Bayesian Structured Prediction using Gaussian Processes

Sébastien Bratières, Novi Quadrianto and Zoubin Ghahramani
Department of Engineering, University of Cambridge, UK

Abstract

We introduce a conceptually novel structured prediction model, *GPstruct*, which is kernelized, non-parametric and Bayesian, by design. We motivate the model with respect to existing approaches, among others, conditional random fields (CRFs), maximum margin Markov networks (M^3N), and structured support vector machines (SVMstruct), which embody only a subset of its properties. We present an inference procedure based on Markov Chain Monte Carlo. The framework can be instantiated for a wide range of structured objects such as linear chains, trees, grids, and other general graphs. As a proof of concept, the model is benchmarked on several natural language processing tasks and a video gesture segmentation task involving a linear chain structure. We show prediction accuracies for GPstruct which are comparable to or exceeding those of CRFs and SVMstruct.

1 Introduction

Much interesting data does not reduce to points, scalars or single categories. Images, DNA sequences and text, for instance, are not just structured objects comprising simpler indendent atoms (pixels, DNA bases and words). The interdependencies among the atoms are rich and define many of the attributes relevant to practical use. Suppose that we want to label each pixel in an image as to whether it belongs to background or foreground (image segmentation), or we want to decide whether DNA bases are coding or not. The output interdependencies suggest that we will perform better in these tasks by considering the structured nature of the output, rather than solving a collection of independent classification problems.

Existing statistical machine learning models for structured prediction, such as maximum margin Markov Network (M^3N) [1], structured support vector machines (SVMstruct) [2] and conditional random field (CRF) [3], have established themselves as the state-of-the-art solutions for structured problems (cf. figure 1 and table 1 for a schematic representation of model relationships).

In this paper, we focus our attention on CRF-like models due to their probabilistic nature, which allows us to incorporate prior knowledge in a seamless

MRF	Markov random field	
GP	Gaussian process	[4]
GPMC	GP multi-class classification	[5]
CRF	conditional random field	[3]
KCRF	kernelized CRF	[6]
BCRF	Bayesian CRF	[7]
M^3N	maximum-margin Markov network	[1]
GPC MAP	GPC maximum a posteriori	
GP seq MAP	GP for sequence labelling	[8]
LR	logistic regression (classification)	
SVM	support vector machine	
SVMMC	multiclass SVM	[9], [10]
SVMstruct	structured SVM	[2]
GPstruct	structured GP classification	this paper

Table 1: Models, acronyms and references. A unified view of structured prediction models related to CRF and SVM is given in [11]. [12] presents techniques and applications resulting from a decade of work on CRFs.

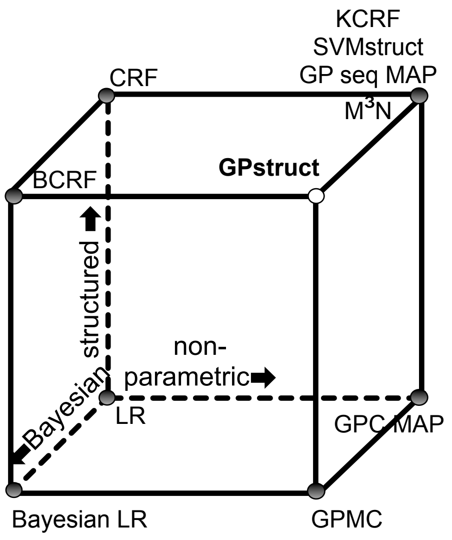

Figure 1: Schematic representation of structured prediction models. The model we are describing here, GPstruct, exhibits all three properties separately present in other, existing, models.

manner. Further, probabilistic models make it possible to compute posterior label probabilities that encode our uncertainty in the predictions.

On the other hand, SVMstruct and M^3N offer the ability to use kernel functions for learning using implicit and possibly infinite dimensional features, thus overcoming the drawbacks of finite dimensional parametric models such as the CRF. In addition, kernel combination allows the integration of multiple sources of information in a principled manner. These reasons motivate introducing Mercer kernels in CRFs [6], an advantage that we wish to maintain.

From training and inference point of views, most CRF models estimate their parameters point-wise using some form of optimisation. In contrast, [7] provide a Bayesian treatment of the CRF which approximates the posterior distribution of the model parameters, in order to subsequently average over this distribution at prediction time. This method avoids important issues such as overfitting, or the necessity of cross-validation for model selection.

