Business Statistics

Tommaso Proietti

DEF - Università di Roma 'Tor Vergata'

Additive Models and Trees

Additive Models

Let us consider the regression model with multiple inputs X

$$Y = f(X_1, X_2, \ldots, X_p) + \epsilon,$$

where $f(X_1, X_2, \ldots, X_p) = \mathsf{E}(Y|X_1, X_2, \ldots, X_p)$ and

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

The functions $f_j(X_j)$ are smooth nonparametric functions, e.g. cubic smoothing splines, of a single input.
This approach stands somewhat between linear (and additive) regression and the non-linear non-additive regression approach, which is too general and prone to the curse of dimensionality.

A training sample is available $\{(y_i, \mathbf{x}_i), i = 1, \ldots, N\}$, $\mathbf{x}_i = (x_{i1}, \ldots, x_{ip})'$.
Estimation of the intercept and the functions f_j is carried out by the following backfitting algorithm:

- Initialisation: set $\hat{\alpha} = \frac{1}{N}\sum_i y_i$ and $\hat{f}_j \equiv 0$
- Compute the partial residuals from the fit excluding the j-th function:
$$e_{i\backslash j} = y_i - \hat{\alpha} - \sum_{k \neq j} \hat{f}_k(x_{ik})$$
- Apply a smoother $\mathcal{S}_j$ to $e_{i\backslash j}$ to obtain $\hat{f}_j$
- Reset $\hat{f}_j$ equal to $\hat{f}_j - \sum_{i=1}^{N} \hat{f}_j(x_{ij})$

The last step ensures that $\sum_{i=1}^{N} \hat{f}_j(x_{ij}) = 0, \forall j$.

- The smoother $\mathcal{S}_j$ can be a cubic smoothing spline with smoothness parameter λ_j or a local polynomial smoother with bandwidth h_j.
- The selection of the parameters governing the complexity of the fit is done by minimizing the test error (e.g. evaluated by cross-validation).

Figure : Additive model using smoothing splines: `gam(y ~ s(x1)+s(x2), data = clothing)`.

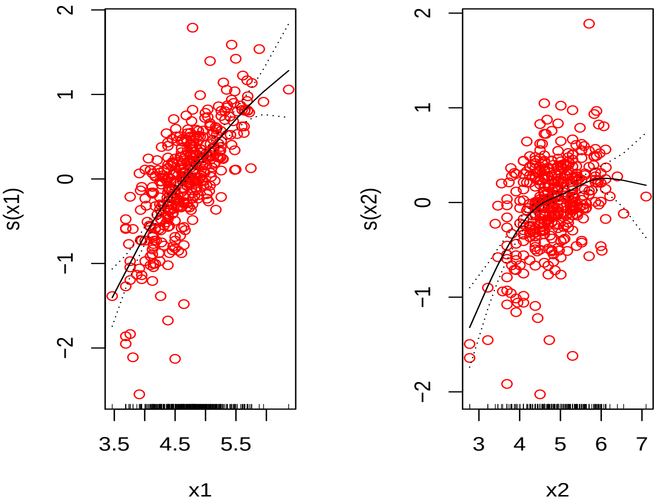
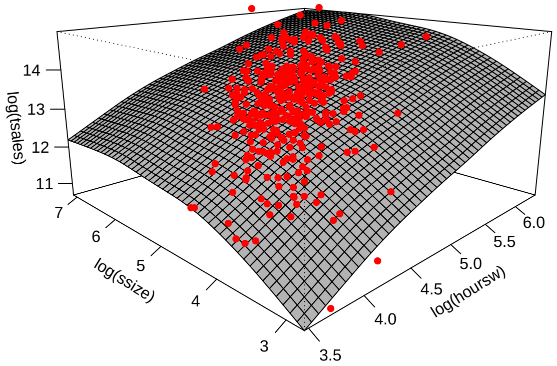


Figure : Additive model using splines: `gam(y ∼ s(x1)+s(x2))`.



