Context Driven Constraints for Gradient Boosted Models

Chapman Siu

Contents

Overview and Scope

This document presents gradient boosted models in the context of constrained models. We will examine in particular the history of boosted models, various approaches to monotonic constraints.

We will also consider super learners and their application to boosted models and how they may be implemented in the context of constraints.

We begin chronologically, provided non-exhaustive survey of the history of boosting, followed by super-learning models and constraint modelling.

A brief literature survey

The majority of works done in the last 20 years in the realm of boosting has been by Scharpire with the rough chronological order of (Schapire & Freund 2012):

1. Adaboost
2. Adaboost as gradient descent with particular loss function (exponential loss)
3. Boosting generalised to gradient boosting with extensions for any kind of loss function

History of boosting

Boosting makes the assumption that there exists a *base learning algorithm* which we will call f, which, given labelled training examples, produces a *base classifier* and was first introduced by (Freund & Schapire 1997). The goal of boosting is to improve the base learning algorithm through calling it repeatedly, whilst treating it as a "black box" without being able to manipulate the particulars of the base learning algorithm.

AdaBoost

AdaBoost or adaptive boosting is one of the first algorithms used to improve the base learner for binary classification. AdaBoost depends on having a distribution D_t over training examples $(x_1, y_1), ..., (x_m, y_m)$, where x_i is every instance from the training data, and y_i is the corresponding label, t represents the t iterative base model built, and f_t representing the base learner f, at the tth iteration.

The weight it assigns to training example i is denoted by $D_t(i)$. The goal is to increase the weights of incorrectly classified examples, and decrease correctly classified examples, thereby giving hard examples higher weight and force the base learner to focus its attention on it.

We can define the weighted error:

$$\epsilon = P_{i \sim D_t}(f_t(x_i) \neq y_i)$$

Where $P_{i \sim D_t}$ is the probability with random selection of index i based on the distribution of training examples D_t.

Algorithm *The boosting algorithm for AdaBoost (Schapire & Freund 2012, p 5.)*
Given: $(x_1, y_1), ..., (x_m, y_m)$ where $x_i \in \chi, y_i \in Y$
Initialize: $D_1(i) = 1/m$ for $i = 1, ..., m$.
For $t = 1, ..., T$:
1. Train base learner $f_t : \chi \to Y$ using distribution D_t.
2. Aim: select f_t to minimalize the weighted error:

$$\epsilon = P_{i \sim D_t}(f_t(x_i) \neq y_i)$$

3. Choose $\alpha_t = \frac{1}{2} \log(\frac{1-\epsilon_t}{\epsilon})$
4. Update, for $i = 1, ..., m$:

$$D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} \text{ if } f_t(x_i) = y_i \\ e^{\alpha_t} \text{ if } f_t(x_i) \neq y_i \end{cases} = \frac{D_T}{Z_T} e^{-y_i \alpha_T f_T(x_i)}$$

where Z_t is a normalization factor (chosen so that D_{t+1} will be a distribution), and defined as

$$Z_t := \sum_{i=1}^{m} D_t(i) e^{-\alpha_t y_i f_t(x_i)}$$

Output the final hypothesis using Platt's callibration (Niculescu-Mizil & Caruana 2005):

$$F(x) = \frac{\sum_{t=1}^{T} \alpha_t f_t(x)}{\sum_{t=1}^{T} \alpha_t}$$

AdaBoost as Gradient Descent for exponential loss

(Breiman 1999) discuss extensions to AdaBoost via gradient descent. The first attempt, was to reframe AdaBoost as an additive model. That is consider the final hypothesis as defined above

$$F(x) = \sum_{t=1}^{T} \alpha_t f_t(x)$$

as a linear combination of weak classifiers computed by AdaBoost. With the goal of minimizing the upper bound for training error via exponential loss, we can construct an additive model which causes the greatest decrease in the exponential loss.

