Probabilistic regressor chains with Monte Carlo methods

Jesse Read, Luca Martino

Probabilistic Regressor Chains with Monte Carlo Methods

Jesse Read[a,1], Luca Martino[b]

[a] *LIX, Ecole Polytechnique, Institut Polytechnique de Paris, France*
[b] *Dept. of Signal Theory and Communications, Universidad Rey Juan Carlos, Spain*

Abstract

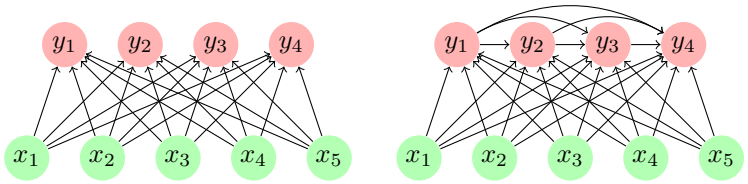
A large number and diversity of techniques have been offered in the literature in recent years for solving multi-label classification tasks, including classifier chains where predictions are cascaded to other models as additional features. Chaining methods have often providing state of the art results, and the idea of extending it to multi-output regression has already been trialed. However, these 'regressor chains' have seen limited applicability, on account of yielding relatively little predictive performance compared to individual regression models, and also limited interpretability. In this work we identify and discuss the main limitations of regressor chains, including an analysis of different base models, loss functions, explainability, and other desiderata of real-world applications. We develop and examine techniques to overcome these limitations. In particular we present Monte Carlo schemes in the framework of probabilistic chains. We show they can be effective, flexible and useful in different areas. Overall, we also place regressor chains in context among general multi-output learning with continuous outputs, and in doing this shed additional light on the applicability of chaining to machine learning tasks.

Keywords: multi-output regression, multi-label classification, regressor chains, classifier chains, Monte Carlo methods, particle filters

1. Introduction

Multi-dimensional data is ever-more present in industrial and scientific contexts. For example, multi-label classification has made a significant impact in the machine learning literature over recent years, where data points

[1] Corresponding author,



(a) Independent models (b) Chained models

Figure 1: The naive/independent approach (Figure 1a) vs chained models (Figure 1b) for multi-output prediction. Each of the target labels (in this example, there are 4 outputs) is learned by a base model, so each y_j-node represents a model where its inputs are shown as incoming arrows and prediction as an outgoing arrow. The class of base model depends on the type of target variable.

are naturally associated with multiple outputs[2]. Rather than a naive approach of building one model per output, advanced methods can model the outputs together, resulting in better predictive performance (and often efficiency). Although the potential of individual models is periodically revived under particular scenarios, the vast majority of literature proposes joint modeling, and as a result show improvement in both predictive performance and efficiency. A recent review of this area containing many useful references, is given in [1].

An established method in multi-label classification is that of *classifier chains*, where a model is trained for each label, but estimates of the other models are used as additional features, in a cascaded chain along the target labels. This is exemplified (and contrasted to the naive approach) in Figure 1.

This 'chaining' mechanism, although simple in its basic form, has proved successful in multi-label classification and provided dozens of modifications, extensions, and improvements, in the literature (see, e.g., [2, 3, 4, 5, 1, 6, 7] and references therein). It is flexible, in the sense that the base model class is a user hyperparameter, easily selected for different domains, and even relatively simple base models (e.g., off-the-shelf logistic regression) lead to greater predictive performance than if they had been used independently for each label.

In view of its success in classification, we are motivated to more thoroughly investigate the use of chaining methods in the regression context,

[2]Distinguished from *multi-class* classification where a single output may take multiple values, but only one such value is assigned per output

with continuous output variables. Although there have been instances of application already to the regression context (note [8, 9, 10, 11, 12]), many limits were shown in terms of functionality. Namely, the methods presented were limited to greedy inference which represents only a small part of classifier chains literature, rather than a generalizable probabilistic inference and the effectiveness (including predictive performance), and wider application was not yet widely explored. In some cases, predictive performance was not noteworthy at all, was not greater that using independent classifiers, or such a comparison was not shown. For example, in [9], good performance was achieved with decision tree regressor base models in ensemble, but this was not shown to be greater than corresponding ensemble of independent models. Again, this was also a greedy chain without providing a probablistic interpretation.

After summarizing background work (Section 2), we provide a rigorous discussion and development of probabilistic regressor chains, including a survey of possible approaches (Sections 3 and 4). Following this, we extend findings to develop a sequential Monte Carlo scheme (Section 5). This scheme allows sampling and evaluation of candidate paths through the target space, which is useful for many applications. We implement and test our approaches on synthetic and real-world datasets involving multiple continuous outputs (Section 6), the results of which we reflect upon in detailed discussion (Section 7). After looking at connections to related and potential future work (Section 8) we draw conclusions and make recommendations (Section 10).

2. Chain Methods

Given a dataset $\mathcal{D} = \{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^N$ of training instances $\boldsymbol{x}_i \in \mathbb{R}^D$ coupled with associated outputs[3] $\boldsymbol{y}_i \in \mathcal{Y} \subseteq \mathbb{R}^L$, we are interested building a model that can output predictions corresponding to the L target variables (labels), for any given input observation. In other words, given any $\boldsymbol{x} = [x_1, \ldots, x_D]$ our task is to produce $\hat{\boldsymbol{y}} = [\hat{y}_1, \ldots, \hat{y}_L]$, having previously observed $\mathcal{D}$.

In the multi-output *classification* scenario [2, 3, 4, 13], each j-th output takes some value $y_j \in \{1, \ldots, K_j\}$. We may highlight the popular case of binary outputs where each $y_i \in \{0, 1\}$, and thus $\mathcal{Y} = \{0, 1\}^L$, which is known often as *multi-label classification*). In the general case of continuous outputs, i.e., multi-output *regression* – then each $y_j \in \mathbb{R}$, and thus $\mathcal{Y} = \mathbb{R}^L$.

[3] In cases where it is not clear from context, we will denote x_{ij} as the value of the j-th attribute of the i-th instance, and y_{ij} as the j-th output associated with this i-th instance

A naive approach is to simply build a separate model h_j (a model class of the user's choice) for each target independently, such that

$$\hat{\boldsymbol{y}} = [\hat{y}_1, \ldots, \hat{y}_L] = [h_1(\boldsymbol{x}), \ldots, h_L(\boldsymbol{x})] \tag{1}$$

where each model h_j has been built from training set $\{(\boldsymbol{x}_i, y_{ij})\}_{i=1}^{N}$, as a traditional single-output classifier or regressor according to the domain of the j-th target label (e.g., clearly a regressor if $y_{ij} \in \mathbb{R}$). The exact class of model (type of regressor) is largely dependent on preference and/or driven by domain assumptions and constraints.

Particularly with regard to multi-output classification, a large volume of literature proposes a plethora of models that out-compete this baseline by modelling the dependencies among outputs; consider, e.g., [2, 3, 14, 4, 5, 13, 1] and references therein. The literature dealing explicitly with multi-output regression is smaller in volume (consider, [8, 9, 10, 12]).

In the classification context, the *classifier chains* method has been presented, analysed, and incorporated into other methods [2, 3, 4, 13] and under dozens of empirical studies provides state-of-the-art results. Due to the relatively large amount of work in the literature on this topic, including continued and recent interest (e.g., [15, 16] among others), we argue it is worth studying this mechanism in the context of multi-output regression, wherein we can call this method *regressor chains*.

In the simplest form of any chaining approach, the estimates of other labels are used as feature inputs for the following classifier, thus augmenting the input space along the chain. Given some test instance $\boldsymbol{x}$, we may obtain an estimate as

$$\hat{\boldsymbol{y}} = [\hat{y}_1, \ldots, \hat{y}_L] = [h_1(\boldsymbol{x}), h_2(\boldsymbol{x}, \hat{y}_1), \ldots, h_L(\boldsymbol{x}, \hat{y}_1, \ldots, \hat{y}_{L-1})] \tag{2}$$

where a recursion takes placed based on $\hat{y}_j = h_j(\ldots)$.

In the classification context, this is known as a *classifier chain* and, specifically, one with *greedy inference*. Note that models h_j can be estimated individually and in parallel at training time, similarly to those in Eq. (1), and – likewise – those models may be any class appropriate for each individual target (a binary classifier for binary output, etc).

The estimate $\hat{\boldsymbol{y}}$ is just one possible path/label combination in space $\mathcal{Y}$ of size 2^L. The development of probabilistic inference in classifier chains [3, 5] led to broadening inference into a search of this space, to select the best according to a given metric, typically 0/1 loss, which implies a maximum

a-posteriori (MAP) estimate[4], hence

$$\begin{aligned}\hat{\boldsymbol{y}} = h(\boldsymbol{x}) &= \operatorname*{argmax}_{\boldsymbol{y}\in\mathcal{Y}} P(\boldsymbol{y}|\boldsymbol{x}) \\ &= \operatorname*{argmax}_{\boldsymbol{y}\in\mathcal{Y}} \prod_{j=1}^{L} P(y_j|\boldsymbol{x}, y_1, \ldots, y_{j-1}),\end{aligned} \tag{3}$$

where distribution[5] $P(y_j|\boldsymbol{x}, \ldots)$ is associated with the j-th model. This is known as a *probabilistic classifier chain*. The need for a probabilistic interpretation can be addressed by building such models, e.g., logistic regression.

Since the search space in $\mathcal{Y}$ in Eq. (3) is exponential with the number of labels L, exhaustive inference (where the entire space of label combinations is explored) is usually prohibitive. Therefore, one may consider the search space as a probability tree, and conduct a tree search, where $P(y_j|\boldsymbol{x}, y_1, \ldots, y_{j-1})$ provides the outgoing branch weights from the node at the end of path $y_1, \ldots, y_{j-1}$. Figure 2 offers some visual intuition.

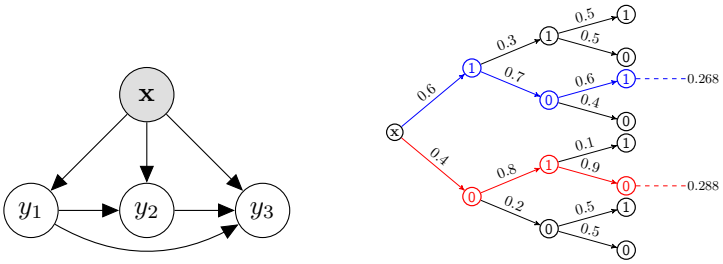


Figure 2: Probabilistic classifier chains where $L = 3, y_j \in \{0, 1\}$: As a probabilistic graphical model (left), and with two explored paths in the probability tree (right). Note that the best path (in red, right) is not found by greedy inference. There are 2^L possible paths ($\mathcal{Y} = \{0, 1\}^3$). The label on each edge indicates $P(Y_j = 1|\ldots)$; shown for explored paths only.

