Preprint typeset in JINST style - HYPER VERSION

New approaches for boosting to uniformity

Aleksandar Bukva[e], **Vladimir Gligorov**[d], **Alex Rogozhnikov**[a,b*], **Andrey Ustyuzhanin**[b,f] **and Mike Williams**[c]

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

ABSTRACT: The use of multivariate classifiers has become commonplace in particle physics. Typically, a series of classifiers is trained rather than just one to enhance the performance; this is known as boosting.
In some applications of high energy physics (i.e., amplitude analyses) classifiers should not only optimize some integrated figure of merit, but produce a uniform selection efficiency in a space of selected physical variables. Recently uBoost technique of boosting was proposed which addresses this issue.
In this paper we introduce several approaches to measure the uniformity of efficiency and propose new boosting techniques.

[*]Corresponding author.

Contents

1. Introduction

Methods of machine learning are playing a important role in particles physics nowadays, in particular, multivariate analysis (MVA). Different classifiers like boosted decision trees (BDTs) and artificial neural networks (ANNs) are often used as an important step in analysis selection criteria.

BDTs are now even used in software triggers [link]. The main point of boosting technique is training many simple classifiers and building a composition of their outputs. Classifiers are trained one-after-another, their inputs are augmented in such a way that new classifier should target more at those events which were poorly classified by previous ones. The resulting classifier obtained by combining them often much more powerful, than individual ones.

In practice, however, there are some restrictions that trained classifier should meet (apart from having good classification quality, which is usually measured by some integrated FOM). For example, in an amplitude analysis obtaining a uniform efficiency in a multivariate space of physics variates, i.e., variates that are of physical interest, is often times more important than any integrated FOM based on the total amount of signal and background. Such analyses often have many variates in which a uniform efficiency is desired. A uniform efficiency reduces systematic uncertainties and helps maintain sensitivity to all hypotheses being tested.

The one approach that was already proposed is uBoost (uniform BOOSTing) — a modification of AdaBoost algorithm [link]

Why do we need such algorithms:

1. The result of classification is stable to check different hypotheses
2. To get unbiased distribution of signal, this enables us to compute real mass of particle
3. Not to get fake peak, thus not get a false discovery

The features, along which we want the prediction to be flat, will be referred to as *uniform variables*. In high energy physics these are usually masses.

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.

One category of boosting works by assigning training events more weight based on classification erros made by previous members of the series. For example, the AdaBoost [**?**] algorithm updates the weight of event i, w_i, according to

$$w_i' = w_i \times \exp\left[-\gamma_i s_i\right], \tag{2.1}$$

where $\gamma = +1(-1)$ 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. The uBoost technique, described in detail in Ref. [**?**], alters the event-weight updating procedure to achieve uniformity in the signal-selection efficiency.

Another approach to obtain uniformity, introduced in this paper, involves defining a more general expression of the AdaBoost criteria:

$$w_i' = w_i \times \exp\left[-\gamma_i \sum_j a_{ij}\right], \tag{2.2}$$

where a_{ij} are the elements of some matrix A[1]. For the case where A is the identity matrix, the AdaBoost weighting procedure is recovered. Other choices of A will induce non-local effects, *e.g.*, consider the sparse matrix A_{knn} given by

$$a_{ij}^{\text{knn}} = \begin{cases} \frac{1}{k}, & j \in \text{knn}(i), \text{ events } i \text{ and } j \text{ belong to the same class} \\ 0, & \text{otherwise,} \end{cases} \tag{2.3}$$

where knn(i) denotes the set of k-nearest-neighbor events to event i. This procedure for updating the event weights, which we refer to as kNNAdaBoost, accounts for the score of each event's k nearest neighbors and not just each event individually.