Reflecting on this rich history of CRF models, we ask a very natural question: can we have a CRF model which is able to use implicit features spaces and at the same time estimates a posterior distribution over model parameters? The main drive for pursuing this direction is to combine the best of both worlds from Kernelized CRFs and Bayesian CRFs. To achieve this, we investigate the use of Gaussian processes (GP) [4] for modelling structured data where the structure is imposed by a Markov Random Field as in the CRF.

Our contributions are the following:

- a conceptually novel model which combines GPs and CRFs, and its coherent and general formalisation;
- while the structure in the model is imposed by a Markov Random Field, which is very general, as a proof of concept we investigate a linear chain instantiation;
- a Bayesian learning algorithm which is able to address model selection without the need of cross-validation, a drawback of many popular structured prediction models;

The present paper is structured as follows. Section 2 describes the model itself, its parameterization and its application to sequential data, following up with our proposed learning algorithm in section 3. In section 4, we situate our model with respect to existing structured prediction models. An experimental evaluation against other models suited to the same task is discussed in section 5.

2 The model

The learning problem addressed in the present paper is structured prediction. Assume data consists of observation-label (or input-output) tuples, which we will note $\mathcal{D} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(N)}, \mathbf{y}^{(N)})\}$, where N is the size of the dataset, and $(\mathbf{x}^{(n)}, \mathbf{y}^{(n)}) \in \mathcal{X} \times \mathcal{Y}$ is a data point. In the *structured* context, $\mathbf{y}$ is an object such as a sequence, a grid, or a tree, which exhibits structure in the sense that it consists of interdependent categorical atoms. Sometimes the output $\mathbf{y}$ is referred to as the *macro-label*, while its constituents are termed *micro-labels*. The observation (input) $\mathbf{x}$ may have some structure of its own. Often, the structure of $\mathbf{y}$ then reflects the structure of $\mathbf{x}$, so that parts of the label map to parts of the observation, but this is not required. The goal of the learning problem is to predict labels for new observations.

The model we introduce here, which we call *GPstruct*, in analogy to the structured support vector machine (SVMstruct) [2], can be succinctly described as follows. Latent variables (LV) mediate the influence of the input on the output. The distribution of the output labels given the input and the LV is defined per *clique*: in undirected graphical models, a clique is a set of nodes such that every two nodes are connected. Let this distribution be:

$$p(\mathbf{y}|\mathbf{x}, \mathbf{f}) = \frac{\exp(\sum_c f(c, \mathbf{x}_c, \mathbf{y}_c))}{\sum_{\mathbf{y}' \in \mathcal{Y}} \exp(\sum_c f(c, \mathbf{x}_c, \mathbf{y}'_c))} \tag{1}$$

where $\mathbf{y}_c$ and $\mathbf{x}_c$ are tuples of nodes belonging to clique c, while $f(c, \mathbf{x}_c, \mathbf{y}_c)$ is a LV associated with this particular clique and values for nodes $\mathbf{x}_c$ and $\mathbf{y}_c$. Let $\mathbf{f}$ be the collection of all LV of the form $f(c, \mathbf{x}_c, \mathbf{y}_c)$. We call the distribution (1) *structured softmax*, in analogy to the softmax (a.k.a. multinomial logistic) likelihood used in multinomial logistic regression. The conditional distribution in (1) is essentially a CRF over the input-output pairs, where the potential for each clique c is given by a Gibbs distribution, whose energy function is $E(\mathbf{x}, \mathbf{y}) = \sum_c -f(c, \mathbf{x}_c, \mathbf{y}_c)$.

In the CRF, potentials are log-linear in the parameters, with basis function $\mathbf{w}^T \phi_c(\mathbf{x}_c, \mathbf{y}_c)$, where $\mathbf{w}$ is the weight parameter and ϕ_c a feature extraction function. Here instead, rather than choosing parametric clique potentials as in the CRF, the GPstruct model assumes that $f(c, \mathbf{x}_c, \mathbf{y}_c)$ is a non-parametric function of its arguments, and gives this function a GP prior. Note that we substitute not only $\mathbf{w}$, but the entire basis function by a LV. In particular, $f(c, \mathbf{x}_c, \mathbf{y}_c)$ is drawn from a GP with covariance function (i.e. Mercer kernel) $k((c, \mathbf{x}_c, \mathbf{y}_c), (c', \mathbf{x}_{c'}, \mathbf{y}_{c'}))$, so that:

$$\mathbf{f} \sim \mathcal{GP}(0, k(\cdot, \cdot)) \tag{2}$$

In summary, the GPstruct is a probabilistic model in which the likelihood is given by a structured softmax, with a Markov random field (MRF) modelling output interdependencies; the MRF's LV, one per factor, are given a GP prior. This MRF could take on many shapes: linear for text, grid-shaped to label pixels in computer vision tasks [13], or even, to take a less trivial example, hierarchical, in order to model probabilistic context-free grammars in a natural language parsing task using CRFs [14].

