Title

Learning from many collider events at once

Permalink

https://escholarship.org/uc/item/1009q5wn

Journal

Physical Review D, 103(11)

ISSN

2470-0010

Authors

Nachman, Benjamin
Thaler, Jesse

Publication Date

2021-06-01

DOI

10.1103/physrevd.103.116013

Peer reviewed

MIT-CTP 5271

E Pluribus Unum Ex Machina:
Learning from Many Collider Events at Once

Benjamin Nachman[1, 2, *] and Jesse Thaler[3, 4, †]

[1] *Physics Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA*
[2] *Berkeley Institute for Data Science, University of California, Berkeley, CA 94720, USA*
[3] *Center for Theoretical Physics, Massachusetts Institute of Technology, Cambridge, MA 02139, USA*
[4] *The NSF AI Institute for Artificial Intelligence and Fundamental Interactions*

There have been a number of recent proposals to enhance the performance of machine learning strategies for collider physics by combining many distinct events into a single ensemble feature. To evaluate the efficacy of these proposals, we study the connection between single-event classifiers and multi-event classifiers under the assumption that collider events are independent and identically distributed (IID). We show how one can build optimal multi-event classifiers from single-event classifiers, and we also show how to construct multi-event classifiers such that they produce optimal single-event classifiers. This is illustrated for a Gaussian example as well as for classification tasks relevant for searches and measurements at the Large Hadron Collider. We extend our discussion to regression tasks by showing how they can be phrased in terms of parametrized classifiers. Empirically, we find that training a single-event (per-instance) classifier is more effective than training a multi-event (per-ensemble) classifier, as least for the cases we studied, and we relate this fact to properties of the loss function gradient in the two cases. While we did not identify a clear benefit from using multi-event classifiers in the collider context, we speculate on the potential value of these methods in cases involving only approximate independence, as relevant for jet substructure studies.

CONTENTS

I. INTRODUCTION

Modern machine learning techniques are being widely applied to enhance or replace existing analysis techniques across collider physics [1–6]. These approaches hold great promise for new particle searches, for Standard Model measurements, and for high-energy nuclear physics investigations. A subset of these proposals have advocated for a multi-event strategy whereby a machine-learned function acts on multiple collision events at the same time [7–14]. This multi-event (per-ensemble) strategy contrasts with more typical single-event (per-instance) machine learning methods that process one event at a time, although both strategies make use of many events during the training process.

Intuitively, an ensemble approach might seem like a more promising learning strategy because there is more information contained in $N > 1$ collision events than in one single event. There is, however, an important distinction between the amount of information contained in a data set and the amount of information needed to encode a machine-learned function. For this reason, there need not be a gain from using multi-event strategies over single-event strategies in the context of machine learning.

In this paper, we show that when directly compared on the same task, there is indeed no informational benefit from training a function that processes multiple events simultaneously compared to training a function that processes only a single event at a time. This fact can be easily understood from the statistical structure of collision data. To test for a practical benefit, we perform empirical comparisons of per-ensemble and per-instance methods on benchmark tasks relevant for the Large Hadron Collider (LHC), finding that single-event (per-instance) methods are more effective for the cases we studied.

[*] bpnachman@lbl.gov
[†] jthaler@mit.edu

To an excellent approximation, collider events are statistically independent and identically distributed (IID). In simulation, this is exactly true up to deficiencies in random number generators. In data, there are some small time-dependent effects from changing conditions and there are also some correlations between events introduced by detector effects with timescales longer than a typical bunch crossing. These event-to-event correlations, however, are truly negligible when considering the set of events typically used for physics analysis that are selected by triggers. The probability for two events next to each other in time to be saved by the triggers is effectively zero, since triggers save only a tiny fraction of events. The IID nature of collision events therefore ensures that the information content is the same for ensembles of events and for single events drawn from an ensemble.

In equations, the probability to observe N events x_i is

$$p(\{x_1,\ldots,x_N\}|\theta)=\prod_{i=1}^{N}p(x_i|\theta), \tag{1}$$

where θ represents possible parameters of the generative model, such as the physics process being studied or the values of coupling constants. The optimal classifier to distinguish whether events have been generated via θ_A or via θ_B depends only on the per-ensemble likelihood ratio [15]:

$$\frac{p(\{x_1,\ldots,x_N\}|\theta_A)}{p(\{x_1,\ldots,x_N\}|\theta_B)}=\prod_{i=1}^{N}\frac{p(x_i|\theta_A)}{p(x_i|\theta_B)}, \tag{2}$$

which by the IID assumption only depends on knowing the per-instance likelihood ratio $p(x_i|\theta_A)/p(x_i|\theta_B)$. This equality explains the informational equivalence of per-ensemble and per-event learning.

Given the simplicity of Eq. (2), why are we writing a whole paper on this topic (apart from the opportunity to invoke a gratuitously Latinate paper title that incorporates an aspiration for national unity)? The studies in Refs. [7–14] find that per-ensemble learning is effective for their respective tasks, in some cases arguing why per-instance learning is deficient. It is certainly true that a set of events $\{x_1,\ldots,x_N\}$ contains more information than a single event x_i drawn from this set. What we will show in this paper is that if one carefully combines the per-instance information, one can recover the per-ensemble benefit, with the potential for a substantially reduced training cost. We emphasize that our analysis does not contradict the studies in Refs. [7–14]; rather this work suggests the possibility of achieving the same or better results by replacing per-ensemble learning with per-instance learning. There may be specialized contexts where per-ensemble learning is superior, particularly if the training procedure itself can be made simpler, such as in the linear regression approach of Ref. [12]. This paper also gives us a chance to mention some facts about loss functions that are well known in the statistics literature but might not be as well appreciated in collider physics. Moving away from the IID case, we speculate on the relevance of our analysis for jet substructure tasks where there is a notion of approximate independence of emissions.

The remainder of this paper is organized as follows. In Sec. II, we provide the formal statistical basis for building multi-event classifiers from single-event classifiers, and vice versa, under the IID assumption. We also explain how regression tasks can be translated into the language of per-instance parametrized classification. In Sec. III, we present empirical studies that corroborate these analytic results. Our conclusions are given in Sec. IV.

II. THE STATISTICS OF PER-ENSEMBLE LEARNING

A. Review of Per-Instance Learning

Suppose that a collider event is represented by features in $\mathbb{E}=\mathbb{R}^M$ and we are trying to train a binary classifier to learn a target in $[0,1]$. Let $c:\mathbb{E}\to[0,1]$ be a function that processes a single event, with the goal of distinguishing events being generated by θ_A ($c\to 1$) versus those generated by θ_B ($c\to 0$). Such a function can be obtained by minimizing an appropriate loss functional, such as the binary cross entropy:

$$L_{\rm BCE}[c]=-\int dx\,\Big(p(x|\theta_A)\log c(x) + p(x|\theta_B)\log(1-c(x))\Big), \tag{3}$$

where $p(x|\theta)$ is the probability density of $x\in\mathbb{E}$ given class θ. Here and throughout this discussion, we consider the infinite statistics limit such that we can replace sums over events by integrals. We have also dropped the prior factors $p(\theta_i)$, assuming that one has equal numbers of examples from the two hypotheses during training. While this is often true in practice, it is not strictly necessary for our main conclusions, though it does simplify the notation. It is well-known [16, 17] (also in high-energy physics [18–30]) that an optimally trained c will have the following property:

$$\frac{c(x)}{1-c(x)}=\frac{p(x|\theta_A)}{p(x|\theta_B)}, \tag{4}$$

such that one learns the per-instance likelihood ratio. By the Neyman–Pearson lemma [15], this defines the optimal single-event classifier.

There are many loss functionals that satisfy this property. Consider a more general loss functional that depends on a learnable function $f:\mathbb{E}\to\mathbb{R}$ (which unlike c may or may not map to $[0,1]$) as well as fixed rescaling functions $A:\mathbb{R}\to\mathbb{R}$ and $B:\mathbb{R}\to\mathbb{R}$:

$$L[f]=-\int dx\,\Big(p(x|\theta_A)\,A(f(x))+p(x|\theta_B)\,B(f(x))\Big). \tag{5}$$

Loss Name	$A(f)$	$B(f)$	$\text{argmin}_f\, L[f]$	Integrand of $-\min_f L[f]$	Related Divergence/Distance
Binary Cross Entropy	$\log f$	$\log(1-f)$	$\frac{p_A}{p_A+p_B}$	$p_A \log \frac{p_A}{p_A+p_B} + (A \leftrightarrow B)$	$2\big(\text{Jensen-Shannon} - \log 2\big)$
Mean Squared Error	$-(1-f)^2$	$-f^2$	$\frac{p_A}{p_A+p_B}$	$-\frac{p_A p_B}{p_A+p_B}$	$\frac{1}{2}\big(\text{Triangular} - 1\big)$
Square Root	$\frac{-1}{\sqrt{f}}$	$-\sqrt{f}$	$\frac{p_A}{p_B}$	$-2\sqrt{p_A p_B}$	$2\big(\text{Hellinger}^2 - 1\big)$
Maximum Likelihood Cl.	$\log f$	$1-f$	$\frac{p_A}{p_B}$	$p_A \log \frac{p_A}{p_B}$	Kullback–Leibler

TABLE I. Examples of loss functionals in the form of Eq. (5), with the associated location and value of the loss minimum, using the shorthand $p_i \equiv p(x|\theta_i)$. We have used the symbol f in all cases to denote the classifier, but some choices require explicit constraints on f to be either non-negative or in the range $[0,1]$. In the last column, we indicate the relation of the loss minimum to statistical divergences and distances, up to an overall scaling and offset. See Ref. [31] for additional relations.

Taking the functional derivative with respect to $f(x)$, the extremum of $L[f]$ satisfies the property:

$$-\frac{B'(f(x))}{A'(f(x))} = \frac{p(x|\theta_A)}{p(x|\theta_B)}. \tag{6}$$

As long as $-B'(f)/A'(f)$ is a monotonic rescaling of f and the overall loss functional is convex, then the function $f(x)$ learned by minimizing Eq. (5) defines an optimal classifier. In many cases, the minimum value of $L[f]$ itself is interesting in the context of statistical divergences and distances [31], and a few examples are shown in Table I.

To simplify the following discussion, we will focus on the "maximum likelihood classifier" (MLC) loss:

$$L_{\text{MLC}}[f] = -\int dx\, \Big(p(x|\theta_A) \log f(x) \\ + p(x|\theta_B)\,(1 - f(x))\Big). \tag{7}$$

This is of the general form in Eq. (5) with $A(f) = \log f$ and $B(f) = 1 - f$. To our knowledge, the MLC was first introduced in the collider physics context in Refs. [32, 33], although with an exponential parametrization of $f(x)$. We call Eq. (7) the MLC loss to distinguish it from the related maximum likelihood loss that is often used to fit generative models [34–36]. Using Eq. (6), the minimum of this loss functional yields directly the likelihood ratio:

$$\underset{f}{\text{argmin}}\, L_{\text{MLC}}[f] = \frac{p(x|\theta_A)}{p(x|\theta_B)}, \tag{8}$$

which will be useful to simplify later analyses.[1] The MLC loss functional value at the minimum is

$$-\min_f L_{\text{MLC}}[f] = \int dx\, p(x|\theta_A) \log \frac{p(x|\theta_A)}{p(x|\theta_B)}, \tag{9}$$

which is the Kullback–Leibler (KL) divergence, also known as the relative entropy from $p(x|\theta_B)$ to $p(x|\theta_A)$. See App. A for an intuitive derivation of Eq. (7).