The gradient boosting [**?**] (GB) algorithm category requires the analyst to choose a differentiable loss function with the goal of building a classifier that minimizes the loss. A popular choice of loss function is the AdaLoss [**?**] function

$$L_{\text{ada}} = \sum_{i=1}^{n} \exp\left[-\gamma_i s_i\right]. \tag{2.4}$$

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*

$$-\frac{\partial L_{\text{ada}}}{\partial s_i} = \gamma_i \exp\left[-\gamma_i s_i\right], \tag{2.5}$$

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 AdaLoss function considers each event individually, but can easily be modified to take into account non-local properties of the classifier as follows:

$$L_{\text{general}} = \sum_{i=1}^{n} \exp\left[-\gamma_i \sum_j a_{ij} s_j\right]. \tag{2.6}$$

For example, the loss function obtained from Eq. 2.6 using A_{knn}, which we refer to as kNNAdaLoss and denote L_{knn}, accounts for the score of each event's k nearest neighbors and not just each event individually. The pseudo-residuals of L_{knn} are

$$-\frac{\partial L_{\text{knn}}}{\partial s_k} = \sum_i \gamma_i a_{ik}^{\text{knn}} \exp\left[-\gamma_i \sum_j a_{ij}^{\text{knn}} s_j\right]. \tag{2.7}$$

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 have 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

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

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:

$$L_{\text{flat(KS)}} = \sum_b w_b \max|F_b(s) - F(s)|, \tag{2.8}$$

where $F_{(b)}(s)$ denotes the cummulative distribution of $f_{(b)}(s)$ and $w_b = \sum \delta(\text{bin}_i - b)/n_{\text{signal}}$, *i.e.* w_b is the fraction of signal events in the bin[2].

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 *flatness* loss function:

$$L_{\text{flat}} = \sum_b w_b \int |F_b(s) - F(s)|^2 ds, \tag{2.9}$$

whose pseudo-residuals are

$$-\frac{\partial L_{\text{flat}}}{\partial s_k} \approx -2\left[F_b(s_k) - F(s_k)\right]/n_{\text{signal}}. \tag{2.10}$$

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

$$L_{\text{ada+flat}} = L_{\text{flat}} + \alpha L_{\text{ada}}, \tag{2.11}$$

where α is a real-valued parameter that is typically chosen to be small. The first term in Eq. 2.11 penalizes non-uniformity, while that second term penalizes poor classification.

The loss function given in Eq. 2.11 can also be constructed without binning the data using k-nearest-neighbor events. The cummulative distribution $F_{\text{knn}}(s)$ is easily obtained and the bin weight, w_b, is replaced by a k-nearest-neighbor weight, w_{knn}. First, each event is weighted by the inverse of the number of times it is included in the k-nearest-neighbor sample of another event. Then, w_{knn} is the sum of such weights in a k-nearest-neighbor sample divided by the total sum of such weights in the full sample. This procedure is followed to offset the fact that some events are found in more k-nearest-neighbor samples than other events.

3. Example Analysis

4. Datasets

This study uses simulated event samples produced using the official LHCb simulation framework. The software used for the generation of the events is described in LHCb publications as follows :

> In the simulation, pp collisions are generated using PYTHIA [4] with a specific LHCb configuration [5]. Decays of hadronic particles are described by EvtGen [6], in which final state radiation is generated using PHOTOS [7]. The interaction of the generated particles with the detector and its response are implemented using the GEANT toolkit [8, 9] as described in Ref. [10].

[2]If weighted events are used, then the fractional sum of weights should be used for w_b.

All simulated event samples are generated inside the LHCb detector acceptance. The signal used in this analysis consists of $D_s^\pm \to \pi^+\pi^-\pi^\pm$ decays, simulated using the D_DALITZ model of EvtGen to simulate the intermediate resonances which contribute to the three pion final state. The background candidates are three pion combinations reconstructed in simulated samples of $c\bar{c}$ and $b\bar{b}$ events, where the charm and bottom quark decays are inclusively modelled by EvtGen. The simulated events contain "truth" information which identifies them as signal or background, and which identifies the physical origin of the three pion combinations reconstructed in the $c\bar{c}$ and $b\bar{b}$ simulated samples.

5. 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: $O(N \ln N)$

6. Summary

7. Source code

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

Acknowledgments

These results were obtained using events generated with the official LHCb simulation, and we are grateful to the LHCb collaboration for this privilege. We particularly acknowledge the work of the LHCb Simulation and Core Computing teams in tuning the simulation software and managing the productions of simulated events.