Algorithm *AdaBoost with iterative updating using exponential loss. (Schapire & Freund 2012, p 178.)*
Given: $(x_1, y_1), ..., (x_m, y_m)$ where $x_i \in \chi, y_i \in Y$ Initialize: $F_0 := 0$
For $t = 1, ..., T$:
1. Choose $f_t : \chi \to Y, \alpha_t \in \mathbb{R}$ to minimize

$$\frac{1}{m} \sum_{i=1}^{m} \exp(-y_i(F_{t-1}(x_i) + \alpha_t f_t(x_i)))$$

over all choices of α_t and f_t
2. Update:

$$F_t \leftarrow F_{t-1} + \alpha_t f_t$$

Output F_T

In order to demonstrate that the additive model for exponential loss is valid, we consider the t iterative call to the algorithm. If $\alpha_t f_t$ does not depend on all previous iterations (i.e. $1, ..., t-1$), then this algorithm is valid. Firstly exponential loss is defined as

$$\mathcal{L}(F_T(x_i)) := \frac{1}{m} \exp(-y_i F_T(x_i))$$

From above we have

$$F(x) = \sum_{t=1}^{T} \alpha_t f_t(x)$$

and

$$D_{T+1}(i) = \frac{D_T}{Z_T} e^{-y_i \alpha_T f_T(x_i)}$$

Then with the recurrence relationship we can rewrite this as

$$\begin{aligned}
D_{T+1}(i) &= \frac{D_T}{Z_T} e^{-y_i \alpha_T f_T(x_i)} \\
&= \frac{D_{T-1}}{Z_{T-1} \times Z_T} e^{-y_i \alpha_{T-1} f_{T-1}(x_i)} \times e^{-y_i \alpha_T f_T(x_i)} \\
&= D_1(i) \frac{e^{-y_i \alpha_1 f_1(x_i)}}{Z_1} \times \cdots \times \frac{e^{-y_i \alpha_T f_T(x_i)}}{Z_T} \\
&= D_1(i) \frac{\exp(-y_i \sum_{t=1}^{T} \alpha_t f_t(x_i))}{\prod_{j=1}^{T} Z_j} \\
&= \frac{1}{m} \frac{\exp(-y_i F_T(x_i)}{\prod_{j=1}^{T} Z_j}
\end{aligned}$$

Rearranging, for any round t and for all examples i:

$$\mathcal{L}(F_T(x_i)) = \frac{1}{m} \exp(-y_i F_T(x_i)) = D_{T+1}(i) \prod_{j=1}^{T} Z_t$$

This implies

$$\begin{aligned}
\frac{1}{m} \sum_{i=1}^{m} e^{-y_i F_{T+1}(x_i)} &= \frac{1}{m} \sum_{i=1}^{m} \exp(-y_i (F_T(x_i) + \alpha_{T+1} f_{T+1}(x_i))) \\
&= \sum_{i=1}^{m} D_{T+1}(i) \prod_{j=1}^{T} Z_j \times e^{-y_i \alpha_{T+1} f_{T+1}(x_i)} \\
&= \sum_{i=1}^{m} D_{T+1}(i) e^{-y_i \alpha_{T+1} f_{T+1}(x_i)} \times \prod_{j=1}^{T} Z_j \\
&\propto \sum_{i=1}^{m} D_{T+1}(i) e^{-y_i \alpha_{T+1} f_{T+1}(x_i)} \\
&= Z_{T+1}
\end{aligned}$$

that is to say that minimizing the exponential loss on t iteration is proportional to minimizing normalization factor Z_t that does not depend on α_t or f_t.

As Z_t is chosen based on

$$\begin{aligned}
Z_t &= \sum_{i=1}^{m} D_t(i) e^{-\alpha_t y_i f_t(x_i)} \\
&= \sum_{i: y_i = f_t(x_i)} D_t(i) e^{-\alpha_t} + \sum_{i: y_i \neq f_t(x_i)} D_t(i) e^{\alpha_t} \\
&= e^{-\alpha_t}(1 - \epsilon_t) + e^{\alpha_t} \epsilon_t
\end{aligned}$$

which is minimized for exactly the choice of α_t by AdaBoost. Thus, for a given f_t we have α_t which minimizes the exponential loss for iteration t. Thus we can see that α_t, f_t does not depend on previous iterations for D_t and by extension knowledge of α_i, f_i for $i = 1, ..., t-1$.