B. Per-Ensemble Binary Classification

To move from single-event classification to multi-event classification, we want to learn a classification function f_N that can process N events simultaneously. Here, we are using $f_N : \mathbb{E}^N \to \mathbb{R}$ instead of $c_N : \mathbb{E}^N \to [0,1]$ to avoid algebraic manipulations like Eq. (4). We will use the vector notation

$$\vec{x} = \{x_1, \ldots, x_N\} \tag{10}$$

to represent an element of $\mathbb{E}^N$. Our goal is to distinguish whether $\vec{x}$ is drawn from $p(\vec{x}|\theta_A)$ ($f_N \to \infty$) or from $p(\vec{x}|\theta_B)$ ($f_N \to 0$). Note that we are trying to classify a pure event ensemble as coming from either θ_A or θ_B, which is a different question than trying to determine the proportion of events drawn from each class in a mixed event ensemble. For $N = 1$, f_1 is the same as f discussed in Eq. (5).

If f_N is trained optimally, then the classification performance of f_N evaluated on $N > 1$ events will be better than the performance of f_1 evaluated on a single event, as relevant to the discussions in Refs. [7–14]. The key point of this paper is that one can construct a classifier $f_{1\to N}$ that is built only from f_1, acts on N events, and has the same asymptotic performance as f_N.

Using the MLC loss in Eq. (7), but now applied to N events, we have

$$L_{\text{MLC}}[f_N] = -\int d^N x\, \Big(p(\vec{x}|\theta_A) \log f_N(\vec{x}) \\ + p(\vec{x}|\theta_B)\,(1 - f_N(\vec{x}))\Big), \tag{11}$$

whose minimum is the per-ensemble likelihood ratio:

$$\underset{f_N}{\text{argmin}}\, L_{\text{MLC}}[f_N] = \frac{p(\vec{x}|\theta_A)}{p(\vec{x}|\theta_B)}. \tag{12}$$

[1] A variation of Eq. (8) holds for $A(f) = \log C(f)$ and $B(f) = 1 - C(f)$, where $C(f)$ is any monotonically increasing function with range that covers $(0, \infty)$. In this case, $C(\text{argmin}_f\, L[f]) = p(x|\theta_A)/p(x|\theta_B)$. This can be useful in practice if $C(f)$ is everywhere positive, since f can take on negative values and still yield a valid likelihood ratio. See Fig. 10 for an empirical study of $C(f) = \exp f$.

By the Neyman–Pearson lemma, this yields the optimal per-ensemble classifier.

On the other hand, once we have trained a single-event classifier f_1 using Eq. (7), we can build a multi-event classifier $f_{1\to N}$ without any additional training:

$$f_{1\to N}(\vec{x}) \equiv \prod_{i=1}^{N} f_1(x_i) \quad \to \quad \frac{p(\vec{x}|\theta_A)}{p(\vec{x}|\theta_B)}, \tag{13}$$

where in the last step we have combined the solution found in Eq. (8) with the IID condition in Eq. (2). Whereas minimizing Eq. (11) requires sampling over $\mathbb{E}^N$, constructing $f_{1\to N}$ only requires sampling over $\mathbb{E}$, which is a considerable reduction in computational burden for large N. The technical details of carrying out this procedure are explained in Sec. III A.

Going in the converse direction, we can learn a single-event classifier $f_{N\to 1}$ starting from a constrained multi-event classifier $\tilde{f}_N$. Using weight sharing, we can minimize Eq. (11) subject to the constraint that $\tilde{f}_N$ takes the functional form:

$$\tilde{f}_N(\{x_1, \ldots, x_N\}) = \prod_{i=1}^{N} f_{N\to 1}(x_i), \tag{14}$$

where $f_{N\to 1}(x)$ is a learnable function. Under the IID assumption, $\tilde{f}_N$ can still learn the per-ensemble likelihood ratio, but the learned $f_{N\to 1}(x)$ will now be the per-instance likelihood ratio, at least asymptotically.[2] An examination of this converse construction is presented in Sec. III B.

C. Comparing the Loss Gradients

We have shown that the per-ensemble classifier f_N and the composite per-event classifier $f_{1\to N}$ have the same asymptotic information content, but one might wonder if there is nevertheless a practical performance gain to be had using per-ensemble learning.

Under the IID assumption, the optimal f_N takes the form of $\tilde{f}_N$ in Eq. (14), and in our empirical studies, we found no benefit to letting f_N have more functional freedom. Therefore, to get a sense of the efficacy of per-ensemble versus per-instance training, we can compare the effective loss functions for $f_{N\to 1}$ and f_1. Since the inputs and outputs of these functions are the same (i.e. $\mathbb{E} \to \mathbb{R}$), we can do an apples-to-apples comparison of their behavior under gradient descent. The following analysis assumes that the neural network training occurs in the vicinity of the global minimum of the loss function.

For the per-ensemble case, plugging Eq. (14) into Eq. (11) and using the IID relation in Eq. (1), we find the effective loss functional:

$$\begin{aligned} L_{\text{MLC}}[f_{N\to 1}] + 1 = -N \int dx\, p(x|\theta_A) \log f_{N\to 1}(x) \\ + \left(\int dx\, p(x|\theta_B) f_{N\to 1}(x) \right)^N . \end{aligned} \tag{15}$$

This is to be contrasted with the per-instance loss functional from Eq. (7), repeated for convenience with the f_1 notation and typeset to be parallel to the above:

$$\begin{aligned} L_{\text{MLC}}[f_1] + 1 = -\int dx\, p(x|\theta_A) \log f_1(x) \\ + \int dx\, p(x|\theta_B)\, f_1(x). \end{aligned} \tag{16}$$

To understand the loss gradients, we can Taylor expand the learned functions about the optimal solution:

$$f_{N\to 1}(x) = \frac{p(x|\theta_A)}{p(x|\theta_B)} + \epsilon(x), \tag{17}$$

$$f_1(x) = \frac{p(x|\theta_A)}{p(x|\theta_B)} + \epsilon(x). \tag{18}$$

Plugging these into their respective loss functionals and looking at the leading-order variations, we have:

$$\begin{aligned} \frac{\delta L_{\text{MLC}}[f_{N\to 1}]}{N} = \int dx\, \frac{\big(p(x|\theta_B)\, \epsilon(x)\big)^2}{2\, p(x|\theta_A)} \\ + \frac{N-1}{2} \left(\int dx\, p(x|\theta_B)\, \epsilon(x) \right)^2 , \end{aligned} \tag{19}$$

$$\delta L_{\text{MLC}}[f_1] = \int dx\, \frac{\big(p(x|\theta_B)\, \epsilon(x)\big)^2}{2\, p(x|\theta_A)}. \tag{20}$$

These expressions are quadratic in $\epsilon(x)$, which means that we are expanding around the correct minimum.

The expression for $\delta L_{\text{MLC}}[f_1]$ involves a single integral over x, so under gradient descent, the value of $\epsilon(x)$ can be independently adjusted at each point in phase space to find the minimum. By contrast, $\delta L_{\text{MLC}}[f_{N\to 1}]$ has an additional piece involving an integral squared, so even if at a given point in phase space x_0 we have achieved $\epsilon(x_0) = 0$, gradient descent will tend to push $\epsilon(x_0)$ away from the correct value until $\epsilon(x) = 0$ everywhere. This correlated structure explains the slower convergence of $L_{\text{MLC}}[f_{N\to 1}]$ compared to $L_{\text{MLC}}[f_1]$ in our empirical studies. While we focused on the MLC loss to simplify the algebra, the appearance of these (typically counterproductive) correlations in the loss gradient appears to be a generic feature of per-ensemble learning.

D. Per-Ensemble Regression

While the discussion above focused on binary classification, the same basic idea applies to regression problems

[2] In the case that the two samples are composed of mixtures of two categories, then the learned $f_{N\to 1}(x)$ will be the ratio of the mixed sample likelihoods, which is monotonically related to the optimal pure sample classifier, as discussed in Ref. [37].

as well. The goal of regression is to infer parameters θ from the data $\vec{x}$. There are a variety of approaches that can be used for this task, and each can be connected to parametrized per-instance classification.

1. *Maximum Likelihood*

Maximum likelihood is the most common strategy for inference in collider physics. Symbolically, we are trying to find

$$\theta_{\rm ML} = \underset{\theta}{\rm argmax}\, p(\vec{x}|\theta). \tag{21}$$

One way to determine $\theta_{\rm ML}$ is with a two-step approach. First, one can train a parametrized classifier $f(x,\theta)$ [26, 38] using, e.g., the per-instance MLC loss:

$$\begin{aligned} L_{\rm MLC}[f] = -\int dx\, \Big(& p(x|\theta)\, p(\theta) \log f(x,\theta) \\ &+ p(x|\theta_0)\, p(\theta)\, (1 - f(x,\theta))\Big). \end{aligned} \tag{22}$$

The top line corresponds to a synthetic dataset where every event is generated from $p(x|\theta)$ with different θ values drawn from the probability density $p(\theta)$. The bottom line corresponds to a synthetic dataset where every event is generated using the same $p(x|\theta_0)$ for fixed θ_0 and then augmented with a value θ that follows from $p(\theta)$ independently of x. Minimizing Eq. (22) with respect to $f(x,\theta)$, the asymptotic solution is the likelihood ratio:

$$f(x,\theta) = \frac{p(x|\theta)}{p(x|\theta_0)}, \tag{23}$$

where the factors of $p(\theta)$ have canceled out. Second, one can estimate $\theta_{\rm ML}$ by using the IID properties of the event ensemble to relate likelihoods to the classifier output $f(x,\theta)$:

$$\begin{aligned} \theta_{\rm ML} &= \underset{\theta}{\rm argmin} \left\{ -\sum_{i=1}^{N} \log p(x_i|\theta) \right\} \\ &= \underset{\theta}{\rm argmin} \left\{ -\sum_{i=1}^{N} \log \frac{p(x_i|\theta)}{p(x_i|\theta_0)} \right\} \\ &\approx \underset{\theta}{\rm argmin} \left\{ -\sum_{i=1}^{N} \log f(x_i,\theta) \right\}. \end{aligned} \tag{24}$$

Thus, even though maximum likelihood regression uses information from the full event ensemble, only a parametrized per-instance classifier is required for this procedure.

2. *Classifier Loss*

Two recent proposals for parameter estimation are explicitly built on classifiers for regression [18, 19]. For any classifier, one can perform the following optimization:[3]

$$\theta_{\rm CL} = \underset{\theta'}{\rm argmax} \left\{ \begin{matrix} \text{Loss of a classifier trained} \\ \text{to distinguish } \theta' \text{ from } \theta_{\rm data} \end{matrix} \right\}. \tag{25}$$

Here, we are imagining that the θ' samples come from synthetic data sets. The appearance of a maximum instead of minimum in Eq. (25) is because, as highlighted in Table I, it is negative loss functions that correspond to statistical divergences and distances.

In general, the $\theta_{\rm CL}$ that minimizes the classifier loss will be different from the $\theta_{\rm ML}$ that maximizes the likelihood. For the special case of the MLC loss, though, they are the same in the asymptotic limit if we set $\theta_A = \theta_{\rm data}$ and $\theta_B = \theta'$. To see this, recall from Eq. (9) that after training, the value of the MLC loss is related to the KL divergence:

$$\begin{aligned} &\underset{\theta'}{\rm argmax}\{\min_f L_{\rm MLC}[f]\} \\ &\quad = \underset{\theta'}{\rm argmax} \left\{ -\int dx\, p(x|\theta_{\rm data}) \log \frac{p(x|\theta_{\rm data})}{p(x|\theta')} \right\} \\ &\quad \approx \underset{\theta'}{\rm argmax} \left\{ \sum_{i=1}^{N} \log \frac{p(x_i|\theta')}{p(x_i|\theta_{\rm data})} \right\} \\ &\quad = \underset{\theta'}{\rm argmin} \left\{ -\sum_{i=1}^{N} \log p(x_i|\theta') \right\} \\ &\quad = \theta_{\rm ML}\,, \end{aligned} \tag{26}$$

where the sum is over data events.

