New approaches for boosting to uniformity

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

[a] Lomonosov Moscow State University, Moscow
[b] Yandex School of Data Analysis, Moscow
[c] Faculty of Physics, Belgrade
[d] Organisation Européenne pour la Recherche Nucléaire (CERN), Geneva
[e] Moscow Institute of Physics and Technology, Moscow
[f] Imperial College, London
[g] Massachusetts Institute of Technology, Cambridge

alex.rogozhnikov@yandex.ru

11 November, 2014

Outline

- What is uniformity (of predictions)?
- How to measure it?
- How to achieve it? (classifiers proposed)

Uniformity

In particle physics, apart from optimizing some FOM of classifier (BDT, ANN), there are cases when we want to have uniformity of predictions

- Dalitz-plot analysis (or any angular or amplitude analysis)
- search for a new particle (not to get fake peak)
- sensitivity for new signal in wide range of mass (lifetime ...), (i.e. train only one classifier, not separate for each mass)

Uniform variables — variables, along which uniformity of selection is desired (Dalitz variables, mass variable).

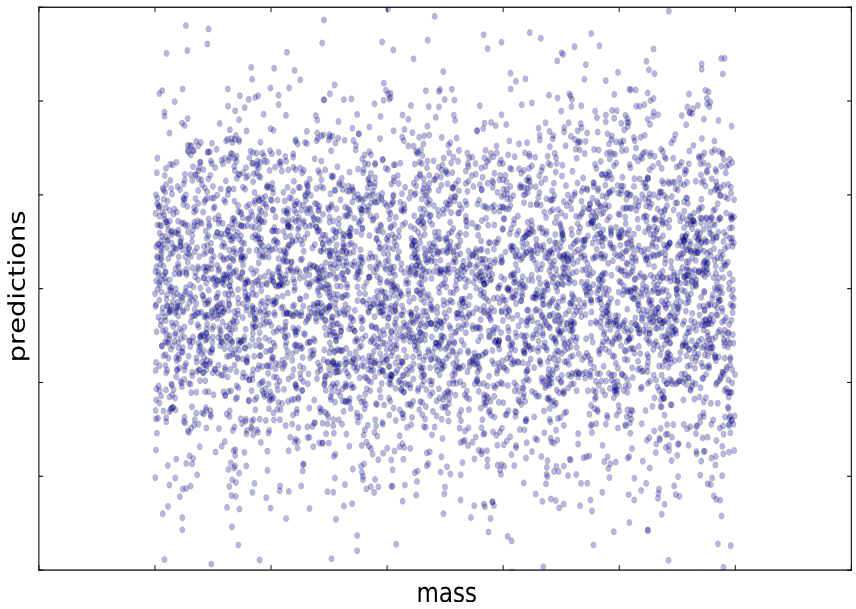
Typical solution: choose such features which don't give an ability to reconstruct 'mass' (or other selected 'uniform variables').

What is uniformity?

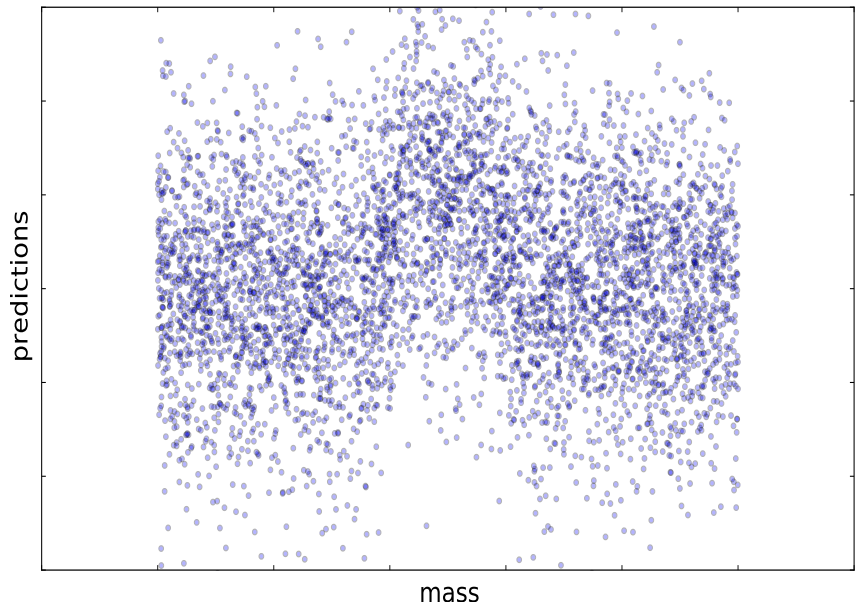
Predictions of some classifier are called *uniform* in variables $var_1, \ldots, var_n$ if prediction and set of this variables is *statistically independent*.
This (and only this) guarantees that any cut of prediction of classifier will produce the same efficiency in every region over $var_1, \ldots, var_n$

What is uniformity?

