Preprint typeset in JINST style - HYPER VERSION

New approaches for boosting to uniformity

Aleksandar Bukva$^{\mathrm{ADD}}$, **Vladimir Gligorov**[d], **Alex Rogozhnikov**[a,b*], **Andrey Ustuzhanin**[b] **and Mike Williams**[c]

[a]*Lomonosov Moscow State University, Moscow, Russia*
[b]*Yandex LLC, Moscow, Russia*
[c]*Massachusetts Institute of Technology, Cambridge, MA, United States*
[d]*Organisation Européenne pour la Recherche Nucléaire (CERN), Geneva, Switzerland*
E-mail: `alex.rogozhnikov@yandex.ru`

ABSTRACT: The use of multivariate classifiers has become commonplace in particle physics. To enhance the performance, a series of classifiers is typically trained; this is a technique known as boosting. This paper explores several novel boosting methods that have been designed to produce a uniform selection efficiency in a chosen multivariate space. Such algorithms have a wide range of applications in particle physics, from producing uniform signal selection efficiency across a Dalitz-plot to avoiding the creation of false signal peaks in an invariant mass distribution when searching for new particles.

[*]Corresponding author.

Contents

1. Introduction

Methods of machine learning play an important role in modern particles physics. Multivariate classifiers, *e.g.*, boosted decision trees (BDTs) and artificial neural networks (ANNs), are now commonly used in analysis selection criteria. BDTs are now even used in software triggers [1, 2]. To enhance the performance, a series of classifiers is typically trained; this is a technique known as boosting. Boosting involves training many simple classifiers and then building a single composite classifier from their responses. The classifiers are trained in series with the inputs of each member being augmented based on the performance of its predecessors. This augmentation is designed such that each new classifier targets more those events which were poorly classified by previous members of the series. The classifier obtained by combining all members of the series is typically much more powerful than any of the individual members.

In particle physics, the most common usage of BDTs is in classifying candidates as signal or background. The BDT is determined by optimizing some figure of merit (FOM), *e.g.*, the signal purity or approximate signal significance. This approach is optimal for a counting experiment; however, in many cases the BDT-based selection obtained in this way is not optimal. For example, in a Dalitz-plot (or any angular or amplitude analysis) analysis, obtaining a selection efficiency on signal candidates that is uniform across the Dalitz-plot is more important than any integrated FOM. Similarly, when measuring a mean particle lifetime, obtaining an efficiency that is uniform in lifetime is what is desired. In both cases, obtaining a uniform selection efficiency greatly reduces the systematic uncertainties involved in the measurement. When searching for a new particle, an analyst may want a uniform efficiency in mass for selecting background candidates so that the BDT-based selection does not generate a fake signal peak. Furthermore, the analyst may also desire a uniform selection efficiency of signal candidates in mass (or other variates) since the new particle mass is not known. In such cases, the BDT is often trained on simulated data generated with several values of mass (lifetime, *etc.*). A uniform selection efficiency in mass ensures that the BDT is sensitive to the full range of masses involved in the search.

2. Uniformity Boosting Methods

The variates used in the BDT are denoted by $\vec{x}$, while the variates in which uniformity is desired are denoted by $\vec{y}$. Some (perhaps all) of the $\vec{x}$ variantes will be *biasing* in $\vec{y}$, *i.e.* they provide discriminating power between signal and background that varies in $\vec{y}$. A uniform BDT selection efficiency can be obtained by removing all such variates; however, this will also reduce the power of the BDT. The goal of boosting algorithms presented in this paper is to *balance* the biases to produce the optimal uniform selection.

Traditional boosting works by assigning training events more weight based on classification erros made by previous members of the series. For example, in gradient boosting [**?**] the analyst chooses a differentiable loss function with the goal of building a classifier that minimizes the loss. A popular choice of loss function is the AdaBoost [**?**] loss function defined as follows:

$$L_{\text{ada}} = \sum_{i=1}^{n} w_i \times \exp\left[-\gamma_i s_i\right], \tag{2.1}$$

where w is the weight of each event[1], $\gamma = +1(-1)$ for signal(background) events and s is the so-called score of each event which is obtained as the sum of scores of all of the previous classifiers in the series. At each stage in the gradient boosting process, a *regressor* (a decision tree in our case) is trained whose purpose is to decrease the loss. This is accomplished using the gradient decent method and the *pseudo-residuals*

$$-\frac{\partial L_{\text{ada}}}{\partial s_i} = w_i \gamma_i \exp\left[-\gamma_i s_i\right], \tag{2.2}$$

which are positive(negative) for signal(background) events and have larger modulai for poorly classified events.