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]

[4] "Sjöstrand, Torbjörn and Mrenna, Stephen and Skands, Peter", *PYTHIA 6.4 physics and manual*, , *JHEP* **05** (2006) 026. [hep-ph/0603175]

[5] "Belyaev, I. and others", *Handling of the generation of primary events in GAUSS, the LHCb simulation framework*, , *NSS/MIC* **IEEE** (2010) 1155.

[6] "Lange, D. J.", *The EvtGen particle decay simulation package*, , *NIM* **A462** (2001) 152-155.

[7] "Golonka, Piotr and Was, Zbigniew", *PHOTOS Monte Carlo: a precision tool for QED corrections in Z and W decays*, , *EPJ* **C45** (2006) 97-107. [hep-ph/0506026]

[8] "Allison, John and Amako, K. and Apostolakis, J. and Araujo, H. and Dubois, P.A. and others", *Geant4 developments and applications*, , *IEEE TNS* **53** (2006) 270.

[9] "Agostinelli, S. and others", *Geant4: a simulation toolkit*, , *NIM* **A506** (2003) 250.

[10] "Clemencic, M and others", *The LHCb simulation application, GAUSS : design, evolution and experience*, , *J. Phys. Conf. Ser.* **331** (2011) 032023.

A. Measures of uniformity

In this section we discuss different methods for measuring the uniformity of prediction. One typical way of 'checking' uniformity of prediction used by physicists is fitting the distribution of the events that were classified as signal (or background) over the feature for which you wish to check uniformity. This approach requires assumptions about the shape of the distribution, which makes quantitative comparisons of different classifiers difficult. Our aim here is to explore uniformity figures of merit which make comparing classifiers easier, analogously to how the area under the ROC curve can be used to compare absolute classifier performance.

The output of event classification is the probability of each event being signal or background, and it is only after we apply a cut on this probability that events are classified. An ideal uniformity of signal prediction can then be defined for a given "uniform feature" of interest. It means that whichever cut we select, the efficiency for a signal event to pass the cut doesn't depend on the uniform feature. Uniformity for background can be defined in the same manner, but for simplicity, in what follows we will only discuss the uniformity of efficiency for signal events.

A trivial example of a classifier that has ideal uniformity is a classifier which returns a random classification probability, but such a classifier is of course not very useful. One can try to design a uniform classifier with respect to a given feature by not using this feature, or any correlated features, in the classification; in practice, however, this approach also tends to lead to poorly performing classifiers. The approach which we take in this paper is to explicitly let the classifier learn how to balance non-uniformities coming from different features in such a way as to generate a classification which is uniform on average. It is then important to be able to accurately measure the uniformity of classification.

Before proceeding, it is useful to define some desirable properties of uniformity metrics

1. The metric shouldn't depend strongly on the number of events used to test uniformity;

2. The metric shouldn't depend on the normalization of the event weights: if we multiply all the weights by some arbitrary number, it shouldn't change at all;

3. The metric should depend only on the order of predictions, not the exact values of probabilities. This is because we care about which events pass the cut and which don't, not about the exact values of predictions. For example: correlation of prediction and mass doesn't satisfy this restriction.

4. The metric should be stable against any of its own free paramters : if it uses bins, changing the number of bins shouldn't affect the result, if it uses k-nearest neighbors, it should be stable against different values of k.

In what follows we will consider different metrics which satisfy these criteria, and then compare their performance in some test cases.

A.1 Standard Deviation of Efficiency on Bins (SDE)

If the space of uniform features is split into bins, it is possible to define the global efficiency

$$\text{eff} = \frac{\text{total weight of signal events that passed the cut}}{\text{total weight of signal events}},$$

as well as the efficiency in every bin,

$$\text{eff}_{\text{bin}} = \frac{\text{weight of signal events in bin that passed the cut}}{\text{weight of signal events in this bin}}.$$

One measure of non-uniformity is the standard deviation of bin efficiencies from the global efficiency:

$$\sqrt{\sum_{\text{bin}} (\text{eff}_{\text{bin}} - \text{eff})^2}.$$

To make the metric more stable against fluctuations in bins which contain very few events, we add weights to the bins (note that $\sum_{\text{bin}} \text{weight}_{\text{bin}} = 1$):

$$\text{weight}_{\text{bin}} = \frac{\text{total weight of signal events in bin}}{\text{total weight of signal events}},$$

giving the weighted standard deviation (SDE) formula

$$\text{SDE}(\text{eff}) = \sqrt{\sum_{\text{bin}} \text{weight}_{\text{bin}} \times (\text{eff}_{\text{bin}} - \text{eff})^2}.$$

This formula is valid for any given cut value. To measure the overall non-flatness of the selection, we take several global efficiencies and use

$$\text{SDE}^2 = \frac{1}{k} \sum_{\text{eff} \in [\text{eff}_1 \dots \text{eff}_k]} \text{SDE}^2(\text{eff})$$

Another power $p \neq 2$ can be used as well, but $p = 2$ is considered as the default value.