Predictions of some classifier are called *uniform* in variables $var_1, \ldots, var_n$ if prediction and set of this variables is *statistically independent*.
This (and only this) guarantees that any cut of prediction of classifier will produce the same efficiency in every region over $var_1, \ldots, var_n$



(a) Uniform predictions

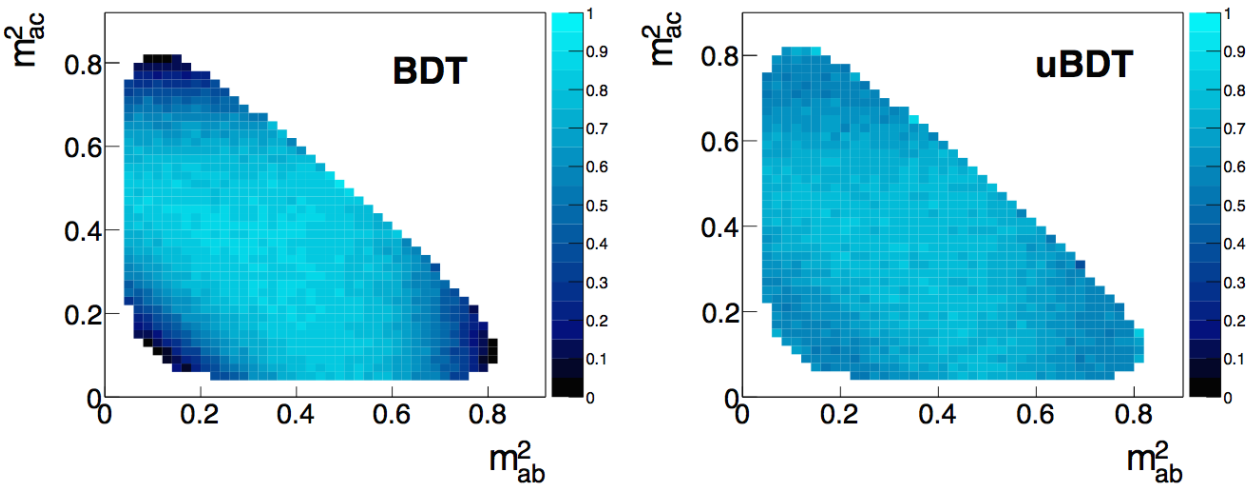


(b) Non-uniform

Boosting to uniformity

Previous work: **uBoost**

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



Model selection efficiency distributions for $\bar{\varepsilon} = 70\%$ from (left) AdaBoost and (right) uBoost.

In this talk I present alternate methods for producing BDTs with uniform selection efficiency.

Boosting: Gradient Boosting with plain AdaLoss

Gradient boosting on trees is widely used algorithm, it's built upon decision tree regressors with usage of some loss function.
Usual AdaLoss ($y_i = +1$ for signal, -1 for background):

$$L_{\text{ada}} = \sum_{i \in \text{events}} \exp[-\text{score}_i \, y_i]$$

Gradient boosting tries to minimize the loss function by 'steepest descent'-like procedure.
Pseudo-residual of AdaLoss:

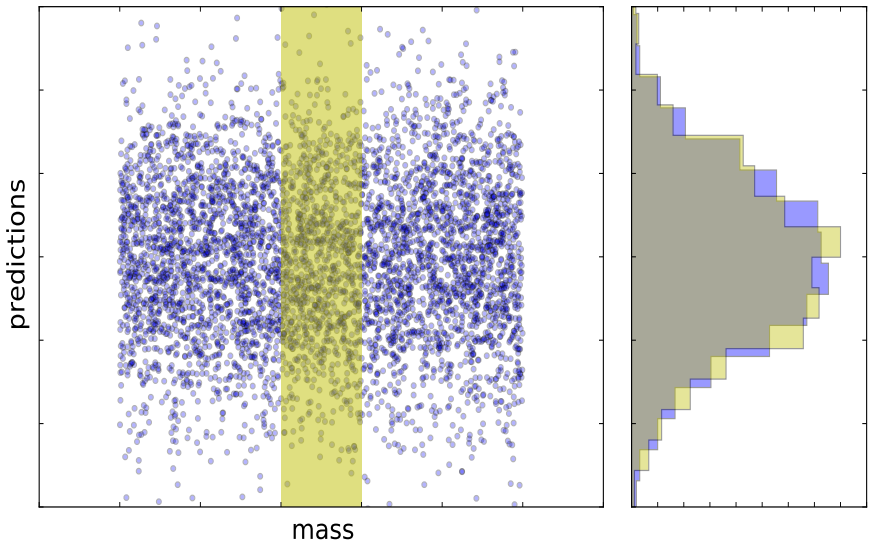
$$-\frac{\partial L_{\text{ada}}}{\partial \, \text{score}_i} = y_i \exp[-\text{score}_i \, y_i],$$