Many search methods have been applied for this purpose (a survey is given in [17]), for example, using Monte-Carlo sampling [4]. Using an adapted notation and terminology (so as to refer to in later sections of this work): this method takes M samples, $m = 1, \ldots, M$ across the chain,

[4]Although configurations for other losses are also possible; see [3]

[5]Formally, we are talking about the full conditional distribution $P(Y_j|\boldsymbol{X} = \boldsymbol{x}, Y_1 = y_1, \ldots, Y_{j-1} = y_{j-1})$ and associated probability mass function denoted $P(y_j|\boldsymbol{x}, \ldots) \equiv P(Y_j = y_j|\boldsymbol{X} = \boldsymbol{x}, \ldots)$ for reasons of brevity

and weighting each sample, as

$$y_j^{(m)} \sim P(\cdot|\boldsymbol{x}, y_1^{(m)}, \ldots, y_{j-1}^{(m)}) \tag{4}$$

$$w_j^{(m)} = w_{j-1}^{(m)} \cdot P(y_j^{(m)}|\boldsymbol{x}, y_1^{(m)}, \ldots, y_{j-1}^{(m)}) \tag{5}$$

where $w_j^{(m)}$ is the weight of the m-th sample at the j-th label; supposing $w_0^{(m)} = 1$. An example is shown in Figure 2 where $M = 2$. A final prediction is obtained as

$$\hat{\boldsymbol{y}} = \boldsymbol{y}^{(m^*)} = \underset{\boldsymbol{y} \in \{\boldsymbol{y}^{(m)}\}_{m=1}^{M}}{\operatorname{argmax}} P(\boldsymbol{y}|\boldsymbol{x}) \tag{6}$$

where $m^* = \operatorname{argmax}_{m \in \{1,\ldots,M\}} w_L^{(m)}$ (index of the maximum weight), and where $\boldsymbol{y}^{(m)} = [y_1^{(m)}, \ldots, y_L^{(m)}]$ is a complete sequence/combination of labels. Note that complexity is determined by $M \ll 2^L$.

In following the maximum, taking a single path, we recover greedy inference

$$\hat{y}_j = \underset{y_j}{\operatorname{argmax}}\, P(y_j|\boldsymbol{x}, \hat{y}_1, \ldots, \hat{y}_{j-1}) \tag{7}$$

across the chain, to obtain a single greedy estimate $\hat{\boldsymbol{y}}$. This method in particular can be applied in an off-the-shelf manner to the multi-output regression context (e.g., [18, 8]). However, as we highlight and discuss in the following section, even though the application is straightforward, there are some major differences that affect the relative results they obtain (i.e., relative to individual models / non-chained methods).

3. The Poor Behavior of Regressor Chains

By *poor* behavior, we mean in particular that regressor chains do not obtain out-of-the-box predictive performance that improves over independent models. This is quite different from the context of classifier chains where such improvement (over independent models) is widely reported. In this section we study and elaborate on the factors behind this; i.e., why regressor chains cannot be expected to outperform independent models without particular considerations.

Applying greedy inference in chains in the case of regression is – exactly as in classification – a case of each output simply being "plugged in" to the following model as an additional feature. Recall that this simply means that predictions $\hat{y}_1, \ldots, \hat{y}_{j-1}$ are treated as fixed observations (i.e., and not random variables) when inferring y_j. This greedy plug-in approach has been

trialed, e.g., in [8]. However, the results obtained do not justify this approach over independent regression models. Recent attention has resulted in more competitive performance, e.g., [10, 12], although – as we have mentioned above (and will expand on in the following and later in Section 7) – it was not exhaustively determined that such performance could not have been obtained by independent counterparts. But aside from strong absolute performance under squared error metrics, such as from these methods, there are also other aspects to consider which motivate a probabilistic approach. We now discuss the different reasons why off-the-shelf applications should be approached carefully.

As can be shown by Eq. (2) above, a basic application of regressor chains simply requires L regression models, trained with their respective extended feature spaces. The method of least squares regression could be considered a first choice, given its prevalence in the wider scientific literature, where a vector of parameters $\boldsymbol{\theta}_j$ represents each j-th model, such that $\hat{y}_j = h_j(\boldsymbol{x}) = \boldsymbol{\theta}_j^\top [x_1, \ldots, x_D, \hat{y}_1, \ldots, \hat{y}_{j-1}]$ (supposing column vectors). For simplicity of illustration, supposing single-dimensional input x and $L = 2$ outputs, we observe that

$$\begin{aligned} \hat{y}_2 &= h_2(x, \hat{y}_1) \\ &= \boldsymbol{\theta}_2^\top [x, \hat{y}_j] = \boldsymbol{\theta}_2^\top [x, \theta_1 x] = \theta_{2,1} x + \theta_{2,2} \theta_1 x = x(\theta_{2,1} + \theta_{2,1}\theta_1) \\ &= \theta' x \end{aligned} \tag{8}$$

i.e., earlier predictions $\hat{y}_1, \ldots, \hat{y}_{j-1}$ are superfluous in predicting $\hat{y}_j$; and a regressor chain only performs a series of linear transformations which can naturally be represented as a single operation. This effect was also noticed in [8].

The effect is similar even in the case where each input dimension x_j is observed only at *time* j; i.e., we estimate $\hat{y}_j = h(x_1, \ldots, x_j, \hat{y}_1, \ldots, \hat{y}_{j-1})$. Unfortunately prior labels $\hat{y}_1, \ldots, \hat{y}_{j-1}$ are still not strictly needed wrt $\hat{y}_j$ since all available information already comes from $x_1, \ldots, x_j$ directly via earlier labels, as illustrated in Figure 3.

Let us consider how data may be generated. Given input instance x, two outputs could be generated by a chain as follows:

$$\begin{aligned} y_1 &= \mu_1(x) + \epsilon_1(x) \\ y_2 &= \tilde{\mu}_2(x, y_1) + \tilde{\epsilon}_2(x, y_1) \\ &\equiv \mu_2(x) + \epsilon_2(x) \end{aligned}$$

with mean function μ_j and error function ϵ_j for each label j.

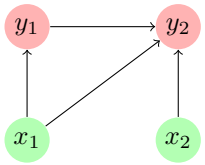


Figure 3: Even when x_2 arrives only at *time* $j = 2$, information can still be carried forward from x_1 (rather than via y_1), thus making the label cascade superfluous wrt the prediction $\hat{y}_2$ as long as $h(\boldsymbol{x})$ is well modeled.

Note that the third line is obtain via function composition: since y_1 is composed of functions of x, so is also y_2. Specifically, we then see that since both outputs are dependent on x and, in general, we may write,

$$\boldsymbol{y} = \boldsymbol{\mu}(\boldsymbol{x}) + \boldsymbol{\Sigma}(\boldsymbol{x}) \tag{9}$$

noting how both mean and covariance functions are dependent on input $\boldsymbol{x}$.

In Eq. (8), with a linear model, we see that correspondingly, there is no need to model y_1 if we are interested in y_2 (and vice versa). This is often not apparent in classifier chains, because a prediction for $\hat{y}_1$ used as a feature adds extra capacity into the model for predicting $\hat{y}_2$ in the form of a *non-linear* feature expansion (a base classifier is always a non-linear function so as to map real-valued input into the range of $y_1 \in \{0, 1\}$). And in that case the reduction in Eq. (8) no longer holds (which is to say: we obtain better results with a chain as opposed to independent models).

Thus far, it is clear that the base models of regressor chains should be in some way non-linear, if regression chains are to be of any benefit. But there is another fundamental difference of chains in the classification vs regression case: the evaluation metric.

In the regression context, we start naturally with the most popular choice for regression is certainly a squared error such as the mean squared error (MSE) loss criterion (considered also in the earlier work on regressor chains of [8, 9] among others). Under random variables $\boldsymbol{Y} = Y_1, \ldots, Y_L$, the mini-

mum MSE (MMSE) estimator under observation $\boldsymbol{x}$ is

$$\begin{aligned}\hat{\boldsymbol{\mu}} &= \mathbb{E}[Y_1, Y_2, \ldots, Y_L|\boldsymbol{x}] \\ &= \mathbb{E}[\boldsymbol{Y}|\boldsymbol{x}] = \int \boldsymbol{y} p(\boldsymbol{y}|\boldsymbol{x}) d\boldsymbol{y} \qquad (10) \\ &= \int \boldsymbol{y} \frac{p(\boldsymbol{y}, \boldsymbol{x})}{p(\boldsymbol{x})} d\boldsymbol{y} \\ &= \int \boldsymbol{y} \frac{p(y_1|\boldsymbol{x}) \prod_{j=2}^{L} p(y_j|\boldsymbol{x}, y_1, \ldots, y_{j-1}) p(\boldsymbol{x})}{p(\boldsymbol{x})} d\boldsymbol{y} \qquad (11) \\ &\propto \int \boldsymbol{y} \cdot p(y_1|\boldsymbol{x}) \prod_{j=2}^{L} p(y_j|\boldsymbol{x}, y_1, \ldots, y_{j-1}) d\boldsymbol{y}.\end{aligned}$$

requiring the full conditional densities $p(\cdot|\boldsymbol{x}, y_1, \ldots, y_{j-1})$ (homologous to the probability functions $P(\cdot|\boldsymbol{x}, y_1, \ldots, y_{j-1})$ of Eq. (3)). Note that $\hat{\boldsymbol{\mu}} = [\hat{\mu}_1, \ldots, \hat{\mu}_L]$ is a vector of L integrals, where we see that each marginal

$$\begin{aligned}\hat{\mu}_j &= \int y_j p(\boldsymbol{y}|\boldsymbol{x}) d\boldsymbol{y} \\ &= \int y_j p(y_j|\boldsymbol{x}) dy_j \qquad (12)\end{aligned}$$

decouples from Eq. (10), i.e., each element $\hat{\mu}_j = \mathbb{E}[Y_j|\boldsymbol{x}]$ is independent of other elements $\hat{\mu}_1, \ldots, \hat{\mu}_{j-1}, \hat{\mu}_{j+1}, \ldots, \hat{\mu}_L$ and they are not needed in this marginal estimation. This is similar to the case of Hamming loss in classifier chains: the risk minimizer shows decoupling [5] and precisely we expect fewer gains using classifier chains (even if we do obtain some, according to the non-linearity, as mentioned above and, e.g., in [19]).

More generally, we are not only interested in obtaining point-wise estimators $\hat{\mu}_j$, but we are also interested in extracting all the statistical information encoded within the posterior $p(\boldsymbol{y}|\boldsymbol{x})$, such as uncertainty measures, credible intervals, quantiles, and so on. Thus, our goal is to approximate complex integrals involving this posterior

$$p(\boldsymbol{y}|\boldsymbol{x}) = p(y_1, y_2, \ldots, y_L|\boldsymbol{x}) = p(y_1|\boldsymbol{x}) \prod_{j=2}^{L} p(y_j|\boldsymbol{x}, y_1, \ldots, y_{j-1}) \qquad (13)$$

Having an estimate of the shape of p then allows us to estimate other values aside from the expected value Eq. (10), such as the median or the mode. We remark again that classifier chains typically predicts a mode in

the form of a MAP estimate, easily tackled in the scenario of a probabilistic classifier chain.

We can illustrate with a synthetic toy example. Following from Eq. (9), we generate a synthetic dataset (henceforth denoted Synth) where $x \sim \mathcal{N}(0, 1)$ and

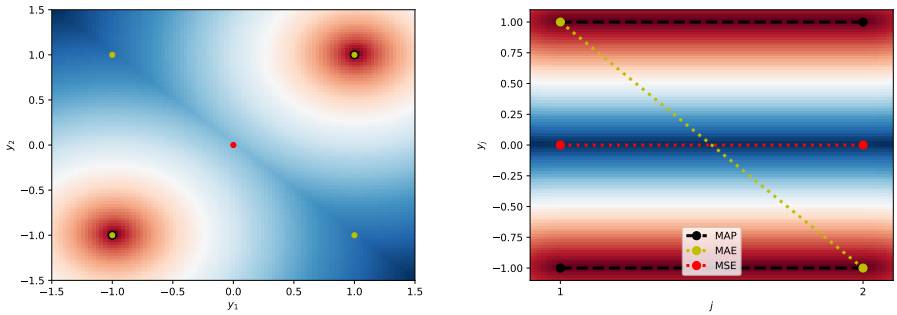
$$\boldsymbol{y} \sim \mathcal{N}(\boldsymbol{\mu}_k, \boldsymbol{\Sigma})$$

where

$$\boldsymbol{\mu} = \begin{cases} [+1, +1] & \text{if } k = 1 \\ [-1, -1] & \text{if } k = 0 \end{cases}, \quad \text{and} \quad \boldsymbol{\Sigma} = \mathbf{I}_2 0.1. \tag{14}$$

and $k \sim \{0, 1\}$ is drawn uniformly and randomly. This dataset thus has two labels, $\boldsymbol{y} \in \mathbb{R}^2$.

