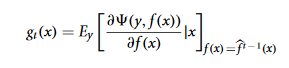
## GRADIENT BOOST ALGORITHM

One can arbitrarily specify both the loss function and the base learner models on demand. In practice, given some specific loss function (*y*,*f*) and/or a custom base-learner *h*(*x*,θ), the solution to the parameter estimates can be difficult to obtain. To deal with this, it was proposed to choose a new function *h*(*x*,θ*t*) to be the most parallel to the negative gradient along the observed data:



Instead of looking for the general solution for the boost increment in the function space, one can simply choose the new function increment to be the most correlated with −*gt*(*x*). This permits the replacement of a potentially very hard optimization task with the classic least-squares minimization one:



To summarize, we can formulate the complete form of the gradient boosting algorithm, as originally proposed by Friedman (2001). The exact form of the derived algorithm with all the corresponding formulas will heavily depend on the design. To design a particular GBM for a given task, one has to provide the choices of functional parameters (*y*,*f*) and *h*(*x*,θ). In other words, one has to specify what one is actually going to optimize, and afterwards, to choose the form of the function, which will be used in building the solution. It is clear that these choices would greatly affect the GBM model properties. The GBM framework provides the practitioner with such design flexibility.

## LOSS-FUNCTION FAMILIES

Given a particular learning task, one can consider different loss functions (*y*,*f*) to exploit. This choice is often influenced by the demand of specific characteristics of the conditional distribution. The most frequent examples of such property is the robustness to outliers, but other opportunities can also be considered.

To use an arbitrary loss function, one has to specify both the loss function and the function to calculate the corresponding negative gradient. Given these two functions, they can be directly substituted into the GBM algorithm. In practice, many of the loss functions have already been derived for the GBM algorithm.

Loss-functions can be classified according to the type of response variable *y*. Specific boosting algorithms have been derived for various families of the response, among which are the regression, classification and time-to-event analysis tasks. Depending on the family of response variable *y* we can systemize the most frequently used loss-functions as follows:

1. Continuous response, *y* ∈ *R*:
   * Gaussian *L*2 loss function
   * Laplace *L*1 loss function
   * Huber loss function, δ specified
   * Quantile loss function, α specified
2. Categorical response, *y* ∈ {0,1}:
   * Binomial loss function
   * Adaboost loss function
3. Other families of response variable:
   * Loss functions for survival models
   * Loss functions counts data
   * Custom loss functions

To provide a better insight into the model design, we will describe the loss-functions for continuous and categorical response variables in more detail. Specific GBM algorithms have also been derived for other types of response like the Poissoncounts and the survival data, but we will not address these models in this paper.

## Loss functions for continuous response

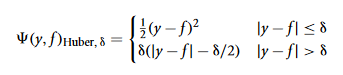
When the response variable *y* is continuous, a regression task is solved. A classic loss function, which is commonly used in practice is the squared-error *L*2 loss:

In the case of the *L*2 loss-function, its derivative is the residual *y* − *f* , which implies that the GBM algorithm simply performs residual refitting. The idea behind this loss function is to penalize large deviations from the target outputs while neglecting small residuals. The illustration of this loss function is provided on **Figure 1A**.

Another example is the absolute *L*1-loss, denoted as the “Laplacian” loss function. The *L*1-loss corresponds to the median of the conditional distribution, thus considered as the robust regression loss. The *L*1 loss function takes the form:

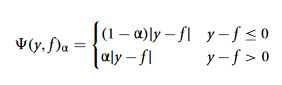
It may be of particular interest in tasks where the response variable has long-tail error distribution. The function is illustrated on **Figure 1B**.

One can also exploit the parameterized loss-functions as well. A robust regression alternative to the *L*1 loss is the Huber loss function. It comprises two parts, corresponding to the *L*2 and *L*1 losses. The Huber loss is designed as follows:



The cutting edge parameter δ is used to specify the robustification effect of the loss-function. The intuition behind this parameter is to specify the maximum value of error, after which the *L*1 loss has to be applied. The Huber loss function is illustrated on **Figure 1C**.

A more general approach is based on predicting a conditional quantile of the response variable (Koenker and Hallock, 2001). This approach is distribution free and in general proves to provide good robustness to outliers. The quantile loss is organized as follows:



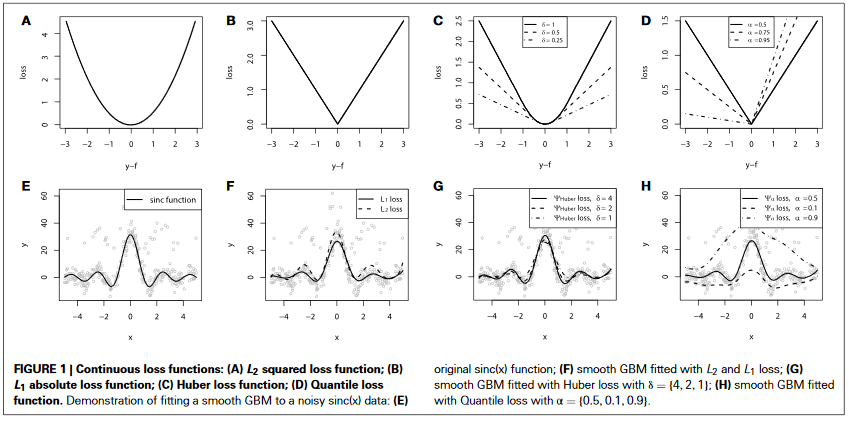
The parameter α in this case specifies the desired quantile of the conditional distribution. One can note that when α = 0.5, this would coincide with the *L*1 loss, thus resulting in the conditional median. Different parameterizations of the quantile loss function are illustrated on **Figure 1D**.

To demonstrate the properties of the described loss functions we will consider an artificially generated dataset. The dataset is sampled from a sinc(*x*) function with two sources of artificially simulated noise: the gaussian noise component ε ∼ *N*(0,σ2) and the impulsive noise component ξ ∼ Bern(*p*). The impulsive noise term is added to illustrate the robustification effects. The generated dataset is illustrated on **Figure 1E**.