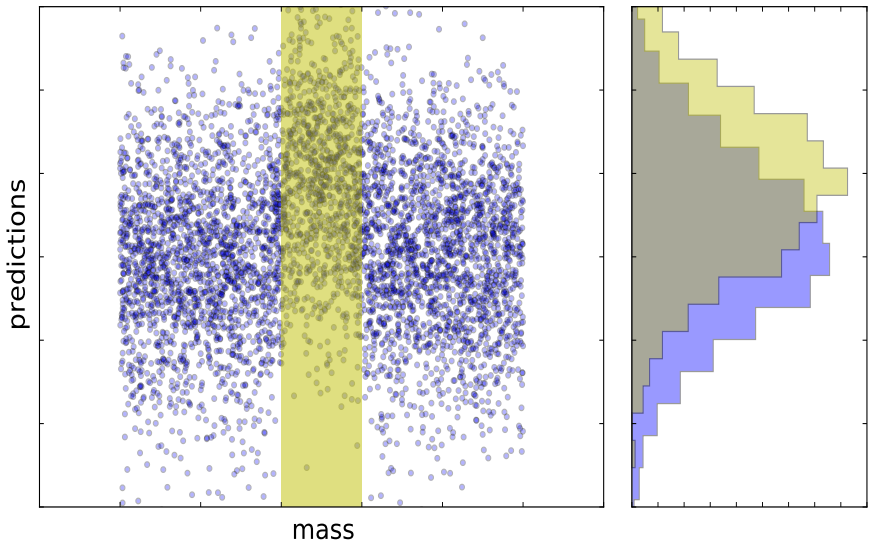
We need to introduce some loss, which penalizes non-unifomity.

Similarity-based approach for measuring uniformity

Idea: uniformity means that distribution of predictions in every bin is equal.

Let's compare the global distribution (blue hist) with distibution in one bin (yellow hist). Yellow rectangle shows the events in selected bin over mass.





Similarity-based approach

Let $F(x) = \mathrm{P}(\textit{prediction} < x)$ — cdf of all predictions,
$F_{\text{bin}}(x) = \mathrm{P}(\textit{prediction in bin} < x)$ — cdf of predictions in bin over mass.
Define $\text{weight}_{\text{bin}} = \frac{\text{weight of events in bin}}{\text{weight of all events}}$

Kolmogorov-Smirnov measure (uninformative)

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

Cramér–von Mises similarity

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

Boosting: Gradient Boosting with FlatnessLoss (uGBFL)

CvM measure of non-uniformity:

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

Let's modify this function:

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

so that it becomes differentiable

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

Boosting: Gradient Boosting with FlatnessLoss (uGBFL)

FL doesn't take into account the quality of predictions, only uniformity.
So what we use in practice is linear combination of FlatnessLoss and AdaLoss:

$$\text{loss} = \text{FL} + \alpha\, L_{\text{ada}}$$

First one penalizes non-uniformity, second one — poor predictions, α is usually taken small.

Boosting: Gradient Boosting with *knn*AdaLoss (uGBkNNAdaLoss)

As another approach we define: *kNN*AdaLoss:

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

It can be written as particular case of:

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

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

This is one particular choice of a_{ij};
in general case matrix a_{ij} even may be non-square.

*knn*AdaBoost

Usual AdaBoost reweighting procedure (p_i is prediction of last classifier):