3. *Direct Regression*

In terms of information content, a regression model trained in the usual way can be built from a parametrized classification model. Suppose that $\theta \in \mathbb{R}^Q$ and $g_N : \mathbb{E}^N \to \mathbb{R}^Q$ is a regression model trained with the mean squared error loss:

$$L_{\rm MSE}[g_N] = -\int d^n x\, p(\vec{x},\theta) \Big(g_N(\vec{x}) - \theta\Big)^2 \tag{27}$$

It is well known that the optimally trained g_N will be related to the expectation value of θ:

$$g_N(\vec{x}) = \mathbb{E}[\theta|\vec{x}] = \int d\theta\, \theta\, p(\theta|\vec{x}). \tag{28}$$

Other loss functions approximate other statistics, as discussed in Ref. [39]. For example, the mean absolute error loss approximates the median of θ. Ultimately, all direct regression methods are functionals of $p(\theta|\vec{x})$.

[3] Note that Ref. [18] used the (non-differentiable) area under the curve instead of the classifier loss, as it is not sensitive to differences in the prior $p(\theta)$ between the two data sets.

We can relate $p(\theta|\vec{x})$ to a parametrized classifier $f_N(\vec{x},\theta)$ trained to distinguish θ from a baseline θ_0:

$$\begin{aligned} p(\theta|\vec{x}) = \frac{p(\vec{x}|\theta)\,p(\theta)}{p(\vec{x})} &= \frac{p(\vec{x}|\theta)\,p(\theta)}{\int d\theta'\,p(\vec{x}|\theta')\,p(\theta')} \\ &= \frac{\frac{p(\vec{x}|\theta)}{p(\vec{x}|\theta_0)}\,p(\theta)}{\int d\theta'\,\frac{p(\vec{x}|\theta')}{p(\vec{x}|\theta_0)}\,p(\theta')} \\ &= \frac{f_N(\vec{x},\theta)\,p(\theta)}{\int d\theta'\,f_N(\vec{x},\theta')\,p(\theta')}, \end{aligned} \tag{29}$$

where $p(\theta)$ is the probability density of θ used during the training of g_N. Following the same logic as Sec. II B, the per-ensemble classifier $f_N(\vec{x},\theta)$ can be related to a per-instance classifier $f_1(x,\theta)$. Therefore, even though g_N acts on N events, it has the same information content as a parametrized classifier that acts on single events.

Performing regression via Eqs. (28) and (29) is straightforward but tedious. In practice, one would train a parametrized per-instance classifier $f_1(x,\theta)$ as in Eq. (23), multiply it to construct $f_N(\vec{x},\theta) = \prod_{i=1}^N f_1(x_i,\theta)$, and then sample over values of θ to approximate the integrals. We show examples of the above regression strategies in Sec. III C.

E. Beyond Regression

In addition to classification and regression, a standard machine learning task is density estimation. While some classical machine learning methods like k-nearest neighbors [40, 41] do require multi-instance information at prediction time, many of the standard deep learning solutions to implicit or explicit generative modeling are built on per-instance functions. Such methods include generative adversarial networks [42],[4] variational autoencoders [44], and normalizing flows [45].

One reason for computing explicit densities is to estimate the distance to a reference density. A common set of tools for this task are the f-divergences mentioned earlier. As discussed in Ref. [31] and highlighted in Table I, there is a direct mapping between the loss value of a per-instance classification task and a corresponding f-divergence between the underlying probability densities.

A related quantity is the mutual information between two random variables X and Y:

$$I(X,Y) = \int dx\,dy\,p(x,y)\log\frac{p(x,y)}{p(x)\,p(y)}. \tag{30}$$

For example, Y could be binary (a class label) and then $I(X,Y)$ would encode how much information (in units of nats) is available in X for doing classification. This can be helpful in the context of ranking input features, and was studied in the context of quark/gluon jet classification in Ref. [46].

Naively, Eq. (30) might seem like it requires estimating the densities $p(x)$, $p(y)$, and $p(x,y)$, which in turn may require ensemble information (see e.g. Ref. [47] for a study in the context of HEP). On the other hand, Eq. (30) takes the same form as the KL divergence in Eq. (9). Therefore, this quantity can be estimated using a similar strategy as in earlier sections, by training a classifier to distinguish data following $p(x,y)$ from data following $p(x)\,p(y)$ using the MLC loss. The value of the loss at the minimum will be an estimate of the mutual information. A simple example of this will be studied in Sec. III D.

III. EMPIRICAL STUDIES

We now present empirical studies comparing per-instance and per-ensemble data analysis strategies to highlight the points made in Sec. II. Our analyses are based on three case studies: a simple two Gaussian example, searching for dijet resonances, and measuring the top quark mass.

A. Classifiers: Multi-Event from Single-Event

As argued in Sec. II B, under the IID assumption we can build multi-event classifiers from single-event classifiers. We now demonstrate how to construct $f_{1\to N}$ defined in Eq. (13), comparing its performance to f_N.

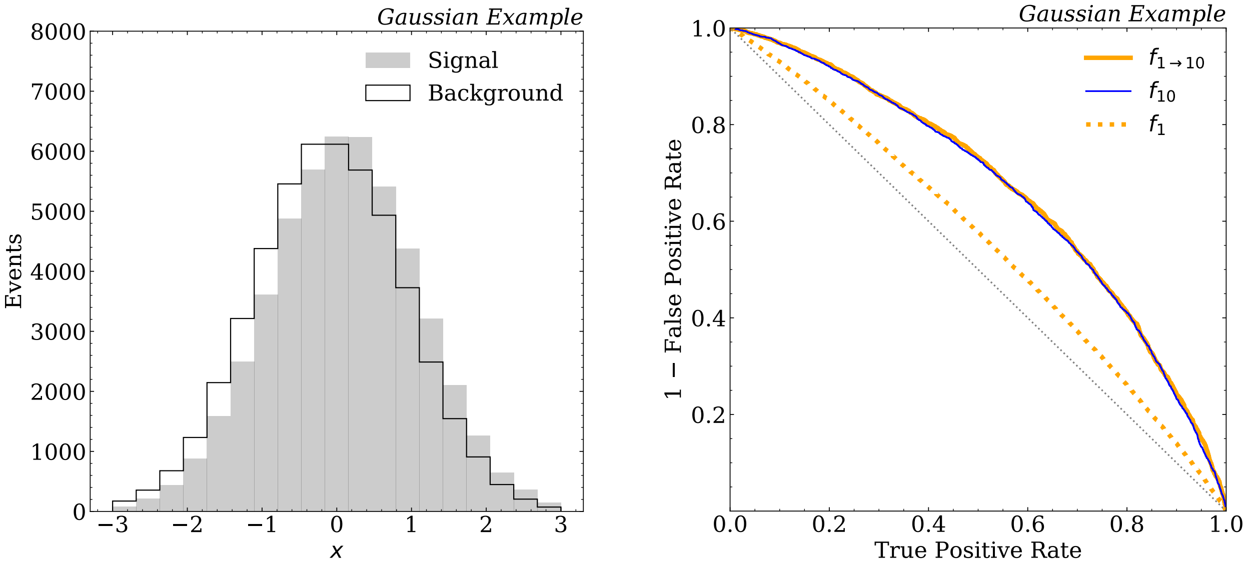
1. *Two Gaussian Example*

Our first case study involves one-dimensional Gaussian random variables. As shown in Fig. 1a, we consider two Gaussian distributions $X \sim \mathcal{N}(\pm\epsilon, 1)$, with slightly different means ($x_0 = \pm\epsilon$) but the same variance ($\sigma = 1$). Here, the "signal" has positive mean while the "background" has negative mean, and we take $\epsilon = 0.1$ for concreteness.

Both the per-instance (f_1) and per-ensemble (f_N) classifiers are parametrized by neural networks and implemented using KERAS [48] with the TENSORFLOW backend [49] and optimized with ADAM [50]. We use the binary cross entropy loss function so Eq. (4) is needed to convert the classifier output to a likelihood ratio. Each classifier consists of two hidden layers with 128 nodes per layer. Rectified Linear Unit (ReLU) activation functions are used for the intermediate layers while sigmoid activation is used for the last layer. The only difference between the per-instance and per-ensemble networks is that the input layer has one input for f_1 but N inputs for f_N.

We train each network with 50,000 events to minimize the binary cross entropy loss function, and we test the performance with an additional 50,000 events. For each

[4] In the context of adversarial training, it may be beneficial to use per-ensemble information in the discriminator to mitigate mode collapse, as utilized in Ref. [14]. This is also the philosophy behind mini-batch discrimination [43].



(a)

(b)

FIG. 1. Classification in the two Gaussian example. (a) A histogram of the Gaussian random variable X, for the "signal" ($x_0 = 0.1$) and background ($x_0 = -0.1$). (b) ROC curves for various binary classifiers. From the single-event classifier f_1, we can construct a multi-event classifier $f_{1\to10}$ that matches the performance of a classifier trained on 10 events simultaneously (f_{10}).

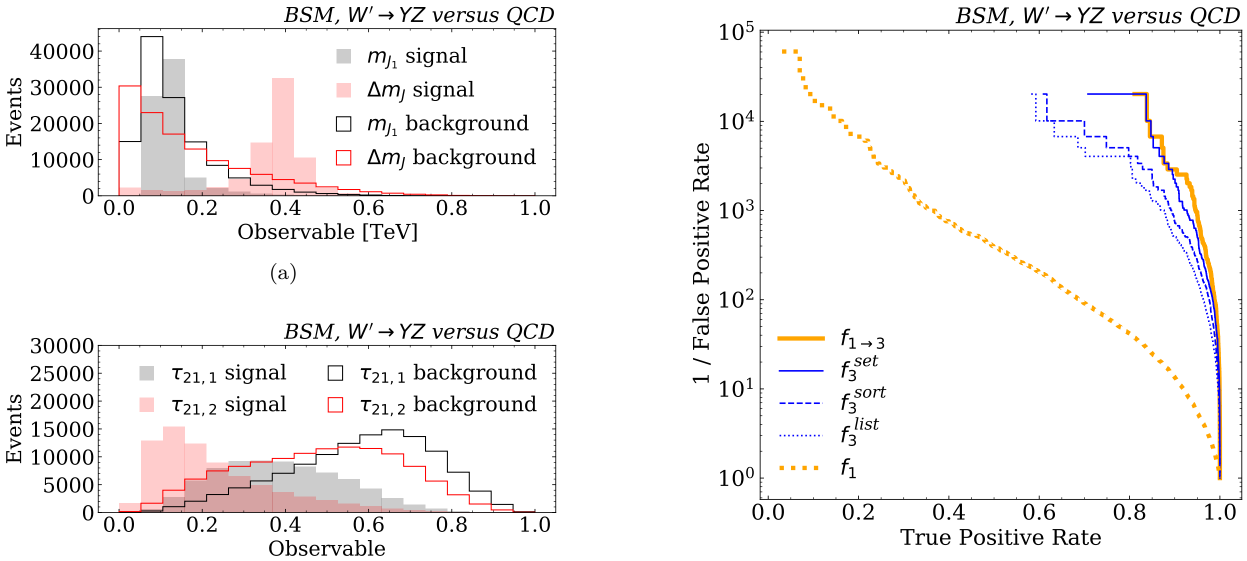
network, we train for up to 1000 epochs with a batch size of 10%, which means that the number of batches per epoch is the same, as is the number of events considered per batch. The training is stopped if the validation loss does not decrease for 20 consecutive epochs (early stopping). For the ensemble network, we take $N = 10$. We did not do any detailed hyperparameter optimization for these studies.