A.2 Theil Index of Efficiency

The Theil Index is frequently used to measure economic inequality:

$$\text{Theil} = \frac{1}{N}\sum_i \frac{x_i}{<x>} \ln \frac{x_i}{<x>}, \qquad <x> = \frac{1}{N}\sum_i x_i$$

In our case we have to alter formula a bit to take into account that different bins have different impact, thus the formula turns into

$$\text{Theil}(\text{eff}) = \sum_{\text{bin}} \text{weight}_{\text{bin}} \frac{\text{eff}_{\text{bin}}}{\text{eff}} \ln \frac{\text{eff}_{\text{bin}}}{\text{eff}}.$$

TODO how to combine Theil for different global efficiencies? VAVA : Suggest to combine a Theil of Theils, i.e. treat each global efficiency as a bin?

$$\text{Theil} = ???\text{fromTheil}(\text{eff})$$

A.3 Distribution Similarity Approach

Instead of measuring uniformity in terms of binned efficiencies, it is possible to consider the distribution of the binned classifier predictions, F_{bin}, directly. Ideal uniformity means that all the distributions F_{bin} are equal and hence equal to the global distribution $F(x)$. To 'measure' non-flatness we can use some distribution distance, like Kolmogorov-Smirnov:

$$\sum_{\text{bin}} \text{weight}_{\text{bin}} \max_x |F_{\text{bin}}(x) - F(x)|,$$

but Cramér–von Mises similarity is more informative (usually $p = 2$ is used):

$$\sum_{\text{bin}} \text{weight}_{\text{bin}} \int |F_{\text{bin}}(x) - F(x)|^p \, dF(x),$$

in particular because Kolmogorov-Smirnov measures are too sensitive to local non-uniformities. The advantage of this method is that we don't need to select some global efficiencies like in the previous metrics.

A.4 Knn-based modifications

Though operating with bins is usually both simple and very efficient, in many cases it is hard to find the optimal size of bins in the space of uniform features (specifically in the case of more than two dimensions). As mentioned earlier, problems can also arise due to bins with very low populations. In these cases we can switch to k-nearest neighbors: for each signal event we find k nearest signal events (including the event itself) in the space of uniform features. Now we can compute the efficiency $\text{eff}_{\text{knn}(i)}$, from the empirical distribution $F_{\text{knn}(i)}$ of nearest neighbors. The weights for $\text{knn}(i)$ are proportional to the total weight of events in $\text{knn}(i)$:

$$\text{weight}_{\text{knn}(i)} = \alpha \sum_{j \in \text{knn}(i)} w_j, \qquad \alpha^{-1} = \sum_i \sum_{j \in \text{knn}(i)} w_j,$$

so again weights are normed to 1: $\sum_i \text{knn}(i) = 1$.

It is then possible to write the knn versions of SDE

$$\text{knnSDE}^2(\text{eff}) = \sum_{i \in \text{events}} \text{weight}_{\text{knn}(i)} \left|\text{eff}_{\text{knn}(i)} - \text{eff}\right|^2,$$

$$\text{knnSDE}^2 = \sum_{\text{eff} \in [\text{eff}_1, \ldots \text{eff}_k]} \text{knnSDE}^2(\text{eff}),$$

the Theil index of Efficiency

$$\text{knnTheil}(\text{eff}) = \sum_{i \in \text{events}} \text{weight}_{\text{knn}(i)} \frac{\text{eff}_{\text{knn}(i)}}{\text{eff}} \ln \frac{\text{eff}_{\text{knn}(i)}}{\text{eff}},$$

$$\text{knnTheil} = ???\text{knnTheil}(\text{eff})$$

and the similarity-based measure:

$$\sum_{i \in \text{events}} \text{weight}_{\text{knn}(i)} \int \left|F_{\text{knn}(i)}(x) - F(x)\right|^p dF(x).$$

The knn approach suffers from a drawback: the impact of different events has very little connection with the weights, because some events are selected as nearest neighbours much more frequently than others. This effect can be suppressed by dividing the initial weight of the event by the number of times it is selected as a nearest neighbour.