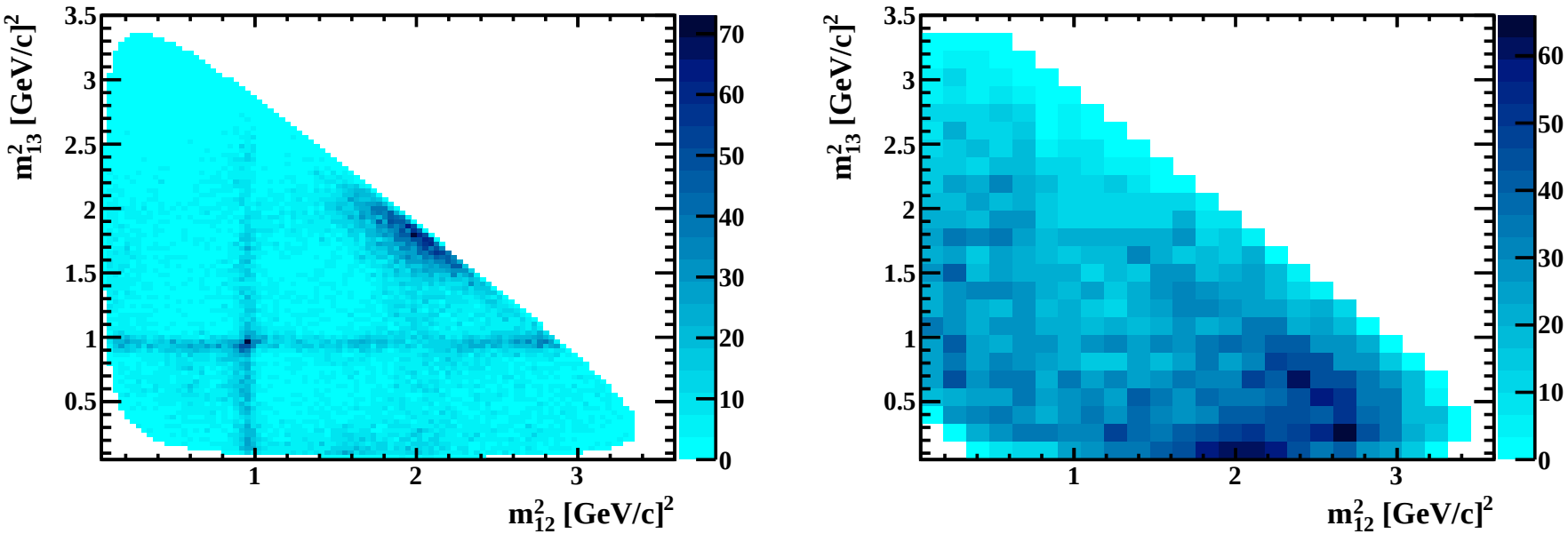
$$w_i' = w_i \times \exp[-y_i\, p_i],$$

*knn*AdaBoost uses mean of predictions of neighbours

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

(neighbours are of the same class).
Thus boosting focuses not on the events that were poorly classified, but on the regions with poor classification.

Example analysis: distibutions



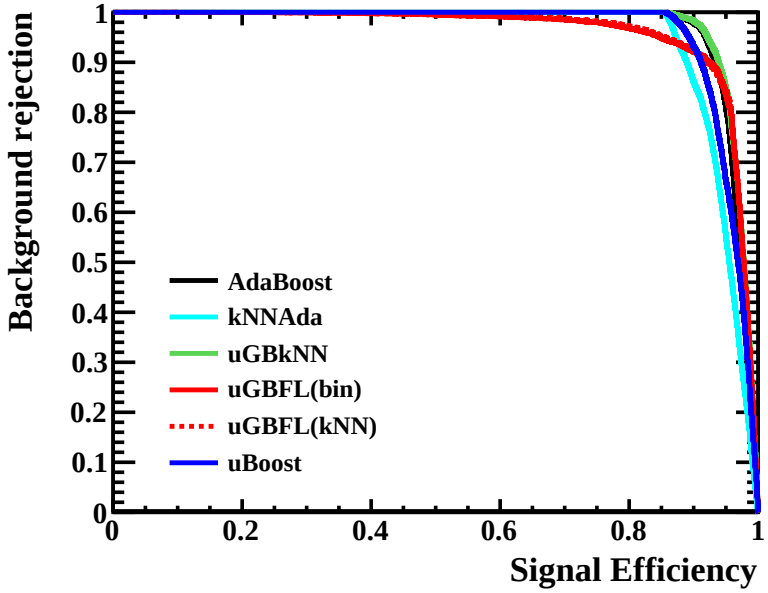
Dalitz-plot distributions for (left) signal and (right) background for the $D_s^{\pm} \to \pi^+\pi^-\pi^{\pm}$. The three pions are labeled here as 1, 2 and 3 and ordered according to increases momentum.

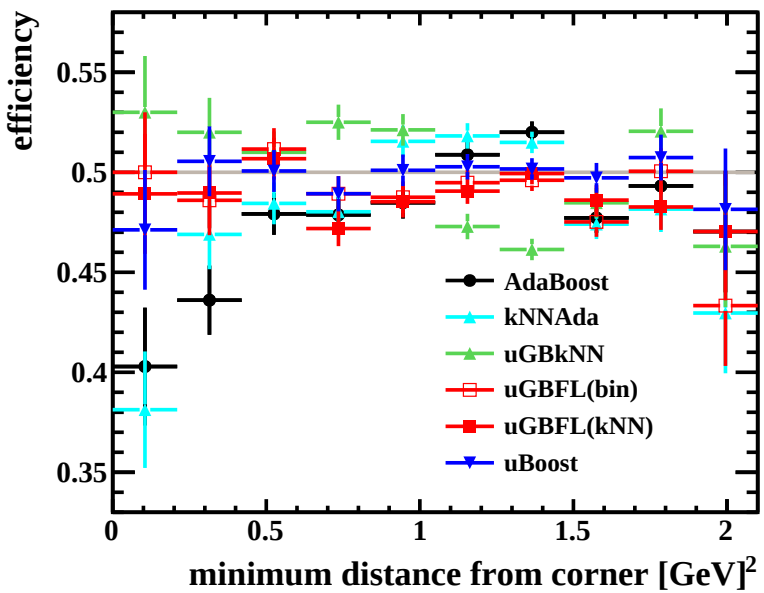
Abbreviations

Table : Description of uniform boosting algorithms.

