Yes, @vollermsj, I certainly miss being in the same room with a blackboard!

Here is my initial response to the proposal as kindly detailed by @DilumAluthge. I'm sorry this does include some more technical discussion. Be assured I appreciate acutely your patience to endure these so far. I am being more verbose than you may like, but only to mitigate further possible misunderstandings.

## 1. Loss estimates in MLJ and their scope of application

At present, the only kind of probablistic supervised learning model that MLJ designed to interface is a model that:

(i) Assumes data $(X_1, y_1), (X_2, y_2), \ldots$ is generated by an i.i.d process; and

(ii) Is capable of delivering, after seeing training data $D$, a probability distribution $p(y|x, D)$, defined for each new *single* input observation $x$.

Given a probabilistic scoring rule (e.g., Brier score) the expected loss of the model is then well-defined. There a number of algorithms, such as cross-validation, implemented in MLJ (and all such ML toolboxes) that take the function $p$ as input and estimate this loss. While not without controversy, these estimates are ubiquitous and well-studied. Furthermore, both the definition of the expected loss, and the algorithms for estimating the loss do not depend on any other feature of the model (e.g., "model is Bayesian", or "model is linear"). It is therefore possible to compare *all* such models in a consistent way using such estimates, which is crucial.

## 2. Goals for Bayesian model / MLJ interaction

Here are goals that have been articulated so far, as best as I can gather:

(i) We integrate into MLJ Bayesian models that fit into the framework outlined in 1.

(ii) Certain functionality of Bayesian models not shared by all models (but not unique to them) is exposed in MLJ. Specifically, "correlated predictions" (see 5. below) should be exposed.

(iii) New functionality is added to MLJ that would allow evaluation of Bayesian models in ways that do not fit into the framework outlined in 1, even though the models themselves may do so. This goal requires (ii). (Here I'm thinking of things like implementing `brier_score(dist::Distribution{Vector{T}}) where T`, as discussed in @DilumAluthge's proposal.)

(iv) We additionally integrate Bayesian models that do not fit into the framework outlined in 1, such as models for non i.i.d. processes. Here "integrate" is not the best word, because currently MLJ has little to offer in the way of meta-algorithms to support such models. But the implication seems to be that realizing

(iii) would change this (?)

## 3. Comment

In principle, I do not have objections to any one goal. However:

- For me (and I expect most general MLJ users) (i) is the highest priority. It seems to me (i) can be achieved independently of the other goals. I would not support adding i.i.d. Bayes models to MLJ that *can* be fit into framework 1 but do no actually implement the necessary part of the API needed to include them. This is not to say that i.i.d. models are only valuable as part of the framework, only that integration into MLJ only makes sense if they participate. I realize that to implement this goal it may be necessary to generalize some measures so that they can deal with (vectors of) `Sampler` predictions, and not just `Distriubtion` predictions, which I would support.

- It seems (ii) can be readily achieved somehow. However, I do see a flaw in the current proposal for doing so, in which goal (i) is compromised. See 6 below. I may also simply misunderstand the proposal.

- I don't believe (iii) is a trivial undertaking. I therefore suggest these enhancements be added by a new third party package. Here are some reasons: (a) MLJBase is already large and the performance measures interface (which should be a package in its own right) is large, growing and a bit complicated; (b) Limited resources now mean it's hard to justify an enhancement that is neither small, simple, nor adding value across the board (to all models); (c ) Having this externalized might help you rally the necessary expertise and would give you independence (you wouldn't have to wait a week+ for every PR review from me).

- Well, (iv) seems to depend on (iii). Realistically, it is probably out-of-scope for now.

## 4. A clarification of the API for probablisitic models

Before responding to the specific design proposal, I think I need to clarify the relationship between the API specs and the framework defined in 1.