In Fig. 1b, we show the performance of the resulting classifiers f_1 and f_{10}. We checked that the f_1 classifier parametrized by a neural network has essentially the same performance as an analytic function derived by taking the ratio of Gaussian probability densities, which means that the neural network f_1 is nearly optimal. As expected, the per-instance classifier f_1 has a worse receiver operating characteristic (ROC) curve than the per-ensemble classifier f_{10}. This is not a relevant comparison, however, because the two are solving different classification tasks (i.e. classifying individual events as coming from signal or background versus classifying an ensemble of $N = 10$ events as all coming from signal or background). With Eq. (13), we can use f_1 to build a 10-instance classifier $f_{1\to10}$, whose ROC curve is nearly identical to f_{10}, if not even slightly better. Thus, as expected from Eq. (2), all of the information in the 10-instance classifier is contained in the per-instance classifier.

2. *Dijet Resonance Search*

We now consider an example from collider physics, motivated by a search for new beyond-the-Standard-Model (BSM) particles in a dijet final state. The simulations used for this study were produced for the LHC Olympics 2020 community challenge [51]. The background process involves generic quantum chromodynamics (QCD) dijet events with a requirement of at least one such jet with transverse momentum $p_T > 1.3$ TeV. The signal process involves the production of a hypothetical new resonance W' with mass $m_{W'} = 3.5$ TeV, which decays via $W' \to XY$ to two hypothetical particles X and Y of masses 500 GeV and 100 GeV, respectively. Each of the X and Y particles decays promptly into pairs of quarks. Due to the mass hierarchy between the W' boson and its decay products, the final state is characterized by two large-radius jets with two-prong substructure. The background and signal are generated using PYTHIA 8.219 [52, 53]. A detector simulation is performed with DELPHES 3.4.1 [54–56] using the default CMS detector card. Particle flow objects are used as inputs to jet clustering, implemented with FASTJET 3.2.1 [57, 58] and the anti-k_t algorithm [59] using $R = 1.0$ for the radius parameter. Events are required to have a reconstructed dijet mass within the range $m_{JJ} < [3.3, 3.7]$ GeV.

Four features are used to train our classifiers: the invariant mass of the lighter jet, the mass difference of the leading two jets, and the N-subjettiess ratios τ_{21} [60, 61] of the leading two jets. The observable τ_{21} quantifies the degree to which a jet is characterized by two subjets or one subjet, with smaller values indicating two-prong substructure. The mass features are recorded in units of TeV so that they are numerically $\mathcal{O}(1)$. Histograms of the four features for signal and background are shown in Figs. 2a and 2b. The signal jet masses are localized at



(a)

(b)

(c)

FIG. 2. Classification in the dijet resonance search example. (a,b) Histograms of the four jet features for the signal ($W' \to XY$) and background (QCD dijet) processes. (c) ROC curves for various binary classifiers. The multi-event classifier $f_{1\to3}$ (built from f_1) outperforms three classifiers trained on triplets of events: f_3^{list} with randomly ordered inputs, f_3^{sort} with sorted inputs, and f_3^{set} based on the deep sets/PFN strategy in Eq. (31) with built-in permutation invariance.

the X and Y masses and the τ_{21} observables are shifted towards lower values, indicating that the jets have two-prong substructure.

We train a per-instance classifier (f_1) and a per-ensemble classifier (f_3) using the same tools as for the Gaussian example above, again using binary cross entropy for the loss function. Because signal and background are so well separated in this example, we restrict our attention to $N = 3$ to avoid saturating the performance. Note that this is an artificially constructed classification problem, since in a more realistic context one would be trying to estimate the signal fraction in an event ensemble, not classify triplets of events as all coming from signal or background.

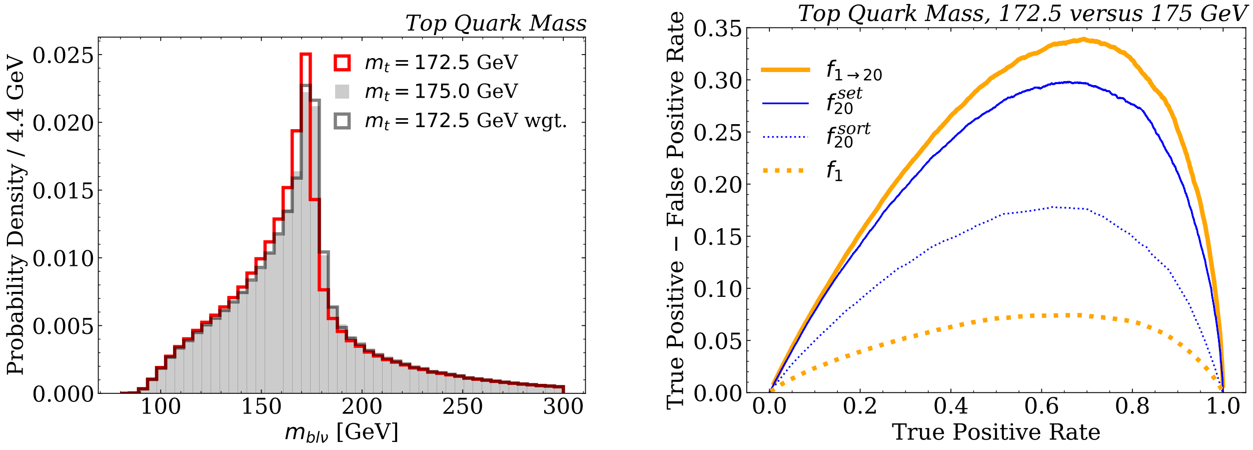
For f_1, the neural network architecture is the same as Ref. [18] with four hidden layers, each with 64 nodes and ReLU activation, and an output layer with sigmoid activation. For f_3, the neural network involves $4 \times 3 = 12$ inputs, and the penultimate hidden layer is adjusted to have 128 nodes, yielding a marginal performance gain. In both cases, about 100,000 events are used for testing and training, with roughly balanced classes. All of the networks are trained for up to 1000 epochs with the same early stopping condition as in the Gaussian case and with a batch size of 10%. Following Eq. (13), we construct a tri-event classifier $f_{1\to3}$ from f_1.

The ROC curves for f_3 and $f_{1\to3}$ are shown in Fig. 2c, with f_1 also shown for completeness. Interestingly, the $f_{1\to3}$ classifier trained on single events significantly outperforms f_3 trained on multiple events. There are a variety of reasons for this, but one important deficiency of the f_3 classifier is that it does not respect the permutation symmetry of its inputs. Because events are IID distributed, there is no natural ordering of the events, but the fully connected architecture we are using imposes an artificial ordering. Inspired by Ref. [12], we can break the permutation symmetry of the inputs by imposing a particular order on the events. Specifically, we train a network f_3^{sort} where the triplet of events is sorted by their leading jet mass. Using f_3^{sort} yields a small gain in performance seen in Fig. 2, but not enough to close the gap with $f_{1\to3}$.

A more powerful way to account for the permutation symmetry among events is to explicitly build a permutation-invariant neural network architecture. For this purpose, we use the deep sets approach [62]. In the particle physics context, deep sets were first used to construct particle flow networks (PFNs) [63], where the inputs involve sets of particles. Here, we are interested in sets of events, though we will still use the PFN code from the https://energyflow.network/ package. Following Refs. [62, 63], we decompose our set-based classifier as:

$$f_N^{\text{set}}(\vec{x}) = F\left(\sum_{i=1}^{N} \Phi(x_i)\right), \tag{31}$$

where $F : \mathbb{R}^L \to [0,1]$ and $\Phi : \mathbb{E} \to \mathbb{R}^L$ are neural networks that are simultaneously optimized. The network Φ embeds single events x_i into a L-dimensional latent space. The sum operator in Eq. (31) guarantees that f_N^{set} is invariant under permutations $x_{\sigma(i)}$ for $\sigma \in S_N$, the permutation group acting on N elements. We use the default parameters from the PFN code, with $L = 128$, Φ having two hidden layers with 100 nodes each, and F having three hidden nodes with 100 nodes each. The same



(a)

(b)

FIG. 3. Classification in the top quark mass example. (a) A histogram of $m_{b_1\mu\nu}$ for top quark masses of 172.5 GeV and 175 GeV. The "wgt." curve is explained later in Sec. III C 2, where we test the performance of a likelihood reweighting. (b) The difference in efficiency for the 172.5 GeV top quark mass sample (true positive) and the 175 GeV top quark mass sample (false positive) as a function of the true positive rate for various binary classifiers. Once again, a multi-event classifier ($f_{1\to20}$) built from the single-event classifier (f_1) has the best performance. For the classifiers trained to process 20 events simultaneously, the deep sets/PFN approach (f_{20}^{set}) does better than sorting the inputs (f_{20}^{sort}).

learning strategy (up to 1000 epochs, early stopping, 10% batch size) as the other networks is used for the PFN.

The performance of f_3^{set} is shown in Fig. 2, which gets much closer to matching the performance of $f_{1\to3}$. Part of this improvement is due to enforcing the permutation symmetry, though there is also a potential gain from the fact the PFN we used for f_3^{set} has more trainable weights than the fully connected network for f_3^{sort}. All of the f_3 variants were considerably more difficult to train than $f_{1\to3}$, likely for the reason discussed in Sec. II C. Thus, we have empirical evidence for the superiority of single-event training for multi-event classification.

3. Top Quark Mass Measurement

Our third and final example is motivated by the top quark mass measurement, as recently studied in Refs. [12, 18]. Extracting the top quark mass is really a regression problem, which we investigate in Sec. III C. Here, we consider a related classification task to distinguish two event samples generated with different top quark masses (172.5 GeV and 175 GeV). This is a realistic hypothesis testing task that requires full event ensemble information, though only per-instance training as we will see.

We use the same dataset as Ref. [18]. Top quark pair production is generated using Pythia 8.230 [52, 53] and detector effects are modeled with Delphes 3.4.1 [54–56] using the default CMS run card. After the production and decay steps $t\bar{t} \to bW^+\bar{b}W^-$, one of the W bosons is forced to decay to $\mu^+\nu$ while the other W boson decays hadronically. Each event is recorded as a variable-length set of objects, consisting of jets, muons, and neutrinos. At simulation-level, the neutrino is replaced with the missing transverse momentum. Generator-level and simulation-level jets are clustered with the anti-k_t algorithm using $R = 0.4$ and the simulation-level jet is labeled as b-tagged if the highest energy parton inside the nearest generator-level jet ($\Delta R < 0.5$) is a b quark. Jets are required to have $p_T > 20$ GeV and they can only be b-tagged if $|\eta| < 2.5$. Furthermore, jets overlapping with the muon are removed.