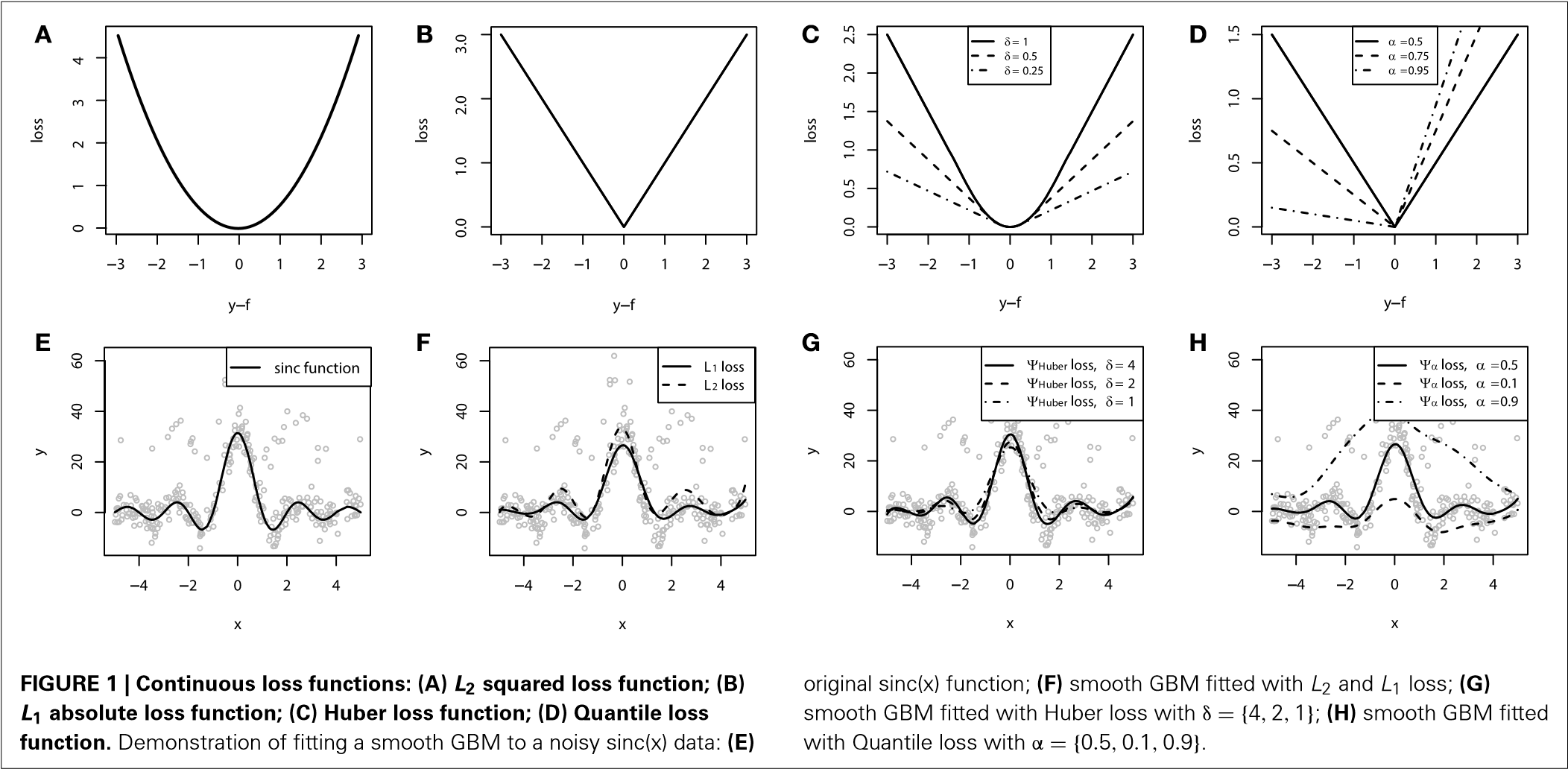
To keep the experiment focused on the loss-function specifics we would assume that the learning was done in an optimal way. In this experiment the base-learner functions applied were the P-splines. The resulting GBM models of this experiment are presented on **Figures 1F–H**.

One can note that the median of the distribution is less affected by the impulsive noise whereas the *L*2 loss function is slightly biased due to the caused deviations. The quantile losses in their turn give a good estimation of the corresponding conditional distribution quantiles.

Following the idea of applying various loss-functions, one can for example model the conditional box-plots. From the computational perspective, this type of modeling would only result in increasing the number of different GBM models built by the number of desired statistics of the conditional distribution. However, it must be kept in mind that the resulting confidence intervals are a model approximation rather than true statistics. It is also important to note that the learned quantile models do not have to be necessary consistent with each other, as they are all learned separately.



## 3.1.2. Loss functions for categorical response

In the case of categorical response, the response variable *y* typically takes on binary values *y* ∈ {0,1}, thus, assuming that it comes from the Bernoulli distribution. To simplify the notation, let us assume the transformed labels *y*¯, putting *y*¯ = 2*y* − 1 and making *y*¯ ∈ {−1,1}. In this case, the probability of class-wise response can be estimated by minimizing the negative loglikelihood, associated with the new class labels:

(*y*,*f*)Bern = log(1 + exp(−2*yf*¯ )) (19)