Note that x is ignored here; but later we look at a more complex case where mean and variance functions are dependent thereon. Figure 4 illustrates the generating distribution; showing the two distinct modes; and the output of different kinds of estimators.



(a) Generating joint distribution (b) Path view of estimators

Figure 4: The generating distribution of the Synth data; viewed as (a) joint distribution over two labels showing the surface over $\boldsymbol{y} = [y_1, y_2]$, and (b) equivalent path view showing y_1, y_2 in sequence vs index j on horizontal axis. Each is equally probable given x. Idealized MSE, MAP and MAE (mean absolute error) estimates (supposing knowledge of the true density) are shown in black, yellow, and red, respectively.

We remark that $\boldsymbol{\mu}$ in Eq. (9) and Eq. (14) is the mean of the k-th *local mode*, whereas the estimate $\hat{\boldsymbol{\mu}}$ of Eq. (10), is an estimated global mean of the distribution. This is an important distinction since, although in a Gaussian distribution the mean and mode is equivalent, this does not hold generally, and to estimate a mode in a bimodal distribution – such as in Synth – we require a model of posterior joint $p(\boldsymbol{y}|\boldsymbol{x})$ as in Eq. (13).

Even when using the chain rule to factorize $p(\boldsymbol{y}|\boldsymbol{x})$ this joint into its components, we find that the integrals are generally intractable; seen already in Eq. (11). This is unlike in probabilistic *classifier* chains where each $P(y_j|\ldots)$ is a discrete distribution, hence producing a finite probability tree (as in Figure 2) that one can explore with tree-search methods. In continuous multi-dimensional output space, there is no such tree.

One more issue is worth attention: It is well-known that classifier chains can suffer from *error propagation* [20] where an initial poor estimation raises the possibility of a cascade of poor predictions down the chain, particularly with greedy inference. In regressor chains this effect can be relativel much worse. In multi-label classification at most one extra bit of wrong information is propagated at each step of the chain, but in regressor chains an estimate may degenerate rapidly and and become progressively lost in $\mathbb{R}^j$ space as we propagate down the chain ($j \to L$). This strongly motivates a study of probabilistic chains, as we address in the following section, since it is known that in the classification case, iterative inference mitigates error propagation [3, 4].

We have thus far identified the main issues confronting the successful application of greedy regressor chains: linear regression models serving no additional benefit within a chaining mechanism, typical loss functions (e.g., MSE) not requiring joint distribution estimation, and thus neither the chain mechanism, and also a potentially catastrophic error propagation. Having identified the lacking in regressor chains, as compared to their classification counterpart, in the following section we develop the idea of probabilistic regressor chains to tackle these challenges, and then we build on this to suggest new methods.

4. Probabilistic Regressor Chains with Monte Carlo Search

As discussed above, probabilistic chains offer several mechanisms to outperform independent models, such as greater flexibility wrt loss metric (and the possibility to minimize 0/1-loss), and inference in the form of efficient tree-search. However, in the regressor chains context there is no inherent probability tree to sample from. We propose to use Monte Carlo methods to build such a tree. The idea is to take samples $y_j^{(m)} \in \mathbb{R}$ and use them as nodes in the tree, assigning an appropriate payoff (weight) to each branch, and to each full path. This implies two requirements: First, to be able to draw samples from the full conditional,

$$y_j^{(m)} \sim p(\cdot|\boldsymbol{x}, y_1^{(m)}, \ldots, y_{j-1}^{(m)}), \tag{15}$$

And secondly, to be able to evaluate each sample to provide a payoff, i.e., to assign a branch weight and also, a path weight. Branch weights are given by

$$w_j^{(m)} = w_{j-1}^{(m)} \cdot p(y_j^{(m)}|\boldsymbol{x}, y_1^{(m)}, \ldots, y_{j-1}^{(m)}). \tag{16}$$

This corresponds approximately to Eq. (4) and Eq. (5). As in that case, due to the recursion on w_{j-1}, it means that w_L carries the full path weight; we could denote $w^{(m)} = w_L^{(m)} = p(\boldsymbol{y}^{(m)}|\boldsymbol{x}) = \prod_{j=1}^{L} p(y_j^{(m)}|y_1^{(m)}, \ldots, y_{j-1}^{(m)}, \boldsymbol{x})$ from root to a leaf. To obtain a categorical distribution per level/label as in the discrete case, we can simply consider normalized weights $\bar{w}_j^{(m)} = w_j^{(m)} / \sum_m w_j^{(m)}$, such that $\sum_m \bar{w}_j^{(m)} = 1$ (noting that the summation over samples/branches, per j-th level of the tree).

Having this tree (effectively a set of weighted samples) is useful to minimize different cost functions and their respective estimators, for example a MMSE estimate,

$$\hat{\boldsymbol{y}} = \hat{\boldsymbol{\mu}} = \sum_{m=1}^{M} \boldsymbol{y}^{(m)} \bar{w}^{(m)} \tag{17}$$

or MAP estimate,

$$\hat{\boldsymbol{y}} = \hat{\boldsymbol{y}}^{(m^*)} \quad \text{where} \quad m^* = \underset{m}{\operatorname{argmax}}\, w^{(m)} \tag{18}$$

which is the homologue of Eq. (6) in the classification case (recall, $w^{(m)} = w_L^{(m)}$). We remark that, strictly, this is only a MAP estimate in the limit, as $M \to \infty$, due to the mode being an infinitesimally small range, i.e., a point.

Generally, given observation $\boldsymbol{x}$, a prediction may be given as path

$$\hat{\boldsymbol{y}} = g\left(\{\boldsymbol{y}^{(m)}, \boldsymbol{w}^{(m)}\}_{m=1}^{M}\right) \tag{19}$$

for appropriate function g over the tree/weighted samples[6].

Figure 5 shows an example illustration of using the weighted samples to form a probability tree where each complete (length-L) path from root to leaf representing a sample $\boldsymbol{y}^{(m)} \in \mathbb{R}^L$, with its associated weights.

To simplify the notation of the following equations, for the remainder of this section we denote

$$p_j := p(y_j|\boldsymbol{x}'_j) := p(y_j|\boldsymbol{x}, y_1, \ldots, y_{j-1}).$$

[6]Although individual branch costs are not always necessary when path cost $w^{(m)} \equiv w_L^{(m)}$ is sufficient, we nevertheless include it so as to depict a full tree

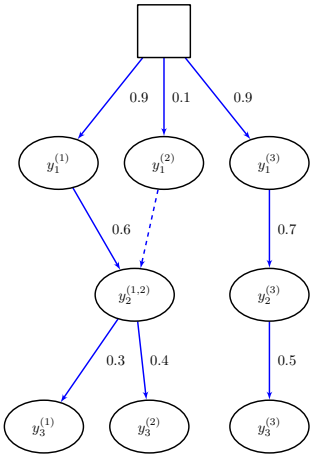


Figure 5: A hypothetical tree through output space, given $M = 3$ samples across $L = 3$ continuous target labels. Nodes correspond to samples $y_j^{(m)}$ (the m-th sample at the j-th label) and edges show $p(y_j^{(m)}|\boldsymbol{x}, y_1^{(m)}, \ldots, y_{j-1}^{(m)})$ (from level/label $j-1$ to j). According to this, the full path weights are $\boldsymbol{w} = [w_3^{(1)}, w_3^{(2)}, w_3^{(3)}] = [0.162, 0.216, 0.315]$ for the three possible paths. The dashed line indicates resampling (covered in Section 5) and is not a branch and has no weight associated.

The fundamental consideration is how to model each local density p_j, and how to draw samples from it, while taking into account the aforementioned issues. Let us now look at suitable methods that can be used to this end, thus, with the combination of Monte Carlo sampling, lead to varieties of probabilistic classifier chains. Later, in the following section, we further develop this idea into a particle filter (sequential Monte Carlo) method.

4.1. Discretization and classification

A simple approach is to discretize the output space of the problem and proceed from a classification perspective. This makes sense, since classifier chains are easy to justify, and a wealth of methods available. Of course not all problems are suited to such discretization, but in some areas it can be effective and is common practice, for example in the domain of reinforcement learning (see, e.g., [21]).

In this case we evaluate the pdf as

$$p_j = p(y_j|\boldsymbol{x}'_j) = P(y_j \in \mathcal{S}_j^{(k)}|\boldsymbol{x}'_j) \Leftrightarrow P(Y_j = k) \text{ for bins } \mathcal{S}_j^{(k)} \subset \mathbb{R},$$

a multi-class classification problem, solvable under a classifier chain over discrete labels and hence with no need to deal with integrals. Clearly it is

fundamental to choose a suitable set of bins $\mathcal{S}^{(k)}$ which define the finite set of class labels for each base model.

We can sample from p_j simply by selecting bin k with probability $|\mathcal{S}_k|/N$ (the proportion of the training set in that bin) and then returning the mean of samples in this bin.

4.2. Bayesian regression

We may elaborate p_j as a Bayesian linear regression model:

$$p_j = \mathcal{N}(y_j|\mu_j, \sigma_j) \tag{20}$$

with sufficient statistics μ_j and σ_j. Sampling from a Gaussian is straightforward, however, it based on a linear combination of $\mu_j = \boldsymbol{w}_j^\top \boldsymbol{x}'_j$ and provides a unimodal Gaussian-shaped estimate that is not suitable for multi-modal data.

4.3. Variational inference

We may approximate each p_j with some other distribution $q(y_j|\theta) \approx p_j$ as in variational Bayesian methods, which turns inference into the optimization problem

$$\theta^* = \operatorname*{argmin}_\theta \mathsf{KL}(q(y_j|\theta) \,\|\, p_j)$$

minimizing Kullback-Leibler divergence (KL); see, e.g., [22]. We can then sample $y_j^{(m)} \sim q(\cdot|\theta)$ as an approximation to $y_j^{(m)} \sim p_j$. Unlike Monte Carlo methods, this approach does not provide an exact model of p_j in the limit (given sufficient samples).

4.4. Kernel regression and density estimation

Noting Bayes rule,

$$p_j = p(y_j|\tilde{\boldsymbol{x}}'_j) = \frac{p(y_j, \tilde{\boldsymbol{x}}'_j)}{p(\tilde{\boldsymbol{x}}'_j)}$$

(where $\tilde{\boldsymbol{x}}'_j$ is our query instance) we may model the target density using a non-parametric method such as a Parzen window (i.e., a kernel density estimate). Given some kernel function K_γ (parametrized by γ),

$$p(y_j, \tilde{\boldsymbol{x}}'_j) = \frac{1}{N}\sum_{i=1}^{N} K_\gamma(\tilde{\boldsymbol{x}}'_j, \boldsymbol{x}'_{ij}) K_\gamma(y_j, y_i) \quad \text{and} \quad p(\tilde{\boldsymbol{x}}'_j) = \frac{1}{N}\sum_{i=1}^{N} K_\gamma(\tilde{\boldsymbol{x}}'_j, \boldsymbol{x}'_{ij})$$

Sampling $y_j^{(m)} \sim p(\cdot|\boldsymbol{x}'_j)$ can be carried out for certain kernels. For example, under the Gaussian kernel (which we use in our implementations later): one may choose an index i with probability proportional to $K_\gamma(\tilde{\boldsymbol{x}}'_j, \boldsymbol{x}'_{ij})$ and then draw from

$$y_j^{(m)} \sim \mathcal{N}(y_{ij}, \gamma)$$

As a lazy non-parametric method, the training set must be recalled and processed for each test example, and for each $j = 1, \ldots, N$, which implies pairwise comparisons and quadratic complexity wrt number of examples, unlike the density based on the discretization in Section 4.1.