The gradient-boosting algorithm is general in that it only requires the analyst specify a loss function and its gradient. The AdaBoost loss function considers each event individually, but can easily be modified to take into account *non-local* properties of the classifier as follows:

$$L_{\text{general}} = \sum_{i=1}^{n} w_i \exp\left[-\gamma_i \sum_j a_{ij} s_j\right], \tag{2.3}$$

where A is a matrix[2]. For the case where A is the identity matrix, the AdaBoost loss function is recovered. Other choices of A will induce non-local effects, *e.g.*, consider the sparse matrix

$$a_{ij}^{\text{knn}} = \begin{cases} 1, & j \in \text{knn}(i), \text{ events } i \text{ and } j \text{ belong to the same class} \\ 0, & \text{otherwise,} \end{cases} \tag{2.4}$$

[1] We use the convention that $\sum w_i = 1$ for both signal and background events at each stage of training.

[2] A natural choice is a square $n \times n$ matrix, but this is not required.

where knn(i) denotes the set of k-nearest-neighbor events to event i. The loss function obtained from Eq. 2.3 using A^{knn}, denoted by L_{knn}, accounts for the score of each event's k nearest neighbors and not just each event individually. The pseudo-residuals are then

$$-\frac{\partial L_{\text{knn}}}{\partial s_k} = -\sum_i w_i \gamma_i a_{ik} \exp\left[-\gamma_i \sum_j a_{ij} s_j\right]. \tag{2.5}$$

One can see that the direction of the gradient will be influenced the most by events whose k-nearest-neighbor events are classified poorly.

Another approach is to include in the definition of the loss function some uniformity metric. Consider first the case where the data has been binned in $\vec{y}$. If the distribution of classifier responses in each bin, $f_b(s)$, is the same as the global response distribution, $f(s)$, then any cut made on the response will produce a uniform selection efficiency in $\vec{y}$. Therefore, performing a one-dimensional goodness-of-fit test of the hypothesis $f_b \equiv f$ in each bin provides an assessment of the selection uniformity. For example, one could perform the Kolmogorov-Smirnov test in each bin and define a loss function as follows:

$$L_{\text{flat(KS)}} = \sum_b w_b \max|F_b(s) - F(s)|, \tag{2.6}$$

where $F_{(b)}(s)$ denotes the cummulative distribution of $f_{(b)}(s)$ and $w_b = \sum \delta(\text{bin}_i - b) w_i$, *i.e.* w_b is the sum of the weights of the signal events in the bin.

The gradient of the Kolmogorov-Smirnov loss function is zero for events with responses greater than the value of s at which $\max|F_b(s) - F(s)|$ occurs. Thus, it is not suitable for gradient boosting due to its instability. Instead, we use the following loss function:

$$L_{\text{flat}} = \sum_b w_b \sum_{i \in b} |F_b(s_i) - F(s_i)|, \tag{2.7}$$

whose pseudo-residuals are

$$-\frac{\partial L_{\text{flat}}}{\partial s_k} = -w_b \text{sign}\left[F_b(s_k) - F(s_k)\right](w_k/w_b - w_k). \tag{2.8}$$

This so-called flatness loss penalizes non-uniformity but does not consider the quality of the classification. Therefore, the full loss function used is

$$L_{\text{ada+flat}} = L_{\text{flat}} + \alpha L_{\text{ada}}, \tag{2.9}$$

where α is a real-valued parameter that is typically chosen to be small. The first term in Eq. 2.9 penalizes non-uniformity, while that second term penalizes poor classification.

ADD KNN HERE ONCE ALGORITHM IS TESTED.

3. Example Analysis

Here should be some description of $D \to hhh$ sample.

4. Timings

The main drawback of uBoost technique is it's high computational complexity: while simple AdaBoost trains M trees, uBoost builds $100 \times M$ trees (contribution of other operations usually can be neglected).

Presented in this paper classifiers are building M trees, though there is more complicated boosting, it takes at each iteration (k is number of neighbors, N is number of events in training sample)

- meanAdaBoost: $O(k \times N)$
- knnAdaLoss: $O(k \times N)$, for the arbitrary matrix A it is $O(\#\text{nonzero elements in the matrix})$
- FlatnessLoss: $O(N \ln N)$

5. Summary

6. Source code

The link on the repository ans links to final notebooks with results in the article.

Acknowledgments

References

[1] R. Aaij *et al.* [LHCb Trigger Group], *The LHCb trigger and its performance*, JINST **8**, P04022 (2013). [arXiv:1211.3055]

[2] V. Gligorov and M. Williams, *Efficient, reliable and fast high-level triggering using a bonsai boosted decision tree*, JINST **8**, P02013 (2013). [arXiv:1210.6861]

[3] J. Stevens and M. Williams, *uBoost: A boosting method for producing uniform selection efficiencies from multivariate classifiers*, JINST **8**, P12013 (2013) [arXiv:1305.7248]

A. Other Flatness Loss Criteria

ADD THEIL, ETC DESCRIPTIONS AND HOW TO IMPLEMENT IN THE ALGORITHM HERE.