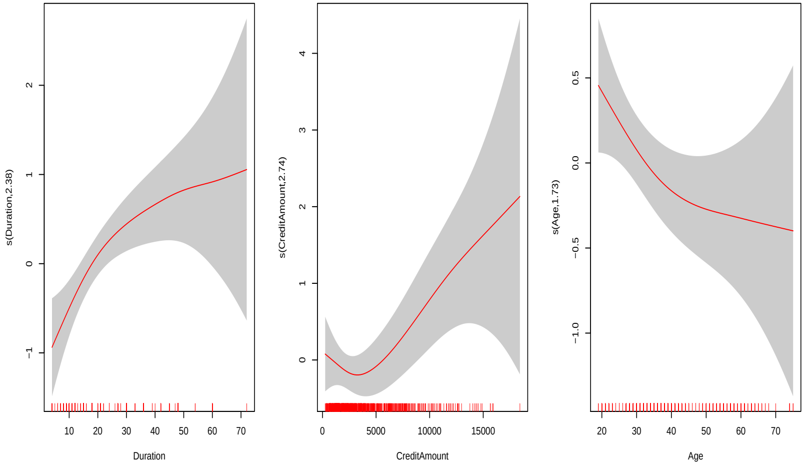
Additive Logistic Regression

The generalized additive logistic regression is additive in the log-odds:

$$\ln \frac{P(G=1|X)}{P(G=0|X)} = \alpha + f_1(X_1) + f_2(X_2) + \cdots + f_p(X_p).$$

where $f_j(X_j)$ is a smooth nonparametric function.
Estimation is carried out by the backfitting algorithm.

Figure : Additive logistic regression. `gam(Group ~ s(Duration)+s(CreditAmount) + s(Age)+factor(...), family=binomial(link="logit")`



Tree-based Methods

Tree-based methods partition the input space into rectangles, using rules to identify regions characterized by a homogeneous response to the inputs.

The idea is to sub-divide, or partition, the space into smaller rectangular regions and to fit a simple model, such as a constant, to the observations falling in the region.

Basic model selection problems, like variable selection, monotonic transformations and interactions are handled by deciding which variables to use in the partitioning and how.

Regression Trees

Suppose we have p input variables $X_1, \ldots, X_p$ and a **quantitative** response variable Y.

Denote the training sample by $\{(y_i, \mathbf{x}_i), i = 1, \ldots, N\}$.

Let $\{R_m, m = 1, \ldots, M\}$ denote a set of disjoint rectangles that partition the input space $(X_1, \ldots, X_p)$.

A regression tree is an additive model of the form

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

Conditional on the splits, the least squares estimate of c_m is $\hat{c}_m = \text{Average}\,[y_i | \mathbf{x}_i \in R_m]$.
The main issue is how to determine the rectangular regions $\{R_m, m = 1, \ldots, M\}$.

A greedy top-down recursive partitioning algorithm is used. The model is fitted sequentially by performing a binary split involving a single input and can be represented as a tree.

- For any splitting variable j and a split point s we define the rectangles
$$R_1 = \{\mathbf{x} : x_j \le s\}, R_2 = \{\mathbf{x} : x_j > s\}.$$
  We select the variable X_j and the split point s which minimise the residual sum of squares
$$\sum_{\mathbf{x}_i \in R_1} (y_i - c_1)^2 + \sum_{\mathbf{x}_i \in R_2} (y_i - c_2)^2.$$
- Repeat the splitting process in each rectangle. Each node is split in two groups.

The subsets created by the splits are called nodes. The subsets which are not split are called terminal nodes. Terminal nodes are also known as leaves of the tree. To each leave there correspond a partition.

We aim at determining the optimal size of the tree.

Suppose there are T terminal nodes and denote by N_m the number of observations belonging to each leaf, and by

$$D_m(T) = \sum_{\mathbf{x}_i \in R_m} (y_i - \hat{c}_m)^2$$

the deviance of the residuals associated to the leaf.

$Q_m(T) = D_m(T)/N_m$ is often referred to as squared-error node impurity measure.

$D(T) = \sum_{m=1}^{T} D_m(T)$ is the deviance for a tree with T leaves.

Usually a large tree is grown, stopping the splitting process when a threshold value for $D_m(T)$ is reached, or N_m becomes very small. Let T_0 be the size of this tree.