Gradient Boosted Machines

In the algorithm above, we have already provided an example of an additive model. To re-frame it as a gradient descent problem, we need to redefine our objective function in terms of a loss function $\mathcal{L}(F)$.

In the examples above, we have a exponential loss function $\mathcal{L}(F) := \frac{1}{m}\sum_{i=1}^{m} \exp(-y_i F(x_i))$, though in this example, we will be generalising our loss function to be any real-valued function on $\mathbb{R}^m$. Then using this idea (Schapire & Freund 2012, p.189) suggests we can repeated update F

$$F(x_i) \leftarrow F(x_i) - \alpha \frac{\partial \mathcal{L}(F)}{F(x_i)}$$

in the gradient descent framework to be the same form as the additive form:

$$F \leftarrow F + \alpha f$$

where $\alpha > 0$ and some base learner f.

With this information we can introduce the "AnyBoost" algorithm as shown by (Mason et al. 1999)(Mason 2000).

Algorithm *AnyBoost, a generic functional gradient descent algorithm (Schapire & Freund 2012, p.190)*
Given: $(x_1, y_1), ..., (x_m, y_m)$ where $x_i \in \chi, y_i \in Y$
Goal: minimization of $\mathcal{L}(F)$
Initialize: $F_0 := 0$.
For $t = 1, \ldots, T$:
1. Select $f_t : \chi \to Y$ that maximizes $-\nabla\mathcal{L}(F_{t-1}) \cdot f_t$ or equivalently minimise $\|-\nabla\mathcal{L}(F_{t-1}) - f_t\|_2^2$.
2. Choose $\alpha_t > 0$.
3. Update: $F_t = F_{t-1} + \alpha_t f_t$.

Output F_T

To see the equivalence between AnyBoost and the distribution of training boost examples D_t in AdaBoost algorithm in the previous section, let

$$\nabla\mathcal{L}(F_t) := \langle \frac{\partial \mathcal{L}(F_t)}{\partial F_t(x_1)}, ..., \frac{\partial \mathcal{L}(F_t)}{\partial F_t(x_m)} \rangle$$
$$l_t(i) := -\frac{\partial \mathcal{L}(F_t)}{\partial F_t(x_i)}$$

(Mason 2000) proposes the relationship:

$$d_t(i) := \frac{|l_t(i)|}{\sum_{j=1}^{m} |l_t(j)|}$$

Where $P_{i \sim D_t}$ is the probability with random selection of index i based on the distribution of training examples D_t. D_t is a distribution and weighted appropriately according the loss function as

$$
\begin{aligned}
l_t(i) &= -\frac{\partial \mathcal{L}(F_t)}{\partial F_t(x_i)} \\
&= -\frac{\mathcal{L}(F_t(x_i)) - \mathcal{L}(F_{t-1}(x_i))}{F_t(x_i) - F_{t-1}(x_i)} \\
&= -\frac{\mathcal{L}(F_t(x_i)) - \mathcal{L}(F_{t-1}(x_i))}{F_{t-1}(x_i) + f_t(x_i) - F_{t-1}(x_i)} \\
&= \frac{\mathcal{L}(F_{t-1}(x_i)) - \mathcal{L}(F_t(x_i))}{f_t(x_i)} \\
&= \frac{\mathcal{L}(F_{t-1}(x_i)) - \mathcal{L}(F_t(x_i))}{\tilde{y}_i}
\end{aligned}
$$

Where $\tilde{y}_i$ is your "pseudolabels". Then maximizing $-\nabla\mathcal{L}(F_{t-1}) \cdot f_t$ becomes equivalent to the relationship below (Schapire & Freund 2012, p.192):

$$
\sum_{i=1}^{m} l_t(i) f_t(x_i) \propto \sum_{i=1}^{m} d(i) \tilde{y}_i f_t(x_i)
$$

To see the equivalence of solving h_t under maximization of $-\nabla\mathcal{L}(H_{t-1}) \cdot h_t$ and minimization of $\|-\nabla\mathcal{L}(F_{t-1}) - h_t\|_2^2$. Consider (Schapire & Freund 2012, p.193)

$$
\|-\nabla\mathcal{L}(F_{t-1}) - f_t\|_2^2 = \sum_{i=1}^{m} \left(-\frac{\partial \mathcal{L}(F_{t-1})}{\partial F_t(x_i)} - f_t(x_i) \right)^2
$$

From the result above, we can substitute the "pseudolabel" and show that it is equivalent to minimizing

$$
\sum_{i=1}^{m} (\tilde{y}_i - f_t(x_i))^2
$$

Which demonstrates any loss-minimization problem can be reduced to a sequence of regression problems. Then on each round the problem is to find h_t that is close in terms of squared difference to the residuals (Schapire & Freund 2012, p.193).