5. Sequential Monte Carlo Regressor Chains

Experimental results in Section 6 show that the Monte Carlo regressor chains method described above can be effective. One potential limitation of, however, is that it relies on the approximation p_j of the conditional densities being suitable for density estimates (evaluation) as well as for obtaining samples. This can restrict model choice considerably in many contexts, for example with high-dimensional input observations or particular restrictions.

In this section we build a probabilistic regressor chain method where sampling and evaluation functions are separate: Particle filter regressor chains (PFRC). This method is inspired by the *particle filter* methodology (PF, see, e.g., [23]), however we make some particular considerations and adaptations for its application as a probabilistic regressor chain.

5.1. The particle filter

A particle filter consists of a model

$$\mathcal{M} : \begin{cases} f(y_j|y_{j-1}) \\ \ell(\boldsymbol{x}_j, y_j) \end{cases} \tag{21}$$

running over time-steps $j = 1, \ldots, L$, encompassing a transition function and observation function, f and ℓ, respectively. See [23] for an in-depth introduction and survey[7].

[7]For readers already familiar with the literature on particle filters and continuous state-space models, it is important to remark that we denote $\boldsymbol{x}$ as the observation and y_j as the label or 'state' label at time j. This may contrast with standard convention for a state-space model where $\boldsymbol{x}$ is the state

The vanilla particle filtering method for obtaining a marginal MMSE estimation for y_j as

$$y_j^{(m)} \sim f(\cdot|y_{j-1}^{(m)}) \tag{22}$$

$$w_j^{(m)} = w_{j-1}^{(m)} \cdot \ell(y_j^{(m)}, \boldsymbol{x}_j) \tag{23}$$

$$\hat{y}_j = \sum_{m=1}^{M} \bar{w}_j^{(m)} y_j^{(m)} \tag{24}$$

where, recall (as in Eq. (17)) that $\bar{w}_j^{(m)}$ denotes normalized weights, such that $\sum_m \bar{w}_j^{(m)} = 1$. The instance $\boldsymbol{x}_j$ is some observation that we observe at step/label j (implying that the test instance is $\boldsymbol{x} = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_L]$).

We highlight the strong connection to Eq. (4)–(6) in Monte Carlo classifier chains, at the same time pointing out the use of two separate functions, f and ℓ, in this context.

There are important differences from typical applications of particle filters, namely 1) in our case the model is learned from training data (i.e., no domain knowledge assumptions); 2) a single observation $\boldsymbol{x}$ is relevant to an entire sequence of states $y_1, \ldots, y_L$ (an *isotemporal* model); and 3) the full cascade can be considered rather than the standard single-order Markovian model (as developed in the following). In brief, at label j we consider $\boldsymbol{x}'_j = \boldsymbol{x}, y_1, \ldots, y_{j-1}$ rather than only $\boldsymbol{x}_j$ and y_{j-1} as in the single-Markov case.

5.2. Training

In our method, the training phase consists of learning the two functions; the transition function f_j and observation function ℓ_j, for each step in the chain $j = 1, \ldots, L$. Unlike the vanilla particle filter model described above, we consider the general case of taking into account the full chain history wrt each j.

For f_j, any suitable model of the density can be considered from which we are able to take samples

$$y_j \sim f_j(\cdot|\boldsymbol{x}, y_1, \ldots, y_{j-1})$$

(one may consider the methods given in sections 4.1–4.4). Optionally, to speed up the training process, we may consider only the marginal dependence $y_j \sim f_j(\cdot|y_{j-1})$.

Each function ℓ_j should provide us with an evaluation of the underlying density up to some normalizing constant, i.e.,

$$\ell_j(y_j, \boldsymbol{x}, y_1, \ldots, y_{j-1}) \propto p(y_j|\boldsymbol{x}, y_1, \ldots, y_{j-1})$$

Here we have more flexibility in function class, but accuracy is particularly important at this step. There is no need to impose a particular choice, as in the spirit of the chaining methodology we may consider the function classes a user-defined hyperparameter. We address several options in our experimental investigations in Section 6.

5.3. Inference

Algorithm 1 elaborates our Sequential Monte Carlo (SMC) scheme which is carried out on a test instance $\boldsymbol{x}$. We also need to specify the number of samples/particles M, depending on accuracy needs and computational constraints, a parameter η determining the threshold for particle degeneracy, and a function g which provides a desired estimator over the weighted samples; namely we can approximate the complex integrals involving of type Eq. (10) which represents a MMSE estimator, or other estimations such as the mode; recall, e.g., Eq. (17) and Eq. (18). We assume that the required functions have been provided by the training process.

The effective sample size (ESS) measurement decides when to carry out a resampling step and thus prevent sample degeneration (i.e., error propagation along the chain). Either

$$\widehat{ESS}(\bar{\boldsymbol{w}}) = \frac{1}{\sum_{m=1}^{M} \bar{w}_m^2}, \quad \text{or} \quad \widehat{ESS}(\bar{\boldsymbol{w}}) = \frac{1}{\max \bar{w}_m}$$

is typically appropriate [24], mapping a set of normalized weights to a scalar output indicating degeneracy, where lower values indicate greater degeneration and correspondingly greater risk of error propagation. The parameter η is a threshold on this value to determine if resampling is carried out, on a scale from $\eta = 0$ (never) to $\eta = 1$ (always). Resampling itself consists simply of drawing M samples from a discrete distribution such that item $y_j^{(m)}$ is taken with probability $\bar{w}_j^{(m)}$ (sampling with replacement).

The Markov Chain Monte Carlo (MCMC) / adaptive importance sampling (AIS) step (line 12) of the algorithm is not strictly necessary, but may be useful in cases if the sequential scheme is struggling; a Metropolis-Hastings variety of AIS based on [25] is given in Algorithm 2, which can be carried out following the resampling proceduce at at any j-th step of the

algorithm. After K iterations of this algorithm, it produces a final set of samples.

Note that in both Algorithm 1 and Algorithm 2 we make multiple use of the notation $1:M$ (or $1:L$) to indicate a vector, e.g., $w_j^{(1:M)} = \boldsymbol{w}_j = [w_j^{(1)}, \ldots, w_j^{(M)}]$; we do this to simplify the pseudocode without creating ambiguity as to the dimensionality of the vector.

Algorithm 1 Sequential Monte Carlo for Probabilistic Regressor Chains

1: **procedure** PREDICT($\boldsymbol{x}, M, \eta \in [0,1], g$)
2: Obtain models $\ell_{1:L}$, and $f_{1:L}$ from training stage
3: Set $w_0^{(m)} = \frac{1}{M}$ for all m.
4: **for** $j = 1, \ldots, L$ **do**
5: **for** $m = 1, \ldots, M$ **do**
6: $y_j^{(m)} \sim f(y_j | y_1^{(m)}, \ldots, y_{j-1}^{(m)})$ ▷ Draw sample
7: $w_j^{(m)} = w_{j-1}^{(m)} \frac{\ell(y_j^{(m)}, \boldsymbol{x}, y_1^{(m)}, \ldots, y_{j-1}^{(m)})}{f(y_j^{(m)} | y_1^{(m)}, \ldots, y_{j-1}^{(m)})}$ ▷ Compute weight
8: $\hat{Z} = \sum_{m=1}^{M} w_j^{(m)}$
9: Set $\bar{w}_j^{(m)} = \frac{1}{\hat{Z}} w_j^{(m)}$ for all m ▷ Normalized weights
10: **if** $\widehat{ESS}(\bar{w}_j^{(1:M)}) \leq \eta M$ **then**
11: $\tilde{y}_j^{(1:M)} \sim \{y_j^{(1:M)}; \bar{w}_j^{(1:M)}\}$ ▷ Resample
12: $\tilde{y}_j^{(1:M)} \leftarrow \text{MCMC/AIS}\left(\boldsymbol{x}, \tilde{y}_j^{(1:M)}\right)$ ▷ Optional, see Alg. 2
13: Set $y_j^{(1:M)} \leftarrow \tilde{y}_j^{(1:M)}$ ▷ Set new particles
14: Set $w_j^{(m)} \leftarrow \frac{1}{M} \hat{Z}$ for all m ▷ Reinitialize weights
return $\hat{\boldsymbol{y}} = g\left(\boldsymbol{y}^{(1:M)}, \boldsymbol{w}^{(1:M)}\right)$ ▷ Prediction*, see Eq. (19)

*where $\boldsymbol{y}^{(m)} = [y_1^{(m)}, \ldots, y_L^{(m)}]$ is a full path across L labels; thus $\boldsymbol{y}^{(1:M)}$ are M such paths.

Figure 6 provides further intuition. The circles refer to conditional samples (as drawn in line 6 of Algorithm 1), and the relative size is indicative of particle weight (computed at line 7). Recall that the form of f and ℓ is determined by the base classifiers. Different estimates are shown (according to function $g(\cdot, \cdot)$ in the final line of Algorithm 1), corresponding to Eq. (17) and Eq. (18), respectively. Training points and their distribution are shown in gray. Figure 7a shows the corresponding *path* view of all $\boldsymbol{y}$, again with inference samples and training samples (in the same colors as Figure 6); and –again– the different estimates.

