leave-one-out cross validation (LOOCV)**.
    * The split of data into K folds can either be based on random assignment of each data point to a fold, or on time sorting of data previously to the split.
    * The larger K, the lower the bias of estimating the (expected) prediction error. However, the larger may be the variance of the estimation, since training sets are very similar to each other. Additionally, the larger K, the higher the computational cost involved in estimating K different models.
    * As discussed below, K-fold CV and LOOCV provide a good estimate of expected prediction error Err instead of prediction error $Err_{\mathcal{T}}$.
    * The theoretical selection of the value of K considers a plot of expected test error Err, or more precisely, $1 - Err$, against the size of training set N_{train} − such curve is named **learning curve**. After some critical point, increasing N_{train} only produces small reductions in Err. In a K-fold CV setting, the larger K, the larger will be the size of training sets. Thus, theoretically, there is an optimal K value that generally will be smaller than N.
    * Putting the question differently, if the data available has a length N whose value of $1-Err$ in the learning curve is located at a point with high slope, then a small value of K − such standard values of $K = 5$ or $K = 10$ − would overestimate Err, since the effective N_{train} would be smaller in such extent that $1 - Err$ is too small.
    * Even so, $K = 5$ or $K = 10$ is still a good compromise in terms of bias-variance trade-off.

* For the purpose of model selection, the procedure of plotting the K-fold CV estimation of (expected) prediction error against model complexity α, in order to select the level of α that minimizes K-fold CV error, may be enhanced through the **one-standard error rule**: one should choose the smaller model complexity α whose K-fold CV error is under the interval of one standard error point from the minimum.

– *Generalized cross-validation:* approximates LOOCV for linear models under the squared-error loss:

$$GCV(\hat{f}) = \frac{1}{N}\sum_{i=1}^{N}\Big[\frac{y_i - \hat{f}(x_i)}{1 - trace(S)/N}\Big]^2 \tag{48}$$

Where S is a matrix NxN for which $\hat{y} = Sy$.

– Always that any kind of model selection must be done from the data in a supervised manner, such as model complexity definition, features selection, shrinkage parameter choice, and so on, all guided by prediction performance, then this model selection should occur inside of K-fold CV estimation.

* For instance, instead of selecting the best subset of features from all data points available, and then to conduct K-fold CV estimation to choose some additional tuning parameter or to estimate (expected) prediction error of a model, the right procedure is to estimate K models from K folds of data, each of which may have a different subset of best features, and then estimate (expected) prediction error from K-fold CV.

* Unsupervised procedures, such as standardization of data, are not affected by considering the whole set of data at once. Another example of procedure that may correctly be applied outside K-fold CV estimation is the selection of features based on some variance criterion.

– If the K-fold CV estimation of (expected) prediction error or any performance metric is the average of the estimate overall folds of data, also the standard error of the K estimates is important for analysis, as the one-standard error rule has pointed out.

- **Bootstrap methods:** also estimate expected prediction error Err instead of test error $Err_{\mathcal{T}}$. The bootstrap performs re-sampling with replacement from the training data, where each of the B samples produced has the same length as the entire dataset available. These B bootstrap datasets can be used either to estimate statistics such as standard error, or to assess prediction accuracy.

– This later alternative is based on fitting the model for each of the B bootstrap datasets. Then,

all B estimates of prediction error are averaged, producing a bootstrap estimate of (expected) prediction error.

- Given $S(Z)$ a statistic computed from data Z, the variance of $S(Z)$, $Var(S(Z))$, has the following bootstrap estimate:

$$\widehat{Var}(S(Z)) = \frac{1}{B-1}\sum_{b=1}^{B}(S(Z^{*b} - \overline{S}^{*})^2 \tag{49}$$

  Where Z^{*b} is the bootstrap sample b from Z and $\overline{S}^{*} = (1/B)\sum_b S(Z^{*b})$.

- Similarly, the bootstrap estimate of (expected) prediction error is given by:

$$\widehat{Err}_{boot} = \frac{1}{B}\frac{1}{N}\sum_{b=1}^{B}\sum_{i=1}^{N} L(y_i, \hat{f}^{*b}(x_i)) \tag{50}$$

  Where $\hat{f}^{*b}(.)$ is the model fitted using the bootstrap sample b.

  * However, expression (50) does not produce a good estimate of (expected) prediction error, since several data points (y_i, x_i) used to calculate the loss function are also present in the training data used to fit $\hat{f}^{*b}(.)$.
  * As shown in page 251, each data point has an approximate probability of 0.632 of belonging to a bootstrap sample b.

- A way for correcting (50) is to consider only those predictions of y_i that has not used (y_i, x_i) in training data. This leads to the **leave-one-out bootstrap** estimate of prediction error:

$$\widehat{Err}^{(1)}\frac{1}{N}\sum_{i=1}^{N}\frac{1}{|C^{-i}|}\sum_{b\in C^{-i}} L(y_i, \hat{f}^{*b}(x_i)) \tag{51}$$

  Where C^{-i} is the set of bootstrap samples that do not contain observation i.

  * Since 0.632 is a relatively high probability for a bootstrap sample to contain a data point, it is important to choose a high value of B so that C^{-i} would not be empty.

- Despite of the adjust in (51), each bootstrap sample still may have a large share of identical observations, which also may lead to bias when estimating prediction error. Then, another alternative is the **.632 estimator**:

$$\widehat{Err}^{(.632)} = 0.368.\overline{err} + 0.632.\widehat{Err}^{(1)} \tag{52}$$

- It was already mentioned that cross-validation provides good estimates of expected prediction error Err, while seems difficult to estimate conditional prediction error (test error) $Err_{\mathcal{T}}$ having at hand only one training set. Pages 254-257 present some simulation reinforcing this argument.

5 Model inference and averaging (Chapter 8, pages 261-292)

Present several methods for model averaging, besides the discussion on model inference based on maximum likelihood, bootstrap and Bayesian models.

- Concerning model inference, pages 261-267 explore maximum likelihood and bootstrap as alternatives for estimating statistics such as standard errors.
  - It is worth notice the difference between *non-parametric bootstrap*, which does not model the data when proceeding the re-sample, then using $\{(x_i, y_i)^b\}_{i=1}^N$, and *parametric bootstrap*, which simulate new values for the response variable Y by adding a Gaussian noise ϵ to predicted values of Y, $y_i^* = \hat{\mu}(x_i) + \epsilon_i$, then using $\{(x_i, y_i^*)\}_{i=1}^N$.
  - Despite of the great effectiveness and efficiency of maximum likelihood estimates, sometimes they are not able to compute statistics such as standard errors, and bootstrap emerges as an option to assess those quantities.
- Still in the field of model inference, pages 267-270 discuss how Bayesian inference can be structured from a sampling model $P(Y|X, \theta)$ and a prior distribution $P(\theta)$, from which a posterior distribution $P(\theta|\mathcal{D})$ can be inferred.
- Pages 272-279 present the EM algorithm, specially useful when it comes to estimate a Bayesian model such as the Bayesian Logistic Regression (BLR) model.
- Pages 279-282 present another method suited to Bayesian inference, the Markov Chain Monte Carlo (MCMC) approach.
- **Bagging:** consists on an application of bootstrap in order to improve performance of learning models. Given B bootstrap samples from an available dataset, each leading to a prediction $\hat{f}^{*b}(x)$, then the bagging estimate for the response variable of a data point x is given by:

$$\hat{f}_{bag}(x) = \frac{1}{B} \sum_{b=1}^{B} \hat{f}^{*b}(x) \tag{53}$$

  - Bagging has potential to affect data modeling only when the estimation from the entire dataset $\hat{f}(x)$ is non-linear or an adaptive function of the data.
  - Since (53) is an average of independent quantities, it is expected a reduction in the variance of prediction.

- If (53) refers to classification rather than regression, then $\hat{f}_{bag}(x)$ is a vector with K non-negative values in the range $[0, 1]$, where $p_k(x)$ indicates the share of bootstrap estimators $\hat{f}^{*b}$ that point to k as the predicted class for data point x. Then, the class prediction is given by $\hat{G}_{bag}(x) = \text{argmax}_k \hat{f}_{bag}(x)$.
  - It is crucial to not take those values $p_k(x)$ inside $\hat{f}_{bag}(x)$ as estimates of class-probabilities. If each $\hat{f}^{*b}(x)$ is a tree, for example, then the class-probability associated with estimation based on bootstrap sample b is the class proportion in the terminal node of $\hat{f}^{*b}(x)$. The bagging estimate of class k probability is then the average of those class proportions.
- Bagging is relevant specially to reduce the variance of unstable procedures. Under squared-error loss, averaging reduces variance and does not affect bias, thus resulting in a reduction in expected prediction error.
- Given 0-1 loss, however, bagging only improves the performance of good classifiers. The bagging effect of reducing variance applies for the aggregation of independent estimates, and bagged trees, for instance, are prone to be high correlated. Thus, modifications in the aggregation should be considered to guarantee a good performance in classification. An example of improvement is *random forests*, which randomizes the set of features that can define the initial split in tree construction.
- A drawback from bagging is the loose of interpretability, since the average of trees, for instance, can not be graphically represented.

- **Model averaging and stacking:** Bayesian inference can be applied to improve accuracy as posterior probabilities of models, and not just parameters, $P(m|\mathcal{D})$, are used to weight the predictions of each model:

$$E(y|x, \mathcal{D}) = \int E(y|x, \mathcal{D}, m) P(m|\mathcal{D}) dm \tag{54}$$

  - *Stacked generalization*, or *stacking* is a method to implement model averaging by optimally choosing the weights to be putted to each prediction from a set of models indexed by $m \in \{1, 2, ..., M\}$. If $\hat{f}_m^{-i}(x_i)$ is a prediction for data point x_i obtained through the estimation of model m fitted with all data available except for observation i, then the vector of stacking weights is given by:

$$\hat{w}^{st} = \underset{w}{\text{argmin}} \sum_{i=1}^{N} \left(y_i - \sum_{m=1}^{M} w_m \hat{f}_m^{-i}(x_i) \right)^2 \tag{55}$$

The final prediction is then:

$$\hat{f}_{st}(x) = \sum_{m=1}^{M} \hat{w}_m^{st} \hat{f}_m(x) \tag{56}$$

  - Accuracy can be enhanced through the constraint that $\hat{w}_m^{st} \geq 0$ and $\sum_m \hat{w}_m^{st} = 1$.
  - Equation (55) considers squared-error loss, however, stacking can be developed to any kind of supervised learning problem.