Events are only saved if they have at least two b-tagged jets and at least two additional non b-tagged jets. The b-jet closest to the muon in rapidity-azimuth is labeled b_1. Of the remaining b-tagged jets, the highest p_T one is labeled b_2. The two highest p_T non-b-tagged jets are labeled j_1 and j_2, and typically come from the W boson. (Imposing the W mass constraint on j_1 and j_2 would yield lower efficiency, though without significantly impacting the results.) The four-momentum of the detector-level neutrino (ν) is determined by solving the quadratic equation for the W boson mass; if there is no solution, the mass is set to zero, while if there are two real solutions, the one with the smaller $|p_z|$ is selected. Four observables are formed for performing the top quark mass extraction, given by the following invariant masses: $m_{b_1\mu\nu}$, $m_{b_2\mu\nu}$, $m_{b_1j_1j_2}$, and $m_{b_2j_1j_2}$. A histogram of $m_{b_1\mu\nu}$ is shown for illustration in Fig. 3a.

We use the same neural network architectures and training procedure as in the BSM example above, with 1.5 million events per fixed-mass sample. The only difference is that the batch size is set to 0.1% in order to keep the number of examples to be $\mathcal{O}(1000)$. For the per-ensemble classifier, we take $N = 20$, though of course for a realistic hypothesis testing situation, N would be as large as the number of top quark events recorded in data. To capture

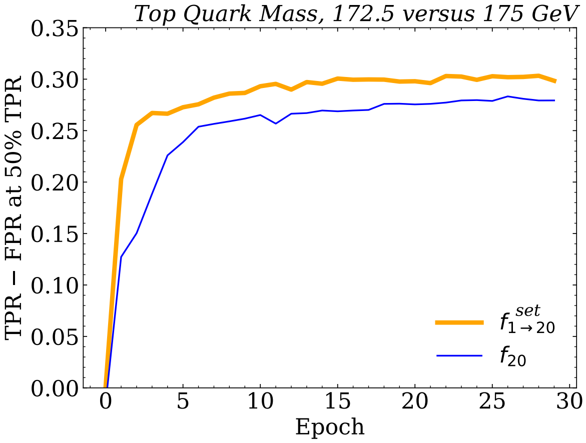


FIG. 4. Computational performance of single-event versus multi-event training. Shown is the efficiency for the 175 GeV sample (false positive) for a fixed 50% efficiency for the 172.5 GeV sample (true positive), plotted as a function of training epoch. Single-event training ($f_{1\to 20}$) outperforms multi-event training ($f^{\rm set}_{20}$), where both methods go through the full data set per epoch.

the permutation invariance of the inputs, we construct $f^{\rm set}_{20}$ using the deep sets approach in Eq. (31). We also build a classifier $f_{1\to 20}$ from the per-instance classifier f_1 using Eq. (13).

In Fig. 3b, we see that $f_{1\to 20}$ and $f^{\rm set}_{20}$ have comparable performance, though $f_{1\to 20}$ is noticeably better. Some of this improvement may be due to differences in the network architecture, but we suspect that most of the gain is due to the more efficient training in the per-instance case. We checked that very poor performance is obtained for a classifier f_{20} lacking permutation invariance, with a ROC curve that was not that much better than f_1 alone. Explicitly breaking the invariance by sorting the inputs based on $m_{b_1\mu\nu}$ does help a little, as indicated by the $f^{\rm sort}_{20}$ curve in Fig. 3b, but does not reach the set-based approach.

Given the similar performance of $f_{1\to 20}$ and $f^{\rm set}_{20}$, it is interesting to examine which learning strategy is more computationally efficient. In Fig. 4, we compare the performance as a function of the training epoch, using the difference of the true and false positive rates at a fixed 50% signal efficiency. In each epoch, both $f_{1\to 20}$ and $f^{\rm set}_{20}$ see the full ensemble of events, so this is an apples-to-apples comparison as far as data usage is concerned. In particular, we plot this information per epoch instead of per compute time to avoid differences due to the structure of the neural networks. (There is not an easy way to control for possible differences in the training time due to the differences in the network structures, since the underlying tasks are different.) The $f_{1\to 20}$ classifier trains much faster, in agreement with the analysis in Sec. II C, even though the ultimate asymptotic performance is similar for both classifiers. Once again, we see better empirical behavior from $f_{1\to 20}$ trained on one event at a time version $f^{\rm set}_{20}$ trained on multiple events simultaneously.[5]

B. Classifiers: Single-Event from Multi-Event

In general, one cannot take a multi-event classifier f_N and extract a single-event classifier f_1. It is, however, possible to construct a special $\tilde{f}_N$ network such that one can interpret a subnetwork as a per-event classifier, as discussed in Sec. II B. When using the MLC loss function, we can use the functional form in Eq. (14), where $\tilde{f}_N$ is a product of $f_{N\to 1}$ terms. Training $\tilde{f}_N$, where the only trainable weights are contained in $f_{N\to 1}$, we can learn a single-event classifier $f_{N\to 1}$ from multi-event samples.

For the binary cross entropy loss used in our case studies, where Eq. (4) is needed to convert the classifier to a likelihood ratio, we have to introduce a slightly different structure than Eq. (14). Let $f^{\rm set}_N$ be a permutation-invariant classifier, as defined in Eq. (31) using the deep sets/PFN strategy. Taking the latent space dimension to be $L = 1$, the Φ network can be interpreted as a single-event classifier. Because the Φ network outputs are pooled via summation, we can build an optimal multi-event classifier if Φ learns the *logarithm* of the likelihood ratio; cf. Eq. (2). With this insight, we can fix the F function to achieve the same asymptotic performance as a trainable F by setting:

$$F(\vec{x}) = \frac{\exp\left(\sum_{i=1}^N \Phi(x_i)\right)}{1+\exp\left(\sum_{i=1}^N \Phi(x_i)\right)}\,. \tag{32}$$

Using Eq. (4), one can check that this F is monotonically related to the ensemble likelihood ratio. Similarly, Φ will be monotonically related to the optimal f_1, which we call $f_{N\to 1}$ for the remainder of this discussion.

This construction is demonstrated in Fig. 5 for the Gaussian example. We see that the deep sets architecture with the fixed form of Eq. (32) ($\tilde{f}^{\rm set}_{10}$) has the same or better performance as the 10-instance fully-connected classifier with more network capacity (f_{10}). Similarly, the Φ function used as a single-event classifier ($f_{10\to 1}$) has nearly the same performance as an independently trained single-event classifier (f_1).

The same conclusion holds for the BSM classification task, shown in Fig. 6. The only difference between the set-based architectures $\tilde{f}^{\rm set}_3$ and $f^{\rm set}_3$ is that the former uses the fixed functional form in Eq. (32). The fact that they achieve nearly the same performance is ensured by the IID relation in Eq. (2). The per-instance $f_{3\to 1}$ network extracted from $\tilde{f}^{\rm set}_3$ is not quite as powerful as

[5] Away from the asymptotic limit, one could try to improve the empirical per-ensemble performance through data augmentation. Data augmentation is a generic strategy to help neural networks learn symmetries, and the IID structure can be reinforced by showing the network new ensembles built from sampling instances from the existing ensembles.

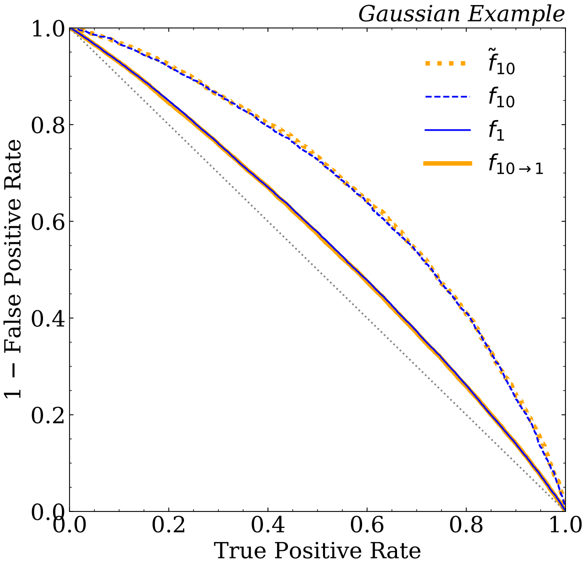


FIG. 5. Revisiting the ROC curves for the two Gaussian example from Fig. 1b. The multi-event classifier $\tilde{f}_{10}$ with the restricted functional form in Eq. (32) has the same performance as f_{10} with no restrictions. Using $\tilde{f}_{10}$, we can construct a single-event classifier $\tilde{f}_{10\to1}$ with the same performance as f_1 trained directly.

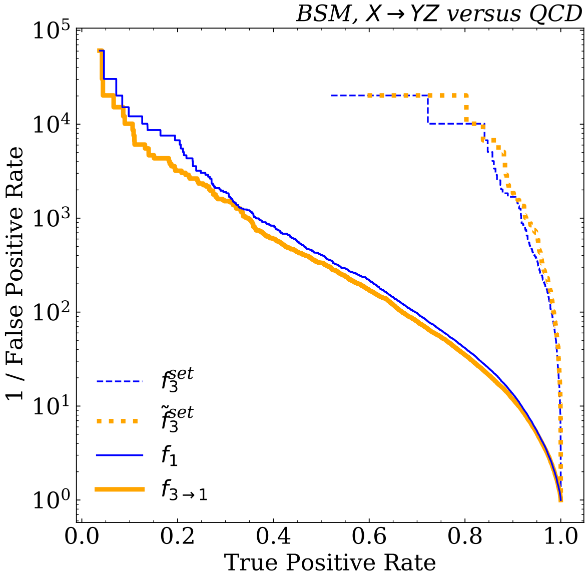


FIG. 6. Revisiting the ROC curves for the dijet resonance search example in Fig. 2c. The set-based multi-event classifiers $\tilde{f}_3^{\text{set}}$ and f_3^{set} have similar performance, but we can use the former to construct a single-event classifier $f_{3\to1}$. This construction is not as effective as performing single-event training directly (f_1).

the f_1 network trained independently on single events, as expected from the gradient issue discussed in Sec. II C. While we found no benefit to extracting a single-event classifier from a multi-event classifier, it is satisfying to see these IID-derived theoretical predictions borne out in these empirical examples.

C. Comparison of Regression Strategies

We now consider the regression methods introduced in Sec. II D. For classification, the mapping between per-instance and per-ensemble information is relatively straightforward. For regression, though, per-ensemble regression is structurally dissimilar from per-instance regression because of the need to integrate over priors on the regression parameters. Nevertheless, we can perform per-ensemble regression by first mapping the problem to per-instance parametrized classification.

We compare three different regression strategies for our empirical studies. The first method is a maximum-likelihood analysis, using the form in Eq. (24) based on the single-event parametrized classifier in Eq. (23). The second method is per-instance direct regression, using the construction in Eqs. (28) and (29) based on the same classifier as above. The third method is per-ensemble direct regression, based on minimizing the mean squared error loss in Eq. (27).

1. *Gaussian Mean Example*

Our first regression study is based on the same one-dimensional Gaussian distributions as Sec. III A 1. The prior distribution for the Gaussian means is taken to be uniform with $\mu \in [-0.5, 0.5]$, while the variance is fixed at $\sigma = 1$. A training dataset is created from 100 examples each from 10,000 values of the Gaussian mean, for a total of one million training data points. For the reference sample $p(x|\theta_0)$ needed to build the single-event parametrized classifier $f(x, \mu)$ in Eq. (23), we create a second dataset with one million examples drawn from a standard normal distribution (i.e. $\mu = 0$). To implement the $p(\theta)$ term in the second line of Eq. (22), each example x_i from the reference dataset is assigned a random mean value picked from the variable-mean dataset.