Recall that, given a model that approximates p_j meeting both require-

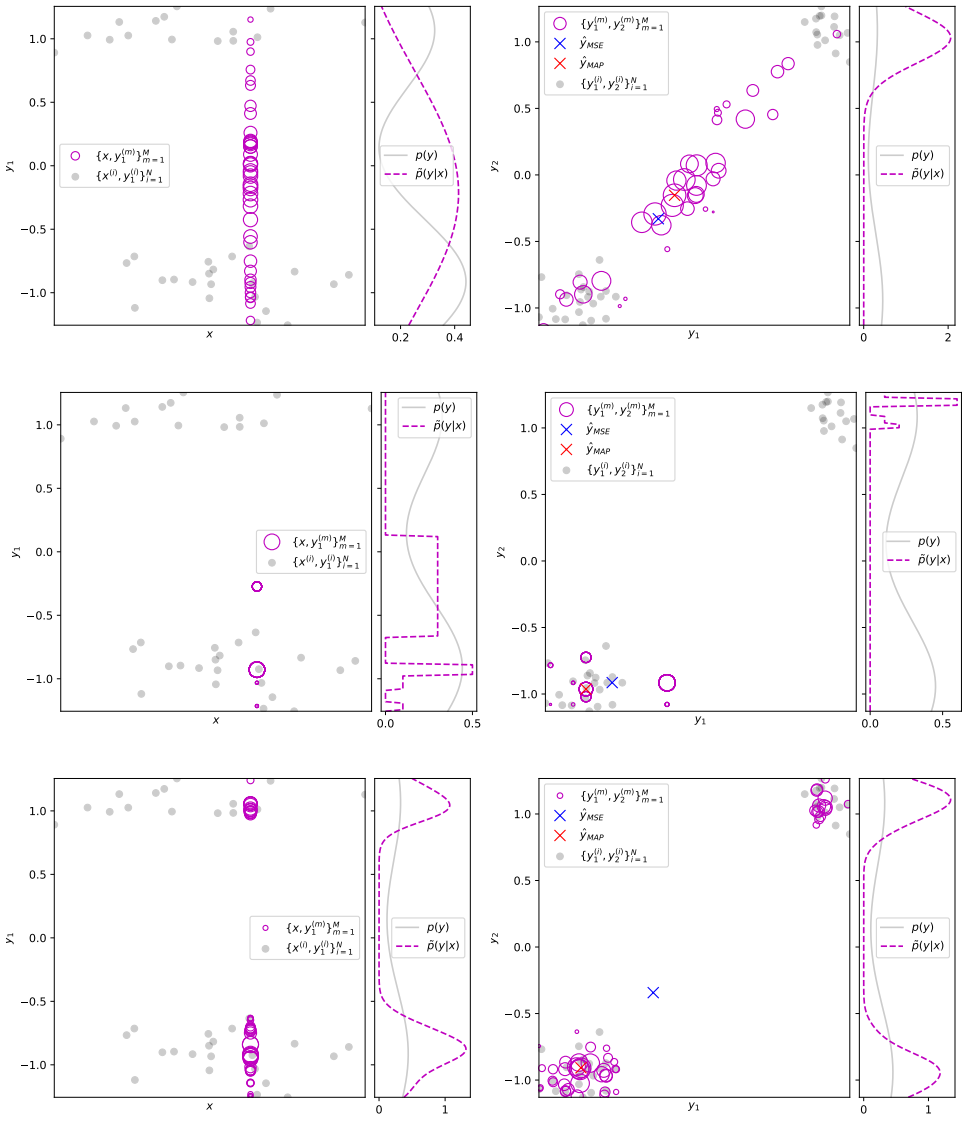
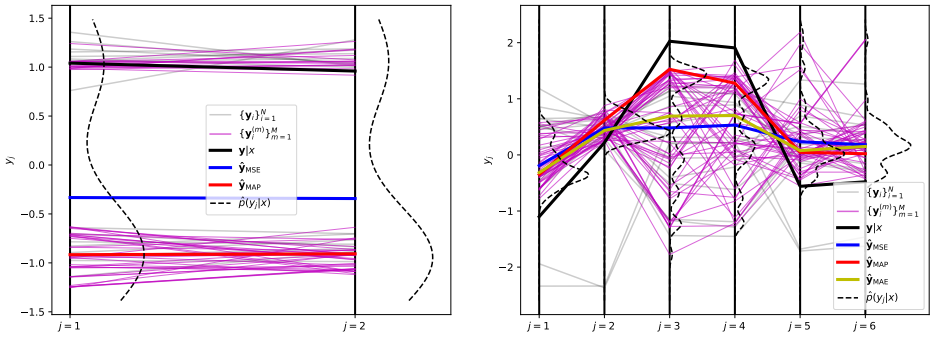


Figure 6: Illustration of Monte Carlo regressor chains; inference on the Synth data (described above in Section 3). For a given test instance x, samples (shown in magenta) are drawn across the chain, $y_1^{(m)} \sim f_1$ (left) and $y_2^{(m)} \sim f_2$ (right) where circle size $\propto \bar{w}_j^{(m)}$. The density estimate is shown in the right of each figure. Training points $x_i, \boldsymbol{y}_i$ are shown in grey. The function class (i.e., base model) used is Bayesian regression (top), discretization (mid); kernel regression/density estimation (bottom) – all described in Section 4.

.

Algorithm 2 Parallel Metropolis-Hastings (MH) Chains

procedure MCMC/AIS($\boldsymbol{x}, \tilde{y}_j^{(1:M)}$)
  Use proposal function q
  Let $\tilde{y}_{j,0}^{(1:M)} \leftarrow \tilde{y}_j^{(1:M)}$
  for $m = 1, \ldots, M$ **do**
    for $k = 1, \ldots, K$ **do**
      $y'_j \sim q(y|y_{j,k-1}^{(m)})$ ▷ Draw from proposal q
      Let $p_j(\cdot) = p_j(\cdot|\boldsymbol{x}, \tilde{y}_1^{(m)}, \ldots, \tilde{y}_{1:j-1}^{(m)})$
      $\alpha = \min\left[1, \frac{p_j(y'_j)q(y_{j,k-1}^{(m)}|y'_j)}{p_j(y_{j,k-1}^{(m)})q(y'_j|y_{j,k-1}^{(m)})}\right]$
      $y_{j,k}^{(m)} \leftarrow \begin{cases} y'_j & \text{with probability } \alpha \\ y_{j,k-1}^{(m)} & \text{with probability } 1-\alpha \end{cases}$
  return $[\tilde{y}_{j,K}^{(1)}, \ldots, \tilde{y}_{j,K}^{(M)}]$



(a) Synth data, $\hat{\boldsymbol{y}} = [\hat{y}_1, \hat{y}_2]$ (b) Andro data, $\hat{\boldsymbol{y}} = [\hat{y}_1, \hat{y}_2, \ldots, \hat{y}_6]$

Figure 7: Samples viewed as paths (for Synth (a) and Andro (b)) along the chain, from single test example $\boldsymbol{x}$. The approximation of each distribution $p_1, \ldots, p_L$ is shown horizontally at respective ticks $j = 1, \ldots, L$. Note that modal estimates cannot be made under MSE. The samples from our method are not guaranteed to be the correct mode/prediction (which is shown in **black**), but nevertheless are likely paths, according to the training data.

ments for sampling *and* evaluation, then line 7 of Algorithm 1 can be simplified to the simple Monte Carlo approach described above, in particular: Eq. (15) and Eq. (16).

6. Experiments

6.1. Methods and Implementation

In this section we empirically test the approaches we have identified, discussed, and developed above; that is the Monte Carlo methods (MCRC) discussed in Section 4, as well as further developed as a Particle Filter method (PFRC) in Section 5. We compare these methods to independent regression models (IR), standard regressor chains with greedy inference (RC), under different configurations, including different base regression models.

For convenience, Table 1 summarizes the methods and their configurations, with an abbreviation key as used in the results below. Each method takes some base model, which in turn may involve their own hyperparameters. Recall that the PFRC approach takes two base models, one is used for sampling and the other for weighting – not necessarily from the same model class. When base-model hyperparameters are indicated as a set, the best of the set was selected under 5-fold internal cross validation (on each training split). If not explicitly stated, then default parameters are used, according to the implementation of each method.

IR and RC are implemented in the well-known Scikit-Learn framework[8]. We implemented our novel contributions using the Scikit-Multiflow framework[9] [26]; which is based on Scikit-Learn.

We used a standard laptop machine (2.60GHz CPUs, with 8GB RAM) for experiments. We set DNF for any kernel methods on datasets with more than 1000 instances, as computational resources were insufficient. This highlights one of the main limitations of kernel methods as a base classifier – in contexts of larger datasets.

We recorded the best hyperparameters selected per dataset per base classifier. However, as there is no particular pattern or interesting conclusion to be drawn from these (a fairly event distribution among the possible values), so we do not discuss further.

[8] `https://scikit-learn.org`

[9] `https://scikit-multiflow.github.io/`; prior to publication, code is at `https://bitbucket.org/jmread/probabilistic_regressor_chains/src/master/`

Table 1: Methods compared in the experiments. Base models f (and ℓ) are shown below. In the case of discretization, the classifier is treated as a hyperparameter h, being one of: extra random trees (ET), logistic regression (LR), or multi-layer perceptron with two hidden layers of 30 units each (MLP). Any parameters not explicitly mentioned are default as per Scikit-Learn.

Key	Method
IR_f	Independent Regression models
RC_f	Regressor Chains (greedy inference)
MC_f	Monte Carlo Regressor Chains MCRC ($M = 100$, $\eta = 0.1$)
PF_f^ℓ	Particle Filter Regressor Chains PFRC ($M = 100$, $\eta = 0.1$)
Key	Base regression model
B	Bayesian Regression
K	Kernel Regression (Gaussian Kernel, $\gamma \in \{0.01, 0.1, 1, 10, 100\}$)
D	Discretization $k \in \{20, 30, \ldots, 60\}$, $h \in \{\mathsf{ET}, \mathsf{LR}, \mathsf{MLP}\}$

6.2. Datasets

We compared methods on the real-world sets described and referenced in [9][10]; covering a number of real-world applications involving predicting the multi-components of sea-water, residential buildings, concrete pouring, and natural resources; i.e., a variety of problem domains. All outputs are standardized.

We also included in the experiments the synthetic dataset Synth described earlier in Section 3 (recall in particular Figure 4). And additionally we developed a more complex dataset, for simulating paths through an urban environment: Traffic. This was inspired by the real-world data gathered for and treated in [27], but in this case considering a continuous output space.

In Traffic, a prediction $\hat{\boldsymbol{y}} = [\hat{y}_1, \ldots, \hat{y}_L]$ corresponds to a path (i.e., trajectory, or route) that a traveler is estimated to take, having observed noisy observation $\boldsymbol{x}$. The particular assumption here is that it is important to estimate paths that fall within appropriate transportation axes. For example, a bus does not pass through a river but over one of the bridges that cross it. This encourages the prediction of a mode (near a commonly traversed path), rather than a mean estimator (average path).

[10] Available online: `http://mulan.sourceforge.net/datasets-mtr.html`

Given representation $\boldsymbol{x} \in \mathbb{R}^3$, each output vector is generated as follows:

$$\begin{aligned}
&\mu_j \sim \rho_j(\cdot|\boldsymbol{x}, y_1, \ldots, y_{j-1}) && \triangleright \text{ select one of } K \text{ modes } \mu_j \sim [\mu_1, \ldots, \mu_K] \\
&\epsilon_j \sim \phi_j(\cdot|\boldsymbol{x}, y_1, \ldots, y_{j-1}) && \triangleright \text{ deviation of } y_j \text{ from mode } \mu_j \\
&y_j = \mu_j + \epsilon_j
\end{aligned}$$

for $j = 1, \ldots, L$, where ρ_j, ϕ_j are models based on those [27]; sufficient to generate realistic mobility among modes. And the K modes are pre-generated as a random transport graph. Notice that covariance among outputs varies according to input $\boldsymbol{x}$ (as in the general case of Eq. (9)). Figure 8 illustrates a subset of data generated under this scheme, with $L = 4$, but for experiments (where we denote Traffic) we set $L = 30$ and $N = 10\,000$ in experiments. Multiple dimensions per real timestep (e.g., position coordinates) are easily considered without loss of generality, since there is no explicit time order in $\{1, \ldots, L\}$ to the chain in our methods (chain order is discussed in Section 9.1).

7. Results and Discussion

In this section we present and discuss empirical results. Namely, our first goal is to demonstrate the performance and behavior of of classifier chains in general, and specifically to the developments in this work. Secondly, we investigate the effectiveness of our proposed methodologies; both in terms of acceptable performance, and potential applicability.

Results are shown in Tables 2–3, respectively for the following loss metrics: *mean squared error*,

$$\mathsf{MSE} = \frac{1}{N} \sum_{i=1}^{N} \sum_{j=1}^{L} (y_{ij} - \hat{y}_{ij})^2$$

and the *uniform cost function*,

$$\mathsf{UCF} = \frac{1}{N} \sum_{i=1}^{N} \begin{cases} 0 & \text{if } \|\hat{\boldsymbol{y}}_i - \boldsymbol{y}_i\|_2 < \frac{\delta}{2} \\ 1 & \text{otherwise} \end{cases}$$

such that a prediction $\hat{\boldsymbol{y}}$ will incur 0 loss if it falls area within δ of the true path $\boldsymbol{y}$ (we set $\delta = 0.1$), and 1 otherwise.