- **Stochastic search: bumping:** consists on a technique to select the best from a set of available models. As with bagging, bumping also generates B bootstrap samples and fits one model to each of them, $\hat{f}^{*b}(x)$. Thus, the best model $\hat{b}$ is selected:

$$\hat{b} = \underset{b}{\operatorname{argmin}} \sum_{i=1}^{N} (y_i - \hat{f}^{*b}(x_i))^2 \tag{57}$$

  - Again, (57) considers squared-error loss, however, bumping can be used on the basis of different loss functions.
  - Usually, the entire training set is included in the set of bootstrap samples, so that the estimation from the entire dataset $\hat{f}(x)$ is also available to be selected.
  - Bumping introduces randomness into the search across the model space by modifying the training data used in each estimation. Therefore, the estimated relationship between response variable Y and inputs X is assessed differently, reducing the likelihood of obtaining a model not representative of the data population (poor solutions).
  - The model complexity should not vary widely through the estimations $\hat{f}^{*b}$, since no validation or cross-validation is done with bumping.

6 Boosting and additive trees (Chapter 10, pages 337-380)

Discusses algorithms for boosting models, such as AdaBoost.M1, its resembled procedure of forward stagewise additive modeling, and finally the gradient tree boosting algorithm. The chapter also presents approaches to define parameters of gradient boosting and to interpret model results.

- Boosting works similarly to bagging, and in the context of classification problems, it is focused on combining predictions from many *weak classifiers* $G_m(x)$, $m \in \{1, 2, ..., M\}$, where each of these classifiers is only slightly better than random guessing and is constructed from modifications applied

to the available data. Then, considering a response variable $Y \in \{-1, 1\}$, prediction from boosting is given by:

$$G(x) = sign\Big(\sum_{m=1}^{M} \alpha_m G_m(x)\Big) \tag{58}$$

- Boosting implements those modifications to the available data by applying weights w_1, w_2, ..., w_N to data points $\{(x_i, y_i)\}_{i=1}^{N}$ when constructing the weak classifiers.
- **AdaBoost.M1:** algorithm that implements boosting classification:
  1. Define initial weights $w_i = 1/N$ for each data point $i \in \{1, 2, ..., N\}$).
  2. For each step $m \in \{1, 2, ..., M\}$:
     (a) Fit a classifier $G_m(x)$ using data points weighted by w_i.
     (b) Calculate the following weighted classification error:

$$err_m = \frac{\sum_{i=1}^{N} w_i I(y_i \neq G_m(x_i)}{\sum_{i=1}^{N} w_i} \tag{59}$$

     (c) Calculate the following quantity:

$$\alpha_m = \log((1 - err_m)/err_m) \tag{60}$$

     Note that the higher err_m, the lower will be α_m.

     (d) Redefine the weights:

$$w_i \leftarrow w_i . \exp(\alpha_m I(y_i \neq G_m(x_i))) \tag{61}$$

     Note that more weight is put to misclassified observations.

  3. Define the output from the algorithm by:

$$g(x) = sign\Big(\sum_{m=1}^{M} \alpha_m G_m(x)\Big) \tag{62}$$

  * The AdaBoost.M1 algorithm should be adapted if instead of a class prediction, the objective is to compute class-probabilities.

- Under a specific loss function, AdaBoost.M1 algorithm is equivalent to another algorithm named forward stagewise additive modeling. This fitting procedure consists on a way of approximating the solution to the estimation of *basis function expansions*:

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m) \tag{63}$$

The estimation of (63) is computationally expensive and is summarized by the following expression:

$$\min_{\{\beta_m,\gamma_m\}_{m=1}^{M}} \sum_{i=1}^{N} L\Big(y_i, \sum_{m=1}^{M} \beta_m b(x_i;\gamma_m)\Big) \tag{64}$$

- **Forward stagewise additive modeling**: as mentioned, this algorithm approximates the solution to (64):
  1. Define $f_0(x) = 0$.
  2. For each step $m \in \{1, 2, ..., M\}$:
     (a) Compute:
     $$(\beta_m, \gamma_m) = \underset{\beta,\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + \beta b(x_i;\gamma)) \tag{65}$$
     (b) Update $f_m(x) = f_{m-1}(x) + \beta_m b(x;\gamma)$.
  - ∗ It can be shown that, for squared-error loss function, at each step the basis function $\beta_m b(x;\gamma_m)$ to be chosen should produce the best fit to the current residual $r_{mi} = y_i - f_{m-1}(x_i)$.
- The loss function that makes AdaBoost.M1 equivalent to forward stagewise additive modeling is the **exponential loss function**:
$$L(y, f(x)) = \exp(-yf(x)) \tag{66}$$
- Pages 343-344 demonstrate how AdaBoost.M1 minimizes the exponential loss function (66) by implementing the forward stagewise additive modeling with $L(.)$ in (65) replaced by (66).

- For all statistical learning methods, the results from estimation depend on the loss function $L(Y, f(X))$ chosen to assess model accuracy. Different loss functions produce different estimates, and some of the functions may not be robust to anomalous data, in the sense that the inclusion/exclusion of extreme data points modifies excessively the fitting.
  - Main loss functions for classification are:
    - ∗ Misclassification loss:
    $$L(y, f(x)) = I(y \neq G(x)) \tag{67}$$
    - ∗ Exponential loss:
    $$L(y, f(x)) = \exp(-yf(x)) \tag{68}$$

* Binomial deviance:

$$L(y, f(x)) = \log(1 + \exp(-2yf(x))) \tag{69}$$

* Squared-error:

$$L(y, f(x)) = (y - f(x))^2 \tag{70}$$

- The **margin** in classification problems with response variable $Y \in \{-1, 1\}$ is defined by $y.f(x)$. A positive margin indicates correct classification, whereas a negative margin means a wrong classification for data point x.
- Independent of the loss function chosen, the main goal in classification problems is to produce positive margins the most.
- A plot of loss against the margin (page 347) shows that the misclassification loss (67) varies sharply as the margin changes, since a penalty of 1 is put to misclassified observations ($yf(x) < 0$), while there is no penalty for data points correctly classified ($yf(x) > 0$).
- Exponential loss (68) and binomial deviance (69), in their turn, are monotone continuous approximations to misclassification loss (67). Both penalize negative margins increasingly more than positive margins. Additionally, exponential loss increases exponentially with the decreasing of the margin, while the changes of binomial deviance are almost linear.
  * Consequently, binomial deviance is more robust than exponential loss, which is particularly relevant for contexts where there may be misspecification of class labels.
  * This higher robustness of binomial deviance can be seen in term of a more equally shared weight put into training data, given that exponential loss will give more weight to data points with large negative margin.
- Squared-error loss in not monotone to margin, since after some point, it increases the loss as positive margin increases. In this setting, the relative weight of misclassified observations decreases, which may not be desired for classification purposes.
- Considering the regression problem, main loss functions are:
  * Squared-error loss:

$$L(y, f(x)) = (y - f(x))^2 \tag{71}$$

  * Absolute loss:

$$L(y, f(x)) = |y - f(x)| \tag{72}$$

∗ Huber loss:

$$L(y, f(x)) = \begin{cases} (y - f(x))^2 & \text{if } |y - f(x)| \leq \delta \\ 2\delta|y - f(x)| - \delta^2 & \text{otherwise} \end{cases} \tag{73}$$

– Since squared-error loss (71) puts excessive weight on observations with large absolute residuals $|y-f(x)|$, it is less robust than absolute loss (72), while Huber loss (73) represents a compromise between them.

– Even though exponential loss (68) and (71) are not as robust as desired, they allow simple and feasible boosting algorithms.

- **Boosting trees:** the predictive rule for regression or classification trees is the following:

$$\text{If } x \in R_j, \text{ then } f(x) = \gamma_j \tag{74}$$

Where R_j is a region of the features space, such that all sets R_1, R_2, ..., R_J are disjoint. Therefore, a given tree is represented by:

$$T(x; \Theta) = \sum_{j=1}^{J} \gamma_j I(x \in R_j) \tag{75}$$

Where $\Theta = \{R_j, \gamma_j\}_{j=1}^{J}$ are estimated through the solution to:

$$\hat{\Theta} = \underset{\Theta}{\operatorname{argmin}} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j) \tag{76}$$

Thus, estimating (75) accounts for defining the appropriate γ_j for each R_j, and for finding the best split of the features space between R_j.

– A boosted tree is given by:

$$f_M(x) = \sum_{m=1}^{M} T(x; \Theta_m) \tag{77}$$

The forward stagewise linear additive modeling algorithm would apply with (65) replaced by:

$$\hat{\Theta}_m = \underset{\Theta_m}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, f_{m-1}(x_i) + T(x_i; \Theta_m)) \tag{78}$$

Where $\Theta_m = \{R_{mj}, \gamma_{mj}\}_{j=1}^{J}$. Given regions R_{mj}, the estimation of γmj is straightforward:

$$\hat{\gamma}_{mj} = \underset{\gamma_{mj}}{\operatorname{argmin}} \sum_{x_i \in R_{mj}} L(y_i, f_{m-1}(x_i) + \gamma_{mj}) \tag{79}$$

- Given the difficulty of defining regions R_{mj} for robust loss functions, a method for fitting boosted trees with general loss functions is very useful.

- **Gradient boosting:** numerical optimization is applied for estimating boosted trees based on any loss function:

$$L(f) = \sum_{i=1}^{N} L(y_i, f(x_i)) \tag{80}$$

Numerical optimization tries to minimize (80) finding the vector of approximating functions $\hat{\mathbf{f}}$ such that:

$$\hat{\mathbf{f}} = \underset{\mathbf{f}}{\text{argmin}} L(\mathbf{f}) \tag{81}$$

Where $\mathbf{f} = (f(x_1), f(x_2), ..., f(x_N))$. Numerical optimization methods find the solution to (81) by summing component vectors:

$$\mathbf{f}_M = \sum_{m=0}^{M} \mathbf{h}_m \text{ for } \mathbf{h}_m \in \mathbb{R}^N \tag{82}$$

For an initial guess $\mathbf{f}_0 = \mathbf{h}_0$. It is worth notice that each $\mathbf{f}_m$ follows:

$$\mathbf{f}_m = \mathbf{h}_0 + \mathbf{h}_1 + ... + \mathbf{h}_{m-1} + \mathbf{h}_m = \mathbf{f}_{m-1} + \mathbf{h}_m \tag{83}$$

Different numerical optimization methods point to different ways of computing the updates $\mathbf{h}_m$.

  - **Steepest descent:** defines $\mathbf{h}_m = -\rho_m \mathbf{g}_m$, where $\rho_m \mathbb{R}$ and $\mathbf{g}_m$ is the gradient vector of $L(\mathbf{f})$ evaluated at $\mathbf{f} = \mathbf{f}_{m-1}$. Each component of $\mathbf{g}_m$ is given by:

$$g_{mi} = \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\Big|_{f(x_i)=f_{m-1}(x_i)} \tag{84}$$

    The scalar ρ_m is defined by:

$$\rho_m = \underset{\rho}{\text{argmin}} L(\mathbf{f}_{m-1} - \rho \mathbf{g}_m) \tag{85}$$

    Then, the current solution (83) with steepest descent method is:

$$\mathbf{f}_m = \mathbf{f}_{m-1} - \rho_m \mathbf{g}_m \tag{86}$$

  - Steepest descent and forward stagewise linear additive modeling are analogous, with (85) being similar to (79). Besides, each $T(x_i; \Theta_m)$ is analogous to each g_{mi} given by (84).