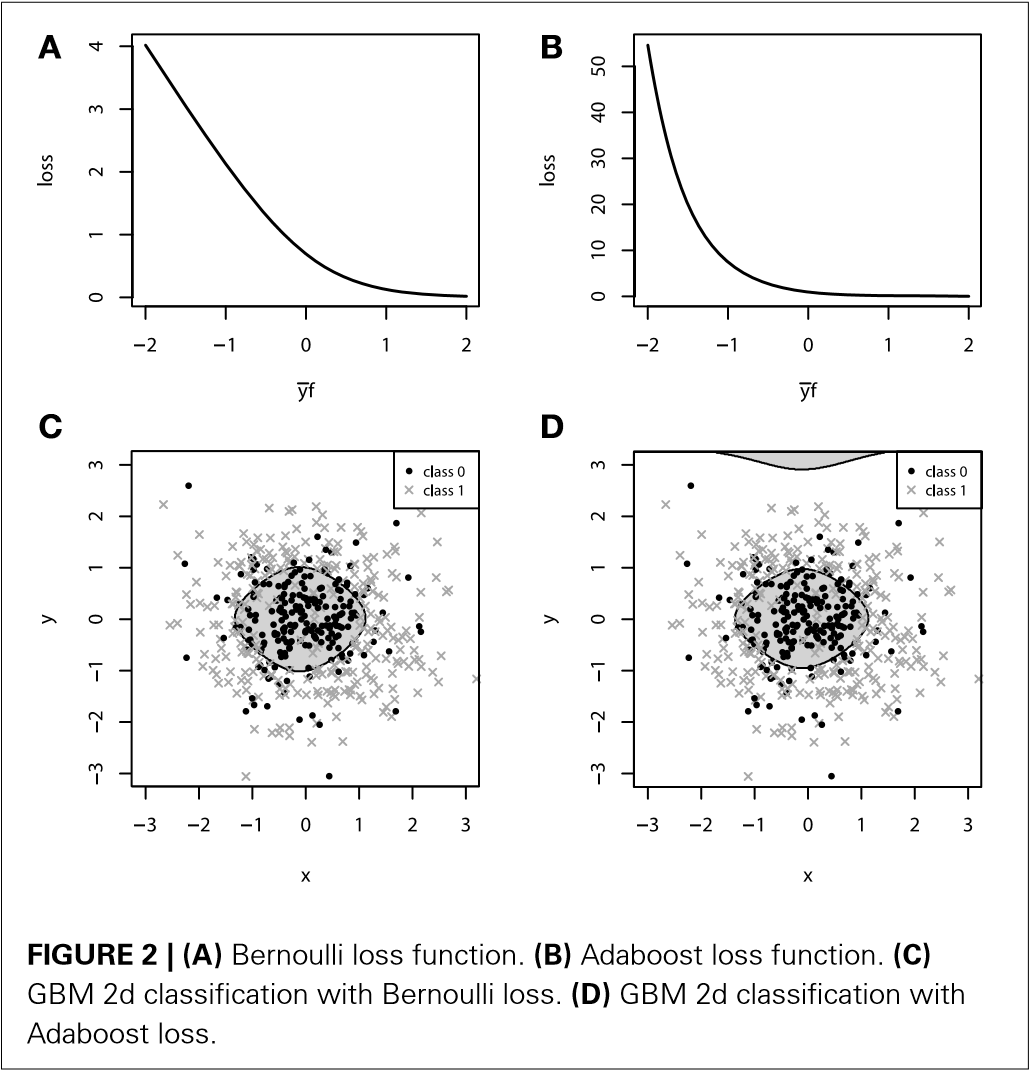
This loss function is commonly referred to as the Bernoulli loss. The illustration of the Bernoulli loss function is given on **Figure 2A**. The chart shows the loss function defined over the values of *yf*¯ . Please note, that in this notation, positive values of *yf*¯ correspond to the correct discrimination.

Another common choice of categorical loss-function is the simple exponential loss, as it is used in the Adaboost algorithm (Schapire, 2002). Following the same notation as in the Bernoulli loss, the Adaboost loss function is therefore defined as:

(*y*,*f*)Ada = exp(−*yf*¯ ) (20)

It is possible to establish a connection between the influence trimming of GBMs with the Adaboost loss function and weight trimming Adaboost algorithm (Friedman, 2001). The illustration of the Adaboost loss is given on **Figure 2B**. The notation for this loss-function chart is the same as we used in the figure with Bernoulli loss.

To demonstrate the properties of the categorical loss functions we will construct another artificial dataset. Originally, all the data comes from a 2-dimensional normal distribution, with zero mean and identity-covariance matrix. The points that lie within the inner circle of unit radius *r* = 1 belong to one class and are colored with black, whereas all the other points are assigned to another class, colored with dark gray. For this setting, we use two sources of noise: 2-dimensional gaussian noise (ε1,ε2),ε*i* ∼ *N*(0,0.32) and a random misclassification error ξ, which randomly switches the class. The random misclassification results



in slightly heavier tails of the distances of class-error distributions and thus, will allow us to contrast the difference between loss functions. The resulting dataset together with the marginal density plots of each class is presented on **Figures 2C,D**.

For both of the inspected models on **Figure 2**, the model complexity was chosen equally in terms of the number of boosting iterations *M*. It is, therefore, interesting to note that the models achieved similar accuracy, with equal confusion matrices. However, despite these two similarities, geometrically these models are considerably different. Due to the fact that the exponential loss of Adaboost model contrasts misclassified points much more, the corresponding model began capturing the boundary, “far”-outlying points, much earlier than the other model.

In the context of loss-functions, we say “much earlier” because it is true that at some point in the learning process we can overestimate the model-complexity and thus overfit the data with both types of loss functions. However, due to the nearly linear impact of outliers to the Bernoulli loss, the Bernoulli model is typically less-sensitive to this type of erroneous labeling in the data.

### 3.2. SPECIFYING THE BASE-LEARNERS

A particular GBM can be designed with different base-learner models on board. A diverse set of base-learners have been introduced in the literature thus far. In this subsection, we will briefly describe and illustrate the base-learner models that are most frequently used in practice.