This can be extended to the canonical gradient boosting models algorithm as defined in (Hastie, Tibshirani & Friedman 2009)

Algorithm *Gradient Boosted Trees (Hastie, Tibshirani & Friedman 2009, p.361)*

1. Initialize model with a constant value:

$$
F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{m} \mathcal{L}(\gamma).
$$

2. For $m = 1$ to T:
   1. Compute so-called "pseudo-residuals":

$$
r_{it} = -\left[\frac{\partial \mathcal{L}(F(x_i))}{\partial F(x_i)} \right]_{F(x) = F_{t-1}(x)} \quad \text{for } i = 1, \ldots, n.
$$

   2. Fit a base learner $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.

3. Compute multiplier γ_m by solving the following one-dimensional optimization problem:

$$\gamma_t = \arg\min_{\gamma} \sum_{i=1}^{m} \mathcal{L}\left(F_{t-1}(x_i) + \gamma f_t(x_i)\right).$$

4. Update the model:

$$F_t(x) = F_{t-1}(x) + \gamma_t f_t(x).$$

3. Output $F_T(x)$.

Where in step 2.3. we have computed a line search in order to determine the optimal magnitude of the gradient to proceed in.

Shrinkage (Learning Rate)

As described in (Hastie, Tibshirani & Friedman 2009), the shrinkage or learning rate is defined as:

$$F_t(x) = F_{t-1}(x) + \nu\gamma_t f_t(x).$$

Smaller values of ν (more shrinkage) result in larger training risk for the same number of iterations M.

Regularisation

Regularisation is implemented through using a regularised objective (Chen & Guestrin 2016)]

$$\sum_{i=1}^{m} \mathcal{L}(F(x_i)) + \sum_{i=1}^{T} \Omega(f_t)$$

Super Learners

There are several definitions of super learners, they are also known as stack generalises and other classes of models. The basis for these models is to build a "model of models". Let $g^k(x)$ be a generalizer g, where a generalizer is an appropriate output for an hypothesis learnt from the training set (Wolpert 1992), based on the training data x, and the subscript k denote the k model built where $k = 1, 2, ..., K$. For example $g_k(x)$ might represent a gradient boosted tree model, or a logistic regression, or a neural network model.

Unlike our treatment of boosting, we will not be updating our generalisers, rather we are seeking an ideal way to combine (already built) models together.

Bagging (Voting Algorithm)

The simplest approach is simply to take a vote or average of the most common classified result (Breiman 1996). This is most famously seen in bagging approaches in random forest algorithm (Bauer et al. 1999). The algorithm for Bagging is as follows

---

Algorithm *Bagging Predictors (Breiman 1996)*

Given base classifier f.
1. The data set is randomly divided into a test set $\mathcal{V}$ and a training set $\mathcal{T}$.
2. For $t = 1$ to T:
   1. A bootstrap sample $\mathcal{T}_B$ is selected from $\mathcal{T}$
   2. Construct a new base classifier f_t using $\mathcal{T}_B$.
3. Output the final classifier

$$F_T = \arg\max_{y \in Y} \sum_{i: f_i(x) = y} 1$$

i.e. the most often predicted label y.

In the construction of the base classifier (Breiman 1996) suggests using 10-fold cross-validation, and select any hyperparameters associated with f_t by using the original learning set $\mathcal{T}$. In other literature (Bauer et al. 1999) this does not appear to be necessarily required.

Weighted Majority Algorithm

Weighted majority algorithm assumes no prior knowledge on the accuracy of algorithms (Littlestone & Warmuth 1994).