Name	Description
uBoost	algorithm introduced in [arXiv:1305.7248]
kNNAda	AdaBoost modification using averaging over knn
uGBkNN	gradient boost using kNNAdaLoss loss function
uGBFL(bin)	gradient boost using flatness loss $+\alpha$ AdaLoss
uGBFL(kNN)	same except kNN events are used rather than bins

Example analysis



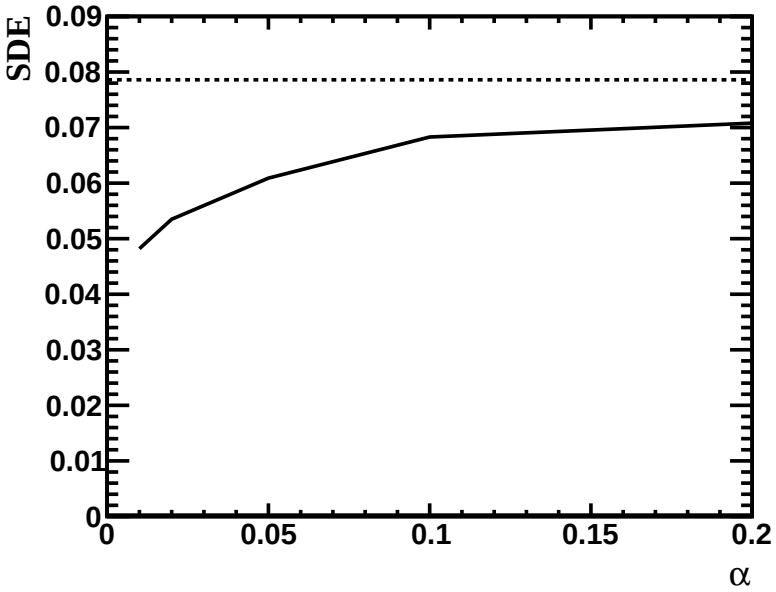


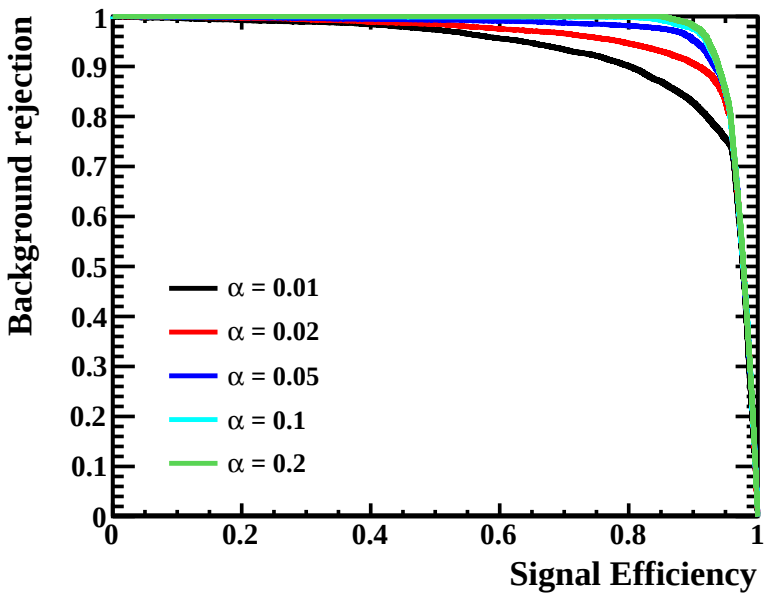
(left) ROC curves for classifiers.
(right) Efficiency *vs* distance to a corner of the Dalitz-plot. An arbitrary working point of 50% integrated efficiency is displayed.

AdaBoost produces a much lower efficiency near corners of Dalitz plot, while uGBFL and uBoost are consistent with flat.

Example analysis, tradeoff

Increasing of α parameter in uGBFL results in better quality of classification, but less uniformity of predictions.