We train a parametrized neural network to distinguish the variable-mean datasets from the reference dataset. This network takes as input two features: one component of $\vec{x}$ and the random mean value μ. The architecture consists of three hidden layers with $(64, 128, 64)$ nodes per layer and ReLU activation. The output layer has a single node and sigmoid activation. Binary cross entropy is used to train the classifier and Eq. (4) is used to convert it to the likelihood ratio form $f(x, \mu)$. The model is trained for 1000 epochs with early stopping and a batch size of 10% of the training statistics.

The same learned function $f(x, \mu)$ is used for both the maximum likelihood analysis and per-instance direct regression. For the maximum-likelihood analysis, the opti-

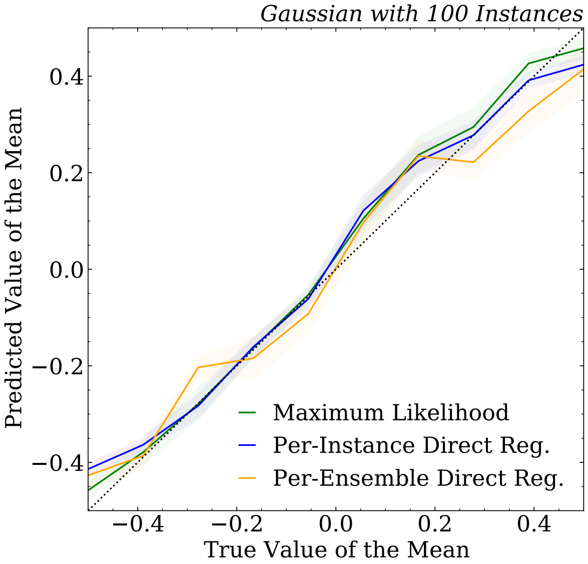


FIG. 7. Comparison of regression methods with the Gaussian example, with the predicted value of the mean plotted against the true value of the mean. The regression involves analyzing 100 instances drawn from the same Gaussian distribution. Bands are the standard deviation of the predictions over 10,000 generated samples. The per-instance direct regression uses single-event training, yet achieves comparable performance to per-ensemble direct regression that processes 100 events simultaneously.

mization in Eq. (24) is performed over a fixed grid with 20 evenly spaced values in $\mu \in [-0.5, 0.5]$. For per-instance direct regression, the function $f_N(\vec{x}, \mu)$ in Eq. (29) is constructed by taking a product of $f(x, \mu)$ outputs over all 100 examples in a given ensemble data point $\vec{x}$. The integrals in Eqs. (28) and (29) are approximated by evaluating $f_N(\vec{x}, \mu)$ at 20 evenly spaced μ values between -0.5 and 0.5 and then adding their values; this is possible because the prior is uniform.

The per-ensemble direct regression approach uses a neural network g_N that takes as input 100 values (i.e. all of $\vec{x}$) and predicts a single mean value. This network has the same architecture as $f(x, \mu)$, except it directly takes as input $\vec{x}$ and has linear (instead of a sigmoid) activation for the output layer, since the predicted mean can be both positive or negative. It is trained to minimize the mean squared error loss in Eq. (27).

In Fig. 7, we see that all three approaches give nearly the same results in terms of bias and variance. Strictly speaking, maximum likelihood and direct regression are different tasks so their behavior could be different. For per-instance and per-ensemble direct regression, they are constructed to yield the same asymptotic behavior, but there will be differences due to, e.g., the finite approximations to the integrals. Note that maximum likelihood and per-instance direct regression only use neural networks that process per-instance inputs; information about the rest of the events is used only through the training procedure. Thus, we have empirical evidence that per-ensemble regression can be accomplished via per-instance training.

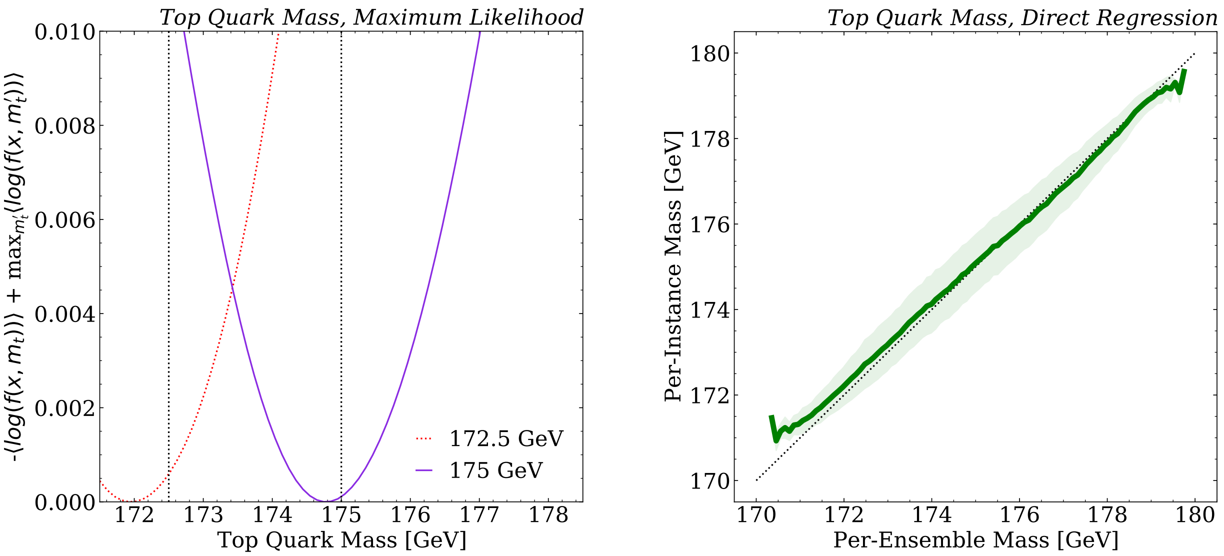
2. *Top Quark Mass Measurement*

As a physics example of regression, we consider extracting the top quark mass. Here, the top quark mass is the regression target and the setup is similar to the Gaussian example above. We use the same event generation as Sec. III A 3, but now with top quark mass parameters sampled uniformly at random in $m_t \in [170, 180]$ GeV. As with the Gaussian example, a variable-mass dataset is created. In this case, we have 100 events for each of 100,000 sampled top quark mass values. The reference sample uses a top quark mass of 172.5 GeV. Due to event selection effects, the actual number of events for each top quark mass value varies from set-to-set, with a mean of about 40 events. Because this event selection has a slight top quark mass dependence, this yields an effective non-uniform prior on m_t, which we account for when assigning dummy mass values to the reference sample.

The parametrized classifier now takes five inputs: the four mass features from Sec. III A 3 ($m_{b_1\mu\nu}$, $m_{b_2\mu\nu}$, $m_{b_1j_1j_2}$, and $m_{b_2j_1j_2}$) plus the top quark mass used for event generation. The neural network has three hidden layers with 50 nodes per layer and ReLU activation, and a single node output layer with sigmoid activation. We train 100 models and take the median as the classifier output, using Eq. (4) to convert it to the likelihood ratio $f(x, m_t)$. Each model is trained for 1000 epochs with early stopping with a patience of 20 epochs and a batch size of 0.1%. To test the fidelity of the training, we extract the estimated likelihood ratio of $m_t = 175$ GeV over $m_t = 172.5$ GeV and use it to reweight the 172.5 GeV sample. From Fig. 3a, we see that we achieve good reweighting performance despite the relatively limited training data.

The maximum likelihood analysis is performed by scanning the learned log likelihood estimate over a fixed grid with 100 uniformly spaced steps in $m_t \in [170, 180]$ GeV. In Fig. 8a, we show this scan where the target data comes from the high statistics 172.5 GeV and 175 GeV samples from Sec. III A 3. As desired, the minimum of the parabolic shapes are near the input top quark masses.

For the per-instance direct regression, we follow the same strategy as in the Gaussian case to convert $f(x, m_t)$ into an estimate of $\mathbb{E}[m_t|\vec{x}]$. The integrals in Eqs. (28) and (29) are approximated by sampling 50 random top quark masses per set of 100 following the probability density from the training dataset. Because 40 events are insufficient to make a precision measurement of the top quark mass, we find a noticeable bias between the estimated and true top mass values, which is exacerbated by edge effects at the ends of the training range. For this reason, we do not show a direct analog to Fig. 7, though this bias could be overcome with much larger training datasets with many more than 100 examples per mass value.



(a)

(b)

FIG. 8. Regression in the top quark mass example. (a) An estimate of the log likelihood for samples generated with 172.5 and 175 GeV top quark masses. The vertical axis has been shifted such that the minimum value is at zero. Note that the axis represents the average log likelihood which is a factor of N_{events} different from the total log likelihood. (b) Correlation between the per-instance predicted mass and the per-ensemble predicted mass in the context of direct regression. The per-ensemble mass values are put in bins of 0.1 GeV width, and the bands represent the standard deviation of the per-instance mass values in each bin.

For the per-ensemble direct regression, we use the deep sets approach in Eq. (31) to handle the permutation-invariance of the inputs. This approach is also well suited to handle the large variation in the number of events in each set due to the event selection effect. We again use PFNs for our practical implementation. We use the default PFN hyperparameters from the `https://energyflow.network/` package, except we use linear activation in the output layer and the mean squared error loss function. We found that it was important for the model accuracy to standardize both the inputs and outputs of the network. Note that this is a different per-ensemble direct regression setup than used in Ref. [12], which found excellent performance using linear regression on sorted inputs.

In Fig. 8b, we compare the output of per-ensemble direct regression to the output of per-instance direct regression. We find a very strong correlation between these two very different approaches to computing the same quantity $\mathbb{E}[m_t|\vec{x}]$. The band in Fig. 8b is the standard deviation over data sets with a true mass in the same one of the 100 bins that are evenly spaced between 170 and 180 GeV. A key advantage of the per-instance approach is that it does not need to be retrained if more events are acquired. By contrast, the per-ensemble approach is only valid for event samples that have the same sizes as were used during training.

D. Beyond Regression Example

As remarked in Sec. II E, the ideas discussed above apply to learning tasks beyond just standard classification and regression. As one simple example to illustrate this, we consider the Gaussian classification task from Sec. III A 1 and compute the mutual information between the Gaussian feature and the label. This quantifies how much information is available in the feature for classification and can be directly compared with other features and other classification tasks.

For this illustration, 10^5 events are generated each from two Gaussian distributions with means $\pm|\epsilon|$ for fixed ϵ. The mutual information is estimated using a per-instance classifier as described in Sec. II E and also computed analytically via Eq. (30). For the per-instance classifier, we use a neural network that processes two inputs (label and feature), has two hidden layers with ReLU activation, and has a single node sigmoid output. The classification task is to distinguish the nominal dataset from one where the labels are assigned uniformly at random to the features. The value of the MLC loss yields an estimate of the mutual information.

The mutual information results are presented in Fig. 9, as a function of ϵ. As expected, the neural network strategy yields an excellent approximation to the analytic calculation. Note that this strategy does require any binning and naturally extends to high-dimensional data, since the core component is a neural network classifier.