Algorithm *Weighted Majority Algorithm (Littlestone & Warmuth 1994).*
1. Initialise all model weights to 1.
2. For each round:
   output prediction based on weighted majority vote for classification
   multiple weights of all models that made a mistake by β, where $0 \leq \beta < 1$

end

The downside to this algorithm is that it only works for classification, whereas other approaches are capable of working with regression problems.

Stacking

If $beta_k$ are our weights and g_k are our final models which are not necessarily of the same base model, then the most simple linear super learners could be frames as a simple linear combination:

$$\sum_{k=1}^{K} \beta_k g_k(x)$$

Stack generalisers by (Wolpert 1992) was one of the first instances which demonstrated how one could combine estimates for neural networks, which was subsequently expanded by (Breiman 1996) under "stacked regression".

In both cases stacking method would aim to minimize

$$\sum_{i=1}^{N} \mathcal{L}(\sum_{k=1}^{K} \beta_k g_k(x))$$

for some loss function $\mathcal{L}$. For example, if we wish to use mean squared error then we would aim to minimize

$$\sum_{i=1}^{N}\left(y_i - \sum_{k=1}^{K}\beta_k g_k(x_i)\right)^2$$

In examples which (Leblanc & Tibshirani 1996) tried; the linear combinations did not have good predictive performance. However, when an additional constraint that the coefficients β_k were to be non-negative, the final model $\sum_{k=1}^{K}\beta_k g_k(x)$ demonstrated better prediction error than any of the individual $g_k(\mathbf{x})$.

(Breiman 1996) also suggested adding an additional restriction $\sum_{k=1}^{K}\beta_k = 1$, where the ensemble is a convex combination of base learners. In his results he noted that this additional restriction did not have much empirical support, and made little difference.

Historically, in stacking implementations, the metalearning algorithm is often some sort of regularized linear model, however, a variety of parametric and non-parametric methods can used as a metalearner to combine the output from the base fits.

Combining Models

Other approaches of combining models can be combined in a probabilistic sense. There are two approaches which will be discussed here:

1. Bayesian Model Averaging
2. Mixture of Experts

Bayesian Model Averaging

Suppose we have K different models indexed as $g_1, g_2, ..., g_K$ with prior probabilities $P(g_k|x)$. Then the predictions using Bayesian model averaging is (Hoeting et al. 1999):

$$\sum_{k=1}^{K} g_k(x_i)P(g_k|x_i)$$

As the model learns, the posterior probability $P(g_k|x_i)$ becomes increasingly focussed on just one of the models (i.e. the "true" model). This is the key difference between Bayesian model averaging and super learning (Bishop 2006).

Mixture of Experts

Mixtures of models are defined where there are mixing coefficients $\pi_k(x)$ which are known as gating functions and the individual component densities g_k which are called experts.

$$\sum_{k=1}^{K}\pi_k(x_i)g_k(x_i)$$

Where

$$0 \geq \pi_k(x_i) \leq 1$$
$$\sum_{k=1}^{K}\pi_k(x_i) = 1$$

The motivation is that each component is to model the distribution in different regions of input space and gating functions determine which components are dominant (Bishop 2006).

Monotonic Extensions

Monotonic constraints are important particularly around regulatory environments where regularots must be convinced of model reasonableness (S. Lee, Antonio, et al. 2015). (S. Lee, Lin, et al. 2015) expresses that there is no scientific way to find the optimal transformation for monotonic constraints in GLM. (Hastie, Tibshirani & Friedman 2009, p.373) proposes visual approaches through partial dependency plots and GBMs to understand monotonic and non-monotonic relationships between variables.

Monotonic constraints have historically been done through imposing certain patterns via splines to force monotonic relationships (Andersson & Elfving 1991) (Hastie, Tibshirani & Friedman 2009, p.365). In boosting machines, (S. Lee, Lin, et al. 2015) suggests that adding monotonic constraints could be simply achieved through imposing restrictions to avoid pattern reversal. The practicalities of GBMs and GLMs over non-linear models such as neural networks or svm lie primarily in interpretable results via information around the relative influence of input variables (Guelman 2012).

Local monotone

Local monotone constraints are only examined for every split and may violate global constraints (Ridgeway 2007). Unlike implementation by (Chen & Guestrin 2016), (Ridgeway 2007) does not maintain in-memory the distribution of the variables in order to maintain global constraints.