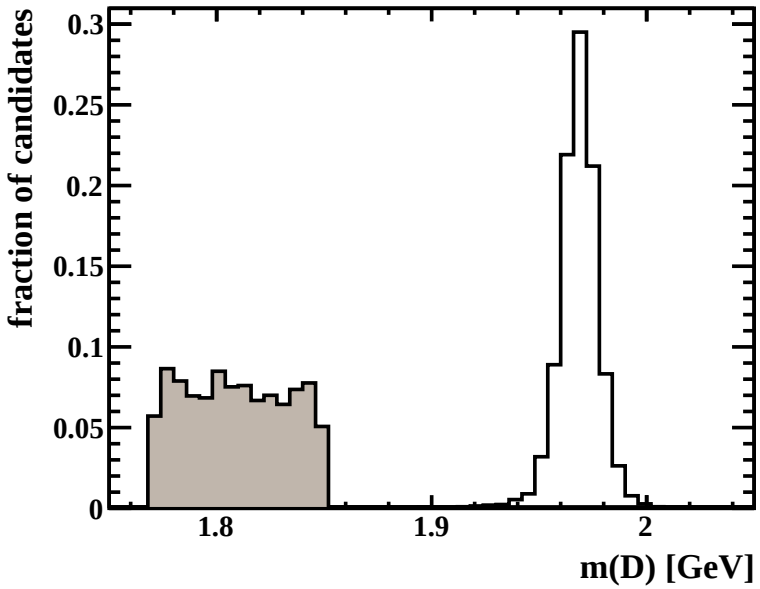


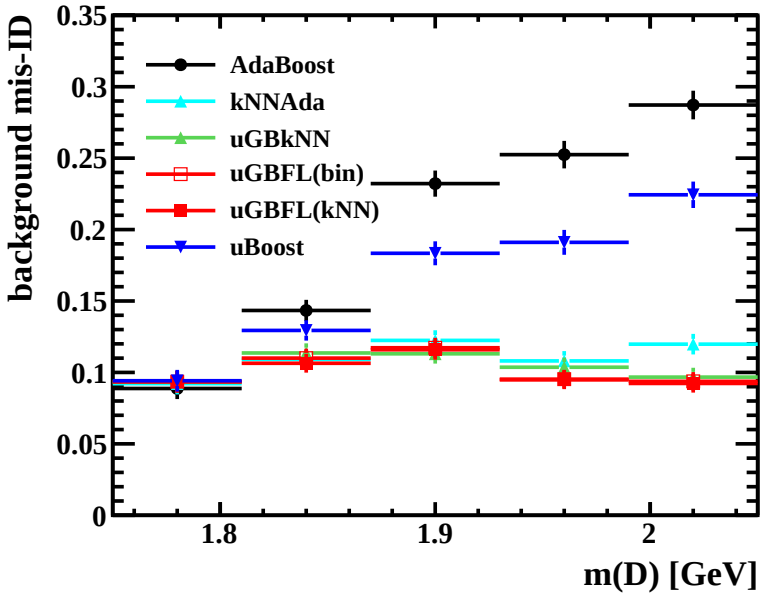


(left) SDE for uGBFL(bin) for different values of α,
SDE = Standard Deviation of Efficiency (0 = perfectly uniform).

(right) ROC curves for uGBFL(bin) for different values of α.

Example analysis 2: training on mass sideband





(left) Signal and (filled) background samples used in training.
(right) Background mis-identification *vs* D candidate mass for an arbitrary working point of 10% background mis-identification in the training region $1.75 < m(D) < 1.85$ GeV is displayed.

Summary on classifiers

New classifiers

- faster (than uBoost)
- can target at uniformity in *both* signal and bck
- *knn*AdaBoost and gradient boosting with knnAdaLoss can be easily implemented, but don't seem to produce good uniformity
- uGBFL is highly tunable and proved to be able to fight severe correlation

uGBFL will be useful for:

- Dalitz-plot or angular/amplitude analyses (as shown);
- mass measurements since both signal and background can be kept unbiased near mass peak
- searches for new particles since efficiency can be kept optimal for mass, lifetime values not used in the generated training samples
- etc

Summary on metrics

- three metrics were introduced (details in the backup)
- despite their difference, the results obtained with metrics proposed are **similar**.
- for higher dimensions: k-nearest neighbours modifications of metrics are available (instead of binning over uniform variables, we can compute nearest neighbours in the space of uniform variables).

Details: backup or article.

Read:
`http://arxiv.org/abs/1410.4140`

Try out (python implementation):
`https://github.com/anaderi/lhcb_trigger_ml`

Feel free to email
`mwill@mit.edu`
`alex.rogozhnikov@yandex.ru`
with any questions or comments.