– Since steepest descent is too focused on training data, an alternative to the direct use of gradient (84) is to choose $\tilde{\Theta}_m$ such that:

$$\tilde{\Theta}_m = \underset{\Theta_m}{\operatorname{argmin}} \sum_{i=1}^{N} (-g_{mi} - T(x_i; \Theta))^2 \tag{87}$$

Equation (87) defines a tree that tries to fit as best as possible the gradient values (84). Given each of these fitted trees $T(x; \tilde{\Theta}_m)$, (79) applies naturally.

– **Gradient tree boosting algorithm:** considering the regression setting and a given loss function $L(y, f(x))$:

1. Define the initial guess:

$$f_0(x) = \underset{\gamma}{\operatorname{argmin}} \sum_{i=1}^{N} L(y_i, \gamma) \tag{88}$$

2. For each step $m \in \{1, 2, ..., M\}$:

   (a) For each observation, compute:

$$r_{mi} = -\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\Big|_{f=f_{m-1}} \tag{89}$$

   (b) Estimate a regression tree, where the response variable is given by r_{mi}, providing regions R_{mj}, with $j \in \{1, 2, ..., J_m\}$.

   (c) For each $j \in \{1, 2, ..., J_m\}$, calculate:

$$\gamma_{mj} = \underset{\gamma}{\operatorname{argmin}} \sum_{x_i \in R_{mj}} L(y_i, f_{m-1}(x_i) + \gamma) \tag{90}$$

   (d) Update the gradient boosting estimate:

$$f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{mj} I(x \in R_{mj}) \tag{91}$$

3. Define the final estimator by $\hat{f}(x) = f_M(x)$.

– The gradient tree boosting algorithm applied for classification should be adapted properly. Mainly, steps 2a-2d are repeated for each class $k \in \{1, 2, ..., K\}$, producing final trees $f_{kM}(x)$.

– The notation r_{mi} in (89) follows from the definition of *generalized*, or *pseudo residuals*, since the gradient from squared-error loss, $(y_i - f(x_i))^2$, is equal to the current residual $y_i - f(x_i)$, where $f(x) = f_{m-1}(x)$.

- Note that step 2b is equivalent to expression (87).
- Two main tuning parameters in gradient boosting are: i) the number of iterations, M; and ii) the sizes J_m of each tree $f_m(x)$, i.e., the number of terminal nodes of each tree.

- The size of each tree J_m should not be computed separately at each iteration, since this would lead to trees with excessive nodes, specially in early iterations, then affecting performance negatively and increasing computational time.
  - The most efficient way to solve this issue is to define $J_m = J \ \forall \ m$. Consequently, J turns to be a tuning parameter of the entire boosting procedure as it is the number of iterations M.
  - Since the size tree is given by the number of terminal nodes, and given that the greater the number of terminal nodes, the greater the number of splits in the tree, and, therefore, the greater the potential number of features involved in tree building, it follows that the parameter J concerns to the interaction between different inputs in their effect on the response variable.
    * The *target function* η defined by:

$$\eta = \underset{f}{\operatorname{argmin}} E_{(Y,X)}(L(Y, f(X))) \tag{92}$$

      This function is that with minimum expected prediction error, then is the one to be estimated, since it captures the deterministic relationship between inputs and the response variable.

    * The analysis of variance (ANOVA) expansion of η follows:

$$\eta(X) = \sum_j \eta_j(X_j) + \sum_{jk} \eta_{jk}(X_j, X_k) + \sum_{jkl} \eta_{jkl}(X_j, X_k, X_l) + ... \tag{93}$$

      The set of functions $\eta_j(X_j)$ is composed from those univariate functions that best approximate $\eta(X)$. Therefore, each $\eta_j(X_j)$ is the main effect of X_j on Y. The functions $\eta_{jk}(X_j, X_k)$ are those with two variables such that, when added to the main effects, best fit $\eta(X)$. Therefore, each $\eta_{jk}(X_j, X_k)$ is the second order interaction involving X_j and X_k. $\eta_{jkl}(X_j, X_k, X_l)$ captures the third order interactions, and so forth. In general, low order interactions tend to dominate when constructing (93).
  - As mentioned, in the context of tree models, the interaction level is defined by the tree size J, because with $J = 2$, only main effects are captured, while with $J = 3$, at most second order interactions are allowed.

- If $J = 2$ seems inappropriate, $J > 10$ is hardly necessary, thus leading to a grid search over the set of integers $4 \leq J \leq 8$.

- **Regularization:** besides of tree size J, the number of iterations M should also be defined through validation or cross-validation procedures. Another parameter of boosting that relates with M and that is also a regularization strategy is the *shrinkage parameter* $0 < v < 1$, which controls the contribution of each tree added to the boosted model:

$$f_m(x) = f_{m-1}(x) + v \sum_{j=1}^{J} \gamma_{mj} I(x \in R_{mj}) \tag{94}$$

  This parameter is also known as the *learning rate* of the boosting procedure.

  - For a given number of iterations M, the smaller v, the larger will be the training error. Thus, smaller values of v require larger values of M for a constant training error.
  - The best approach for calibrating v and M is to define a low v and choose M through *early stopping*.

- **Stochastic gradient boosting** is another alternative to regularization, and it is based on sampling (without replacement) a fraction η from the entire available data at each iteration m, where this sub-sample should be used to fit the m-th tree in the boosting model.

  - Since there are four parameter in this setting of gradient boosting, J, M, v and η, a convenient approach follows the definition of suitable values for J, v and η, leaving M as the free parameter.

- **Relative importance of predictors:** given a tree T with internal nodes indexed by $\{1, 2, ..., J-1\}$, $\hat{\iota}_t$ is defined as the maximum reduction in loss function obtained through the split of features space in a given node t. If $v(t)$ is the index of the feature that produces the split in node t, then the relative contribution of input X_l for the tree T is given by:

$$\mathcal{I}_l^2(T) = \sum_{t=1}^{J-1} \iota_t^2 . I(v(t) = l) \tag{95}$$

  For boosted trees, the average of (95) over all trees is taken:

$$\mathcal{I}_l^2 = \frac{1}{M} \sum_{m=1}^{M} \mathcal{I}_l^2(T_m) \tag{96}$$

Thus, (96) is the relative importance of input X_l for explaining the response variable Y. For K-class classification, since there are K different functions f_{mk} to be estimated at each iteration, for each of the K different classes to be separated from each other, then (96) applies for each class k:

$$\mathcal{I}_{kl}^2 = \frac{1}{M}\sum_{m=1}^{M}\mathcal{I}_l(T_{mk}) \tag{97}$$

Consequently, for the entire boosted tree:

$$\mathcal{I}_l^2 = \frac{1}{K}\sum_{k=1}^{K}\mathcal{I}_{kl}^2 \tag{98}$$

Thus, (98) is the relative importance of input X_l for separating class k from the remaining classes.

- **Partial dependence plots:** try to solve the dimensional problem of plotting $f(X)$ against some of the inputs in a multivariate setting. Given the disjoint split of the input vector X into a pair of vectors $(X_{\mathcal{S}}, X_{\mathcal{C}})$, the partial dependence of $f(X)$ on $X_{\mathcal{S}}$ is defined by:

$$f_{\mathcal{S}}(X_{\mathcal{S}}) = E_{X_{\mathcal{C}}}(f(X_{\mathcal{S}}, X_{\mathcal{C}})) \tag{99}$$

The expression in (99) is useful to understand the relationship between $f(X)$ and a low-dimensional subset of the inputs, specially when $X_{\mathcal{S}}$ has few interactions with $X_{\mathcal{C}}$ and when $X_{\mathcal{S}}$ is a set of inputs strongly related with the response, which can be assessed through relative importances. This partial dependence shows how $f(X)$ relates with $X_{\mathcal{S}}$ after taking the average relationship between $f(X)$ and the complement $X_{\mathcal{C}}$.

  - The estimation of (99) follows:

$$\overline{f}(X_{\mathcal{S}}) = \frac{1}{N}\sum_{i=1}^{N} f(X_{\mathcal{S}}, x_{i\mathcal{C}}) \tag{100}$$

  - Note that (99) is different from the effect of $X_{\mathcal{S}}$ ignoring those from $X_{\mathcal{C}}$:

$$\tilde{f}(X_{\mathcal{S}}) = E(f(X_{\mathcal{S}}, X_{\mathcal{C}})|X_{\mathcal{S}}) \tag{101}$$

    The expression (101) is the linear regression equation of $f(X)$ against $X_{\mathcal{S}}$. Equations (99) and (101) are equivalent only when $X_{\mathcal{S}}$ and $X_{\mathcal{C}}$ are independent.

- **General procedures for data mining:** exploring data for research, industrial and commercial purposes has predictive learning as a central issue. Therefore, choosing the statistical learning

method to develop and executing all activities that precede model estimation are fundamental for accomplishing predictive goals.

1. Issues to consider previously to model estimation:
   (a) Computational aspects: dimension of datasets, etc.
   (b) Data types, both for response variable and inputs.
   (c) Missing values identification and treatment.
   (d) Standardization of inputs.
   (e) Inputs distributions and relevance of using transformations.
   (f) Extreme values identification and treatment.
2. Additional aspects to consider before model estimation:
   (a) Subset selection among the entire set of inputs (even if regularization is to be implemented).
   (b) Desired level of model interpretability.
3. The choice of which statistical learning method will base model construction must consider a set of pros and cons that are context-specific. Some general points are addressed in page 351. In what concerns gradient boosting, this method is specially useful for modeling interactions, selecting inputs and handling outliers and missing data.

7 Ensemble learning (Chapter 16, pages 605-623)

Discusses the relationship between boosting with shrinkage, forward stagewise linear regression and (linear) ensemble models with L1 regularization. Presents the Importance Sampled Learning Ensemble (ISLE) algorithm for constructing ensembles from a collection of weak learners.

- The ensemble learning collects a set of simple learners to produce a more efficient predictor from their aggregation. Bagging and random forest are methods that create a committee of trees, while boosting produces a committee of weak learners that evolve throughout the iterations.
- In general, ensemble learning develops a population of base learners and then combines them to produce a composite predictor. Additionally, regularization in the context of ensemble learning can be seen as a way of searching across the entire space of weak learners, selecting the more promising ones to be part of the ensemble.

- A (linear) ensemble model with L1 regularization is approximated by an algorithm named *forward stagewise linear regression*. In its turn, (regression) tree boosting with shrinkage resembles this algorithm. Therefore, (regression) tree boosting with shrinkage resembles a (linear) ensemble model with basis functions defined in a large space of weak learners and constrained to L1 regularization. In few words, there is a close connection between boosting with shrinkage and the lasso estimation.