For selecting a tree with size $T \leq T_0$, we minimize the cost complexity, defined as

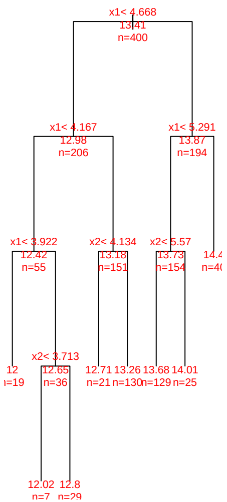
$$D(T) + \alpha T,$$

where the complexity parameter, $\alpha \geq 0$, regulates the trade-off between goodness of fit (as measured by the deviance) and the tree size.

For $\alpha = 0$ the solution is the full tree T_0. On the other hand, if $\alpha \to \infty$ the fit tends to the global mean $\bar{y}$ (no partition).

Estimation of α is done by (5 or 10-fold) crossvalidation. Given $\hat{\alpha}$, the corresponding T_α is obtained by weakest link pruning, i.e. we prune the leaves that produce the smallest increase in the deviance, when collapsed.

Figure : Regression tree, clothing dataset.



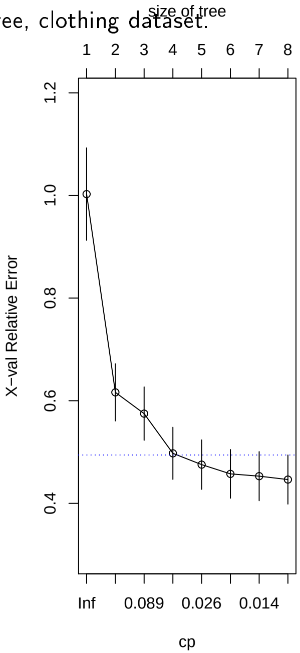
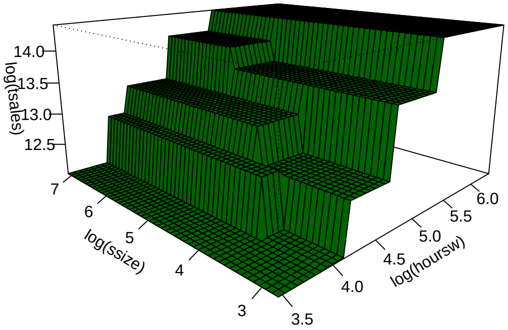


Figure : Regression tree, clothing dataset.



Classification Trees

Let Y denote a nominal variable and let Y_k denote the dummy variables for the k-th response category.

Given a set $\{R_m, m = 1, \dots, M\}$ of disjoint rectangles that partition the input space, we compute the mean of $Y_k, k = 1, \dots, K$, over the N_m units belonging to the partition; this is an estimate of $P(G = k|\mathbf{x} \in R_m) = p_{mk}$.

The units falling in this region will be classified according to the usual majority rule. Let κ be the modal class and let $\hat{p}_{m\kappa}$ be the associated fraction of cases.

The tree is grown using different measures of the value of a split.

Let $\hat{p}_{mk}$ denote the fraction of the observations in the region R_m belonging to class k.
Three measures of node impurity are popular:

- The missclassification error: $1 - \hat{p}_{m\kappa}$, i.e. the fraction of units missclassified in node m.
- The Gini index: $1 - \sum_{k=1}^{K} \hat{p}_{mk}^2$.
- Cross-entropy or deviance: $-\sum_{k=1}^{K} \hat{p}_{mk} \ln \hat{p}_{mk}$

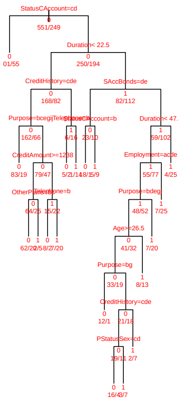
Both Gini and cross-entropy are measures of the *heterogeneity* of the probability distribution $\{\hat{p}_{mk}, k = 1, \ldots, K\}$.