Q&A

Backup

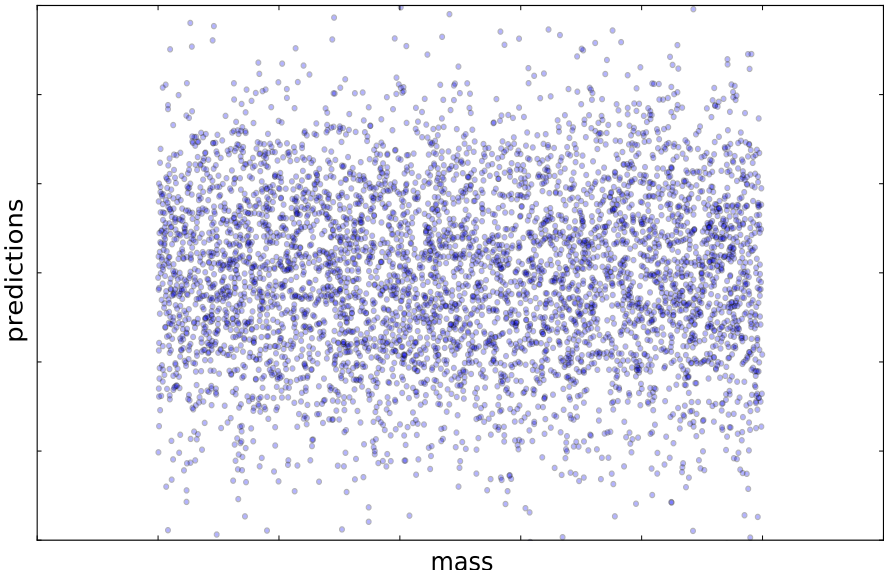
Desirable properties of metrics

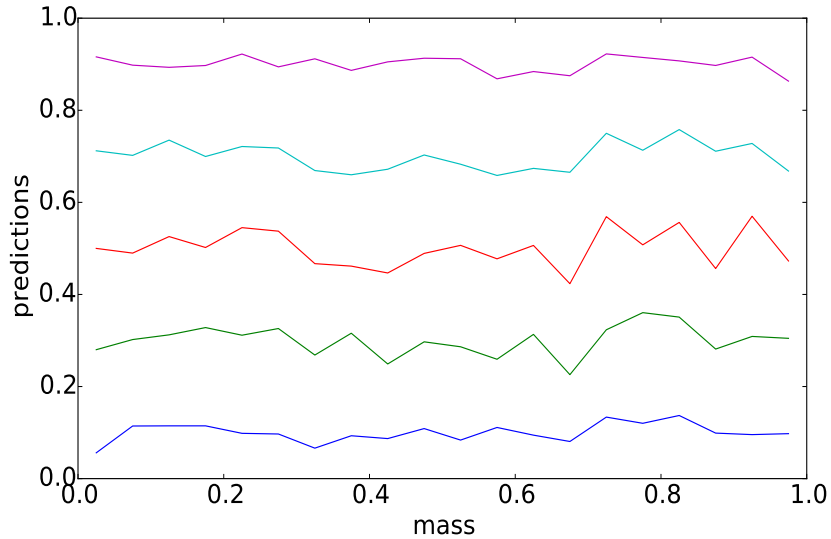
The metric should ...

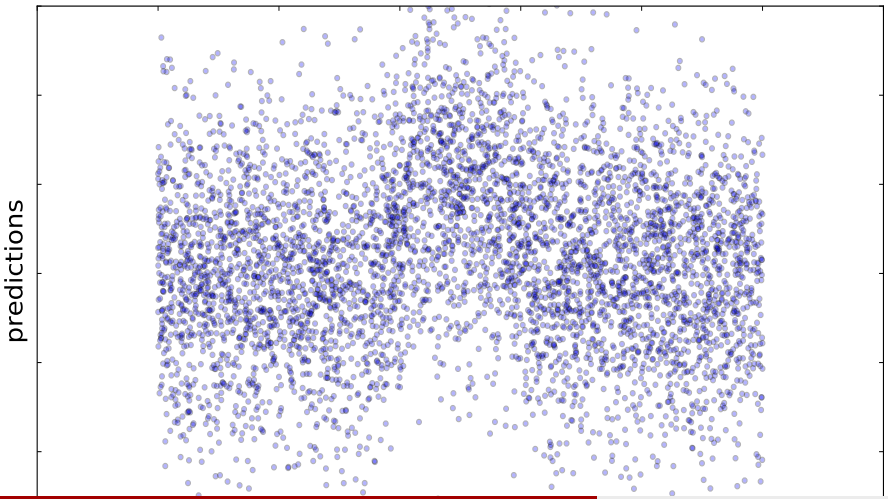
1. not depend strongly on the number of events used to test uniformity
2. not depend on the total weight
3. depend on order of predictions, not the exact values of predictions (example: Pearson correlation does not satisfy this property)
4. be stable against free parameters (number of bins, k in knn)

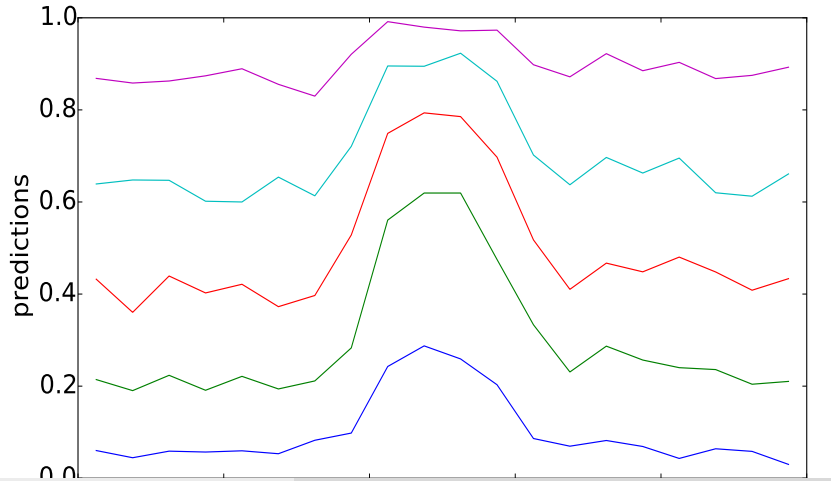
Cut-based approach (1/2)

Select some set of efficiencies (in examples: 0.1, 0.3, 0.5, 0.7, 0.9), for each one can compute global cut and look at efficincies in each bin:









Cut-based approach (2/2)