Averaged results over 10-fold cross validation are provided in Tables 2 and 3 for MSE and UCF, respectively, under all datasets, for the methods considered in Table 1. Note that numbers in the tables are set in bold if

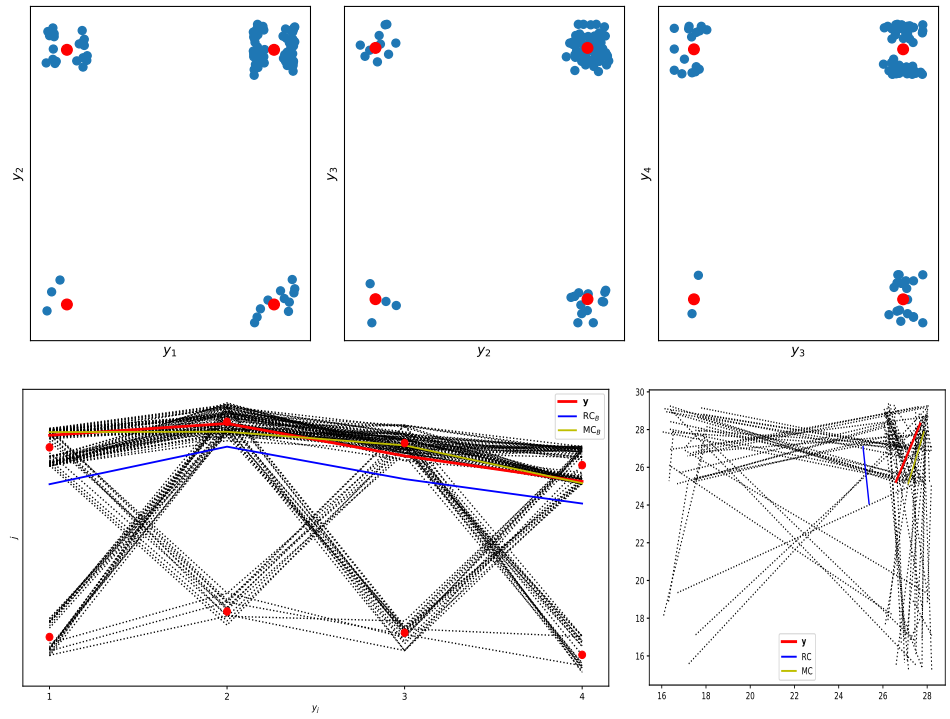


Figure 8: A subset of the generated Traffic dataset ($N = 100, L = 4$). The top row of plots shows the data; y_j vs y_{j+1} for $j = 1, 2, 3$. The centroids μ_k of local modes are shown in red. Note the high non-linearity of the distributions and also varying density. At the bottom left is a corresponding path-plot (comparable to those in Figure 7), and bottom right shows the same paths cast into a 2-dimensional map such that path y_1, y_2, y_3, y_4 becomes a line from point (y_1, y_2) to (y_3, y_4). Training data is shown in black, alongside some true $\boldsymbol{y}$ and two example estimates; our Monte Carlo method (MC) and baseline regressor chains (RC) – which we see falls outside of commonly-traversed paths.

they are the best value in each row (per dataset), however results are then rounded to two decimal places for display, so minor differences may not be visible.

It is easy to confirm that greedy regressor chains (RC) shows little to no advantage against independent regression models (IR) when a linear base model is used (only a small exception under the Andro dataset) by comparing RC_B and IR_B (recall, B denotes Bayesian regression – a linear model in this case). These findings are in line with the analysis so far. Models for classification involve an inherent non-linearity (such as for example the sigmoid function in logistic regression) which adds predictive power via the chain structure, leading to good performance of these methods, and a non-

Table 2: Results of 10-fold cross validation, under MSE.

Dataset [L]	IR_B	IR_K	RC_B	RC_K	MC_B	MC_D	MC_K	PF_D^B	PF_K^B
synth [2]	1.05	1.05	1.05	1.10	**1.03**	1.07	1.08	1.05	1.06
traffic [30]	0.31	DNF	0.30	DNF	0.31	**0.03**	DNF	0.12	DNF
andro [6]	0.64	0.27	0.53	**0.26**	0.54	0.41	0.36	0.55	0.49
atp1d [6]	0.18	0.56	0.19	0.57	0.19	**0.18**	0.28	0.23	0.27
atp7d [6]	0.30	0.70	0.31	0.70	0.32	**0.27**	0.64	0.40	0.64
edm [2]	0.59	0.45	0.59	**0.43**	0.60	0.64	0.53	0.58	0.82
enb [2]	0.10	DNF	0.10	DNF	0.10	0.02	DNF	**0.02**	DNF
jura [3]	0.38	**0.33**	0.38	0.33	0.38	0.37	0.48	0.40	0.86
oes10 [16]	**0.13**	0.84	0.14	0.84	0.14	0.31	0.30	0.50	0.29
oes97 [16]	**0.19**	0.84	0.19	0.85	0.19	0.43	0.31	0.46	0.31
osales [12]	0.55	DNF	**0.55**	DNF	0.56	3.55	DNF	2.99	DNF
rf1 [8]	0.34	DNF	0.36	DNF	0.36	**0.10**	DNF	0.99	DNF
rf2 [8]	0.19	DNF	0.19	DNF	0.19	**0.10**	DNF	0.10	DNF
scm1d [16]	**0.15**	DNF	0.15	DNF	0.15	1.08	DNF	1.09	DNF
scm20d [16]	0.41	DNF	**0.41**	DNF	1.01	1.01	DNF	1.10	DNF
sf1 [3]	**1.00**	1.00	1.00	1.01	1.10	2.12	1.10	2.26	1.10
sf2 [3]	0.94	DNF	**0.94**	DNF	1.09	4.06	DNF	4.22	DNF
slump [3]	0.49	**0.42**	0.49	0.42	1.14	1.37	1.19	1.03	1.08
wq [14]	0.91	DNF	**0.91**	DNF	1.11	1.10	DNF	1.11	DNF
Avg Rank	2.89	4.30	2.84	5.50	4.47	4.11	5.90	5.00	6.10

Table 3: Results of 10-fold cross validation, under UCF ($\delta = 0.1$)

Dataset [L]	IR_B	IR_K	RC_B	RC_K	MC_B	MC_D	MC_K	PF_D^B	PF_K^B
synth [2]	1.00	1.00	1.00	1.00	1.00	0.56	**0.55**	0.62	0.62
traffic [30]	0.96	DNF	0.95	DNF	0.94	0.31	DNF	**0.15**	DNF
andro [6]	1.00	0.71	0.96	**0.69**	0.97	0.73	0.77	0.94	0.94
atp1d [6]	0.54	0.46	0.55	0.46	0.68	**0.43**	0.54	0.68	0.77
atp7d [6]	0.83	0.51	0.83	0.51	0.87	**0.32**	0.51	0.66	0.74
edm [2]	0.89	0.67	0.87	0.63	0.87	0.55	**0.42**	0.93	0.53
enb [2]	0.22	DNF	0.22	DNF	0.22	**0.08**	DNF	0.27	DNF
jura [3]	0.73	0.71	0.73	0.70	0.74	**0.69**	0.72	0.82	0.91
oes10 [16]	0.94	0.93	0.94	**0.93**	0.99	0.96	1.00	0.98	1.00
oes97 [16]	**1.00**	1.00	**1.00**	1.00	1.00	1.00	1.00	1.00	1.00
osales [12]	**0.97**	DNF	0.97	DNF	1.00	1.00	DNF	1.00	DNF
rf1 [8]	0.45	DNF	0.46	DNF	0.75	**0.02**	DNF	0.98	DNF
rf2 [8]	0.36	DNF	0.36	DNF	0.59	**0.02**	DNF	0.13	DNF
scm1d [16]	**0.95**	DNF	0.95	DNF	1.00	1.00	DNF	1.00	DNF
scm20d [16]	**1.00**	DNF	1.00	DNF	1.00	1.00	DNF	1.00	DNF
sf1 [3]	0.43	0.43	0.43	0.40	0.80	1.00	**0.18**	1.00	0.49
sf2 [3]	**0.32**	DNF	**0.32**	DNF	0.83	1.00	DNF	1.00	DNF
slump [3]	0.82	**0.70**	0.85	0.71	0.99	0.99	0.95	0.81	0.94
wq [14]	**1.00**	DNF	**1.00**	DNF	**1.00**	**1.00**	DNF	**1.00**	DNF
Avg Rank	3.53	3.20	4.11	3.10	5.58	3.37	4.10	5.42	6.30

linearity is therefore also needed for chaining to obtain such an advantage in the regression scenario.

We do not need to discuss the power of linear vs non-linear modeling for regression, as this is an elementary concept, but it is particularly interesting to observe how well regressor chains with a non-linear base learner (e.g., RC_K) can perform better on average (i.e., in terms of average rank) against independent models (IR_K). This again highlights the need for a non-linear base learner in regressor chains.

Although regressor chains may thus be effective, we can recall the potential severity of the degeneration of estimates across the chain with poorly regularized base models. Indeed, we found this regressor chains with basic (non-regularized) linear regression models, estimates diverged so quickly and so far (obtaining even greater than 1000 MSE) that it there was no value to even include these results. Hence, the reason we chose Bayesian regression as a baseline base model; the inherent regularization prevents catastrophic divergence. For a similar reason we included resampling mechanisms in our probabilistic regressor chains (MCRC and PFRC); to remove samples that diverge away from the main probability mass.

Results show that some of the best predictive performance can be obtained by the Monte-Carlo approaches MCRC and PFRC (MC and PF, respectively, in the tables). For example, we can highlight that MC_D, MC_K, and PF_D^B obtained 7, 3 and 1 wins, respectively under UCF. It is not surprising that MC_B did not obtain top results, particularly under the UCF metric, due to the unimodal nature of Bayesian regression (this could be observed earlier in Figure 6), although correspondingly it performed reasonably well under MSE. Although greedy RC_K indeed performs well in many settings, particularly under UCF, we can remark that being tied to a kernel method has particular major limitations in scalability, incurring many DNFs.

It is worth taking a closer look at particular performance aspects, for example those which are distinguished on the synthetic dataset Synth: a loss of 1.00 (all standard IR and greedy RC approaches) vs 0.55–0.62 (among our approaches) under UCF. We note that 0.50 would be the Bayes optimal result for this data as we have controlled the distribution to have two randomly assigned modes with that probability. The gains are even sharper under Traffic, and echoed under MSE.

Recall (Figure 4 and in Figure 6) that any path crossing a low density region is not likely to ever occur in the data, even though the best estimate for MSE lies precisely across such a region. Our proposed Monte-Carlo approaches model the underlying density and thus are able to identify its modes, and to traverse this high-density area along the chain.