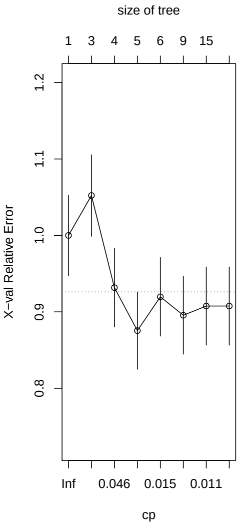
We look for the split that yields minimum heterogeneity.

Maximum heterogeneity corresponds to the case when $\hat{p}_{mk} = 1/K$ (the posterior probability are uniformly distributed - no majority vote), and the Gini index is equal to $1 - 1/K$ whereas cross-entropy equals $\ln K$.

On the contrary, heterogeneity is a minimum when $\hat{p}_{m\kappa} = 1$ and $\hat{p}_{mk} = 0, \forall k \neq \kappa$, in which case both indices are equal to 0.

Figure : Classification tree, German Credit dataset.





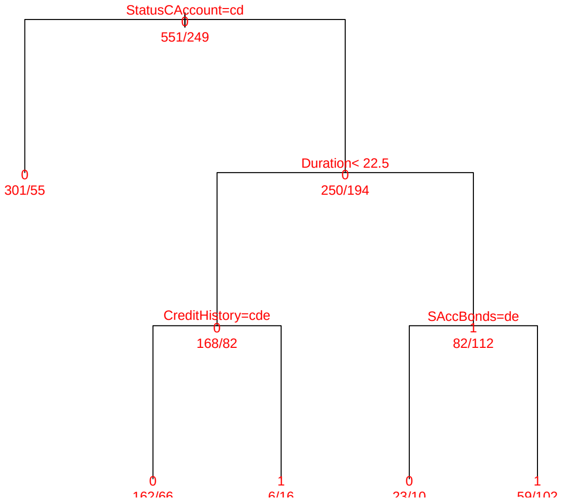


Figure : Pruned tree, Credit dataset.

Bagging

One serious limitation of trees is their volatility, which is related to the hierarchical structure of the splitting process. Small changes in the training sample produce a different sequence of splits. *Bagging*, which averages predictions over trees drawn from the same population, provides a solution, reducing the variance of the predictions.

A *bootstrap sample* is a sample of size N drawn with replacement from the training data $(y_i, \mathbf{x}_i)$.

Suppose B such samples are drawn independently. They can be used to assess the uncertainty of a method or model (parameter uncertainty, prediction uncertainty) by looking at the variability of the results.

In our case, however, interest lies in the prediction $\hat{f}(\mathbf{x})$ for a unit with input feature $\mathbf{x}$.

For each boostrap sample we estimate the model or apply the model and compute $\hat{f}_b(\mathbf{x}), b = 1, \ldots, B$.

The bagging estimate of $f(\mathbf{x})$ is

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

For linear methods (such as linear regression) bagging tends to the linear fit $\hat{f}(\mathbf{x})$ as B increases. Hence, it is of little use in those frameworks.

For regression trees, the bagged estimate is the average prediction from B trees.

For classification trees, we use bagging to estimate the probability that a unit with feature $\mathbf{x}$ belongs to group k, $\hat{p}_k(\mathbf{x})$, as the proportion of the B trees predicting class k.

Alternatively, we can average across the values $\hat{p}_{b,k}(\mathbf{x})$ computed on the bootstrap samples:

$$\hat{p}_{bag,k}(\mathbf{x}) = \frac{1}{B}\sum_{b=1}^{B}\hat{p}_{b,k}(\mathbf{x}).$$

See HTF, section 8.7 for more information and illustrations of this idea.

Boosting

Boosting originates as a method starting with a base classifiers and gradually improving its performance by refitting the data assigning higher weight to misclassified observations.

It is a learning algorithm that improves (boosts) the performance of a weak classifier $\hat{G}(\mathbf{x})$ (i.e. an unsophisticated method that would otherwise perform similarly to a random guess, like a tree with a single split). The classifier is applied sequentially to a modified data set which weighs more the missclassified units.

We start from the original AdaBoost algorithm.

Assume that there are two classes, labelled by $Y = \{-1, 1\}$ so that the output of the classifier is either -1 or 1 (rather than $\{0, 1\}$).

