New Approaches to Uniform Boosting

August 19, 2014

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 Measurement

In this section we come up with some approaches on how to measure uniformity of prediction. The 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 mass (or some other variable).

This approach is hardly formalizable, and not automatable — each time you should assume some kind of distribution. Ideally we want to have some easy-to-use out-of-the-box metrics like FOMs in machine learning (like area under the ROC, f1 or).

2.1 Ideal uniformity

We start from the simplest case — when we are fully satisfied by predictions of our classifier. Remember that output of classification is probabilities of each event being a signal and background event, only after we select some cut on probability we get classification.

Ideal uniformity of signal prediction means that whichever cut we select, the efficiency (part of signal event that passed the cut) doesn't depend on uniform variables: in every region of uniform variables space the part of signal events that passed the cut is the same.

In practice, of course, this never happens.

There is uniformity of predictions on signal and uniformity of efficiency on background, which can be defined one from another by swapping classes. In what follows in this section we are writing about the efficiency on signal events.

A good example of classifier that has close to ideal uniformity is classifier which returns a random probability in [0, 1] range. What a pity: in practice it's absolutely useless.

2.2 Some Restrictions on Uniformity Metrics

There are some additional conditions which we expect metric to meet:

1. shouldn't depend much on the number of events (i.e., if we randomly select half of the events, the metrics should roughly be the same)

2. shouldn't depend on the weights renormalization: if we multiply all the weight by some arbitrary number, it shouldn't change at all.

3. depends 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.

   Example: correlation of prediction and mass doesn't satisfy this restriction.

4. parameter stability: if it uses bins, changing the number of bins shouldn't affect the metrics value much, if it uses k-nearest neighbors, it should be stable to small deviations of k.

2.3 Standard Deviation of Efficiency on Bins (SDE)

Let's split the space of uniform features into bins, the uniformity in these terms is following: when we select some probability cut, the part of signal events that passes the cut is equal in all bins. Assume we selected some cut, then we have global efficiency

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

Efficiency in every bin is defined respectively,

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

So, basically, what we want to have in our dreams:

$$\text{eff}_{\text{bin}} = \text{global efficiency} \qquad \forall \text{ bin}$$

To measure how far we are from ideal situation we use standard deviation:

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

What is bad in this formula that every bin has some impact in the result, which does not depend on how many events are there, so metrics becomes very unstable to deviations in bins with only few events. To cure this, we add weights to the bins (note that $\sum_{\text{bin}} \text{weight}_{\text{bin}} = 1$):

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

so we have SDE formula:

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

In fact, the expression depends on the cut, but for cuts which produce equal efficiency, this is

Finally we note that the weighted average of eff_{bin} is eff:

$$\text{eff} =< \text{eff}_{\text{bin}} >= \sum_{\text{bin}} \text{weight}_{\text{bin}} \times \text{eff}_{\text{bin}},$$

and this is why the introduced metrics was named SDE — this is a weighted standard deviations of array of bin efficiencies.

But this is how we measure the non-uniformity for only one fixed cut, to measure the overall non-flatness, we take several global efficiencies (for instance, [0.5, 0.6, 0.7, 0.8, 0.9], because in practice usually we are interested in cuts with high global efficiency) and use

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

Some other power $p \neq 2$ can be used as well, but $p = 2$ is considered as the default value:

$$\text{SDE}^p(\text{eff}) = \sum_{\text{bin}} \text{weight}_{\text{bin}} \times |\text{eff}_{\text{bin}} - \text{eff}|^p, \qquad \text{SDE}^p = \frac{1}{k} \sum_{\text{eff} \in [\text{eff}_1 \ldots \text{eff}_k]} \text{SDE}^p(\text{eff}).$$

2.4 Theil Index of Efficiency

One more measure uses Theil Index 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?

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

2.5 Distribution Similarity Approach

Let's start from reformulation of what is uniform predictions in signal. First we split all signal events into some bins in uniform variables. There is some empirical distribution F_{bin} of predictions in each bin. 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):

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

The good point is we don't need to select some global efficiencies like in the other metrics.

2.6 Connection Between SDE and Distance Similarity Approach

SDE and DSA based on Cramér–von Mises similarity can be shown to have connection. Let's consider the SDE with global efficiencies $= [1/N, 2/N, \ldots, N/N]$. In the limit $N \to \infty$

$$\lim_{N\to\infty} \text{SDE}^2 = \lim_{N\to\infty} \frac{1}{N} \sum_{\text{eff}} \text{SDE}^2(\text{eff}) = \int_0^1 \text{SDE}^2(\text{eff}) d\,\text{eff} = \int_0^1 \sum_{\text{bin}} \text{weight}_{\text{bin}} \left|\text{eff}_{\text{bin}} - \text{eff}\right|^2 d\,\text{eff}$$

From the other side, we can write the expression for similarity-based measure (for $p = 2$)

$$\sum_{\text{bin}} \text{weight}_{\text{bin}} \int |F_{\text{bin}}(x) - F(x)|^2 \, dF(x) = \int \sum_{\text{bin}} \text{weight}_{\text{bin}} |F_{\text{bin}}(x) - F(x)|^2 \, dF(x)$$

The hard thing now is to believe this is literally the same and these two expressions are equal.

2.7 Knn-based modifications

Though operating with bins is usually both simple and very efficient, in many cases it is hard to find optimal size of bins in the space of uniform variables (specifically in the case of more than two dimensions). One more situation when bins-based approach fails, is when we have too few events to obtain a good statistics at least in several bins.

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 variables. Now we can compute the efficiency $\text{eff}_{\text{knn}(i)}$, 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: $\sum_i \text{knn}(i) = 1$.

Now we are ready to write knn version of SDE:

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

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

knn version of Theil index of Efficiency

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

knnTheil =???knnTheil(eff)

and knn version of similarity-based measure:

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

K-nearest neighbors approach suffers from the other drawback: the impact of different events has very little connection with the weights, because some events are met in knn of other events much more frequently while the other. This effect can be suppressed by dividing initial weight of the event by the number of times it is met in knn.

2.8 Advantages and Disadvantages of Different Metrics

... TODO, here some plots with comparison, maybe timings ...

3 Approaches Proposed

3.1 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.

3.2 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:

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

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.

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

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

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

so the loss can be written as

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

it's negative gradient is easily computed:

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

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.

3.3 Gradient Boosting with Flatness Loss (FL)

Let's use the metrics introduces in the section 2.5:

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

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:

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

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

$$= w_i \, p \, \left|F_{\text{bin}(i)}(x) - F(x)\right|^{p-1} \text{sgn}[F_{\text{bin}(i)}(x) - F(x)] \Big|_{x=\text{score}_i}$$

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.

4 Dataset

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

5 Plots

6 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: $O(N \ln N)$

7 Summary

8 Acknowledgments

9 Source code

The code for classifiers ans links to final notebooks with results in the article.

References

[1] Author, *Title*, Journal/Editor, (year)