Global monotone

Global monotone constraints can be defined as, if the input dataset is x and has K number of features. Let the kth feature is denoted by x_k, then if we have monotonic constraint for x_k then either

$$f'(x_k|x_1, x_2, ..., x_{k-1}, x_{k+1}, ..., x_l) \geq 0, \forall \mathbf{x}$$

or

$$f'(x_k|x_1, x_2, ..., x_{k-1}, x_{k+1}, ..., x_l) \leq 0, \forall \mathbf{x}$$

(Chen & Guestrin 2016) proposes a global monotone constraint through "global proposal" algorithm when gradient boosting machine proposes its splits. This differs from R's implementation (Ridgeway 2007) where the constraint is only applied for the current split and level. This is done through mapping continuous features into potential bucket splits and aggregating the relevant statistics to find the best solution based on aggregated statistics.

Algorithm *XGBoost Approximate Algorithm for Split Finding(Chen & Guestrin 2016)*
Define: $g(i) = \frac{\partial \mathcal{L}(f(x_i))}{\partial f(x_i)}$
Define: $h(i) = \frac{\partial^2 \mathcal{L}(f(x_i))}{\partial f^2(x_i)}$
Input: I, instance of current node
Input: d, feature dimension
1. for $k = 1$ to m do
  Propose $S_k = \{s_{k1}, s_{k2}, \ldots, s_{kl}\}$ by percentiles on feature k.
  Proposal can be done per tree (global), or per split (local)
end
2. for $k = 1$ to m do
  $G_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq x_{jk} > s_{k,v-1}\}} g(j)$

$H_{kv} \leftarrow= \sum_{j \in \{j | s_{k,v} \geq x_{jk} > s_{k,v-1}\}} h(j)$
end
3. for $k = 1$ to m do
$G_L \leftarrow 0, H_L \leftarrow 0$
for j in S_k (our proposals)
$G_L \leftarrow G_L + G_{kv}, H_L \leftarrow H_L + H_{kv}$
$G_R \leftarrow G_{kv} - G_L, H_R \leftarrow H_{kv} - H_L$
score $\leftarrow \max(\text{score}, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda})$
end

Strict monotone

Strict monotone constraints is similar to the global monotone constraints, except the equality is removed. That is if the kth feature have strict monotone constraint applied then

$$f'(x_k | x_1, x_2, ..., x_{k-1}, x_{k+1}, ..., x_l) > 0, \forall \mathbf{x}$$

or

$$f'(x_k | x_1, x_2, ..., x_{k-1}, x_{k+1}, ..., x_l) < 0, \forall \mathbf{x}$$

is true.

Currently (S. Lee, Lin, et al. 2015) suggests that this is typically solved through additive models such as constrained spline based models. Tree based methods like gradient boosted machines fall short of this constraint due to the nature of recursive partitioning, which leads to "flat" spots within its decision boundaries which are not allowed under strict monotone constraints.

Surrogate Models

Surrogate models are another approach through constraint based modelling. They are models which are trained based on the original inputs and predictions of another model. An example could be models built based on the predictions of trend filtering.

Local Surrogate Models

(Ribeiro, Singh & Guestrin 2016) presents a method for building local surrogate models around single observations to understand how decisions are made for specific observations. Constructing models through this nature will help impose constraints around pre-determined "explanatory records". The premise of (Ribeiro, Singh & Guestrin 2016) model, "Local Interpretable Model-agnostic Explanations" (LIME) is to derive a set of pre-known explanatory records which are similar (based on some kernel) to the observation of interest. A regularised linear model is trained on weighted explanatory set, and the parameters are used to help explain the prediction for the select record.

Algorithm *LIME: Local Interpretable Model-agnostic Explanations (Ribeiro, Singh & Guestrin 2016)*
Input: H, a provided complex model
Input: $D = \{x_1, x_2, ..., x_d\}$, a set of provided explanatory records
Parameter: Similarity kernel π
for each record to be explained
Weigh all explanatory records in D by closeness to that record by kernel π
Based on all explanatory records deemed to be close enough by π, build surrogate model

end

Global Surrogate Models

Global surrogate models are used in many applications such as modelling or optimization techniques, including modelling aerospace applications or approximating surfaces (Forrester & Keane 2009). In order to add constraints through the use of surrogate models, one approach is simply to penalize surrogates in regions of failures by imputing large objective function values at failed points (Forrester & Keane 2009).