1. Initialization: set the observation weights $w_i = \frac{1}{N}, i = 1, \ldots, N$.
2. For $m = 1, \ldots, M$:
   2.1 Fit the classifier $\hat{G}_m(\mathbf{x})$ to the weighted training data.
   2.2 Compute the weighted missclassification rate
   $$\mathsf{err}_m = \frac{\sum_i w_i I(y_i \neq \hat{G}_m(\mathbf{x}_i))}{\sum_i w_i}$$
   2.3 Compute the weight
   $$\alpha_m = \ln \frac{1 - \mathsf{err}_m}{\mathsf{err}_m}$$
   2.4 Update the observation weights for the missclassified units:
   $$w_i = w_i \exp\left[\alpha_m I(y_i \neq \hat{G}_m(\mathbf{x}_i))\right]$$
3. Combine the predictions from the M classifiers through a weighed majority vote:
$$\hat{G}(\mathbf{x}) = \mathsf{sign}\left(\sum_m \alpha_m \hat{G}_m(\mathbf{x})\right)$$

Notice that as err_m decreases α_m increases and the w_i become more concentrated.

Boosting can be thought of fitting an additive model using a greedy forward stagewise approach.

At each iteration a new classifier $\hat{G}_m(\mathbf{x})$ enters additively, so that a particular loss function, the exponential loss function, is minimised. See HTF, ch. 10 for more details.

While bagging is a variance reduction technique, boosting can be thought of as a bias reduction technique.

The algorithm has been generalized by Friedman, who proposes a class of gradient boosting algorithms.

Gradient Boosting (GB)

More generally, boosting can be viewed as an additive (linear) combination of simple predictors (e.g. decision trees, or linear regression); an ”ensemble" method for obtaining predictors with an additive structure.

Typically, a base procedure (e.g. a tree) is fit to the residuals of the previous prediction. The residual is defined in terms of the derivative of a loss function.

Let $L(y, f(\mathbf{x}))$ be a loss function, e.g. if the output is quantitative, $L(y, f(\mathbf{x})) = (y - f(\mathbf{x}))^2$.

Given a training sample, our aim is to obtain $\hat{f}(x)$ as the minimiser of the empirical risk function

$$C(f) = \sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i)).$$

GB provides a solution to this optimisation problem.

Gradient Boosting Algorithm

1. Initialise $\hat{f}_0(\mathbf{x})$ as the minimiser of $C(f)$ when f is constant (e.g. set $\hat{f}_0(\mathbf{x}) = \bar{y}$).
2. For $m = 1, \ldots, M$, compute the negative **gradient**
$$g_i = -\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\bigg|_{f(\mathbf{x}_i)=\hat{f}_{m-1}(\mathbf{x}_i)}, \quad i = 1, \ldots, N.$$
3. Predict the negative gradient g_i from $\mathbf{x}_i$, using the base procedure. Let $\hat{g}_m(\mathbf{x}_i)$ denote the fitted value. This is an approximation to $\mathsf{E}(g_i|\mathbf{x}_i)$.
4. Update
$$\hat{f}_m(\mathbf{x}) = \hat{f}_{m-1}(\mathbf{x}) + \nu\hat{g}_m(\mathbf{x}_i),$$
where $0 < \nu \leq 1$ is a step-length factor, e.g. $\nu = 0.1$, a small number.
5. Iterate steps 2-4 until M is reached.

Notes:

- *Shrinkage*: reduce the complexity by regularizing the fit, using only a fraction ν (known as learning rate) of each new base learner (e.g. a tree) that is successively added. The fraction ν is used to shrink the contribution of each base learner (tree) as it is added to the model.
- The value of ν can also be determined by minimising $\sum_{i=1}^{N} L(y_i, \hat{f}_{m-1}(\mathbf{x}) + \nu \hat{g}_m(\mathbf{x}_i))$ with respect to ν.
- In *Stochastic Gradient Boosting* at each iteration $m = 1, \dots, M$, a subsample of the training data of size $\eta, 0 < \eta < 1$ is drawn at random (without replacement) from the training data set.
  The subsample is then used, instead of the full training data set, to fit the base learner (to grow the tree) for the current iteration.
- The optimal number of iterations of the steps 2-4 is determined by cross validation or by some information criterion.