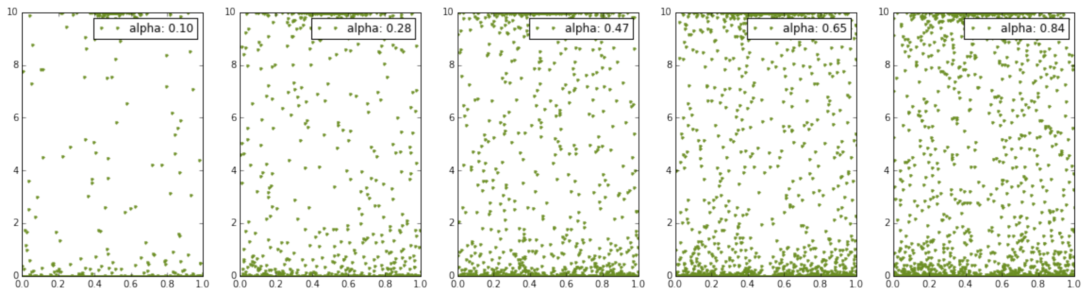
A.5 Advantages and Disadvantages of Different Metrics

A.5.1 Theil and SDE

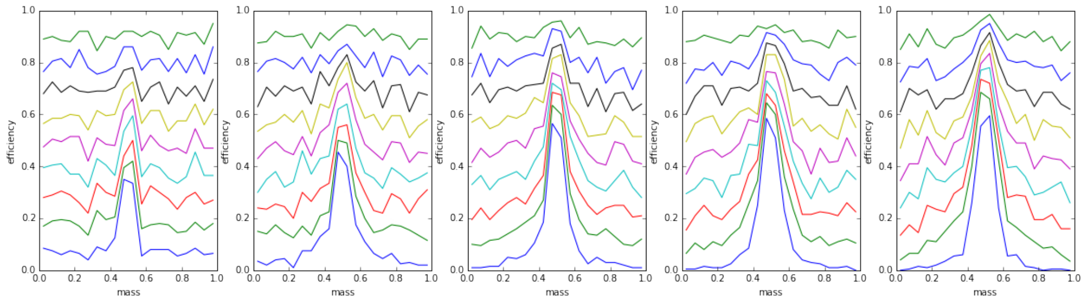
Let us compare two metrics that have proven most appropriate for our problem, SDE and Theil. We have some linearly spaced array of mass of particles from interval $[0,1]$, some constant α linearly sapced in interval $[.1,1]$. The predictions are correlated with mass via beta distribution. We have two distributions that are different in a way that they are symmetrical. SDE should show no changes if we flip the distribution, while the Theil should make difference between pit and peak. First distribution with a peak is given by:

$$p[\text{i}] = \textbf{beta}(\alpha \cdot \bar{m}, \text{abs}(m[i] - \bar{m}))$$

while the second one has minus sign in front.