  – **Penalized regression:** given a (linear) ensemble model based on all possible K-terminal nodes tree $\mathcal{T} = \{\mathcal{T}_k\}$:

  $$f(x) = \sum_{k=1}^{K} \alpha_k T_k(x) \tag{102}$$

  Where $K = card(\mathcal{T})$. Since $\mathcal{T}$ is by definition too large, it is advisable to estimate (102) with regularization:

  $$\min_{\alpha} \Big(\sum_{i=1}^{N} \Big(y_i - \sum_{k=1}^{K} T_k(x_i) \Big)^2 + \lambda J(\alpha) \Big) \tag{103}$$

  Where $J(\alpha)$ is a penalty function given by $\sum_k |\alpha_k|$, if regularization is L1 (lasso), or given by $\sum_k \alpha_k^2$, if regularization is L2 (ridge regression). Considering the lasso penalty function, with a large set $\mathcal{T}$, it is expected that many $\hat{\alpha}_k(\lambda)$ will equal zero for a given regularization parameter λ. Besides, the greater λ, more $\hat{\alpha}_k(\lambda)$ will equal zero.

  – The difficulty when trying to estimate (103) leads to the **forward stagewise linear regression** algorithm:

    1. Initialize $\check{\alpha}_k = 0$, an arbitrarily small constant ϵ and a large number of iterations M.
    2. For each $m \in \{1, 2, ..., M\}$:
       (a) $(\beta^*, k^*) = \underset{(\beta,k)}{\operatorname{argmin}} \sum_{i=1}^{N} \Big(y_i - \sum_{l=1}^{K} \check{\alpha}_l T_l(x_i) - \beta T_k(x_i) \Big)^2$.
       (b) Update $\check{\alpha}_{k^*} \leftarrow \check{\alpha}_{k^*} + \epsilon . sign(\beta^*)$.
    3. Define the final predictor as $f_M(x) = \sum_{k=1}^{K} \check{\alpha}_k T_k(x)$.

    * Initially, with all $\check{\alpha}_k = 0$, it holds that $\lambda \to \infty$.
    * There is a first resemblance with boosting in 2(a), since the constructed $T_{k^*}(x)$ should provide the best fit to the current residuals $y_i - \sum_l \check{\alpha}_l(x_i)$.
    * Another similarity applies for the parameters v in boosting with shrinkage and ϵ in the forward stagewise linear regression algorithm.

* If $K < N$ and M grows indefinitely so as to result in $\beta^* = 0$, the solution $\{\check{\alpha}_k\}$ is equivalent to $\{\hat{\alpha}_k^{OLS}\}$ and $\lambda = 0$.
* For $M < \infty$, some $\check{\alpha}_k$ will be zero, while the remaining coefficients will be non-zero but smaller than $\hat{\alpha}_k^{OLS}$.
* This algorithm approximates the lasso in the context of (linear) ensemble learning, and if all $\{T_k\}$ are uncorrelated, or if $\hat{\alpha}^{lasso}(\lambda)$ is a monotone function of λ, then the forward stagewise linear regression algorithm leads to the precise lasso solution.

- **The bet on sparsity principle:** even though L2 regularization is computationally more efficient, regularization L1 is more appropriate for *sparse contexts*, where there is a large set of regressors (inputs, basis functions, weak learners), and then only a few of them will be part of the model with non-zero coefficients.
  - Since no approach has a good performance under *dense contexts*, preference should be given to those approaches which perform well under sparse contexts.
  - How much sparsity there will be in a given context depends on the true function to be estimated, as well as on the set $\{T_k\}$. In special, the larger $\{T_k\}$, the more sparse should be the problem. Additionally, the bigger the training dataset and the higher the noise-to-signal ratio (NSR), more precisely the coefficients can be estimated, then increasing the set of non-zero coefficients.
- As misclassification error is less sensitive to the variance component than mean-squared error, boosting for classification is generally more resilient to overfitting as M increases.
- Given the (linear) ensemble model:

$$f(x) = \alpha_0 + \sum_{T_k \in \mathcal{T}} \alpha_k T_k(x) \tag{104}$$

Where $\mathcal{T}$ is a dictionary of basis functions, or weak learners, it was discussed before how boosting with shrinkage resembles the estimation of (104) under L1 regularization. Thus, a finite set of basis functions $\mathcal{T}_L = \{T_1(x), T_2(x), ..., T_M(x)\}$ is defined from the training data, and then the following estimation is implemented for a general loss function $L(y, f(x))$:

$$\alpha(\lambda) = \underset{\alpha}{\operatorname{argmin}} \Big(\sum_{i=1}^{N} L(y_i, \alpha_o + \sum_{m=1}^{M} \alpha_m T_m(x_i)) \Big) + \lambda \sum_{m=1}^{M} |\alpha_m| \tag{105}$$

– Some improvements can be introduced to the estimation of (105) so that the components of the ensemble are less correlated.

– Defining the following objective function:

$$f(x) = \int \beta(\gamma) b(x, \gamma) d\gamma \tag{106}$$

It is desired γ_m for $m \in \{1, 2, ..., M\}$ such that $f_M(x) = \alpha_0 + \sum_m \alpha_m b(x; \gamma_m)$ approximates well $f(x)$. In addition to that, γ_m should imply in $b(x; \gamma_m)$ few correlated across m.

– The **measure of relevance of a learner** is defined by:

$$Q(\gamma) = \min_{c_0, c_1} \sum_{i=1}^{N} L(y_i, c_0 + c_1 b(x_i; \gamma)) \tag{107}$$

Imposing some randomness to the choice of γ: $Q(\gamma) \geq Q(\gamma^*)$, where $\gamma^* = \underset{\gamma}{\text{argmin}} Q(\gamma)$. Thus, so as to control over bias and variance, γ should be defined from the definition of the **characteristic window**:

$$\sigma = E_{\mathcal{S}}(Q(\gamma) - Q(\gamma^*)) \tag{108}$$

Where $\mathcal{S}$ is a sampling schema from the training data.

– The **Importance Sampled Learning Ensemble (ISLE)** algorithm is a way for introducing randomness to the estimation of each γ_m:

1. Initialize $f_0(x) = \underset{c}{\text{argmin}} \sum_{i=1}^{N} L(y_i, c)$.
2. For each $m \in \{1, 2, ..., M\}$:
   (a) Estimate $\gamma_m = \underset{\gamma}{\text{argmin}} \sum_{i \in S_m(\eta)} L(y_i, f_{m-1}(x_i) + b(x_i; \gamma))$. Where $S_m(\eta)$ is a random sample of size $N.\eta$ produced in iteration m, typically generatd without replacement.
   (b) Update $f_m(x) = f_{m-1}(x) + v.b(x; \gamma_m)$.
3. Then, the final ISLE ensemble of learners is given by $\mathcal{T}_{ISLE} = \{b(x; \gamma_1), b(x; \gamma_2), ..., b(x; \gamma_M)\}$.

* The smaller η, the more randomness will be introduced, reducing the variance at the cost of more bias (i.e., increasing the characteristic window σ).
* The parameter v introduces memory into the randomization process.

– Some special cases of ISLE are:

* Bagging: $\eta = 1$, randomization with replacement and $v = 0$.

    * Random forest: $\eta < 1$, where $\eta < 1/2$ is equivalent to reducing the number of features allowed to produce initial splitting at each tree construction.
    * GBM with shrinkage: $\eta = 1$.
    * Stochastic GBM: all steps 1-3.

- **Rule ensembles:** for a given tree T_m, a set of rules T^m_{rule} can be derived from the splits into the features space that generate the tree. From a collection of set of rules, follows the ensemble:

$$T_{rule} = \bigcup_{m=1}^{M} T^m_{rule} \tag{109}$$

  The ensemble (109) can be regularized and each input X_j can be added to it.

  – Check the file *Rules Ensemble Models.html* containing notes from the paper *Predictive Learning via Rule Ensembles* by Friedman and Popescu.

8 Additive models, trees, and related methods (Chapter 9, pages 295-335)

Non-parametric methods are explored, some of which constitute very specific approaches (MARS, PRIM), while others (tree-based methods) converge to more complex learning methods. Generalized additive models (GAMs) are a generalization of splines models, such as polynomial regression, piecewise constant regression, regression splines, smoothing splines, and local regression.

- All learning methods presented and discussed in this chapter are considerably more flexible than usual parametric models, specially those that define a linear relationship between the response variable Y and the inputs X, whether they are transformed or not.

- **Generalized additive models (GAMs)**: this class of models considers very broad assumptions towards how inputs X may affect the response Y, and by doing this it allows flexible relationships between them. In the regression setting, a GAM model takes the form:

$$E(Y|X_1, X_2, ..., X_p) = \alpha + f_1(X_1) + f_2(X_2) + ... + f_p(X_p) \tag{110}$$

  Where each $f_j(.)$ is a unspecified smooth and non-parametric function. The estimation of a GAM requires the estimation of each such function by applying a scatterplot smoother (regression spline,

smoothing spline, and so on). In the two-class classification setting, instead of an additive model for the conditional expectation off the response variable, a model for the logit is defined:

$$\log\Big(\frac{\mu(X)}{1-\mu(x)}\Big) = \alpha + f_(X_1) + f_(X_2) + ... + f_p(X_p) \tag{111}$$

Where $\mu(X) = P(Y = 1|X)$. Equation (111) constitutes an *additive logistic regression model*. More generally, for $\mu(X) = E(Y|X)$, a generalized additive model is given by:

$$g(\mu(X)) = \alpha + f_(X_1) + f_(X_2) + ... + f_p(X_p) \tag{112}$$

If the *link function* $g(.)$ is the identity function $g(z) = z$, then equation (112) refers to a model for Gaussian data in a regression setting; if $g(z) = logit(z)$ or $g(z) = probit(z)$, then it refers to a model for binary response variable; if $g(z) = \log(z)$, then equation (112) applies for log-additive models used when there is Poisson count data.

- In addition to their more flexible fit to the data, GAMs also share the properties that make linear models appealing for interpretation.
- Functions $f_j(X_j)$ in (112) do not need to be non-parametric. They can also be linear, or of any parametric relationship. Besides, the components of the GAM can also capture effects other than main effects, i.e., they can model interaction effects between different inputs.
  - The following equation defines a *semi-parametric model*:

    $$g(\mu) = \alpha_k X_1 \beta + f(X_2) \tag{113}$$

    Where α_k distinguishes the intercept for level k of a given categorical input, X_1 is a set of inputs to be modeled linearly, and X_2 a set of inputs to be modeled non-parametrically.
  - Another specification possibility is given by:

    $$g(\mu) = f(X_1) + g_k(X_2) \tag{114}$$

    Where $g_k(X_2)$ models the interaction between a categorical input and a set of numerical inputs.
  - A third model follows:

    $$g(\mu) = f(X_1) + g(X_2, X_3) \tag{115}$$

    Here, $g(X_2, X_3)$ models the interaction between sets of numerical inputs X_2 and X_3.