The commonly used base-learner models can be classified into a three distinct categories: linear models, smooth models and decision trees. There is also a number of other models, such as markov random fields (Dietterich et al., 2004) or wavelets (Viola and Jones, 2001), but their application arises for relatively specific practical tasks. The base-learner model systematization with the corresponding examples of functions is organized as follows:

1. Linear models:
   * Ordinary linear regression
   * Ridge penalized linear regression
   * Random effects
2. Smooth models:
   * P-splines
   * Radial basis functions
3. Decision trees
   * Decision tree stumps
   * Decision trees with arbitrary interaction depth
4. Other models:
   * Markov Random Fields
   * Wavelets
   * Custom base-learner functions

An important design opportunity is that nothing prevents the practitioner from specifying a complex model, utilizing several classes of base-learner models in one GBM. This means that the same functional formula can include both smooth additive components and the decision trees components at the same time. Another example would be to split the explanatory variables into the categorical and smooth subspaces and fit different boosted base-learner models to each of the subspaces simultaneously.

Another important feature of the base-learner specification is that they can be designed for different models of variable interactions. If we consider the ANOVA decomposition of the function different interrelationships between explanatory variables: estimate *f*, different interaction terms would correspond to the

*f*(21)

*j jk jkl*

## 3.2.1. Additive base-learners

Using the additive base-learner models explicitly assumes that there is no interaction between the explanatory variables. Yet, there has been mounting empirical evidence that for most practical tasks, simple additive models corresponding to the first term of the ANOVA decomposition, provide considerably accurate results (Schapire, 2002; Wenxin, 2002). Another important observation is that the resulting additive models are interpretable by design, allowing the practitioner to investigate each of the model components separately.

The learning algorithm for additive GBM models slightly differs from the algorithm we described earlier. At each iteration, several additive base-learner candidates, built atop some randomly chosen variables, are fitted simultaneously. Next, the best of these models is chosen, based on the residual sum of squares criterion. One property of this learning process is that it often leads to the situation, when many of the explanatory variables are omitted, thus, naturally leading to a sparser solution.

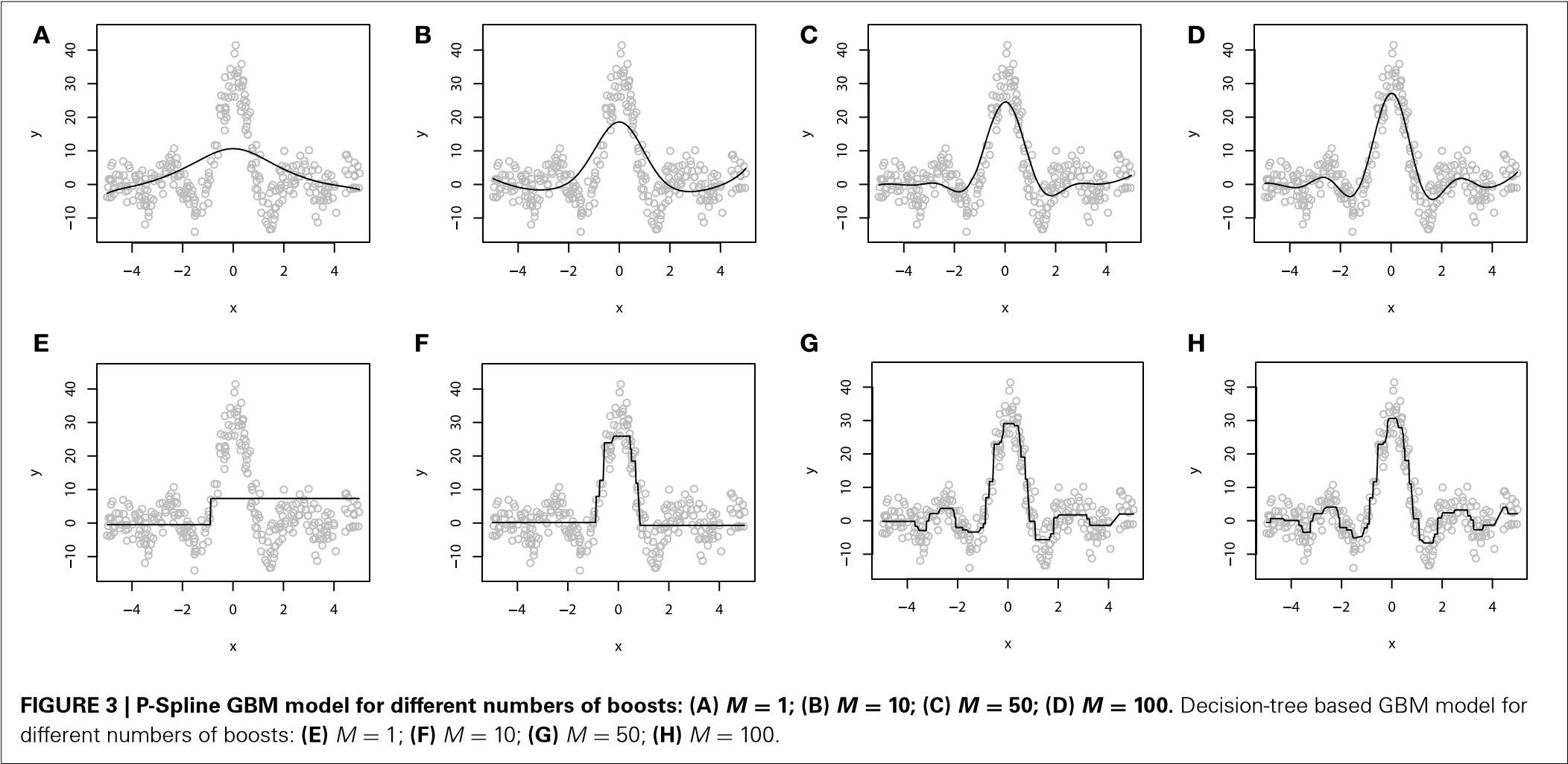
Motivation for using the additive representation with linear and generalized linear models (GLM) instead of the common GLM model with some penalty, is particularly based on the desire to fit a sparse model. This becomes especially important in tasks with many categorical variables, like the data that comes from medical and biological experiments. The resulting gradient boosting fitting leads to a relatively easy variable-selection procedure by design.