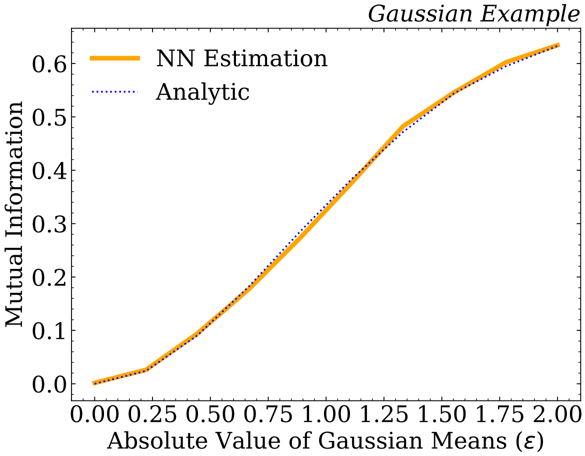


FIG. 9. Mutual information between a Gaussian feature and a label, where the " signal" ($x_0 = \epsilon$) and " background" ($x_0 = -\epsilon$) have opposite means. The estimate using the MLC loss approach shows good agreement with the exact analytic expression.

We leave an investigation of this approach in the particle physics context to future work.

IV. CONCLUSIONS

We have demonstrated a connection between classifiers trained on single events and those that process multiple events at the same time. One can take a generic single-event classifier and build an N-event classifier using simple arithmetic operations. Such classifiers tend to out-perform generic N-event classifiers, since we can enforce the IID assumptions into the learning task. This performance gap can be mostly recovered by deploying a classifier that respects the permutation invariance of the set of N events. We used the deep sets/PFN architecture [62, 63] for this purpose, but other set-based architectures such as graph neural networks [64, 65] would also be appropriate.

An amusing feature of the deep sets approach is that we can use it to reverse-engineer a single-event classifier from a multi-event classifier by restricting the latent space to be one-dimensional and fixing a static output function. Even after enforcing these additional structures, though, we found both theoretically and empirically that the loss function gradients are better behaved for single-event classifiers than multi-event classifiers. Going beyond classification, we explained how various regression tasks can be phrased in terms of per-instance parametrized classification, yielding similar performance to per-ensemble direct regression. We also mentioned how to compute distances and divergences between probability densities without requiring explicit density estimation. These results hold for any data sample satisfying the IID property.

Ultimately, we did not find any formal or practical advantage for training a multi-event classifier instead of a single-event classifier, as least for the cases we studied. With a carefully selected multi-event architecture, one can achieve similar performance to a scaled-up per-event classifier, but the latter will typically train faster. For direct regression, the per-ensemble strategy might be conceptually simpler than the per-instance method, though the per-instance methods allow for a simpler treatment of variably-sized data sets. Note that there may be situations where a simplifying assumption (e.g. the linear regression model in Ref. [12]) could yield better per-ensemble behavior than indicated by our case studies. At minimum, we hope this paper has demystified aspects of per-ensemble learning and highlighted some interesting features of the MLC loss function.

Going beyond the IID assumption, the duality between per-instance classifiers and per-ensemble classifiers could have applications to problems with approximate independence. For example, flavor tagging algorithms have traditionally exploited the approximate independence of individual track features within a jet [66, 67]. Similarly, emissions in the Lund jet plane [68, 69] are approximately independent, with exact independence in the strongly ordered limit of QCD. In both contexts, the instances are particles (or particle-like features) and the ensemble is the jet. A potentially powerful training procedure for these situations might be to first train a per-particle classifier, then build a per-jet classifier using the constructions described in this paper, and finally let the network train further to learn interdependencies between the particles.

CODE AND DATA

The code for this paper can be found at https://github.com/bnachman/EnsembleLearning. The physics datasets are hosted on Zenodo at Ref. [70] for the top quark dataset and Ref. [71] for the BSM dataset.

ACKNOWLEDGMENTS

We thank Anders Andreassen, Patrick Komiske, and Eric Metodiev for discussions about the MLC loss. We thank Rikab Gambhir and Ian Convy for discussions about mutual information. We thank Adi Suresh for discussions about the regression task with the classifier loss. We thank Katherine Fraiser, Yue Lai, Duff Neill, Bryan Ostdiek, Mateusz Ploskon, Felix Ringer, and Matthew Schwartz for useful comments on our manuscript. BN is supported by the U.S. Department of Energy (DOE), Office of Science under contract DE-AC02-05CH11231. JT is supported by the National Science Foundation under Cooperative Agreement PHY-2019786 (The NSF AI Institute for Artificial Intelligence and Fundamental Interactions, http://iaifi.org/), and by the U.S. DOE Office of High Energy Physics under grant number DE-SC0012567. BN would also like to thank NVIDIA for providing Volta GPUs for neural network training.

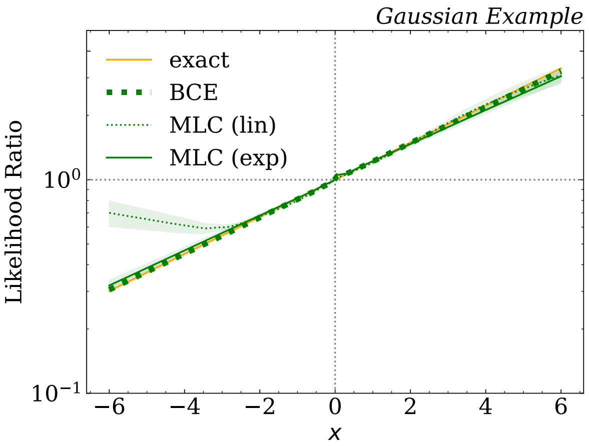


FIG. 10. A demonstration of the MLC loss for learning the likelihood ratio directly, using the Gaussian example from Fig. 1a. The linear (lin) and exponential (exp) parametrizations perform similarly. Shown for comparison is likelihood ratio computed using the binary cross entropy (BCE) loss that requires the manipulation in Eq. (4).

Appendix A: Deriving Maximum Likelihood Classifier Loss

Beyond just the practical value of learning the likelihood ratio, the MLC loss in Eq. (7) has a nice interpretation in terms of learning probability distributions.

Consider trying to learn a function $f(x)$ that is a normalized probability distribution, up to a Jacobian factor $j(x)$:

$$\int dx\, j(x) f(x) = 1. \tag{A1}$$

We are given samples from a probability distribution $q(x)$, and we want to learn $f(x)$ such that

$$f(x) \to \frac{q(x)}{j(x)}. \tag{A2}$$

In other words, we want to learn a function $f(x)$ that reproduces the sampled distribution $q(x)$ after including the Jacobian factor. This problem was studied in Ref. [34], albeit in a context where $f(x)$ had a restricted functional form such that Eq. (A1) was automatically enforced.

One strategy to accomplish this is to minimize the cross entropy of $f(x)$ with respect to $q(x)$, since the smallest cross entropy is obtained when $f(x)$ has the same information content as $q(x)$. The associated loss functional is:

$$L[f] = -\int dx\, q(x) \log f(x) - \lambda \left(1 - \int dx\, j(x) f(x)\right), \tag{A3}$$

where the first term is the cross entropy and λ is a Lagrange multiplier to enforce the normalization condition in Eq. (A1). Taking the functional derivative of Eq. (A3) with respect to $f(x)$ and setting it equal to zero, we find the extremum condition:

$$-\frac{q(x)}{f(x)} + \lambda\, j(x) = 0. \tag{A4}$$

Multiplying both sides of this equation by $f(x)$ and integrating over x to set the Lagrange multiplier, we find that Eq. (A4) is solved for

$$\lambda = 1, \qquad f(x) = \frac{q(x)}{j(x)}, \tag{A5}$$

so $f(x)$ learns the $q(x)/j(x)$ ratio as desired.

In the special case that $j(x)$ is itself a normalized probability distribution, we can substitute for the Lagrange multiplier and rewrite Eq. (A3) in the following form:

$$L[f] = -\int dx \left(q(x) \log f(x) + j(x)(1 - f(x))\right). \tag{A6}$$

Identifying $q(x) = p(x|\theta_A)$ and $j(x) = p(x|\theta_B)$, this is precisely the MLC loss in Eq. (7). Therefore, we have an intuitive understanding of the MLC loss as trying to maximize the (log) likelihood of $f(x)$ with respect to $p(x|\theta_A)$, subject to the constraint that $f(x)\, p(x|\theta_B)$ is a proper probability distribution.

In Fig. 10, we plot the learned likelihood ratio between the two Gaussian samples from Fig. 1a, comparing the performance of MLC against binary cross entropy and the exact analytic expression. In all cases, a network is trained with 100 epochs and early stopping with a patience of 10 epochs. We also compare the MLC loss against the $C(f) = \exp f$ variant discussed in footnote 1. We see that both the linear (i.e. $C(f) = f$) and exponential parametrizations perform similarly in the region with ample data. That said, the exponential parametrization has a more robust extrapolation towards the edges, yielding similar behavior to binary cross entropy. Note that the exponential parametrization of the MLC loss was used in Ref. [32].

[1] A. J. Larkoski, I. Moult, and B. Nachman, Jet Substructure at the Large Hadron Collider: A Review of Recent Advances in Theory and Machine Learning, Phys. Rept. **841**, 1 (2020), arXiv:1709.04464 [hep-ph].

[2] D. Guest, K. Cranmer, and D. Whiteson, Deep Learning and its Application to LHC Physics, Ann. Rev. Nucl. Part. Sci. **68**, 161 (2018), arXiv:1806.11484 [hep-ex].

[3] K. Albertsson *et al.*, Machine Learning in High Energy Physics Community White Paper, (2018), arXiv:1807.02876 [physics.comp-ph].

[4] A. Radovic *et al.*, Machine learning at the energy and intensity frontiers of particle physics, Nature **560**, 41 (2018).

[5] D. Bourilkov, Machine and Deep Learning Applications in Particle Physics, Int. J. Mod. Phys. A **34**, 1930019 (2020), arXiv:1912.08245 [physics.data-an].

[6] HEP ML Community, A Living Review of Machine Learning for Particle Physics.

[7] Y. S. Lai, Automated Discovery of Jet Substructure Analyses, (2018), arXiv:1810.00835 [nucl-th].

[8] C. K. Khosa, V. Sanz, and M. Soughton, Using Machine Learning to disentangle LHC signatures of Dark Matter candidates, (2019), arXiv:1910.06058 [hep-ph].

[9] Y.-L. Du, K. Zhou, J. Steinheimer, L.-G. Pang, A. Motornenko, H.-S. Zong, X.-N. Wang, and H. Stöcker, Identifying the nature of the QCD transition in relativistic collision of heavy nuclei with deep learning, Eur. Phys. J. C **80**, 516 (2020), arXiv:1910.11530 [hep-ph].

[10] A. Mullin, H. Pacey, M. Parker, M. White, and S. Williams, Does SUSY have friends? A new approach for LHC event analysis, (2019), arXiv:1912.10625 [hep-ph].

[11] S. Chang, T.-K. Chen, and C.-W. Chiang, Distinguishing W' Signals at Hadron Colliders Using Neural Networks, (2020), arXiv:2007.14586 [hep-ph].

[12] F. Flesher, K. Fraser, C. Hutchison, B. Ostdiek, and M. D. Schwartz, Parameter Inference from Event Ensembles and the Top-Quark Mass, (2020), arXiv:2011.04666 [hep-ph].

[13] M. Lazzarin, S. Alioli, and S. Carrazza, MCNNTUNES: tuning Shower Monte Carlo generators with machine learning, (2020), arXiv:2010.02213 [physics.comp-ph].

[14] Y. S. Lai, D. Neill, M. Płoskoń, and F. Ringer, Explainable machine learning of the underlying physics of high-energy particle collisions, (2020), arXiv:2012.06582 [hep-ph].

[15] J. Neyman and E. S. Pearson, On the problem of the most efficient tests of statistical hypotheses, Phil. Trans. R. Soc. Lond. A **231**, 289 (1933).

[16] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning*, Springer Series in Statistics (Springer New York Inc., New York, NY, USA, 2001).