– The estimation of a GAM derives from a penalized sum of squared residuals (PRSS) very similar to that for smoothing splines. Therefore, given equation (GAM) to which is summed an additive error term:

$$PRSS(\alpha, f_1, f_2, ..., f_p) = \sum_{i=1}^{N} \Big(y_i - \alpha - \sum_{j=1}^{p} f_j(X_j)\Big)^2 + \sum_{j=1}^{p} \lambda_j \int f_j''(t_j)^2 dt_j \tag{116}$$

Where $\lambda_j \geq 0$ are tunning parameters. As it was for smoothing splines, in which there is only one non-parametric function $f(X)$, the solution to the minimization of (116) is a natural cubic spline $\mathcal{S}_j$ for each function $f_j(X_j)$ having one knot at each x_{ij}, for $i \in \{1, 2, ..., N\}$.

– Not only is necessary to impose a restriction such as $\sum_{i=1}^{N} f_j(x_{ij}) = 0$ in order to obtain a unique solution, but to get it is required to apply an algorithm, since no closed form solution is available.

– **Backfitting algorithm for additive models:**

1. Initialize $\hat{\alpha} = (1/N) \sum_{i=1}^{N} y_i$ and $\hat{f}_j 0 \ \forall \ j \in \{1, 2, ..., p\}$.
2. Repeating over $\{1, 2, ..., p, 1, 2, ..., p, ...\}$ until $\hat{f}_j$ changes less than a predefined threshold:

$$\hat{f}_j \leftarrow \mathcal{S}_j\Big(\{y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k(x_{ik})\}_{i=1}^{N}\Big) \tag{117}$$

$$\hat{f}_j \leftarrow \hat{f}_j - (1/N) \sum_{i=1}^{N} \hat{f}_j(x_{ij}) \tag{118}$$

Where $\mathcal{S}_j(Z)$ is a cubic smoothing spline using z as its response variable and X_j as its input.

– Backfitting algorithm also applies if different scatterplot smoothers $\mathcal{S}_j$ are used instead of a smoothing spline.

– The additive logistic regression model is also estimated through the backfitting algorithm, but it requires an additional algorithm for maximizing the log-likelihood function at each iteration.

– **Local scoring algorithm for the additive logistic regression model:**

1. Initialize $\hat{\alpha} = \log(\overline{y}/(1 - \overline{y}))$ and $\hat{f}_j = 0 \ \forall \ j \in \{1, 2, ..., p\}$.
2. Define $\hat{\eta}_i = \hat{\alpha} + \sum_j \hat{f}_j(x_{ij})$ and $\hat{p}_i = 1/(1 + \exp(-\eta_i))$. Iterate the following steps:
   (a) Construct a target variable:

$$z_i = \eta_i + \frac{y_i - \hat{p}_i}{\hat{p}_i(1 - \hat{p}_i)} \tag{119}$$

(b) Construct weights $w_i = \hat{p}_i(1 - \hat{p}_i)$.

(c) Estimate an additive model using z_i as the response and the original inputs x_i, besides weights w_i and a weighted backfitting algorithm. This will provide new estimates for $\hat{\alpha}$ and $\hat{f}_j \ \forall \ j \in \{1, 2, ..., p\}$.

3. Continue step 2 until a convergence criterion is satisfied.

– A similar procedure would be applied for multinomial classification.

– When measuring test set error rate, errors may be treated differently if the cost incurred in one type of misclassification is larger than the other one. Two alternatives may be implemented: first, the Bayes rule under which a score is compared against a threshold in order to predict a given class may account for this difference in loss for different misclassifications; second, observations in class 0 may be multiplied by L_{01}, the loss incurred when a class 0 is assigned to class 1, while observations in class 1 may receive a weight equals to L_{10}, the loss incurred in classifying a data point as being of class 0 when it actually belongs to class 1.

– Backfitting algorithm may not be appropriate for contexts where there are a large amount of inputs. Even so, some penalty functions may serve for sparse models, providing a penalty similar to Lasso. Finally, boosting procedures can be a promising alternative for GAMs in large problems.

- **Tree-based methods:** decision trees partition the features space into disjoint regions in order to assign some constant value for the response variable whenever a data point belongs to one of those regions.

  – Although not necessary, the usual procedure is to take binary partitions of the features space (when a data point satisfies the split condition, it is assigned to the left branch of the split). Therefore, three main parameters of a decision tree are: input variables for the splits, split points, and a number of successive splits to be made.

  – An important advantage of decision trees is interpretability, since no matter how many features are involved in a model, a relatively easy to understand representation displays the reasons for any given prediction. This applies only for the tree representation, as a consequence of the impossibility to represent more than three variables in a features space. Consequently, the partition of features space is visually interpretable only if $p \leq 3$.

– The prediction from a decision tree that creates M disjoint regions $\{R_1, R_2, ..., R_M\}$ in the features space can be algebraically defined by:

$$f(x) = \sum_{m=1}^{M} c_m I(x \in R_m) \tag{120}$$

– The estimation of a decision tree must accomplish the choice of split variables and split points, values for the response variable for data points in each produced region, besides of its shape, given mainly by the number of splits.

– For a **regression tree**, for the first split it is necessary to choose a variable X_j, a split point s, and values c_1 and c_2 such that:

$$\min_{j,s} \Big(\min_{c_1} \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 + \min_{c_2} \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \Big) \tag{121}$$

For any choice of j and s, the optimal values for c_1 and c_2 are:

$$\hat{c}_1 = Ave(y_i | x_i \in R_1(j,s)) \text{ and } \hat{c}_2 = Ave(y_i | x_i \in R_2(j,s)) \tag{122}$$

In its turn, for each variable X_j, the choice of the better split point s is straightforward. Thus, the first split is finished by choosing the best variable X_j that should minimize expression (121) the most. This procedure is repeated for both constructed regions R_1 and R_2, and repeated again for all subsequent regions after this second round. The iteration ends when a predefined stopping rule is reached, such as minimum node size.

– Once understood how to grow a decision tree, it is convenient to prune it, given that **tree size** is a crucial parameter: it should not be too large, in order to not overfit the data, but it should also not be small in such extent that relevant structure is lost by the model.

– **Cost-complexity pruning** is a procedure for selecting the best subtree, where a subtree T is defined as a subset of the original tree T_0. A subtree $T \subset T_0$ is obtained by collapsing any number of non-terminal nodes (check handwritten notes "collapsing_trees.pdf" for an example). The **cost-complexity criterion** is given by:

$$C_\alpha(T) = \sum_{m=1}^{|T|} N_m Q(T) + \alpha |T| \tag{123}$$

Where:

$$N_m = |\{x_i \in R_m\}|$$

$$Q_m(T) = \frac{1}{N_m} \sum_{x_i \in R_m} (y_i - \hat{c}_m)^2$$

$$\hat{c}_m = \frac{1}{N_m} \sum_{x_i \in R_m} y_i$$

Parameter α controls the complexity of the tree: a large value favors a small tree and a less complex model, while a small value tolerates larger trees.

* The objective is to first found the best subtree T_α for each value of α. Then, α is defined through validation techniques, such as K-folds CV.
* The calculation of T_α is made by the **weakest link pruning** procedure. The collapse of non-terminal nodes occurs by choosing the internal node whose collapsing leads to the smallest increase in $\sum_m N_m Q_m(T)$. This is repeated until the single-node (root) tree is achieved. The best subtree T_α belongs to these ensemble of trees, and is found by minimizing $C_\alpha(T)$ as given by (123).

– The construction of a **classification tree** requires changing criteria for splitting nodes and pruning trees, by modifying (121) and (123), respectively. In a classification setting, the following measure is crucial for defining the quality of fit, the proportion of observations in terminal node m that belongs to class k:

$$\hat{p}_{m,k} = \frac{1}{N_m} \sum_{x_i \in R_m} I(y_i = k) \tag{124}$$

The class assignment, then, follows $k(m) = argmax_k\ \hat{p}_{m,k}$. Having defined that, three main measures for node impurity $Q_m(T)$ are **misclassification error rate**:

$$miss\_class = \frac{1}{N_m} \sum_{i \in R_m} I(y_i \neq k(m)) = 1 - \hat{p}_{m,k(m)} \tag{125}$$

The **Gini index**:

$$Gini = \sum_{k \neq k'} \hat{p}_{m,k} \hat{p}_{m,k'} = \sum_{k=1}^{K} \hat{p}_{mk}(1 - \hat{p}_{mk}) \tag{126}$$

And the **cross-entropy**, or **binomial deviance**:

$$cross\_entropy = -\sum_{k=1}^{K} \hat{p}_{mk} \log(\hat{p}_{mk}) \tag{127}$$

Comparing these three measures, the last two are differentiable, and thus are more suited for numerical optimization. Besides, they are more sensitive to node changes, which implies that impure nodes (higher level of class mixture in a given terminal node) are more penalized by Gini index and cross-entropy than by misclassification rate.

  * Consequently, Gini index and cross-entropy are preferred for splitting nodes and growing a tree. With respect to tree pruning and the definition of T_α, any of those three measures are appropriate, with preference being posed to misclassification error rate.
  * Interpretation of Gini index: equation (126) is the expected training error rate of assigning observations to class k with probability $\hat{p}_{mk}$. Additionally, the variance of a binary variable that assumes 1 when an observation is from class k and 0 otherwise is equal to $\hat{p}_{mk}(1-\hat{p}_{mk})$; summing over all k leads to the Gini index.

- Further topics on tree-based methods:
  - The discussion above applies mainly for numerical inputs. When dealing with **categorical variables**, some additional steps are required. First, categories should be ordered according with their proportion of observations falling in class 1. Then, each category is considered as a possible splitting point for the categorical variable, and this in turn can be treated as a numerical variable. The same applies for the regression setting, but instead of class proportion, the ordering of categories is made based on the average of response variable.
    * For multi-class classification, the procedure above does not apply immediately, and modifications are needed.
    * When the number of categories is big, the more likely the tree-based model is to suffer from overfitting.
  - **Loss matrix:** it helps appropriately assigning the cost of misclassification, and has as elements $L_{kk'}$ the loss when classifying an observation of class k to class k', while generally $L_{kk} = 0$. As mentioned previously, two main approaches for estimation making use of losses are: i) constructing thresholds based on values of $L_{kk'}$; ii) weighting observations using $L_{kk'}$. The second option is suited for binary classification, while for multi-class classification a better option would involve the modification of Gini index to $\sum_{k \neq k'} L_{kk'} \hat{p}_{mk} p_{mk'}$.
  - **Missing values:** as an alternative for general strategies for treating missing values, such as

the imputation of zero or the mean of the variable, tree-based models can directly handle them. For categorical inputs, the straightforward procedure is to create a new category for missing data. For numerical inputs, one possibility is the creation of **surrogate variables**. This strategy applies when choosing variables to produce splits, and is based on the definition of substitutes (first best, second best, and so on) for each splitting variable that should be used when the primary (or the first, second, and so on surrogate) variable is missing for a given observation that is about to be assigned to a terminal node.