The choice of boosting the additive models is also sometimes dictated by the computational considerations. Consider using the spline base-learner functions for boosting the generalized additive model (GAM). In order to fit a multivariate spline model with respect to interactions, the number of knots in the spline grid will grow exponentially with the number of variables.

To illustrate the additive GBM model with additive baselearners, we again refer to the artificial dataset, simulated from sinc(*x*) function. For demonstrative purposes, we will omit the impulsive noise component and choose the *L*2 loss and use the smooth spline base-learner functions. To provide a better intuition into the process of fitting a smooth additive boosting models, the resulting fits are evaluated for different number of boosting iterations *M*. The resulting demonstration is presented on **Figures 3A–D**.

When *M* = 1, we obtain a single penalized spline model, partially fitting the central part of the wave function. When we increase the number of iterations *M*, the accuracy of the fitted model grows gradually until the function is fitted considerably well. More details on the properties of GLM and GAM boosting models can be found in Buhlmann (2006) and Schmid and Hothorn (2007).

## 3.2.2. Decision tree base-learners

A computationally-feasible way of capturing interactions between variables in GBM models is based on using the decision tree models. Although interactions between several explanatory variables would remove the interpretability property of additive models, this can not be considered a significant drawback as there are still several tools for tree-based GBM interpretation.

The idea behind a decision tree is to partition the space of input variables into homogenous rectangle areas by a tree-based rule system. Each tree split corresponds to an if-then rule over some input variable. This structure of a decision tree naturally encodes and models the interactions between predictor variables. These trees are commonly parameterized with the number of splits, or equivalently, the *interaction depth*. It is also possible to have one of the variables be split in a particular several times.

A special case of a decision tree with only one split (i.e., a tree with two terminal nodes) is called a tree stump. Therefore, if one wants to fit an additive model with tree base-learners, it is possible to do this using the tree stumps. In many practical applications small trees and tree-stumps provide considerably accurate results (Wenxin, 2002). Moreover, there is much evidence that even complex models with rich tree structure (*interaction depth* > 20) provide almost no benefit over compact trees (*interaction depth* ≈ 5).

One important property of the decision trees is that by design, a single decision tree always extrapolates the function with the constant value. An implication of this is that even a simple function like a straight line with a non-zero angle can not be approximated correctly with a single decision tree.

To demonstrate a GBM designed with the decision tree baselearners, we will use the same sinc(*x*) dataset as we used to illustrate the additive models. For this experiment, we also used the *L*2 loss. As the dimension of the explanatory variables is equal to one, we chose to use the tree-stumps. The resulting fitted models are shown on **Figures 3E–H**).

To demonstrate the progress of the fitting procedure, the number of iterations *M* was varied from 1 to 500. The similar behavior of consecutive improvements in the fit accuracy, when the number of iterations *M* increases, is apparent on this chart.

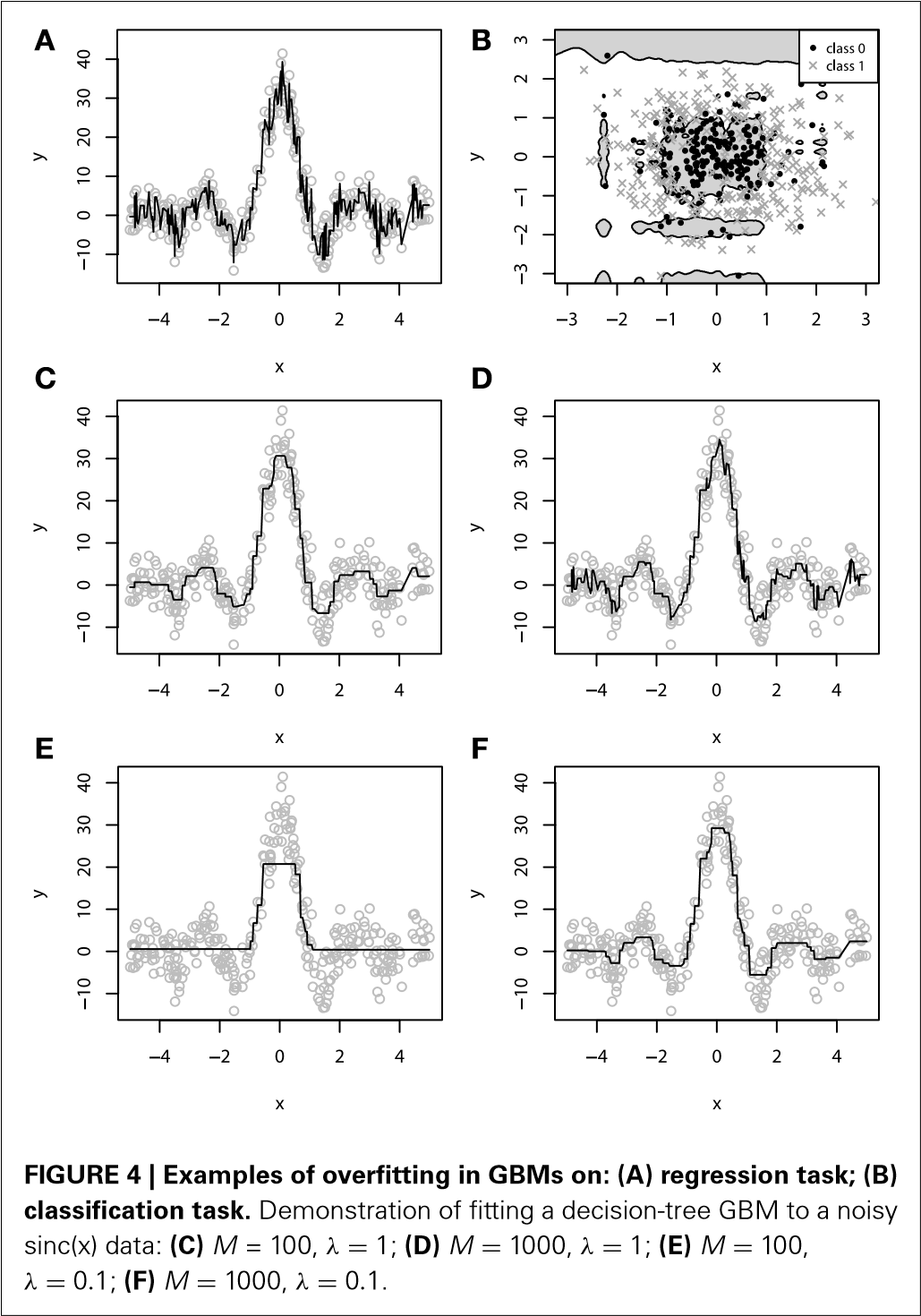
To conclude this section we must note that although there is a wide variability of possible design options, in most practical tasks one doesn’t have to exhaustively try every possible combination of them. The choice of the loss function is often a matter of a particular task, whether to make the model more robust or not. Therefore, we advise the practitioners to first try fitting their models with classical loss functions, i.e., *L*2 loss for regression and Bernoulli loss for classification. As for the base-learner model, we would recommend to first try using tree stumps or lowinteraction trees, because they usually perform reasonably well on many real-world datasets.