[17] M. Sugiyama, T. Suzuki, and T. Kanamori, *Density Ratio Estimation in Machine Learning* (Cambridge University Press, 2012).

[18] A. Andreassen, S. Hsu, B. Nachman, N. Suaysom, A. Suresh, Parameter Estimation using Neural Networks in the Presence of Detector Effects, (2020), arXiv:2010.03569 [hep-ph].

[19] A. Andreassen and B. Nachman, Neural Networks for Full Phase-space Reweighting and Parameter Tuning, Phys. Rev. D **101**, 091901(R) (2020), arXiv:1907.08209 [hep-ph].

[20] M. Stoye, J. Brehmer, G. Louppe, J. Pavez, and K. Cranmer, Likelihood-free inference with an improved cross-entropy estimator, (2018), arXiv:1808.00973 [stat.ML].

[21] J. Hollingsworth and D. Whiteson, Resonance Searches with Machine Learned Likelihood Ratios, (2020), arXiv:2002.04699 [hep-ph].

[22] J. Brehmer, K. Cranmer, G. Louppe, and J. Pavez, Constraining Effective Field Theories with Machine Learning, Phys. Rev. Lett. **121**, 111801 (2018), arXiv:1805.00013 [hep-ph].

[23] J. Brehmer, K. Cranmer, G. Louppe, and J. Pavez, A Guide to Constraining Effective Field Theories with Machine Learning, Phys. Rev. D **98**, 052004 (2018), arXiv:1805.00020 [hep-ph].

[24] J. Brehmer, F. Kling, I. Espejo, and K. Cranmer, MadMiner: Machine learning-based inference for particle physics, Comput. Softw. Big Sci. **4**, 3 (2020), arXiv:1907.10621 [hep-ph].

[25] J. Brehmer, G. Louppe, J. Pavez, and K. Cranmer, Mining gold from implicit models to improve likelihood-free inference, Proc. Nat. Acad. Sci. , 201915980 (2020), arXiv:1805.12244 [stat.ML].

[26] K. Cranmer, J. Pavez, and G. Louppe, Approximating Likelihood Ratios with Calibrated Discriminative Classifiers, (2015), arXiv:1506.02169 [stat.AP].

[27] C. Badiali, F. Di Bello, G. Frattari, E. Gross, V. Ippolito, M. Kado, and J. Shlomi, Efficiency Parameterization with Neural Networks, (2020), arXiv:2004.02665 [hep-ex].

[28] A. Andreassen, B. Nachman, and D. Shih, Simulation Assisted Likelihood-free Anomaly Detection, Phys. Rev. D **101**, 095004 (2020), arXiv:2001.05001 [hep-ph].

[29] A. Andreassen, P. T. Komiske, E. M. Metodiev, B. Nachman, and J. Thaler, OmniFold: A Method to Simultaneously Unfold All Observables, Phys. Rev. Lett. **124**, 182001 (2020), arXiv:1911.09107 [hep-ph].

[30] M. Erdmann, B. Fischer, D. Noll, Y. Rath, M. Rieger, and D. Schmidt, Adversarial Neural Network-based data-simulation corrections for jet-tagging at CMS, in *Proc. 19th Int. Workshop on Adv. Comp., Anal. Techn. in Phys. Research, ACAT2019* (2019).

[31] X. Nguyen, M. J. Wainwright, and M. I. Jordan, On surrogate loss functions and f-divergences, arXiv Mathematics e-prints , math/0510521 (2005), arXiv:math/0510521 [math.ST].

[32] R. T. D'Agnolo and A. Wulzer, Learning New Physics from a Machine, Phys. Rev. D **99**, 015014 (2019), arXiv:1806.02350 [hep-ph].

[33] R. T. D'Agnolo, G. Grosso, M. Pierini, A. Wulzer, and M. Zanetti, Learning Multivariate New Physics, (2019), arXiv:1912.12155 [hep-ph].

[34] A. Andreassen, I. Feige, C. Frye, and M. D. Schwartz, JUNIPR: a Framework for Unsupervised Machine Learning in Particle Physics, Eur. Phys. J. C **79**, 102 (2019), arXiv:1804.09720 [hep-ph].

[35] J. Brehmer and K. Cranmer, Flows for simultaneous manifold learning and density estimation (2020), arXiv:2003.13913 [stat.ML].

[36] B. Nachman and D. Shih, Anomaly Detection with Density Estimation, Phys. Rev. D **101**, 075042 (2020), arXiv:2001.04990 [hep-ph].

[37] E. M. Metodiev, B. Nachman, and J. Thaler, Classification without labels: Learning from mixed samples in high energy physics, JHEP **10**, 174, arXiv:1708.02949 [hep-ph].

[38] P. Baldi, K. Cranmer, T. Faucett, P. Sadowski, and D. Whiteson, Parameterized neural networks for high-energy physics, Eur. Phys. J. C **76**, 235 (2016), arXiv:1601.07913 [hep-ex].

[39] S. Cheong, A. Cukierman, B. Nachman, M. Safdari, A. Schwartzman, Parametrizing the Detector Response with Neural Networks, JINST **15**, P01030, arXiv:1910.03773 [physics.data-an].

[40] E. Fix and J. L. Hodges Jr., Discriminatory analysis-nonparametric discrimination: consistency properties, USAF School of Aviation Medicine, Project Number 21-49-004, Report Number 4 (1951).

[41] T. Cover M. and P. E. Hart, Nearest neighbor pattern classification, IEEE Transactions on Information Theory **13**, 21 (1967).

[42] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio, Generative adversarial nets, in *Advances in Neural Information Processing Systems*, Vol. 27, edited by Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Q. Weinberger (Curran Associates, Inc., 2014) pp. 2672–2680.

[43] T. Salimans, I. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen, Improved Techniques for Training GANs, arXiv e-prints , arXiv:1606.03498 (2016), arXiv:1606.03498 [cs.LG].

[44] D. P. Kingma and M. Welling, Auto-encoding variational bayes., in *ICLR*, edited by Y. Bengio and Y. LeCun (2014).

[45] D. Rezende and S. Mohamed, Variational inference with normalizing flows, in *Proceedings of the 32nd International Conference on Machine Learning*, Proceedings of Machine Learning Research, Vol. 37, edited by F. Bach and D. Blei (PMLR, Lille, France, 2015) pp. 1530–1538.

[46] A. J. Larkoski, J. Thaler, and W. J. Waalewijn, Gaining (Mutual) Information about Quark/Gluon Discrimination, JHEP **11**, 129, arXiv:1408.3122 [hep-ph].

[47] N. Carrara and J. Ernst, On the estimation of mutual information (2019), arXiv:1910.00365 [physics.data-an].

[48] F. Chollet, Keras, `https://github.com/fchollet/keras` (2017).

[49] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, *et al.*, Tensorflow: A system for large-scale machine learning., in *OSDI*, Vol. 16 (2016) pp. 265–283.

[50] D. Kingma and J. Ba, Adam: A method for stochastic optimization, (2014), arXiv:1412.6980 [cs].

[51] G. Kasieczka, B. Nachman, and D. Shih, R&D Dataset for LHC Olympics 2020 Anomaly Detection Challenge, 10.5281/zenodo.2629073 (2019), https://doi.org/10.5281/zenodo.2629073.

[52] T. Sjöstrand, S. Mrenna, and P. Z. Skands, PYTHIA 6.4 Physics and Manual, JHEP **05**, 026, arXiv:hep-ph/0603175 [hep-ph].

[53] T. Sjöstrand, S. Mrenna, and P. Z. Skands, A Brief Introduction to PYTHIA 8.1, Comput. Phys. Commun. **178**, 852 (2008), arXiv:0710.3820 [hep-ph].

[54] J. de Favereau, C. Delaere, P. Demin, A. Giammanco, V. Lemaître, A. Mertens, and M. Selvaggi (DELPHES 3), DELPHES 3, A modular framework for fast simulation of a generic collider experiment, JHEP **02**, 057, arXiv:1307.6346 [hep-ex].

[55] A. Mertens, New features in Delphes 3, *Proceedings, 16th International workshop on Advanced Computing and Analysis Techniques in physics (ACAT 14): Prague, Czech Republic, September 1-5, 2014*, J. Phys. Conf. Ser. **608**, 012045 (2015).

[56] M. Selvaggi, DELPHES 3: A modular framework for fast-simulation of generic collider experiments, *Proceedings, 15th International Workshop on Advanced Computing and Analysis Techniques in Physics Research (ACAT 2013): Beijing, China, May 16-21, 2013*, J. Phys. Conf. Ser. **523**, 012033 (2014).

[57] M. Cacciari, G. P. Salam, and G. Soyez, FastJet User Manual, Eur. Phys. J. **C72**, 1896 (2012), arXiv:1111.6097 [hep-ph].

[58] M. Cacciari and G. P. Salam, Dispelling the N^3 myth for the k_t jet-finder, Phys. Lett. **B641**, 57 (2006), arXiv:hep-ph/0512210 [hep-ph].

[59] M. Cacciari, G. P. Salam, and G. Soyez, The anti-k_t jet clustering algorithm, JHEP **04**, 063, arXiv:0802.1189 [hep-ph].

[60] J. Thaler and K. Van Tilburg, Maximizing Boosted Top Identification by Minimizing N-subjettiness, JHEP **02**, 093, arXiv:1108.2701 [hep-ph].

[61] J. Thaler and K. Van Tilburg, Identifying Boosted Objects with N-subjettiness, JHEP **03**, 015, arXiv:1011.2268 [hep-ph].

[62] M. Zaheer, S. Kottur, S. Ravanbhakhsh, B. Póczos, R. Salakhutdinov, and A. J. Smola, Deep sets, in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17 (Curran Associates Inc., Red Hook, NY, USA, 2017) p. 3394–3404.

[63] P. T. Komiske, E. M. Metodiev, and J. Thaler, Energy Flow Networks: Deep Sets for Particle Jets, JHEP **01**, 121, arXiv:1810.05165 [hep-ph].

[64] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, The graph neural network model, Trans. Neur. Netw. **20**, 61–80 (2009).

[65] J. Shlomi, P. Battaglia, and J.-R. Vlimant, Graph Neural Networks in Particle Physics 10.1088/2632-2153/abbf9a (2020), arXiv:2007.13681 [hep-ex].

[66] M. Aaboud *et al.* (ATLAS), Measurements of b-jet tagging efficiency with the ATLAS detector using $t\bar{t}$ events at $\sqrt{s} = 13$ TeV, JHEP **08**, 089, arXiv:1805.01845 [hep-ex].

[67] S. Chatrchyan *et al.* (CMS), Identification of b-Quark Jets with the CMS Experiment, JINST **8**, P04013, arXiv:1211.4462 [hep-ex].

[68] B. Andersson, G. Gustafson, L. Lonnblad, and U. Pettersson, Coherence Effects in Deep Inelastic Scattering, Z. Phys. C **43**, 625 (1989).

[69] F. A. Dreyer, G. P. Salam, and G. Soyez, The Lund Jet Plane, JHEP **12**, 064, arXiv:1807.04758 [hep-ph].

[70] A. Andreassen, S.-C. Hsu, B. Nachman, N. Suaysom, and A. Suresh, Srgn: Pythia + delphes $pp \to t\bar{t}$, 10.5281/zenodo.4067673 (2020).

[71] G. Kasieczka, B. Nachman, and D. Shih, Official Datasets for LHC Olympics 2020 Anomaly Detection Challenge, 10.5281/zenodo.4287846 (2019).