- The tree construction presented here considers *recursive binary splits*, which has as alternatives **multiway splits** and **linear combination splits**. The later uses as split points linear combinations $\sum_j a_j X_j \leq s$, instead of a single variable X_j with $a_j = 1$.
  - The presented approach is named **CART**, for classification and regression tree.
- Three main problems with tree-based models are the following: i) instability, i.e., high variance of estimations and, therefore, of predictions; ii) lack of smoothness of the estimated hyperplane, which is specially harmful for the regression setting; and iii) difficulty to capture additive structure, such as $Y = c_1 I(X_1 \leq t_1) + c_2 I(X_2 \leq t_2)$.

- **Patient Rule Induction Method (PRIM):** this method is, as the tree-based ones, non-parametric and also explores the features space in order to capture patterns for best assigning values for the response variable from inputs. PRIM partitions the features space producing boxes that are represented by rules of the form $(a_1 \leq X_1 \leq b_1)$ & $(a_2 \leq X_2 \leq b_2)$. The boxes sequentially produced by PRIM try to increase more and more the average of response variable for observations inside the box. PRIM handles both categorical predictors and missing values in the same way as CART. Besides, even though PRIM is designed for regression tasks, binary classification can also be accomplished by this method if the binary response variable is 0-1 encoded.
  - **Algorithm for fitting a PRIM model:**
    1. Starting with all training data, a reference box is constructed containing all available data.
    2. The box should then be compressed in a face given by input X_j, and a proportion α of observations with the highest or the lowest values of X_j should be dropped out of the box. The proportion α should be set so as to maximize the average of Y inside the constrained box (in general, $\alpha \in \{0.05, 0.1\}$).

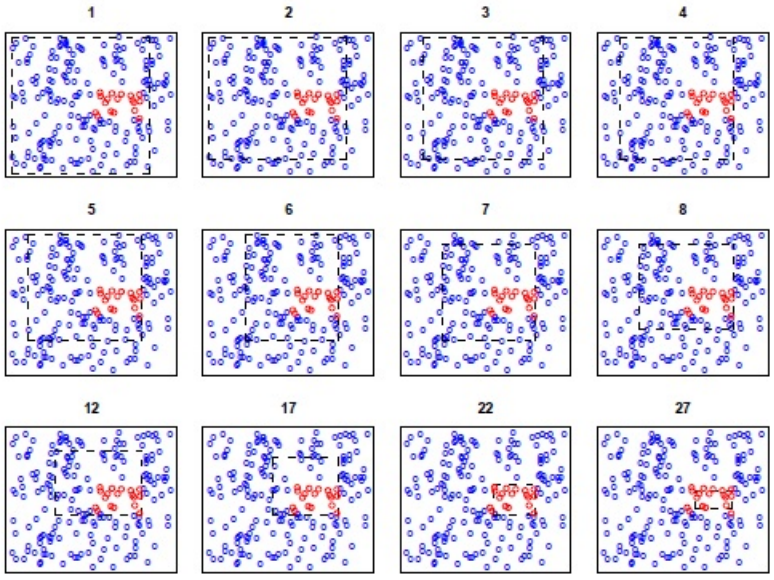

FIGURE 9.7. *Illustration of PRIM algorithm. There are two classes, indicated by the blue (class 0) and red (class 1) points. The procedure starts with a rectangle (broken black lines) surrounding all of the data, and then peels away points along one edge by a prespecified amount in order to maximize the mean of the points remaining in the box. Starting at the top left panel, the sequence of peelings is shown, until a pure red region is isolated in the bottom right panel. The iteration number is indicated at the top of each panel.*

Figure 8.1: PRIM

3. Step 2 is repeated until a predefined minimal number of observations remain in the box (in general, 10).
4. Reversely, the final box should be sequentially enlarged in any of its faces, provided that the average of response variable increases even further.
5. Steps 1-4 give a collection of boxes of different sizes. Cross-validation may help choosing the most appropriate among them, B_1.
6. Now, excluding training data points belonging to B_1 and repeating steps 2-5 produces boxes B_2, B_3, ..., B_k, with k previously defined. Each box constitutes rules based on subsets of inputs.

- **Multivariate Adaptive Regression Splines (MARS):** this methods is a generalization of step-

wise linear regression, since it uses as basis functions piecewise linear functions such as:

$$(x-t)_+ = \begin{cases} x-t & \text{if } x > t \\ 0 & \text{otherwise} \end{cases} \quad \text{and } (t-x)_+ = \begin{cases} t-x & \text{if } x < t \\ 0 & \text{otherwise} \end{cases} \tag{128}$$

The estimation of a MARS model considers a complete collection of basis functions for which the knots are training data observations x_{ij}:

$$\mathcal{C} = \{(X_j - t)_+, (t - X_j)_+\}_{j \in \{1,2,\ldots,p\},\ t \in \{x_{1j}, x_{2j}, \ldots x_{Nj}\}} \tag{129}$$

Given $h_m(X)$ a product between two or more components of (129), such as $(X_1 - x_{31,1})_+ * (x_{27,4} - X_4)_+$, MARS model follows:

$$f(X) = \beta_0 + \sum_{m=1}^{M} \beta_m h_m(X) \tag{130}$$

Since (130) is a linear model, parameters β_m are estimated through least squares.

- The estimation of a MARS model is made upon a forward stagewise approach. First, the constant $h_0(X) = 1$ is added to the model, and all pairs in (129) are candidate to be interacted with $h_0(X)$ and then to be added into the model.
- With no loss in generality, supposing that $(X_2 - x_{72})_+$ and $(x_{72} - X_7)_+$ when multiplied by $h_0(X) = 1$ and added into (130) produce the largest decrease in training error, then the collection of $h_m(X)$ now becomes $h_0(X) = 1$, $h_1(X) = 1.(X_2 - x_{72})_+$ and $h_2(X) = 1.(x_{72} - X_2)_+$.
- Another pair of piecewise linear functions is taken from (129) and interacted with any of functions $h_0(X)$, $h_1(X)$, and $h_2(X)$. The products that again reduce the most the training error will be selected; supposing they are given by $(x_{72} - X_2)_+.(X_1 - x_{51})_+$ and $(x_{72} - X_2)_+.(x_{51} - X_1)_+$, then the functions added into (130) are: $h_3(X) = (x_{72} - X_2)_+.(X_1 - x_{51})_+$ and $h_4(X) = (x_{72} - X_2)_+.(x_{51} - X_1)_+$.
- Since model (130) will be large enough, it is likely to be leading to overfitting of data. Therefore, it is implemented a backwards procedure of deletion of those $h_m(X)$ that increases the less the training error. This results in a collection $\{\hat{f}_\lambda\}_{\lambda=1}^{M}$ of models with different number of elements λ.

- For computational reasons, generalized cross-validation is used for selecting the best λ. Thus, the following measure is minimized in λ:

$$GCV(\lambda) = \frac{\sum_{i=1}^{N}(y_i - \hat{f}_\lambda(x_i))^2}{(1 - M(\lambda)/N)^2} \tag{131}$$

  Where $M(\lambda)$ is the effective number of degrees of freedom in the model, the number of parameters plus the observational values tried out for constituting knots.

- MARS uses those piecewise linear functions because of their ability to operate locally, instead of being applied for the entire features space, as it will be the case for regression splines, etc. Additionally, the forward stagewise procedure is built up hierarchically, since new products are only constructed from products already in the model.

- The model is restricted so that only one input can participate in multiway products. A further restriction would limit the highest order of possible interactions, for instance, allowing only pairwise products, and not products between three or more piecewise linear functions.

- MARS is designed for regression tasks, but can be adapted for classification settings as well. For binary classification, the response may be encoded with 0-1 and MARS normally fitted. The indicator response approach (matrix Y with rows for observations and columns for possible classes) combined with MARS makes possible multi-class classification.

- MARS deals with categorical inputs in the same way as CART.

- Finally, if piecewise linear functions are replaced by stepwise functions $I(X_j < t_j)$ and $I(X_j > t_j)$, and if any model term $h_m(X)$ can not be used more than once for interactions, then MARS converges to CART.

- **Hierarchical Mixture of Experts (HME):** this method shares the same structure of tree-based methods, such as CART. Differently from them, however, the splits are not definite, but follow probability distributions, in addition to being (generally) multiway splittings based on linear combinations of multiple inputs, instead of binary splits from unique inputs.

  - Considering a two-level HME model, there are two layers of *gating networks* and a final layer of *expert networks*, given that terminal nodes are here defined as *experts*.

  - The HME model for a two-level architecture is based on the following equations. The first

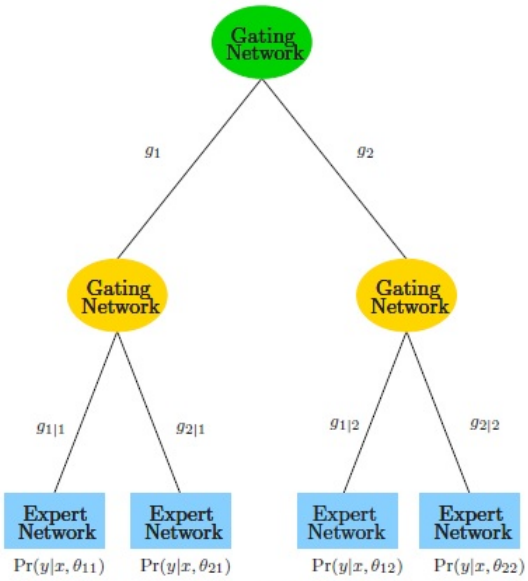

FIGURE 9.13. *A two-level hierarchical mixture of experts (HME) model.*

Figure 8.2: Hierarchical Mixture of Experts (HME) model

applies for the top gating network:

$$g_j(x;\gamma_j) = \frac{\exp(x\gamma_j)}{\sum_{k=1}^{K}\exp(x\gamma_k)} \text{ for } j \in \{1,2,...,K\} \tag{132}$$

From the initial node, $g_j(x;\gamma_j)$ is the probability of assigning a data point x to branch j. For a given j from top gating network, the equation for second level gating network is given by:

$$g_{l|j}(x;\gamma_{jl}) = \frac{\exp(x\gamma_{jl})}{\sum_{k=1}^{K}\exp(x\gamma_{jk})} \text{ for } l \in \{1,2,...,K\} \tag{133}$$

Now, $g_{l|j}(x;\gamma_{jl}$ is the probability of assigning x to branch l, given that it has been firstly assigned to branch j in the previous gating network. Finally, for each terminal node (expert), corresponds a model $Y \sim P(y|x;\theta_{jl})$. In the case of a regression problem, $\theta_{jl} = (\beta_{jl}, \sigma^2_{jl})$ and:

$$y = x\beta_{jl} + \epsilon \text{ where } \epsilon \sim N(0, \sigma^2_{jl}) \tag{134}$$

In the context of a binary classification problem:

$$P(y = 1|x;\theta_{jl}) = \frac{1}{1+\exp(-x\theta_{jl})} \tag{135}$$

- The total probability of $Y = y$ depends on x and a collection of parameters $\Phi = (\gamma_j, \gamma_{jl}, \theta_{jl})$:

$$P(Y = y|x; \Phi) = \sum_{j=1}^{K} g_j(x; \gamma_j) \sum_{l=1}^{K} g_{l|j}(x; \gamma_{jl}) P(y|x; \theta_{jl}) \tag{136}$$

  The data log-likelihood $\sum_i \log(P(y|x; \Phi))$ should be maximized over Φ for the estimation of HME model. Solving the optimization problem usually requires the application of EM algorithm.