# 4. REGULARIZATION

The most important concern about building a machine-learning model from data is the resulting model’s generalization capabilities. If the learning algorithm is not applied properly, the model can easily overfit the data. This means that it will predict the training data itself rather than the functional dependence between input and response variables. These concerns are obviously the same for GBMs.

It is easy to imagine a situation where new base-learners are added to the ensemble until the data is completely overfitted. Overfitting a GBM is possible with different types of base-learners with very different loss-functions. On **Figures 4A,B**, we illustrate overfitting for both regression and classification tasks.

To decrease the overfitting effects in GBMs, a number of different approaches were introduced. They help to constrain the



fitting procedure and thus balance the predictive performance of the resulting model (Sutton, 2005; Zhang and Yu, 2005; Zou and Hastie, 2005). In this section, we will describe the most efficient regularization techniques that are most frequently used in GBMs.

## 4.1. SUBSAMPLING

The simplest of the regularization procedures introduced for GBMs is subsampling. The subsampling procedure has shown to improve the generalization properties of the model, at the same time reducing the required computation efforts (Sutton, 2005).

The idea behind this method is to introduce some randomness into the fitting procedure. At each learning iteration only a random part of the training data is used to fit a consecutive base-learner. The training data is typically sampled without replacement, however, replacement sampling, just as it is done in bootstrapping, is yet another possible design choice.

The subsampling procedure requires a parameter called the “bag fraction.” Bag fraction is a positive value not greater than one, which specifies the ratio of the data to be used at each iteration. For example, *bag* = 0.1 corresponds to sampling and using only 10% of the data at each iteration. Another useful property of the subsampling is that it naturally adapts the GBM learning procedures to large datasets when there is no reason to use all the potentially enormous amounts of data at once.

When the amount of data, measured by the number of data points *N* is not of practical concern, setting the default value *bag* = 0.5 gives a reasonable result for many practical tasks. If an optimal bag fraction is of interest, one can simply estimate it by comparing predictive performance under different parameter values.

However, one should also consider the effect of reducing the sample size on the model estimates. If the number of points becomes too low, one might receive a poorly fit model due to the lack of degrees of freedom. Therefore, some basic sanity-check analysis is essential before reducing the sample size.

It is also important to note the “big data” argument, as a consequence of the sample size reduction. In general, the more data there is available for the fitting a base-learner, the more accurate will the estimate be, if sufficient data was used. Therefore, when there are large amounts of data, one may consider a trade-off between the number of points, used for fitting each of the baselearners and the accuracy improvement, achieved by each of the base-learners.

One can easily arrive at a situation, when it is more efficient to have a larger number of base-learners, learnt with the lower *bag* rate. This means that the GBM ensemble will reach the desired accuracy with a larger number of base-learners and lower *bag* than the one with smaller amount of more carefully fitted base-learners with larger *bag*.

## 4.2. SHRINKAGE

The classic approach to controlling the model complexity is the introduction of the regularization through shrinkage. Shrinkage is commonly used in ridge regression where it literally shrinks regression coefficients to zero and, thus, reduces the impact of potentially unstable regression coefficients.

In the context of GBMs, shrinkage is used for reducing, or shrinking, the impact of each additional fitted base-learner. It reduces the size of incremental steps and thus penalizes the importance of each consecutive iteration. The intuition behind this technique is that it is better to improve a model by taking many small steps than by taking fewer large steps. If one of the boosting iterations turns out to be erroneous, its negative impact can be easily corrected in subsequent steps.

The simplest form of regularization through shrinkage is the direct proportional shrinkage (Friedman, 2001; Hothorn et al. 2010). In this case the effect of shrinkage is directly defined as the parameter λ ∈ (0,1]. The regularization is applied to the final step in the gradient boosting algorithm:



*ft* ← *ft* −1 + λρ*th*(*x*,θ*t*) (22)

It is a common pattern that the smaller parameter λ and therefore, the lower the shrinked boosted increments are, the better generalization is achieved. But, the cost of improving the generalization properties is the convergence speed. Choosing a stronger value of λ will increase the number of iterations *M*, required for convergence to a similar empirical loss minimum. For example, a decrease in λ by a factor of 10 implies an increase in the number of iterations *M* by a similar factor, slightly higher than 10.

An example of exploiting the shrinkage regularization is illustrated on **Figures 4C–F**. For this demonstration we used the *L*2 loss and the decision-tree base-learners. We didn’t separate the dataset into training and validation set, because we wanted to show the geometric effects of shrinkage.