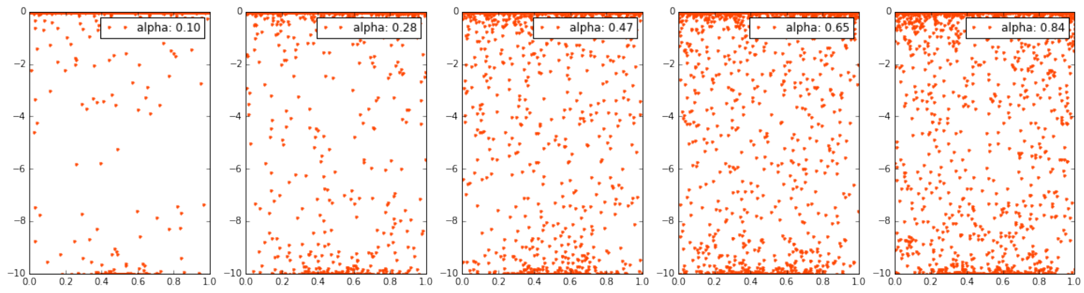


(a) Mass vs prediction

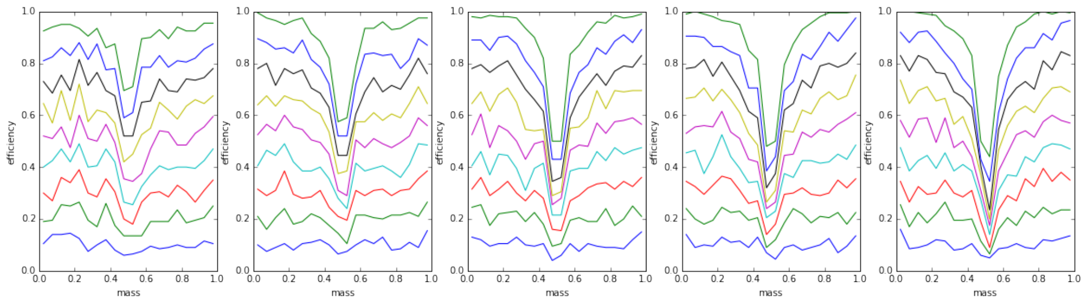


(b) Efficiencies

Figure 1: Peak distribution



(a) Mass vs prediction



(b) Efficiencies

Figure 2: Pit distribution

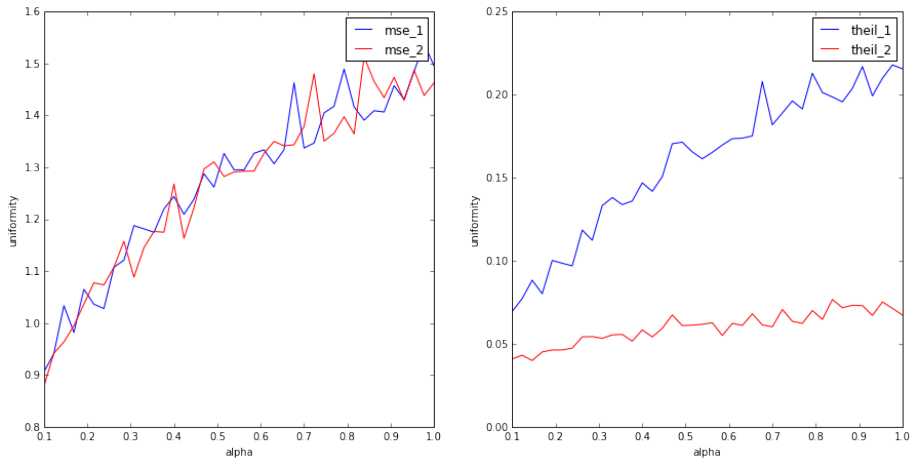
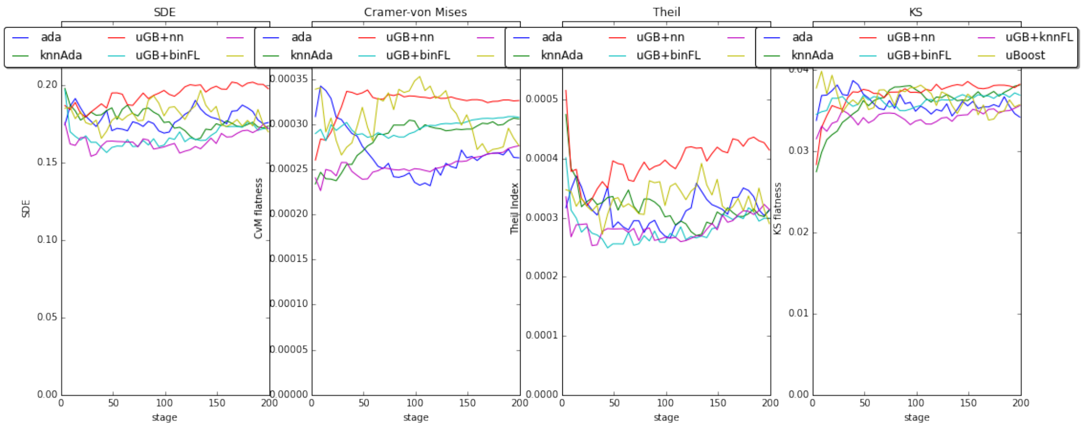