While the MLJ API specifies that each `Probabilistic` model should implement a `predict(mach, Xnew)` method that returns a *vector* of probability distributions `[d1, d2, ..., dk]` for each multi-observation input `X` (a table with `k` rows, say) it is tacitly assumed that this method *is equivalent to broadcasting a single observation predict method*, corresponding to the distribution $p$ above. In other words, `predict` should just be an implementation of the vector-valued function $P$, given by

$$P(y_1, y_2, \ldots, y_k|x_1, x_2, \ldots, x_k, D) = (p(y_1|x_1, D), p(y_2|x_2, D), \ldots, p(y_k|x_k, D)).$$

This assumption is necessary unless we agree to depart from the framework 1 (which would exclude us from comparing all models in a consistent way).

Let me note here a trivial corollary of our assumption: the single component of $P(y_1|x_1, D)$ is the same thing as the first component of $P(y_1, y_2|x_1, x_2, D)$, or in MLJ syntax:

```julia
predict(mach, Xnew[1, :])[1] == predict(mach, Xnew)[1, :]
```

for any table `Xnew` with two rows. I will call this property *consistency* below.

## 5. Correlated predictions for "mixture models"

Distilling previous disucssions:

Given a family of probabilistic predictors $p_\theta$, parameterized by $\theta$ (each fitting into the framework of 1 above) and a mixing pdf $w(\theta)$ (possibly depending on the training data $D$) then we can construct a multivariate distribution function

$$p_{corr}(y_1, \ldots, y_k|x_1, \ldots, x_k|D) = \int \prod_i p_\theta(y_i|x_i, D) d\theta$$

whose marginals are generally correlated. This framework includes Bayesian models, where $w(\theta)$ is the posterior for model hyperparameters.

Note that if we take the special case $k = 1$ our multivariate distribution becomes a univariate one, and we obtain a candidate $p(y|x, D)$ for placing a mixture model into the framework 1 (I'm assuming the setting is i.i.d data). However, this does not appear to factor into @DilumAluthge's proposal.

## 6. On the proposal to integrate models into MLJ

To achieve goal 2(ii) @DilumAluthge is proposing that for a class of models with the new subtype `ProbablisiticJoint`, we should declare that `predict(mach, Xnew)` return a representation of correlated predictions $p_{corr}$, evaluated on the rows `x_1`, ..., `x_k` of the table `Xnew`. (Actually, version 2 of the proposal just says this needs to be probability distribution, but in that case there is no suggestion as to how to fit the model into framework 1.)

As I understand it (and maybe I have this wrong) one then obtains the "vector of distributions" required for fitting the model into framework 1 by computing the marginals of `p_corr`? If that is so, and we call the result of this operation `predict_marginals(mach, Xnew)`, then it must be consistent, in the sense of 4. That is, we require

```julia
predict_marginal(mach, Xnew[1, :])[1] == predict_marginal(mach, Xnew)[1, :]
```

for any table `Xnew` with two rows. This is evidently not the case. An equal mixture of two binary classifiers already provides a counter example.

I would be very surprised if there is *any* way to construct a *consistent* "vector of distributions" from the correlated predictions `predict(mach, Xnew)`. Of course, the `predict` *function* (as opposed to a single *evaluation*) can be used to get this, as the last observation in 5 shows. But the idea that we can convert a `ProbablisiticJoint` model into a regular `Probabilistic` model by simply composing it with a marginalization operation (or *any* operation) would not appear to work, right?

## 7 Comment

Very happy to see a revised update to the proposal or have my misunderstandings corrected. However, my own view is that this is not the right approach. Since i.i.d Bayes models are expected to implement framework 1, they must share all the behaviour of the existing `Probabilistic` models and so ought to have this type. Extra functionality goes on top by adding methods, such as `predict_joint`. To flag those models that support the extra functionality, we could introduce a subtype `JointProbabilistic <: Probabilistic` or a trait. (Actually, the `implemented_methods` trait may already serve this purpose.)

I understand @DilumAluthge has already given this approach a lot of thought, which I appreciate:

> I believe that this would lead to way too much code churn throughout the entire MLJ ecosystem. Additionally, it would require a lot of breaking changes in both the MLJ ecosystem as well as all Julia packages that currently implement Probabilistic MLJ models. I think that this would be quite disruptive and would require a lot of person-hours.

It's not clear to me is why *adding* functionality should be disruptive. Could you give an example?