2.1 Linear chain parameterization

While this model is very general, we will now instantiate it for the case of sequential data, on which our experiments are based. Both the input and output consist of a linear chain of equal length T, and where the micro-labels all stem from a common set, i.e. $\mathcal{Y} = \underset{t=1,\ldots,T}{\times} \mathcal{Y}_t$ with $\forall t, \mathcal{Y}_t = \mathcal{L}$ (and the same for $\mathcal{X}$ and $\mathcal{X}_t$).

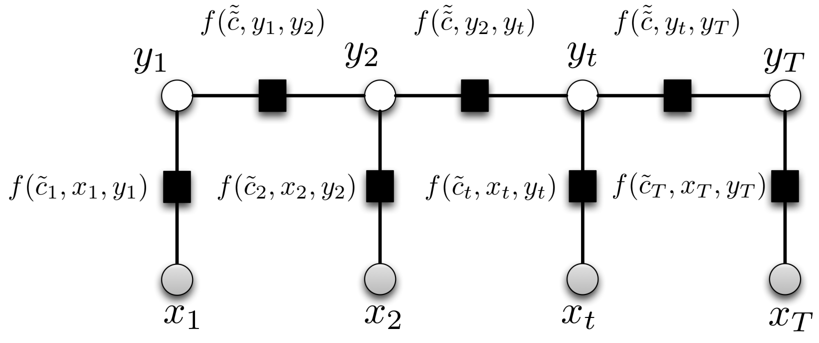
$f(\tilde{\tilde{c}}, y_1, y_2)$
$f(\tilde{\tilde{c}}, y_2, y_t)$
$f(\tilde{\tilde{c}}, y_t, y_T)$
y_1
y_2
y_t
y_T
$f(\tilde{c}_1, x_1, y_1)$
$f(\tilde{c}_2, x_2, y_2)$
$f(\tilde{c}_t, x_t, y_t)$
$f(\tilde{c}_T, x_T, y_T)$
x_1
x_2
x_t
x_T

Figure 2: Linear-chain factor graph for sequence prediction.

We will therefore write $\mathbf{y} = (y_1, \ldots, y_t, \ldots, y_T)$ and $\mathbf{x} = (\mathbf{x}_1, \ldots, \mathbf{x}_t, \ldots, \mathbf{x}_T)$. In this context, macro-labels can be called label sequences, and micro-labels just "labels", without risk of confusion.

In our experiments below, we tackle text data: the input consists of sentences (chains of words), and outputs of corresponding chains of task-specific micro-labels. A common natural language processing task is word/ sentence/ topic segmentation: here the (2-class) micro-labels are B, to label the beginning of a segment, and I, to label the inside of a segment.

We will now expose the design decisions involved in instantiating GPstruct to a linear-chain MRF. A priori, there is one LV per $(c, \mathbf{x}_c, \mathbf{y}_c)$, i.e. per clique and node values.

Parameter tying amounts to grouping (tying) some of these LV, thus decreasing the number of LV which need to be learnt. In particular, we will be interested in grouping LV according to their clique type (or *clique template* [6]) or to node values. In the linear chain model, there are two clique types: pairwise cliques (y_t, y_{t+1}), and unary cliques $(\mathbf{x}_t, y_t)$.

In our treatment of linear chain GPstruct, we tie LV as follows: we distinguish each individual unary clique, but tie all pairwise cliques, so that we can denote them by $\tilde{c}_t$ resp. $\tilde{\tilde{c}}$; further, we do not tie based on node values. This is illustrated in figure 2. By distinguishing each unary clique, we obtain a non-parametric model, where the number of unary LV grows with the data. Alternatively, we could decide to group all $\tilde{c}_t$ to $\tilde{c}$, or maybe to group all $\tilde{c}_t$ except for the edge positions $t = 1$ or $t = T$, where the task may dictate a special behaviour. The same goes for the pairwise LV, where alternative choices may be relevant to the domain. Parameter tying for the linear chain is essentially the same as e.g. for a grid.