- One issue about HME models concerns defining the topology of the model. A possible solution involves running a CART model to define an appropriate depth.

- **Missing data:** when for some inputs training data have missing values, two main alternatives are: dropping those observations with missing values, and using information on non-missing values to impute data. A first, but crucial step is to check whether missing data is at random, or if there is some underlying relationship with the process that relates inputs to response.

  - **Missing at random (MAR)** is an assumption under which the mechanism producing missing data is independent of the unobserved values. Given y N x 1 the vector of response variable values, R the N x p matrix indicating missing value status, X the N x p matrix of data, X_{obs} the matrix of observed values in X, $Z = (y, X)$ and $Z_{obs} = (y, X_{obs})$, the MAR assumption requires $P(R|Z; \theta) = P(R|Z_{obs}; \theta)$, where θ is a vector of parameter governing the process of missing values generation.

  - A stronger assumption is **missing completely at random (MCAR)**, which requires $P(R|Z; \theta) = P(R|\theta)$. Thus, the occurrence of missings does even depend on the observed values. Imputation procedures usually rely on the MCAR assumption.

  - To understand the process generating missing values, it is important to have a clear picture of the data collection process. Categorical inputs, however, indicate if their missings are biased by fitting a model opposing the response variable to dummies for categories, including one for the missing data level. If this variable is relevant for predicting the response, then missings are definitively not at random.

  - In addition to dropping observations with missing values and to imputation, some learning algorithms implicitly deal with missing values, as it is the case of CART procedure, which uses surrogate variables for the case of missing data. MARS and PRIM have similar procedures.

– Imputation can make use of mean or median values of non-missing values for the inputs. Another possibility is to fit a model which relates a given inputs to the remaining ones. Then, whenever a missing value is detected, the model can be used to predict a value for imputation. In this application, CART, or any other learning method that handles missing data is an appropriate choice for the estimation method, since they do not require dropping observations for fitting the imputation model.

9 Random forests (Chapter 15, pages 587-601)

Describes an additional method based on decision trees. The algorithm for random forest estimation de-correlates trees that constitute the ensemble.

- Bagging or bootstrap aggregation (chapter 8) and boosting (chapter 10) are methods constructed upon several different estimators. While bagging tries to reduce the variance of prediction by averaging distinct predictors, boosting goes for reducing the bias by generating weak learners that evolve through estimations.

- These methods perform specially well when using base learners with low bias and high variance, such as tree-based methods. Boosting has generally dominated bagging, but **random forests** is a strong competitor is terms of popularity. This method tries to reduce even further the variance of an ensemble of estimators by de-correlating them before averaging predictions. If boosting may outperform random forests in many applications, the latter has the advantage of being simpler to tune.

- Trees generated for bagging are identically distributed, so that the bias (difference between actual value being predicted and expected value of prediction) of the average is the same of each individual predictor. While bias is unchanged, variance may be reduced by averaging. This because the variance of B identically distributed variables each with variance σ^2 is:

$$\rho\sigma^2 + \frac{1-\rho}{B}\sigma^2 \tag{137}$$

  Where ρ is the correlation between each pair of variables in the sequence. Therefore, with $B > 1$, expression (137) is smaller than σ^2. However, even with B large enough, further reductions in (137) are limited by ρ. Random forests provide a way for reducing (137) using large values of B and making use of procedures for reducing ρ.

- At each tree growing and at each node split, a random subset of $m \leq p$ inputs is considered for splitting. The following algorithm describes the estimation procedure for random forests.

  1. For $b \in \{1, 2, ..., B\}$:
     (a) Create a bootstrap sample of size N from the training data.
     (b) Grow a tree $T(x; \Theta_b)$, where Θ_b captures split variables, cutpoints and terminal-node values, based on this sample by sequentially executing the following steps until a minimum node size n_{min} (value for the minimum number of data points in terminal nodes) is reached.
        i. Randomly select m inputs from the p original variables.
        ii. Find the best combination of splitting variable and point.
        iii. Perform the binary split.
  2. Output the ensemble of trees $\{T(\Theta_b)\}_{b=1}^{B}$.
  3. For regression, the random forest predictor is given by:

$$\hat{f}_{rf}^{B}(x) = \frac{1}{B} \sum_{b=1}^{B} T(x; \Theta_b) \tag{138}$$

     While for classification, given $\hat{C}_b(x)$ the predicted class for data point x by tree b:

$$\hat{C}_{rf}^{B}(x) = \text{majority vote}\{\hat{C}_{rf}^{b}(x)\}_{b=1}^{B} \tag{139}$$

- Main hyper-parameters of random forests are minimum node size n_{min}, which controls the complexity of grown trees, and m, the number of randomly selected variables for splitting.
  - The smaller n_{min} the larger a tree will likely be, since more splits have to be produced in order to make the smallest terminal node have fewer data points.
  - The smaller m the smaller the correlation between trees and, hence, the smaller the variance of the final estimator.
  - Suggested values for m are $m = floor(\sqrt{p})$ for classification problems and $m = floor(p/3)$ for regression problems. For n_{min}, suggested values are 1 and 5, respectively.

- Bagging and random forests perform better when averaging over highly non-linear estimators. Indeed, for linear estimators, bagging does not change properties of estimation.

- An interesting way of implementing random forests is by assessing the **out-of-bag (OOB) error**. This measure is given by the average over all data points x_i of the training error calculated using only those trees in the random forest constructed upon bootstrap samples that do not contain x_i.

- **Variable importance** can be used for interpretation purposes similarly to its application for boosting models. The importance of a given feature considers its contribution for reducing training error as it is used for splitting, and this improvement is averaged over all trees in the random forest. The fact that the set of candidate inputs for splitting is randomly defined increases the probability of any input to present a positive importance in comparison to boosting, which is likely to concentrate only in most relevant variables.

- A drawback of random forest occurs when m is set to small values and the set of inputs is large, but the subset of relevant ones is small.

- Random forests are robust to overfitting when it comes to setting the number of trees B. Even so, irrespective of the random forest size, the model still can overfit if the trees are excessively big.

10 Support vector machines and flexible discriminants (Chapter 12, pages 417-454)

Presents, discusses and derives support vector methods, namely, maximal margin classifier, support vector classifier and support vector machine. These explicitly estimate decision boundaries in the input space to separate (perfectly or not) classes, or additionally, to perform regression.

- Both logistic regression and linear discriminant analysis (LDA) models fit a linear decision boundary in the input variables space. Support vector methods explicitly construct hyperplanes aiming to separate classes. The first of such methods, *maximal margin classifier*, tries to generate a hyperplane that perfectly separates the two classes. *Support vector classifier*, in its turn, allows overlap of classes, while still fitting a linear decision boundary. *Support vector machines* consist on a generalization which not only allows misclassification of training data points, but also produce non-linear decision boundaries in the original input space. Finally, *flexible discriminant models* generalizes LDA models.

- A hyperplane over the $\mathbb{R}^p$ space is defined by the set of points:

$$\{x \in \mathbb{R}^p | f(x) = x^T\beta + \beta_0 = 0\} \tag{140}$$

Given a two class $Y\{-1, 1\}$ classification problem characterized by a **perfect separation** between classes, a hyperplane $f(x)$ can be found such that points above it ($f(x) > 0$) are assigned to $Y = 1$, and points below it ($f(x) < 0$) are assigned to $Y = -1$. Therefore, the classification rule induced by $f(x)$ is:

$$G(x) = sign(x^T\beta + \beta_0) \tag{141}$$

The perfect separation can be summarized by the following property of the *margin* for all data points available: $y_i f(x_i) > 0 \ \forall \ i \in \{1, 2, ..., N\}$. Indeed, maximizing the margin $M \in \mathbb{I}_+$ that separates the set of points for which $Y = 1$ from those that $Y = -1$ is the way how to define the perfect separating hyperplane $f(x)$:

$$\max_{\beta, \beta_0} M \tag{142}$$

$$\text{subject to } ||\beta|| = 1$$

$$y_i(x_i^T\beta + \beta_0) \geq M \ \forall \ i \in \{1, 2, ..., N\}$$

More precisely, $2M$ is the margin between the two classes, since M is actually the distance between the closest data points from $Y = 1$ and $Y = -1$ to the hyperplane. Optimization problem (142) can be redefined as:

$$\min_{\beta, \beta_0} ||\beta|| \tag{143}$$

$$\text{subject to } y_i(x_i^T\beta + \beta_0) \geq 1 \ \forall \ i \in \{1, 2, ..., N\}$$

Where M is taken to be equal to $1/||\beta||$.

- The estimated function $\hat{f}(x) = x^T\hat{\beta} + \hat{\beta}_0$ that follows from (143) is named the **maximal margin classifier**.
- Optimization problem (143) is a quadratic optimization problem, with a quadratic objective function and linear inequality constraints.
- Note that the constraint $y_i(x_i^T\beta + \beta_0) \geq M \ \forall \ i$ requires that all training data points are far from the hyperplane for a distance of at least M, since $y_i(x_i^T\beta + \beta_0)$ captures the *sign distance* between data points and the hyperplane. Besides, the fact that $M > 0$ requires that $y_i(x_i^T\beta + \beta_0) > 0 \ \forall \ i$, i.e., that all training data points are correctly classified.

– The margin is the minimal perpendicular distance between points of a class and the hyperplane. From this follows the objective of maximizing this distance with respect to the nearest data point of a class.

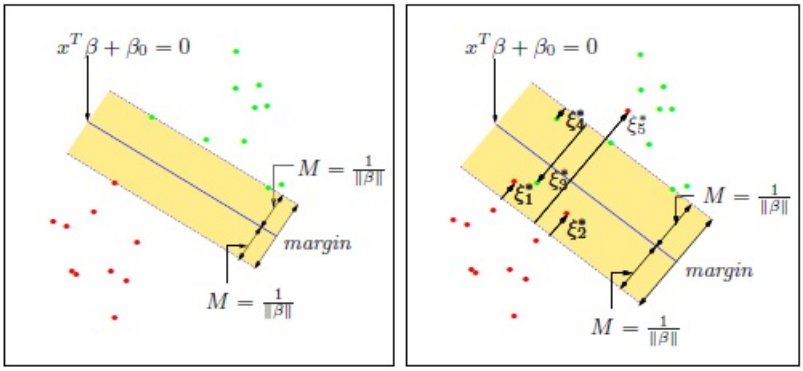

FIGURE 12.1. *Support vector classifiers. The left panel shows the separable case. The decision boundary is the solid line, while broken lines bound the shaded maximal margin of width $2M = 2/\|\beta\|$. The right panel shows the nonseparable (overlap) case. The points labeled ξ_j^* are on the wrong side of their margin by an amount $\xi_j^* = M\xi_j$; points on the correct side have $\xi_j^* = 0$. The margin is maximized subject to a total budget $\sum \xi_i \leq$ constant. Hence $\sum \xi_j^*$ is the total distance of points on the wrong side of their margin.*

Figure 10.1: Separating hyperplane

- The **non-separable case** demands the change of the main constraint of (142) to:

$$y_i(x_i^T\beta + \beta_0) \geq M - \xi_i \tag{144}$$

Or:

$$y_i(x_i^T\beta + \beta_0) \geq M(1 - \xi_i) \tag{145}$$

For all data points, with $\xi_i \geq 0$ and $\sum_i \xi_i \leq K$. Considering the constraint given by (145), ξ_i is the proportion to which the data point indexed by i is distanced from its correct margin.