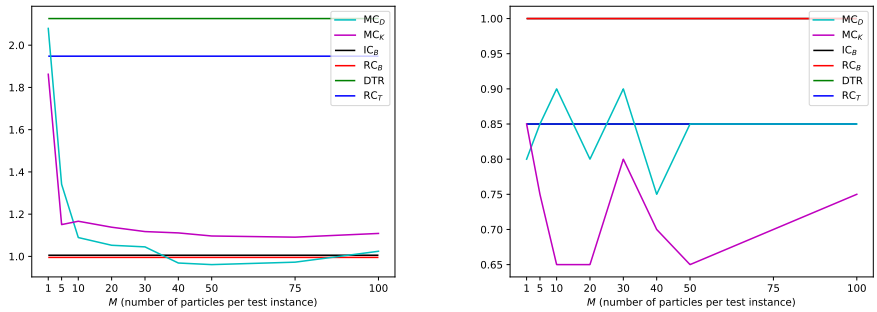
Having a probabilistic model of the density and its different characteristics offers explainability regarding the actual paths taken (as seen already in, e.g., Figure 6 and, particularly, Figure 7). One can study a set of hypothetical paths accounting for uncertainty, rather than a single estimation. This is useful in many domains, such as time series forecasting and anomaly detection, where a set of possible explanations is significantly more useful than single best guess. Indeed, in predicting routes (such as we simulated in Traffic), the best estimate under MSE may correspond to a road vehicle traversing a lake, yet a set of likely paths (e.g., around each shore of the lake) is of more practical use in many applications.

Overall, the need for interpretable models is of increasing interest as machine learning methods are used in more sectors, especially such as medical, security, and legal domains where providing a detailed explanation of results is a requirement [28].

In terms of running time complexity (per test instance), both MC and PF take M-times longer than RC to output a prediction, due to the M samples taken at each step in the chain. However, we also show that methods are effective with a relatively small number of samples (see Figure 9) and a few dozen samples are sufficient to achieve asymptotic convergence regarding MSE (vs RC and IR) as well as best results under UCF. Although the linear increase in running time is obvious in the figure, we emphasise that this is still only a fraction of a second per test example, and equivalent to obtaining votes from an ensemble of M models; yet an ensemble being relatively much more costly at training time. We can also bear in mind that most variance in running time in our experiments does not come from the difference in the specific type of chain inference, but from their base regression models. In particular, kernel methods, as mentioned above. Especially we remark that *training time* is identical for an identical base estimator in greedy regressor chains (RC methods) as compared to our methods.

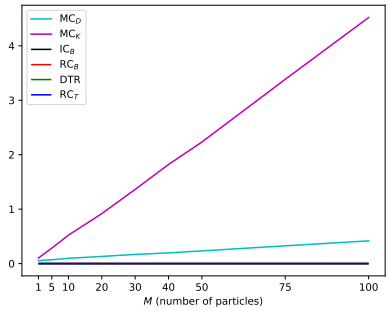
Although in this work we have been specifically interested in the comparison among probablistic regressor chain methods (and their respective baselines, of independent regression models), as reflected in the main results, the reader should also recall recent work on regression trees (as we have discussed above) for multi-output regression. And Figure 9 additionally includes a small comparison versus both multi-output decision tree regression and regressor chains with decision trees as base regression models.

We specifically chose synthetic elements of Traffic to highlight a context which is favorable to probabilistic regressor chains (recall: Figure 8); namely non-linear and multi-modal aspects. We can remark that these aspects can also be found in real-world data. Indeed, we found such characteristics



(a) MSE

(b) UCF



(c) Running time (seconds)

Figure 9: Corresponding to the data of Figure 8; the relationship between the number of samples (horizontal axes) and evaluation metrics MSE, UCF, and running/test time, respectively. A number of baseline methods are provided. There is a linear cost in running time for this performance.

precisely in the datasets on which our probabilistic chain inference performs best in Table 3. For example, Figure 10 displays an analysis wrt the dataset rf2.

8. Related Methods

To the best of our knowledge this is the first work treating regressor chains in depth, and particularly from a probabilistic point of view with a sequential Monte Carlo approach. Nevertheless, since multi-output regression is a well-established problem it is also important to discuss related methods and techniques. We do so in this section.

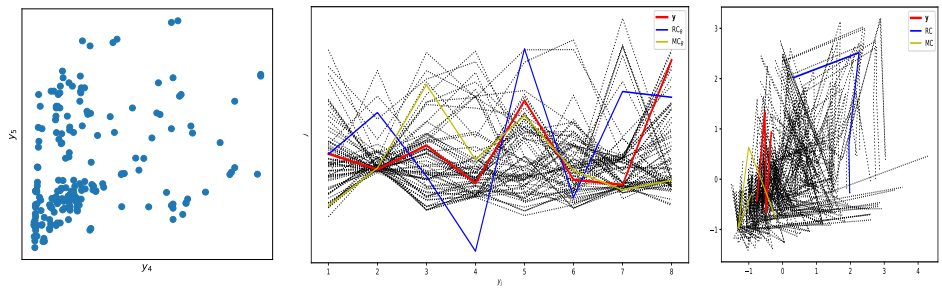


Figure 10: Real datasets exhibits non-linear and multi-modal aspects. In this example, the dataset rf2 is examined (see also, Figure 8 for comparison).

8.1. Multi-output Regression

The multi-output regression problem has been approached several times in recent decades, particularly wrt joint regularization [29]. Several such approaches take the form of

$$\boldsymbol{y} = \mathbf{B}\boldsymbol{\theta}\boldsymbol{x}$$

where $\mathbf{B}$ is a matrix used to to perform shrinkage on the ordinary (independent) linear regression estimates $\boldsymbol{\theta}$. See [8] and references therein for details and variations.

8.2. Neural networks

Greedy inference in a regressor (or classifier) chain can be seen as a particular feed-forward pass across a neural network, where base models represent activation functions. In particular, it is a network with *skip layers*, since each node passes the input directly to further nodes down the chain. Skip layers are common in modern neural network architectures such as ResNets [30], although often in a layer-wise rather than node-wise fashion, and only the final layer is used as output. However, layer-wise multi-label architectures have indeed been proposed (including, e.g., [31, 19], although these cited works deal primarily with classification).

8.3. Probabilistic models

A closely related approach to the inference in probabilistic regressor chains can be seen as performing a standard regression with *noisy inputs* [32, 33]. Indeed, suppose that $L = 2$, and

$$\begin{aligned} y_1 &\sim p(y_1|\boldsymbol{x}), \\ y_2 &\sim p(y_2|y_1, \boldsymbol{x}). \end{aligned} \tag{25}$$

Under the assumption of additive noise, we can write

$$y_1 = h_1(\boldsymbol{x}) + \epsilon_1 \tag{26}$$
$$y_2 = h_2(\boldsymbol{x}, y_1) + \epsilon_2, \tag{27}$$

where h_j are the regression models. For performing proper inference, all the statistical features of y_1 should be taken into account (i.e., the uncertainty), hence a regression problem with noisy inputs. Gaussian Processes (GPs) provide a method relevant to this context, e.g., [34, 35], and in particular warped GPs [36, 37, 38] which are a kind of hierarchical GP, thereby implying a form of 'depth'.

8.4. State space models

The previous considerations have direct application to the inference and prediction in so-called state-space models. Such models are formed by a transition dynamic equation and an observation equation[11],

$$\begin{cases} y_{t+1} = h_d(y_t) + \epsilon_{d,t}, \\ x_{t+1} = h_o(y_{t+1}) + \epsilon_{o,t}, \end{cases} \tag{28}$$

where y_t is the *state* at time t, and ϵ is perturbation noise. Note that even if the mappings h and noises ϵ are known, the inference and prediction in this stochastic system can be interpreted, at each t, also as a noisy input regression problem since

$$x_{t+1} = h_o(h_d(y_t) + \epsilon_{d,t}) + \epsilon_{o,t}. \tag{29}$$

showing strong parallels with the noisy regression model above. Furthermore, when the elements of the dynamic system are not deterministically known the problem becomes even more complex. In [39, 40], the authors suggest modeling h_d and h_o as two GPs.

9. Other Chaining Considerations

9.1. Chain order

The classifier chain literature many papers find that predictive performance can change significantly after permuting the order of labels in the

[11] Once more, we have opted for notation where x is the observation

chain, and many methods leverage this fact, for example with an ensemble of random orders ([2, 14]), hill-climbing towards a local maximum (e.g., [13, 4]) and even using a heuristic based on label dependence to order the chain (e.g., [41, 42]). Some of these have already been looked at in the regression context, e.g., [9, 10].

A full methodological development of chain ordering/structuring mechanisms is beyond the scope of this work. Nevertheless, we investigated the effect of different order on the methods we propose and respective baselines. Namely, Figure 11 shows that, despite the additional stochasticity of the inference of the Monte Carlo methods (as with any such method drawing random samples), the samples offer a stabilizing affect; and different orders have a relatively small impact on results.

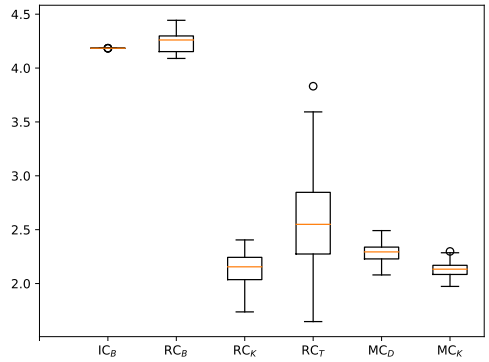


Figure 11: Summary of MSE results for different methods on the Andro data under 50 different chain orders (with an identical train-test split). MC uses $M = 100$ particles, and this helps it remain reasonably robust to changes in chain order; unlike greedy regressor chains with tree-based (RC_T) or, to a lesser extent, kernel-based (RC_K) base models, in this case.

Thus chain order is not a primary issue in our case. On the other hand, we recall that the space and complexity of structure is exactly the same in regression and classification, and most ensemble and hill-climbing methods need no specific adaptation (to measure dependence of continuous variables, in the case where such a heuristic is considered). Therefore, we can point the interested reader to the classifier chains literature for a study of the application of such methods and their respective complexity.

9.2. Multiple passes

Another consideration is the possibility of making more than one pass over the set of variables (i.e., down the chain) multiple times per test instance. This is an interesting proposal and would be an easy extension to our methods: simply extend the chain twice in length. One can view this as a special case of *regressor stacking*, as mentioned in [8] (in the general case, we simply feed all predictions as inputs to a second model – not necessarily a chain model). On the other hand, time complexity increases significantly with each pass along the chain.

With greedy inference, one can also see a connection to recurrent neural networks (RNN), as in recent work by [43], where the network is unrolled across time. From the probabilistic perspective, an approach of many passes can be seen as a form of Gibbs sampling on an undirected chain. In fact, an undirected and fully connected network (rather than a 'chain') removes the question of label order entirely. This has been developed in the context of *conditional dependency networks* [44] for classification, but we notice that this framework is also applicable to the regression case, whenever sampling from the conditional is possible. Namely, adjusting the equations for continuous target variables, we see that Gibbs sampling provides an estimate the full conditionals:

$$y_j^{(m)} \sim p(y_j|\boldsymbol{x}, y_{\neq j}) \tag{30}$$

$$\hat{\mu}_j = \frac{1}{M - M_0} \sum_{m=M_0}^{M} y_j^{(m)} \tag{31}$$

where $\neq j$ denotes all indices except j, and M_0 indicates an initial number of burn-in samples, which are later discarded. Precisely on account of this burn-in time the number of samples must be relatively much greater than the Monte Carlo samples in our methods.

9.3. Other areas of application

Regressor chains can be applied to any problem involving multiple continuous output variables. Time series forecasting is a natural application, e.g., [34], where one could simply include future values as the labels, and past/present values as the input instance. Previously classifier chains was applied to a discretized version of route prediction in urban traffic modeling [27]. An application of regressor chains in continuous space is natural in this setting, as we have initiated in Section 6.