Let $T^{(n)}$ denote the length of $\mathbf{y}^{(n)}$ (which need not be constant across label sequences). In our chosen parameterization, there is one unary LV for each position t, and each value y_t, so there are $\sum_n T^{(n)} \times |\mathcal{L}|$ unary LV. This number usually dominates the number of pairwise LV.

Note that we had to define LV for all possible labels y_t (more generally, $\forall \mathbf{y} \in \mathcal{Y}$), not just the ones observed. This is because in (1), the normalisation runs over $\mathbf{y}' \in \mathcal{Y}$, and also because we want to evaluate $p(\mathbf{y}|\mathbf{x}, \mathbf{f})$ for any potential $\mathbf{y}$. By contrast, the input $\mathbf{x}$ and therefore x_t is always assumed observed in our supervised setting, and so we do not need to define LV for ranges of x_t values.

Now turning to pairwise LV: there is one pairwise LV per (y_t, y_{t+1}) tuple; and so there are $|\mathcal{Y}_t| \times |\mathcal{Y}_{t+1}| = |\mathcal{L}|^2$ pairwise LV.

2.2 Kernel function specification

The Gaussian process prior defines a multivariate Gaussian density over any subset of the LV, with usually zero mean and a covariance function given by the positive definite kernel (Mercer kernel) k [15]. Our choice of kernel decomposes into a unary and pairwise kernel function:

$$\begin{aligned} &k((c, \mathbf{x}_c, \mathbf{y}_c), (c', \mathbf{x}_{c'}, \mathbf{y}_{c'})) = \\ &\qquad \mathbb{I}[c, c' \in \{\tilde{c}_t | t\}]\ k_u((t, \mathbf{x}_t, y_t), (t', \mathbf{x}_{t'}, y_{t'})) \\ &\qquad + \mathbb{I}[c = c' = \tilde{\tilde{c}}]\ k_p((y_s, y_{s+1}), (y_{s'}, y_{s'+1})) \end{aligned} \tag{3}$$

In the above, we make use of Iverson's bracket notation: $\mathbb{I}[P] = 1$ when condition P is true and 0 otherwise. The positions of c, c' are noted t, t', and $\mathbf{x}_t, \mathbf{x}_{t'}, y_t, y_{t'}$ are the corresponding input resp. output node values.

We give the unary kernel the form

$$k_u((t, \mathbf{x}_t, y_t), (t', \mathbf{x}_{t'}, y_{t'})) = \mathbb{I}[y_t = y_{t'}] k_x(\mathbf{x}_t, \mathbf{x}_{t'}). \tag{4}$$

k_x is an "input-only" kernel, for instance the linear kernel $\langle \mathbf{x}_t, \mathbf{x}_{t'} \rangle$, or the squared exponential kernel, defined as the inverse of the exponentiated Euclidean distance: $\exp(-\frac{1}{\gamma} ||\mathbf{x}_t - \mathbf{x}_{t'}||^2)$, where γ is a kernel hyperparameter.

Further, our pairwise kernel takes on the form

$$k_p((y_t, y_{t+1}), (y_{t'}, y_{t'+1})) = \mathbb{I}[y_t = y_{t'} \wedge y_{t+1} = y_{t'+1}] \tag{5}$$

With the proper ordering of LV, the Gram matrix has a block-diagonal structure:

$$K = cov[\mathbf{f}] = \begin{pmatrix} K_{\text{unary}} & 0 \\ 0 & K_{\text{pairwise}} \end{pmatrix} \tag{6}$$

It is a square matrix, of length equal to the total number of LV $\sum_n T^{(n)} \times |\mathcal{L}| + |\mathcal{L}|^2$. K_{unary} is block diagonal, with $|\mathcal{L}|$ equal square blocks, each the Gram matrix of k_x of size $\sum_n T^{(n)}$, and $K_{\text{pairwise}} = \mathbf{I}_{|\mathcal{L}|^2}$.

Summing up sections 2 to 2.2, designing an instance of a GPstruct model requires three types of decisions: the choice of the MRF, mainly dictated by the task and the output data structure; parameter tying; kernel design. The next section now details attractive properties of this model.

2.3 Model properties

The GPstruct model has the following appealing statistical properties:

Structured: the output structure is controlled by the design of the MRF, which is very general. The only practical limitation is the availability of efficient inference procedures on the MRF.

Non-parametric: the number of LV grows with the size of the data. In the linear chain case, it is the number of unary LV which grows with the total length of input sequences.

Bayesian: this is a probabilistic model that supports Bayesian inference, with the usual benefits of Bayesian learning. At prediction time: error bars and reject options. At learning time: model selection and hyperparameter learning with inbuilt Occam's razor effect, without the need for cross-validation.