Figure 3: SDE and Theil

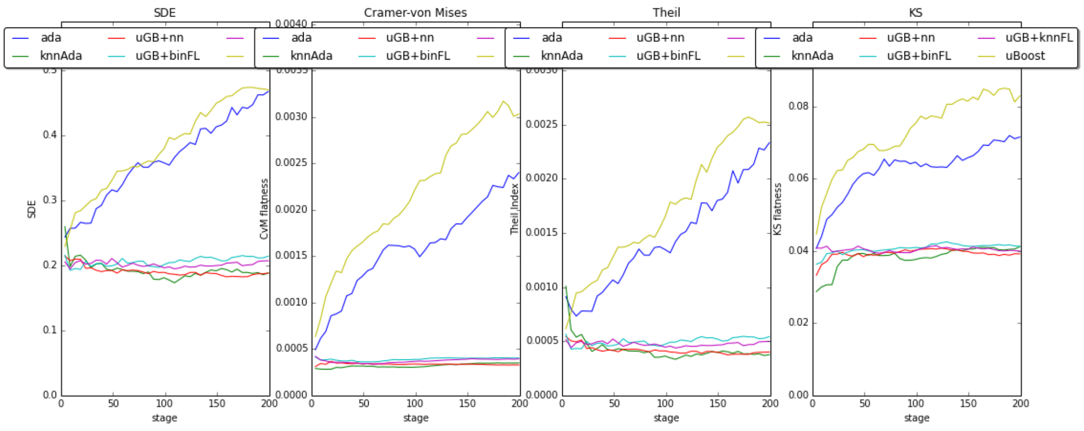
From these figures we can see that SDE has the same value and it is not able to make difference between these distributions. In the second case Theil has lower value which indicates that distribution is flatter. In the combination with SDE it can be used to detarmine the type of a distribution. Lower value of SDE can indicate that distribution is flat or has some narrow peaks or pits, and by using Theil we can distinguish between these two cases.

A.5.2 $D \to hhh$

When our classifiers where applied to data set $D \to hhh$ we got uniformity measures in a graph below:



(a) Metrics for $D \to hhh$ background



(b) Metrics for $D \to hhh$ signal

Figure 4: Metrics for $D \to hhh$ data set

From these graphs we can see the all metrics have shown similar results, and that there is no significant difference on real data sets. SDE and Theil can be used in limit cases to more precisely determine nature of a distribution.

B. Boosting approaches

B.1 Mean Ada Boost

This is a modification of AdaBoost algorithm. In AdaBoost one multiplies weights in such a way:

$$w_i = w_i \times \exp[-y_i \, \text{score}_i],$$

to enlarge the weights of poorly classified events (y_i is $+1$ for signal and -1 for background).

But now we use the mean of prediction of k nearest neighbors (of the same class)

$$w_i = w_i \times \exp[-y_i \frac{1}{k} \sum_{j \in \text{knn}(i)} \text{score}_j]$$

Thus boosting focuses not on the events that were poorly classified, but on the regions with poor classification.

B.2 Gradient Boosting with AdaLoss Modification (knn-Ada)

Gradient boosting on trees is widely used algorithm[link], it's built upon decision tree regressors with usage of some loss function.

Let's start with examples. One of the popular losses used is AdaLoss:

$$\sum_{i \in \text{events}} w_i \times \exp[-\text{score}_i \, y_i]$$

where y_i is either +1 (for signal events) or -1 (for background events). Good classification supposes that signal events should have large positive scores, while background ones should have large negative.

The predictions of separate regressors are simply summed up to form a score:

$$\text{score}_i = \sum_{r \in \text{regressors}} \text{pred}_r(i),$$

which can be 'translated' into probabilities by logistic function.