Dealing with continuous action spaces in reinforcement learning is another possibility for regressor chaining. Forming a tree to search on top

of a continuous space, as in [21], is indeed related to the problem we have tackled in this paper. Interestingly, we noticed that the authors of this cited work use a kernel regression approach which is also what we have empirically found to work well in this work on regressor chains. Unlike our study, there is no use of a chain in the sense of a cascade but such an investigation of this aspect could be promising.

10. Conclusions

In this work we have looked at chaining models in the regression context, called regressor chains. This was motivated by the fact that, although such chaining models have been established in the multi-label classification literature and the its application to multi-output regression already attracted initial interest, the resulting behavior of regressor chains failed to achieve satisfactory and wide-reaching results. We carried out an in-depth study to identify, unravel, and overcome the weaknesses of this method in the regression context.

Namely, we could identify and explain (and expand, where relevant findings existed) several important issues in this regard:

- Regressor chains are only useful when non-linearities are used in the base models. This is equally applicable in the non-isotemporal case where where attribute values arrive over time
- It is difficult to justify regressor chains if minimizing a squared error is the goal, unless paying particular attention to the previous point
- Error propagation along the chain can be much more severe in the regression case due to the unconfined output space
- Even taking into account the above points, inference is more difficult, in the sense that 'off-the-shelf' results versus independent models are more difficult to obtain, and when obtained, such models still fail to provide any useful representation or interpretation

To our knowledge, this is the first work which looks in-depth at regressor chains in the probabilistic context. We surveyed a number of applicable methods, and developed our own based on Monte Carlo methodology, particularly crafted to tackle these identified issues: MCRC and PFRC. This had the following aspects:

- In line with the chains methodology, we considered base models as a flexible hyperparameter, which in practice may be selected according to the problem domain, rather than an aspect hardwired to the overall method. This should not be seen as a 'nuisance' parameter but as an attractive feature for adaptation across different areas.

- The inference process results in a set of particles which can be used as a mode-seeking mechanism or an estimator for minimum mean squared error, as desired.

- Error propagation is controlled by a resampling scheme, which prevents degenerate particles from propagating.

- Interpretation can be provided via a point cloud of sample paths, which offer a description of the underlying conditional posterior distribution.

Probabilistic regressor chains have peculiarities that distinguish them from other models, such as the full cascade involving all outputs and hyperparametrization of the base models. However, to properly place regressor chains in context with the wider literature, we also identified and discussed connections to a range of related work including neural networks and Gaussian processes.

The analysis not only facilitates understanding the performance of regressor chains, but also throws more light onto the performance considerations of classifier chains, modifications of which are still under active development and recent publication. For example, it becomes clearer that much of the power of classifier chains comes as being used as labels being used as feature representations; in addition to the fact that the chain is modeling statistical dependence among outputs.

The Monte Carlo methods we develop suggest to be promising especially for tasks where path explainability (i.e., different hypotheses regarding the path taken through output space) is of more importance than outputting the result of a minimum mean squared error estimator. Such tasks include medical research, anomaly detection, de-noising, and missing-value imputation; providing a multitude of application lines along which to develop this work further and built additional connections with related areas of the literature. In future work we also intend to explore areas (such as dynamic chains and recurrent models) that are being developed in parallel in the classification context to approach themes relating to chain order and structure.

References

[1] W. Waegeman, K. Dembczyński, E. Hüllermeier, Multi-target prediction: a unifying view on problems and methods, Data Mining and Knowledge Discovery 33 (2019) 293–324.

[2] J. Read, B. Pfahringer, G. Holmes, E. Frank, Classifier chains for multi-label classification, Machine Learning 85 (2011) 333–359.

[3] K. Dembczyński, W. Cheng, E. Hüllermeier, Bayes optimal multilabel classification via probabilistic classifier chains, in: ICML '10: 27th International Conference on Machine Learning, Omnipress, Haifa, Israel, 2010, pp. 279–286.

[4] J. Read, L. Martino, D. Luengo, Efficient Monte Carlo methods for multi-dimensional learning with classifier chains, Pattern Recognition 47 (2014) 1535–1546.

[5] K. Dembczyński, W. Waegeman, W. Cheng, E. Hüllermeier, On label dependence and loss minimization in multi-label classification, Mach. Learn. 88 (2012) 5–45.

[6] J. Read, L. Martino, P. M. Olmos, D. Luengo, Scalable multi-output label prediction: From classifier chains to classifier trellises, Pattern Recognition 48 (2015) 2096 – 2109.

[7] J. Read, L. Martino, J. Hollmén, Multi-label methods for prediction with sequential data, Pattern Recognition 63 (2017) 45 – 55.

[8] H. Borchani, G. Varando, C. Bielza, P. Larrañaga, A survey on multi-output regression, Wiley Int. Rev. Data Min. and Knowl. Disc. 5 (2015) 216–233.

[9] E. Spyromitros-Xioufis, G. Tsoumakas, W. Groves, I. Vlahavas, Multi-target regression via input space expansion: treating targets as inputs, Machine Learning (2016) 1–44.

[10] J. M. Moyano, E. L. Gibaja, S. Ventura, An evolutionary algorithm for optimizing the target ordering in ensemble of regressor chains, in: 2017 IEEE Congress on Evolutionary Computation (CEC), pp. 2015–2021.

[11] G. Tsoumakas, E. Spyromitros-Xioufis, A. Vrekou, I. Vlahavas, Multi-target regression via random linear target combinations, in: ECML PKDD 2014.

[12] G. Melki, A. Cano, V. Kecman, S. Ventura, Multi-target support vector regression via correlation regressor chains, Information Sciences 415-416 (2017) 53 – 69.

[13] K. Dembczyński, W. Waegeman, E. Hüllermeier, An analysis of chaining in multi-label classification, in: ECAI: European Conference of Artificial Intelligence, volume 242, IOS Press, 2012, pp. 294–299.

[14] G. Tsoumakas, I. Katakis, I. Vlahavas, Random k-labelsets for multilabel classification, IEEE Transactions on Knowledge and Data Engineering 23 (2011) 1079–1089.

[15] X. Jun, Y. Lu, Z. Lei, D. Guolun, Conditional entropy based classifier chains for multi-label classification, Neurocomputing 335 (2019) 185 – 194.

[16] P. Teisseyre, CCnet: Joint multi-label classification and feature selection using classifier chains and elastic net regularization, Neurocomputing 235 (2017) 98 – 111.

[17] D. Mena, E. Montañés, J. R. Quevedo, J. J. Coz, An overview of inference methods in probabilistic classifier chains for multilabel classification, Wiley Int. Rev. Data Min. and Knowl. Disc. 6 (2016) 215–230.

[18] G. Tsoumakas, E. Spyromitros-Xioufis, I. Vlahavas, Drawing parallels between multi-label classification and multi-target regression, in: ECML PKDD 2014 Workshop on Multi-Target Prediction.

[19] J. Read, J. Hollmén, Multi-label Classification using Labels as Hidden Nodes, Technical Report 1503.09022v3, ArXiv.org, 2017. ArXiv.

[20] R. Senge, J. J. del Coz, E. Hüllermeier, On the problem of error propagation in classifier chains for multi-label classification, in: M. Spiliopoulou, L. Schmidt-Thieme, R. Janning (Eds.), Data Analysis, Machine Learning and Knowledge Discovery, Springer International Publishing, Cham, 2014, pp. 163–170.

[21] T. Yee, V. Lisy, M. Bowling, Monte carlo tree search in continuous action spaces with execution uncertainty, in: Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI'16, AAAI Press, 2016, pp. 690–696.

[22] D. Barber, Bayesian Reasoning and Machine Learning, Cambridge University Press, 2012.

[23] P. M. Djuric, J. H. Kotecha, J. Zhang, Y. Huang, T. Ghirmai, M. F. Bugallo, J. Miguez, Particle filtering, IEEE Signal Processing Magazine 20 (2003) 19–38.

[24] L. Martino, V. Elvira, F. Louzada, Effective sample size for importance sampling based on discrepancy measures, Signal Processing 131 (2017) 386 – 401.

[25] M. F. Bugallo, L. Martino, J. Corander, Adaptive importance sampling in signal processing, Digital Signal Processing 47 (2015) 36 – 49.

[26] J. Montiel, J. Read, A. Bifet, T. Abdessalem, Scikit-MultiFlow: A multi-output streaming framework, Journal of Machine Learning Research 19 (2018) 1–5.

[27] J. Read, L. Martino, J. Hollmén, Multi-label methods for prediction with sequential data, Pattern Recognition 63 (2017) 45–55.

[28] C. Molnar, Interpretable Machine Learning, 2019. `https://christophm.github.io/interpretable-ml-book/`.

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

[30] K. He, X. Zhang, S. Ren, J. Sun, Deep residual learning for image recognition, 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016) 770–778.

[31] M. Cisse, M. Al-Shedivat, S. Bengio, Adios: Architectures deep in output space, in: Proceedings of The 33rd International Conference on Machine Learning, volume 48, PMLR, New York, New York, USA, 2016, pp. 2770–2779.

[32] P. Dellaportas, D. A. Stephens, Bayesian analysis of errors-in-variables regression models, Biometrics 51 (2009) 1085–1095.

[33] J. E. Johnson, V. Laparra, G. Camps-Valls, A derivative-based variance estimate for Gaussian Process regression, Submitted (2018) 1–20.

[34] J. Quiñonero-Candela, A. Girard, C. Rasmussen, Prediction at an uncertain input for Gaussian Processes and Relevance Vector Machines application to multiple-step ahead time-series forecasting, Technical Report, no. 1 (2003) 1–14.

[35] P. Dallaire, C. Besse, B. Chaib-draa, An approximate inference with Gaussian Process to latent functions from uncertain data, Neurocomputing 74 (2011) 1945 – 1955.

[36] E. Snelson, Z. Ghahramani, C. Rasmussen, Warped Gaussian Processes, in: Advances in Neural Information Processing Systems 16, 2003, pp. 1–8.

[37] M. Lázaro-Gredilla, Bayesian Warped Gaussian Processes, in: Advances in Neural Information Processing Systems 25, 2012, pp. 1619–1627.

[38] P. Dallaire, C. Besse, B. Chaib-draa, Deep Gaussian Processes, Proceedings of the Sixteenth International Workshop on Artificial Intelligence and Statistics (AISTATS) (2013) 207–215.

[39] M. P. Deisenroth, M. F. Huber, U. D. Hanebeck, Analytic moment-based Gaussian process filtering, in: Proceedings of the 26th Annual International Conference on Machine Learning (ICML), pp. 225–232.

[40] H. Bijl, T. B. Schon, J. W. van Wingerden, M. Verhaegen, System identification through online sparse Gaussian Process regression with input noise, in: arXiv:1601.08068, pp. 1–25.

[41] D. Mena, E. Montañés, J. R. Quevedo, J. J. del Coz, Using a* for inference in probabilistic classifier chains, in: Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence, IJCAI 2015, Buenos Aires, Argentina, July 25-31, 2015, pp. 3707–3713.

[42] J. Read, C. Bielza, P. Larrañaga, Multi-dimensional classification with super-classes, Transactions on Knowledge and Data Engineering 26 (2014) 1720–1733.

[43] J. Nam, E. Loza Mencía, H. J. Kim, J. Fürnkranz, Maximizing subset accuracy with recurrent neural networks in multi-label classification, in: Advances in Neural Information Processing Systems 30, pp. 5413–5423.

[44] Y. Guo, S. Gu, Multi-label classification using conditional dependency networks, in: IJCAI '11: 24th International Conference on Artificial Intelligence, IJCAI/AAAI, 2011, pp. 1300–1305.