GB in regression problems (quantitative output)

- When the loss function is $(y - f(x))^2$, $g_i = -2(y_i - \hat{f}_{m-1}(\mathbf{x}_i))$, and step 3 amounts to residual fitting.
- A base procedure for the regression case can be selecting the best variable in a simple linear regression model.
- The base learner amounts to fitting in every iteration the best predictor variable reducing the residual sum of squares most.
- The base learner is characterised by large bias, but small variance.
- Then, at every iteration one predictor variable is selected, and $\hat{g}_m(\mathbf{x}_i) = \hat{\beta}_{(m)} x_{i,(m)}$, where $x_{i,(m)}$ is the value for the variable selected at step m and $\hat{\beta}_{(m)}$ is the coefficient estimated from the regression of the residuals on that variable.
- As m goes to infinity $\hat{f}(\mathbf{x})$ converges to the full regression model fit.

- Regression trees are also a very popular choice for the base learner.
- When applied to regression trees, at each iteration m, a regression tree partitions the input space into L-disjoint regions $\{R_{lm}, l = 1, \dots, L\}$ and predicts a separate constant value in each one.
- The parameters regulating the fit are the size of the trees T_m (tree complexity) and the number M of boosting iterations.
- Usually T_m is fixed for all m.
- Too large an M exposes to the dangers of overfitting. M is usually estimated by plotting the test error versus the boosting iterations.
- Typically grow small trees: the base procedure has to have small variance at the price of large estimation bias.

GB for classification (qualitative outputs)

In classification, $f(\mathbf{x})$ is the log-odds-ratio

$$f(\mathbf{x}) = \ln \frac{p(\mathbf{x})}{1 - p(\mathbf{x})}$$

and a candidate loss function (it is differentiable!) is the deviance:

$$L(y, f(\mathbf{x})) = -2[y \ln p(\mathbf{x}) + (1 - y) \ln(1 - p(\mathbf{x}))].$$

The 0-1 loss cannot be used as it is not differentiable.
If we transform the output variable, $y \to \tilde{y} = 2y - 1$, so that it takes the values $\{-1, +1\}$, and set

$$f(\mathbf{x}) = \frac{1}{2} \ln \frac{p(\mathbf{x})}{1 - p(\mathbf{x})},$$

(one-half the log-odds ratio), then

$$L(\tilde{y}, f(\mathbf{x})) = \ln[1 + \exp(-2\tilde{y} f(\mathbf{x}))].$$

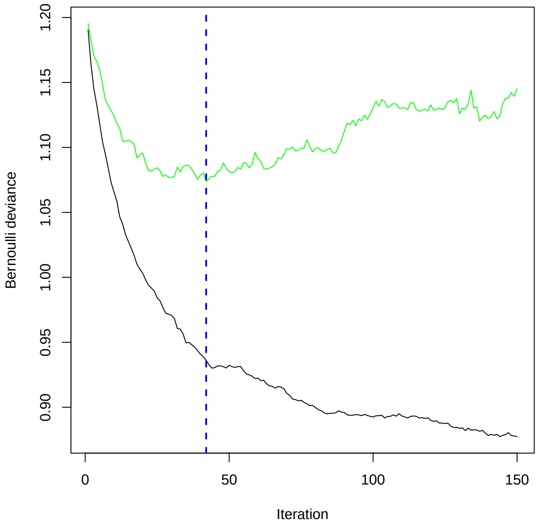
Under this setting, GB is known as LogitBoost or BinomialBoosting.

Software

- In R (Rstudio)
  - The `gbm` package implements Friedman's tree-based GB for regression and classification.
  - The package `GAM-Boost` implements boosting for spline-based additive models.
  - The package `mboost` implements L2-Boosting and BinomialBoosting.
- In SAS Enterprise Miner, the Gradient Boosting Node implements Friedman's tree-based GB for regression and classification.

The various implementation differ wrt the base procedure (tree, regression, logistic regression, splines) and the random resampling of the training set at each iteration.