So the goal of algorithm is now to minimize the loss function. At each stage it trains one more regressor, which should decrease the value of loss, the most vivid way is to train it on negative gradient of loss. In the case of AdaLoss this can be done pretty easy:

$$-\frac{\partial \, \text{AdaLoss}}{\partial \, \text{score}_i} = w_i \, y_i \exp[-\text{score}_i \, y_i],$$

so it is positive for signal events, negative for background events and has larger modulus for the events which are poorly classified.

After a new tree was built, it's output is altered: for each leaf we can compute such a value, that the result will give the smallest possible value for loss.

...computation of optimal value goes here...

This loss function can be easily modified to take in account not only the score for individual elements, but also 'finds' regions with lower-than-average quality.

$$\text{knnAdaLoss} = \sum_{i \in events} \exp[-y_i \times \sum_{j \in \text{knn}(i)} \text{score}_j],$$

where the index j goes over the k nearest neighbors of ith event, which belongs to the same class (signal or background).

We can introduce a supplementary sparse matrix $A \in \mathbb{R}^{N \times N}$ (N is number of events), which is defined as

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

so the loss can be written as

$$\text{knnAdaLoss} = \sum_i \exp[-y_i \sum_j a_{ij} \, \text{score}_j],$$

it's negative gradient is easily computed:

$$-\frac{\partial \, \text{knnAdaLoss}}{\partial \, \text{score}_k} = y_k \sum_i a_{ik} \exp[-y_i \sum_j a_{ij} \, \text{score}_j],$$

from this we can see that new algorithm will pay more attention to the events, which has poorly classified neighbors (and neighbors of neighbors). The named loss targets to obtain uniformity in both signal and background.

And the post-correction can be handled by using second-order Newton-Raphson optimization step

... TODO computation of optimal value goes here ...

One can note, that the matrix A doesn't need to be necessarily square. One can introduce M groups of events (which may intersect), each group consists of several events with close uniform variables (and close events). Then one introduces $A \in \mathbb{R}^{M \times N}$:

$$a_{mj} = \begin{cases} 1, & \text{event } j \text{ is in group } m \\ 0, & \text{otherwise} \end{cases}$$

In particular, if we take A to be identity matrix: $A = I$ (each event to be in it's own group), knnAdaLoss turns into a simple AdaLoss.

B.3 Gradient Boosting with Flatness Loss (FL)

Let's use the metrics introduces in the section A.3:

$$\sum_{\text{bin}} \text{weight}_{\text{bin}} \int |F_{\text{bin}}(x) - F(x)|^p \, dF(x),$$

which is good as a measure, but due to the non-smoothness of $F(x)$ it's gradient is singular, so we use instead

$$\text{FL} = \sum_{\text{bin}} \text{weight}_{\text{bin}} \int |F_{\text{bin}}(x) - F(x)|^p \, dx.$$

So, the derivative looks like:

$$\frac{\partial}{\partial \,\text{score}_i} \text{FL} = \sum_{\text{bin}} \text{weight}_{\text{bin}} \frac{\partial}{\partial \,\text{score}_i} \int |F_{\text{bin}}(x) - F(x)|^p \, dx$$

Let $\text{bin}(i)$ be a bin to which event i belongs, then we can compute:

$$\begin{aligned} -\frac{\partial}{\partial \,\text{score}_i} \text{FL} &= -\text{weight}_{\text{bin(i)}} \frac{\partial}{\partial \,\text{score}_i} \int \left|F_{\text{bin}(i)}(x) - F(x)\right|^p dx \cong \\ &\cong \text{weight}_{\text{bin(i)}} \, p \left|F_{\text{bin}(i)}(x) - F(x)\right|^{p-1} \text{sgn}[F_{\text{bin}(i)}(x) - F(x)] \frac{w_i}{\text{weight}_{\text{bin(i)}}} \Bigg|_{x=\text{score}_i} = \\ &= w_i \, p \left|F_{\text{bin}(i)}(x) - F(x)\right|^{p-1} \text{sgn}[F_{\text{bin}(i)}(x) - F(x)] \Bigg|_{x=\text{score}_i} \end{aligned}$$

TODO give explanations here

The next thing we need to point is FL doesn't take into account the quality of predictions. So what we use in practice is linear combination of FlatnessLoss and AdaLoss:

$$\text{loss} = \text{FL} + \alpha \, \text{AdaLoss}$$

First one penalizes the non-uniformity, second one — poor predictions.