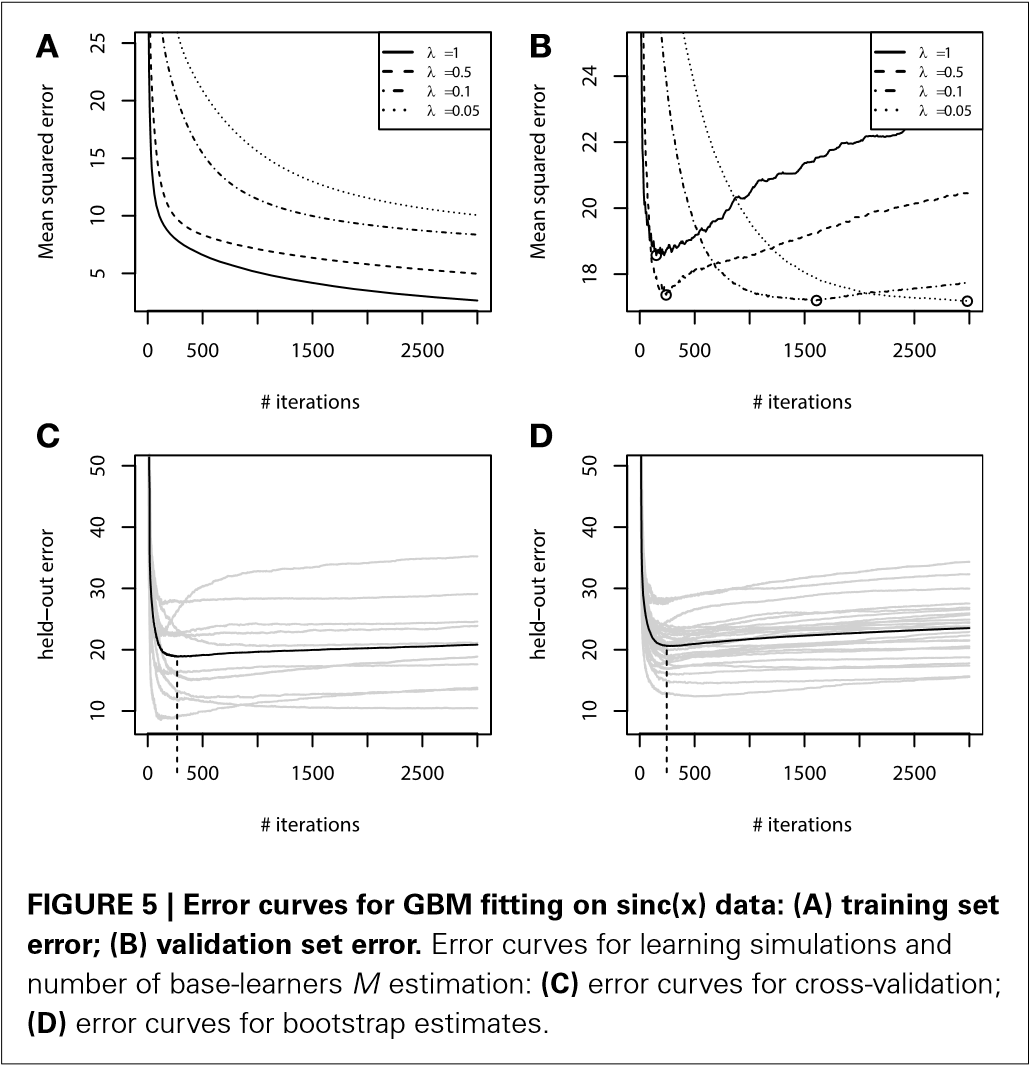
From **Figure 4** we can deduce some interesting patterns. First of all, decreasing the shrinkage parameter requires more iterations to achieve the accuracy, compared to the non-regularized learning. Besides, we can see that using shrinkage results in capturing more details as it relies on a larger amount of boosts and thus, provides more continuity. This is especially important for decision-trees because, as previously discussed, they are very limited to capturing details by design.

Exploiting shrinkage in learning allows the decision-tree GBMs to capture more continuity in the modeled effects. The same effect of smoothing the decision-tree ensemble would also hold true for higher dimensional data, and that’s why authors claim that it is desirable to train GBMs with infinitesimal stepsizes (Friedman, 2001; Buhlmann, 2006).

Analyzing **Figure 4**, one can note the effect of overfitting on the **Figure 4D**. This naturally leads us to the question of how does the shrinkage affect overfitting, or in the case of GBM, how does it affect the dependence between the learning error and the number of iterations.

To investigate this question in more detail, let us now consider the fitting experiment with both training and validation sets. Of the 300 initial points, we use randomly resampled 200 of them for training, and another 100 for validation. All the other experiment parameters remain unchanged. The learning error curves for GBMs with different λ parameters are presented on **Figure 5**.

From **Figure 5A** we can see that the training set error is substantially falling, but the speed of this improvement heavily depends on the shrinkage parameter λ. A much more important



effect from a practical point of view is the validation set error behavior, which is shown on **Figure 5B**. The validation-error hyperparameter *M*, corresponding to the error minima of each of the models, is highlighted with circles. We can see that increasing the shrinkage leads to both finding a better hyperparameter *M* minima and to improving the generalization of the model. The latter corresponds to the fact that shrinked models have a flatter plateau beyond their error minimas, and it takes them many more iterations to initiate overfitting. Yet, it also means that these models will naturally take longer to learn.

## 4.3. EARLY STOPPING

Using regularization techniques described above, one can significantly improve the generalization properties of a GBM model. However, given a shrinkage parameter λ, the optimal number of iterations *M*opt, in the sense of the validation set performance, can be different from the initially pre-specified one *M*. We have illustrated this phenomenon on the **Figure 5**.

Once important practical consideration that can be derived from **Figure 5** is that one can greatly benefit from early stopping (Zhang and Yu, 2005). This means that if the ensemble was trimmed by the number of trees, corresponding to the validation set minima on the error curve, the overfitting would be circumvented at the minimal accuracy expense. Another observation is that the optimal number of boosts, at which the early stopping is considered, varies with respect to the shrinkage parameter λ. Therefore, a trade-off between the number of boosts and λ should be considered.