References

Andersson, L.E. & Elfving, T. 1991, 'Interpolation and approximation by monotone cubic splines', *Journal of Approximation Theory*, vol. 66, no. 3, pp. 302–33.

Bauer, E., Kohavi, R., Chan, P., Stolfo, S. & Wolpert, D. 1999, 'An Empirical Comparison of Voting Classification Algorithms: Bagging, Boosting, and Variants', *Machine Learning*, vol. 36, no. August, pp. 105–39.

Bishop, C.M. 2006, *Pattern Recognition and Machine Learning (Information Science and Statistics)*, Springer-Verlag New York, Inc., Secaucus, NJ, USA.

Breiman, L. 1996, 'Bagging Predictors', *Machine Learning*, vol. 24, no. 421, pp. 123–40.

Breiman, L. 1999, 'Prediction games and arcing algorithms', *Neural computation*, vol. 11, no. 7, pp. 1493–517.

Chen, T. & Guestrin, C. 2016, 'XGBoost : Reliable Large-scale Tree Boosting System', *arXiv*, pp. 1–6.

Forrester, A. & Keane, A. 2009, 'Recent advances in surrogate-based optimization', *Progress in Aerospace Sciences*, pp. 1–77.

Freund, Y. & Schapire, R. 1997, 'A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting', *Journal of Computer and System Sciences*, vol. 55, no. 1, pp. 119–39.

Guelman, L. 2012, 'Gradient boosting trees for auto insurance loss cost modeling and prediction', *Expert Systems with Applications*, vol. 39, no. 3, pp. 3659–67.

Hastie, T., Tibshirani, R. & Friedman, J. 2009, 'The Elements of Statistical Learning', *Elements*, vol. 1, pp. 337–87.

Hoeting, J.A., Madigan, D., Raftery, A.E. & Volinsky, C.T. 1999, 'Bayesian Model Averaging: A Tutorial', *Statistical Science*, vol. 14, no. 4, pp. 382–417.

Leblanc, M. & Tibshirani, R. 1996, 'Combining Estimates in Regression and Classification', *Journal of the American Statistical Association*, vol. 91, no. 436, pp. 1641–50.

Lee, S., Antonio, K., Lee, S. & Antonio, K. 2015, *Why High Dimensional Modeling in Actuarial Science ?*, vol. 61, no. 0, pp. 0–28.

Lee, S., Lin, S., Antonio, K. & Lee, S.C. 2015, *Delta Boosting Machine and its Application in Actuarial Modeling*, vol. 61, no. 0, pp. 0–21.

Littlestone, N. & Warmuth, M. 1994, 'The Weighted Majority Algorithm', *Information and Computation*, vol. 108, no. 2, pp. 212–61.

Mason, L. 2000, 'Boosting algorithms as gradient descent', *Nips*, vol. 3, no. 1, pp. 1–11.

Mason, L., Baxter, J., Bartlett, P.L. & Frean, M. 1999, 'Functional gradient techniques for combining

hypotheses', *Advances in Neural Information Processing Systems*, no. January 2000, pp. 221–46.

Niculescu-Mizil, A. & Caruana, R. 2005, 'Obtaining Calibrated Probabilities from Boosting', *Uai*, pp. 413–20.

Ribeiro, M.T., Singh, S. & Guestrin, C. 2016, *'Why Should I Trust You?': Explaining the Predictions of Any Classifier*, vol. 39, no. 2011, p. 117831.

Ridgeway, G. 2007, 'Generalized Boosted Models : A guide to the gbm package', *Compute*, vol. 1, no. 4, pp. 1–12.

Schapire, R.E. & Freund, Y. 2012, *Boosting: Foundations and Algorithms*, The MIT Press.

Wolpert, D.H. 1992, 'Stacked generalization', *Neural Networks*, vol. 5, no. 2, pp. 241–59.