– Since $y_i(x_i^T\beta + \beta_0)$ is the sign distance between the data point and the hyperplane, if this measure is smaller than M, then even if the point is located in the correct side of the hyperplane, it will be in the wrong side of its margin, given that the distance has not surpassed M.

– Misclassification occur when $\xi_i > 1$, since this implies in a negative sign distance. The constraint under which $\sum_i \xi_i \leq K$ limits the number of training data points that are allowed to be misclassified.

- The optimization problem for the non-separable case is constructed upon (143), and makes use of constraint (145):

$$\min_{\beta,\beta_0} ||\beta|| \tag{146}$$
$$\text{subject to } y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i \ \forall \ i \in \{1, 2, ..., N\}$$
$$\xi_i \geq 0, \ \sum_i \xi_i \leq K$$

- Considering the *slack variables* ξ_i necessary for defining a solution to (146), the main constraint shows that training data points very far from the hyperplane ($y_i(x_i^T\beta + \beta_0) \gg 0$) are less important to the definition of the hyperplane than data points near to it. This because the latter require $\xi_i > 0$ and progressively larger.

- Support vector classifier does not only accomplish the task of classification in the non-separable case, but also it is more robust to overfitting than maximal margin classifier, even when perfect separation would be possible.

- The characterization of the solution to the optimization problem for estimating the **support vector classifier** involves another redefinition of the problem, this time to:

$$\min_{\beta,\beta_0} \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N} \xi_i \tag{147}$$
$$\text{subject to } y_i(x_iT\beta + \beta_0) \geq 1 - \xi_i \ \forall \ i \in \{1, 2, ..., N\}$$
$$\xi_i \geq 0 \ \forall \ i \in \{1, 2, ..., N\}$$

Parameter C in (147) relates inversely to K in (146). The higher C, the higher the penalty to misclassification, while the smaller will be K, since less training data points would be allowed to be misclassified.

- The Lagrangian of (147) that should be minimized with respect to β, β_0 and ξ_i is given by:

$$L = \frac{1}{2}||\beta||^2 + C\sum_{i=1}^{N} \xi_i - \sum_{i=1}^{N} \alpha_i[y_i(x_i^T\beta + \beta_0) - (1 - \xi_i)] - \sum_{i=1}^{N} \mu_i\xi_i \tag{148}$$

The first order conditions are the following:

$$\beta = \sum_{i=1}^{N} \alpha_i y_i x_i \tag{149}$$

$$0 = \sum_{i=1}^{N} \alpha_i y_i \tag{150}$$

$$\alpha_i = C - \mu_i \text{ for } i \in \{1, 2, ..., N\} \tag{151}$$

$$\alpha_i [y_i (x_i^T \beta + \beta_0) - (1 - \xi_i)] = 0 \text{ for } i \in \{1, 2, ..., N\} \tag{152}$$

$$\mu_i \xi_i = 0 \text{ for } i \in \{1, 2, ..., N\} \tag{153}$$

$$y_i (x_i^T \beta + \beta_0) - (1 - \xi_i) \geq 0 \text{ for } i \in \{1, 2, ..., N\} \tag{154}$$

$$\xi_i, \alpha_i, \mu_i \geq 0 \text{ for } i \in \{1, 2, ..., N\} \tag{155}$$

- From condition (149), the estimated hyperplane should have $\hat{\beta} = \sum_{i=1}^{N} \hat{\alpha}_i y_i x_i$. Given condition (152), the only data points that will contribute with $\hat{\beta}$ will be those for which $y_i (x_i^T \beta + \beta_0) = 1 - \xi_i$. These observations are called **support vectors**, and may be either such that $\hat{\xi}_i = 0$, i.e., data points placed exactly over the margin, or such that $\hat{\xi}_i > 0$, i.e., data points in the wrong side of its margin.
- Those support vectors for which $\hat{\xi}_i = 0$ will have $\mu_i > 0$, and thus from (151) and (153), support vectors with $\xi_i = 0$ will have $0 < \hat{\alpha}_i < C$. The same argument implies that $\hat{\alpha}_i = C$ for support vectors such that $\hat{\xi}_i > 0$.
- The definition of $\hat{\beta}_0$ follows from condition (154) for support vectors with $\hat{\xi}_i = 0$, and using $\hat{\beta}$ given by (149).
- Given the solutions for $\hat{\beta}$ and $\hat{\beta}_0$, and considering a data point x, its classification considers:

$$\hat{G}(x) = sign(\hat{f}(x)) = sign(x^T \hat{\beta} + \hat{\beta}_0) \tag{156}$$

- The hyper-parameter that should be tuned for optimizing predictive performance is C, and cross-validation techniques may be applied for the definition of an optimum value of C. The larger C, the narrower the margin will be, since less training data points can be misclassified.

- Support vectors are data points that solely define the estimated hyperplane. So, if $\hat{f}(x)$ depends only on their location, changing the position of any of those observations will change the estimated classifier. Note that, in the case of support vector classifiers (and SVMs), support vectors may be data points placed precisely on its margin, or in the wrong side of its margin, or even in the wrong side of the hyperplane.

- **Support vector machines** have the same structure of support vector classifiers. The main difference concerns the use of basis functions to enlarge the features space in order to produce linear decision boundaries in that enlarged space, but non-linear boundaries in the original space, which may help increasing predictive performance. The hyperplane equation is now given by $f(x) = h(x)^T\beta + \beta_0$, where $h(x)$ is a vector of basis functions constructed upon the original inputs.

  - Considering the support vector classifier setting, the hyperplane equation is estimated by:

$$\hat{f}(x) = x^T\hat{\beta} + \hat{\beta}_0 = \sum_{i=1}^{N} \hat{\alpha}_i y_i x^T x_i + \hat{\beta}_0 = \sum_{i=1}^{N} \hat{\alpha}_i y_i < x, x_i > +\hat{\beta}_0 \tag{157}$$

    The generalization provided by SVMs replaces x and x_i by the vector of basis functions $h(x)$ and $h(x_i)$:

$$\hat{f}(x) = h(x)^T\hat{\beta} + \hat{\beta}_0 = \sum_{i=1}^{N} \hat{\alpha}_i y_i h(x)^T h(x_i) + \hat{\beta}_0 = \sum_{i=1}^{N} \hat{\alpha}_i y_i < h(x), h(x_i) > +\hat{\beta}_0 \tag{158}$$

  - Instead of using the standard inner product function:

$$< h(x), h(x') >= h_1(x)h_1(x') + h_2(x)h_2(x') + ... + h_k(x)h_k(x') \tag{159}$$

    It may be replaced by a more general **kernel function** $K(x, x') =< h(x), h(x') >$ that will compute inner products in the transformed space. A kernel is a positive semi-definite function of vectors x and x' whose calculation equals the inner product between large vectors of basis functions. Some popular options are:

$$K(x, x') = (1+ < x, x' >)^d \tag{160}$$

$$K(x, x') = \exp(-\gamma||x - x'||^2) \tag{161}$$

$$K(x, x') = \tanh(\kappa_1 < x, x' > + \kappa_2) \tag{162}$$

The d-th degree polynomial, the radial basis, and the neural network kernels, respectively. Then, (158) is generalized to:

$$\hat{f}(x) = \sum_{i=1}^{N} \hat{\alpha}_i y_i K(x, x_i) + \hat{\beta}_0 \tag{163}$$

- The use of kernels makes unnecessary the definition of the vector of basis expansions $h(x)$, given that each kernel function and its specification will calculate the inner product of some basis expansions vectors.
- A large value of C penalizes more misclassification of training data points. Then, $||\hat{\beta}||$ will be larger than otherwise, leading to a more flexible function $\hat{f}(x)$.
- Note that the support vector classifier is a particular case of SVM when the kernel is linear: $K(x, x') =< x, x' >= \sum_j x_j x'_j$.
- The radial basis kernel produces a local behavior, since the larger the distance between a data point x and a training data point x', the larger $||x - x'||$ will be, and thus the smaller $K(x, x')$ will be. Consequently, far training data points x' will contribute less with the prediction $\hat{f}(x)$. The parameter γ controls this sensitivity, with large values producing flexible classifiers.

- **SVMs and regularization:** the optimization problem that leads to the SVM classifier can be redefined as:

$$\min_{\beta, \beta_0} \sum_{i=1}^{N} \max[0, 1 - y_i f(x_i)] + \frac{\lambda}{2}||\beta||^2 \tag{164}$$

Where $\lambda = 1/C$. Loss function $L(y, f) = \max[0, 1 - yf(x)]$ is the **hinge loss function**, and it has a behavior similar to those for (negative) log-likelihood and binomial deviance. The optimization problem (164) depicts the SVM estimation as an intrinsically regularized learning method, since β in $f(x) = h(x)^T\beta + \beta_0$ will be shrunk toward zero.

- Since the use of kernels implies in basis functions $h(x)$ that can be very large, SVMs are affected by the **curse of dimensionality**. The existence of noise input variables in x is not taken into account when estimating the model. All inputs and their basis expansions are treated equally, differently from adaptive models, which focus on more promising features.

- Support vector methods do not explicitly produce estimates for class probabilities. Some additional algorithms may perform this estimation by regressing true labels against predicted labels from a SVM model, for instance. These outputs are then used as inputs to an auxiliary estimation method, which can be a logistic regression (*Platt scaling*) or a piecewise regression (*isotonic regression*).

- **SVMs for regression tasks:** optimization problem (164), that reveals how SVM estimation is implicitly regularized, allows the use of SVMs for a regression problem. Considering for simplicity the support vector classifier setting, the following cost function is minimized for estimating a support vector regressor:

$$H(\beta, \beta_0) = \sum_{i=1}^{N} V(y_i - f(x_i)) + \frac{\lambda}{2}||\beta||^2 \tag{165}$$

  Where $V(.)$ is a continuous version of the hinge loss function in (164) given by:

$$V_\epsilon(r) = \begin{cases} 0 & \text{if } |r| \leq \epsilon \\ |r| - \epsilon & \text{, otherwise} \end{cases} \tag{166}$$

  This cost function is very similar to Huber loss (73).

- Support vector machines may also be applied for **multiclass classification problems**. Two main possibilities arise: one SVM model for each pair of classes, or a collection of SVM models each opposing a class to the remaining ones.