Figure : German credit data: gradient boosting - plot of performance versus number of trees for the train and the test sample.



Random forests

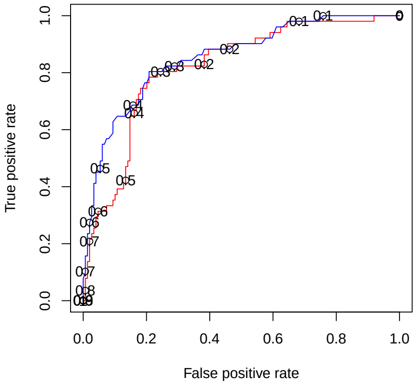
In some situations both N (number of observations) and p (number of inputs) are very large.

Several decision trees are grown by selecting a random subsample of both the units (with replacement) and of the inputs.

Each decision tree is built to its maximum size, (no pruning).

The trees are combined into a single classifier by averaging the individual classifiers.

Figure : ROC curves for gradient boosting tree and Random forests (blue).



Variable importance

There are two essential ways of measuring the relative importance of each input variable in predicting the response.

The *split-based variable importance* measure is based on the number of times a variable is selected for splitting, weighted by the squared improvement in the impurity measure as a result of each split, and averaged over all trees. This is subsequently scaled so that the sum adds to 100, with higher numbers indicating stronger influence on the response.

Partial Dependence plots are obtained by integrating out the effects of other predictors to show the partial dependence of the fit on the predictors of interest.

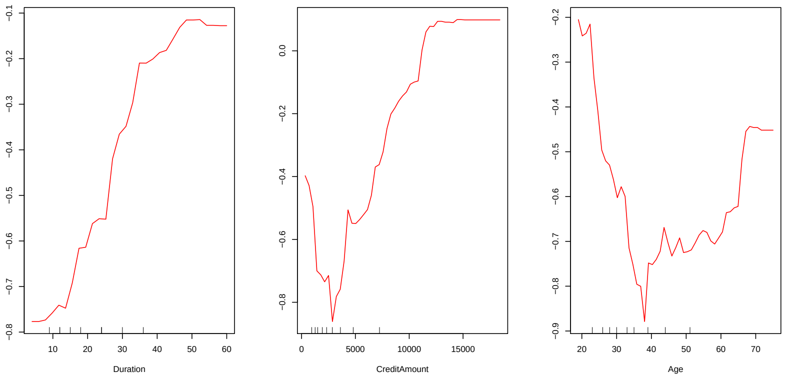


Figure : Random forests. Partial dependence plots (logits) for `Duration`, `CreditAmount` and `Age`.

The receiver operating characteristic (ROC) curve

Consider the confusion matrix:

	$\hat{G}(X)$ (prediction outcome)	
G (actual value)	0	1
0	True negative (TN)	False positive (FP)
1	False negative (FN)	True positive (TP)

The true positive rate (TPR) is defined as

$$P(\hat{G}(X) = 1 | G = 1) = TPR = \frac{TP}{TP + FN}$$

this is also referred to as the sensitivity rate.
The false positive rate (FPR) is defined as

$$P(\hat{G}(X) = 1 | G = 0) = FPR = \frac{FP}{TN + FP}$$

The specificity rate is $P(\hat{G}(X) = 0 | G = 0) = \frac{TN}{TN+FP}$;

- The **ROC curve** displays the true positive rate (TPR, aka sensitivity) on the vertical axis and false positive rate (FPR, 1-specificity) on the horizontal axis.
- The point (0,1) is the perfect classification point. The 45 degrees line is the line of no discrimination (random guess). Below the line the classifier performs worse than a pure random guess.
- ROC is used to illustrate the trade-off between TPR and FPR.
- In logistic regression $\hat{p}_i, y_i$ are sorted according to the values of $\hat{p}_i$ from the largest to the smallest. For each threshold p from 1 to 0 we compute the TPR and FPR when the classifier allocates to Group 1 if $\hat{p}_i > p$.
- The **area under the curve** (AUC) is often used as a measure of accuracy.
  AUC is 0.5 for a random classifier. Its maximum is 1.