Standard deviation of efficiency

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

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

Theil index of $x_1, \ldots, x_n$

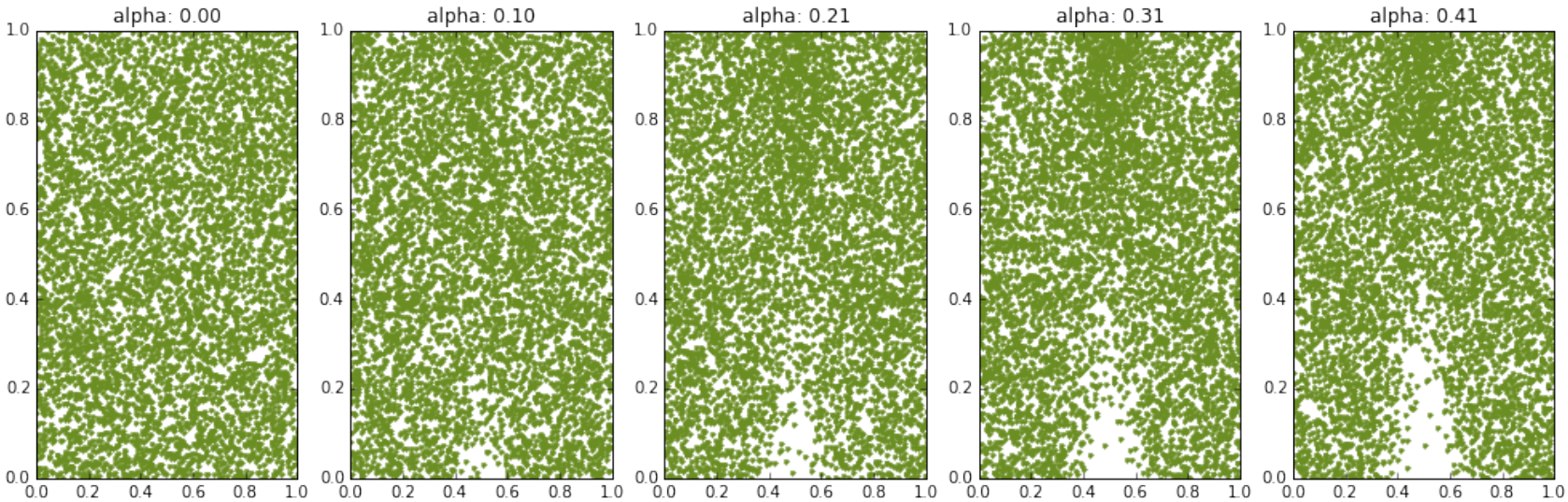
$$\mathrm{Theil} = \frac{1}{N} \sum_i \frac{x_i}{< x >} \ln \frac{x_i}{< x >},$$

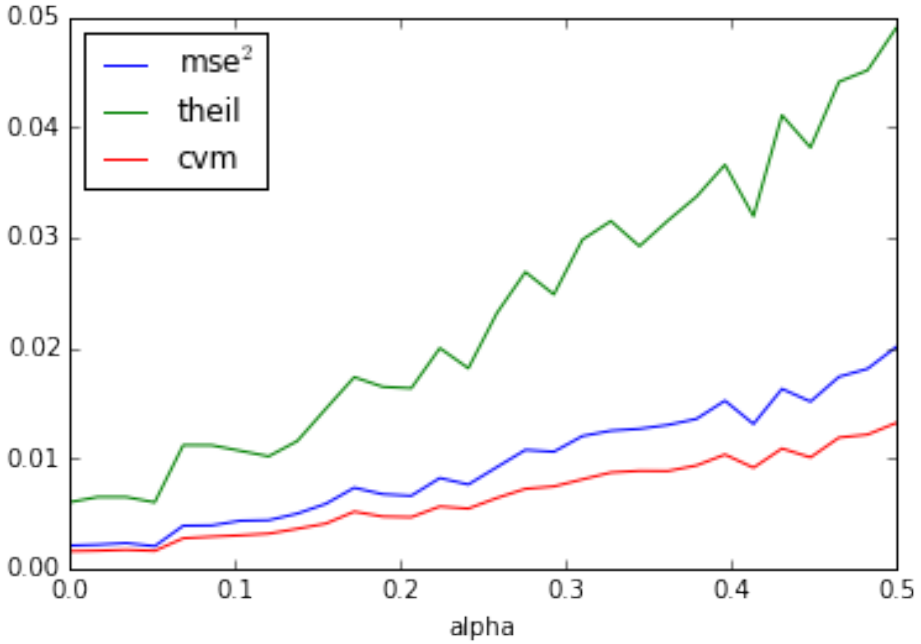
Theil index of efficiency

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

$$\mathrm{Theil} = \frac{1}{k} \sum_{\mathrm{eff} \in [\mathrm{eff}_1 \ldots \mathrm{eff}_k]} \mathrm{Theil}(\mathrm{eff}).$$

Example


alpha: 0.00
alpha: 0.10
alpha: 0.21
alpha: 0.31
alpha: 0.41


mse^2
theil
cvm
alpha

Timings of classifiers

One drawback of the uBoost technique is that it has a high degree of computational complexity: while AdaBoost trains M trees (a user-defined number), uBoost builds $100 \times M$ trees. The algorithms presented only build M trees; however, the boosting involves some more complicated algorithms. Training each of the M trees scales as follows for N training events:

- kNNAdaBoost: $O(k \times N)$;
- uGBkNNknn: $O(k \times N)$ for A_{knn}, and $O(\#\text{nonzero elements in the matrix})$ for arbitrary matrix A;
- uGBFL(bin): $O(N \ln N)$;
- uGBFL(kNN): $O(N \ln N + Nk \ln k)$.

Usually the training time for these new algorithms is within a factor of two the same as AdaBoost.