In practice one typically chooses the shrinkage parameter λ beforehand and varies the number of iterations *M* with respect to the chosen shrinkage. One possible approach to choosing the number of iterations *M* would be to use an information criterion like Akaike’s AIC or some sort of minimum description length criteria. However, they have been shown to overshoot the true number of iterations (Hastie, 2007) and thus are not recommended for practical usage.

The most frequently used approach to deal with this trade-off relies on the cross-validation procedure. Cross-validation provides means for testing the model on withheld portions of data, while still using all of the data at the processing stages.

First, the shrinkage parameter λ, the maximum number of iterations *M*max and the cross-validation parameter *k*, corresponding to the number of validation folds, are specified. The data is then partitioned into *k* disjoint non-overlapping subsets. Afterwards, for each of the *k* subsets of the data, one of them is set aside as the validation set and the others are used for fitting a GBM model. The fitted GBM is then tested on the validation set to produce the held-out estimates of the predictive performance. At last, the validation performance is aggregated from each of the folds, for example, by averaging the validation set performances. This aggregated measure serves as estimate of model generalization on the validation set.

One may note that besides cross-validation one can use a similar procedure to test the model on bootstrap samples (Hofner et al., 2012). Bootstrap is essentially useful for parameter estimation when the training dataset is considerably small. In bootstrapping, we choose the number of bootstrap samples *B* similarly to the number of folds *k* in cross-validation. Afterwards, at each iteration we randomly sample with replacement the original data, which leads to approximately 63% of the unique original data entries in each sample. This means that if we had a sample of {1,2,3}, the resulting bootstrap samples can, for example, be {1,1,3} or {3,2,2}. The held-out estimates are evaluated on the left-out original data entries, the so-called “out of the bag” values. These values are then aggregated in the same fashion, as in the cross-validation.

The results of this procedure are illustrated in **Figures 5C,D**. For this experiment we used the same parameter setting as in all the other regularization experiments, with the same training and validation sets. For the hyperparameter specification we chose λ = 0.5, *M*max = 3000, *k* = 10 folds for cross-validations and *B* = 25 for boostsrapping.

As we can see from the simulation plots, the average behavior of the held-out errors is rather similar. And from both of these plots we can deduce very similar estimates of the optimal number of iterations *M*. Namely, the cross-validation estimate is *M*optCV = 255, the bootstrap estimate is *M*optboot = 241, while the optimum of the validation set was *M*opt = 245. It means that both methods provide us with considerably good estimates of the number of iterations.

# 5. MODEL INTERPRETATION

In practice, it can be of great utility to be able to interpret the resulting model. As we have previously discussed, additive GBM models can be trivially explained, as the additive components correspond to the marginal dependence plots by design. One only has to predict each additive component over a grid of values of the corresponding variable and plot it.

When one uses an ensemble of decision trees with high interaction depth, the same visualization approach is inapplicable.

And despite the simplicity of a simple decision tree, when there are thousands of trees in the ensemble it becomes challenging to interpret such models. However, even decision tree GBMs can be interpreted with the appropriate tools.

Several tools have been designed to alleviate interpretation problems in decision-tree based GBMs. Therefore, even high interaction-based GBMs should not be considered completely black boxes, as the resulting models can provide important insights into the captured dependencies. In this section we describe the most common tools for GBM interpretation.

## 5.1. RELATIVE VARIABLE INFLUENCE

A common practical task is to identify the variable importance. To perform feature selection in decision-tree ensembles the main modeled effects are not separated from the effects caused by interactions. Therefore, one cannot strictly analyze the captured effects in a similar fashion to the regression coefficients. For this purpose, the variable influence for the decision tree ensembles, based on the decision trees influences(Breiman et al., 1983), was proposed (Friedman, 2001).

If we consider a likelihood framework of GBMs, and for simplicity assume the *L*2 loss, it follows that the increase in log likelihood is proportional to the increase in sums of squares explained by the model. Each split on a variable in a decision tree increases the log likelihood of the whole ensemble and the sum of log likelihood increases across all trees.

Let us define the influence of the variable *j* in a single tree *T*. Consider that the tree has *L* splits, therefore we are looking for all the non-terminal nodes from the root to the *L* − 1 level of the tree. This gives rise to the definition of the variable influence:

*L*−1

Influence*j* *j*) (23)

*i*=1

This measure is based on the number of times a variable is selected for splitting, i.e., current splitting variable *Si* is the same as the queried variable *j*. The measure also captures weights of the influence with the empirical squared improvement *Ii*2, assigned to the model as a result of this split. To obtain the overall influence of the variable *j* in the ensemble, this influence should be averaged over all trees.

*M*

1

Influence*j* =  Influence*j*(*Ti*) (24)

*M*

*i*=1

The influences are further standardized so that they add up to 100%. Influences do not provide any explanations about how the variable actually affects the response. The resulting influences can then be used for both forward and backward feature selection procedures.

## 5.2. PARTIAL DEPENDENCE PLOTS

Visualization is one of the most comprehensive ways of interpretation. We have already stated that additive GBMs can be plotted fairly easily. In decision-tree GBMs similar model representation can be achieved with partial dependence plots. Partial dependence implies the demonstration of the effect of a variable on the modeled response after marginalizing out all other explanatory variables.

Although the correct way of obtaining the marginal plots would be to numerically integrate out other variables over a suitable grid of values, it can be very computationally consuming in practice. An easier approach is therefore commonly used, when the marginalized variables get fixed with a constant value, equal to their sample mean.

These graphs might not be a perfect representation of the captured effects, especially if the variable interactions significantly impact the resulting model. However, partial dependence plots can provide a useful basis for interpretation that has been noted practical in different applications (De’ath, 2007; Hutchinson et al., 2011; Pittman and Brown, 2011).

The same idea with visualization can be applied to couples of variables, therefore allowing one to inspect and analyze the most important interactions. To identify the interactions of interest, one might first use the relative variable influence and then produce pairwise dependence plots.

We shall illustrate the described interpretations options in the following section on several real world application examples.