Kernelized: a joint (input-output) kernel is defined over the LV. Kernels potentially introduce several hyperparameters, making grid search for cross-validation intractable. Kernelized Bayesian models like GPstruct do not suffer from this, as they define a posterior over the hyperparameters.

3 Learning

Our learning algorithms are Markov chain Monte Carlo procedures, and as such are "any-time", in that they have no predefined stopping criterion.

3.1 Prediction

Given an unseen test point $\mathbf{x}_*$, and assuming the LV $\mathbf{f}_*$ corresponding to x_* to be accessible, we wish to predict label $\hat{\mathbf{y}}_*$ with lowest loss. Now, given $\mathbf{f}_*$, the underlying MRF is fully specified. In tree-shaped structures, belief propagation gives an exact answer in linear time $O(T)$; in the linear chain case, under 0/1 loss $\ell(\mathbf{y}, \hat{\mathbf{y}}) = \delta(\mathbf{y} = \hat{\mathbf{y}})$, we predict the jointly most probable output sequence obtained from the Viterbi procedure, and under Hamming (micro-label-wise) error $\ell(\mathbf{y}, \hat{\mathbf{y}}) = \sum_t \delta(\mathbf{y}_t = \hat{\mathbf{y}}_t)$, we predict the micro-label-wise most probable output sequence. For other cases than trees, where exact inference is impossible, approximate inference methods such as loopy belief propagation [16] are available.

Given $\mathbf{f}$, due to the GP marginalisation property, the test point LV $\mathbf{f}_*$ are distributed according to a multivariate Gaussian distribution (cf. e.g. [17, section 2] for a derivation): $\mathbf{f}_*|\mathbf{f} \sim N(K_*^T K^{-1}\mathbf{f}, K_{**} - K_*^T K^{-1} K_*)$, where matrices K, K_*, K_{**} have their element (i,j) equal to $k(\mathbf{f}^i, \mathbf{f}^{(j)})$ resp. $k(\mathbf{f}^i, \mathbf{f}_*^{(j)})$ resp. $k(\mathbf{f}_*^{(i)}, \mathbf{f}_*^{(j)})$, with k the kernel described section 2.2.

Uncertainty over $\mathbf{f}_*|\mathbf{f}$ is accounted for correctly by *Bayesian model averaging*: $\hat{\mathbf{y}}_* = \arg\max_{\mathbf{y}_* \in \mathcal{Y}_*} p(\mathbf{y}_*|\mathbf{f})$, with

$$p(\mathbf{y}_*|\mathbf{f}) = \frac{1}{N_{\mathbf{f}_*|\mathbf{f}}} \sum_{\mathbf{f}_*|\mathbf{f}} p(\mathbf{y}_*|\mathbf{f}_*) \tag{7}$$

where $N_{\mathbf{f}*|\mathbf{f}}$ is the number of samples from $\mathbf{f}_*|\mathbf{f}$.

3.2 Sampling from the posterior

We wish to represent the posterior distribution $\mathbf{f}|\mathcal{D}$ (as opposed to performing a MAP approximation of the posterior to a single value $\mathbf{f}_{MAP}$). The training data likelihood is $p(\mathcal{D}|\mathbf{f}) = \prod_n p(\mathbf{y}^{(n)}|\mathbf{f}, \mathbf{x}^{(n)})$, with the single point likelihood given by (1). Training aims at maximizing the likelihood, for which we propose to use elliptical slice sampling (ESS) [18], an efficient MCMC procedure for ML training of tightly coupled LV with a Gaussian prior. In all our experiments below, we discard the first third of the samples before carrying out prediction, to allow for burn-in of the MCMC chain.

The computation of the likelihood itself is a non-trivial problem due to the presence of the normalising constant, which ranges over $\mathbf{y}' \in \mathcal{Y}$, of size exponential in the number of micro-labels $|\mathcal{L}|$. In the case of tree-shaped MRFs, however, belief propagation yields an exact and usually efficient procedure; in the linear case, it is referred to as forwards-backwards procedure, and runs in $O(T|\mathcal{L}|^2)$.

ESS requires computing the full kernel matrix, of size $O(N_{LV}^2)$, where N_{LV} is the total number of LV, and its Cholesky, obtained in $O(N_{LV}^2)$ time steps. The large size of the matrices is a limiting factor to our implementation.

ESS yields samples of the posterior. To perform prediction, it is necessary to introduce one more level of Bayesian model averaging: continuing from (7), $p(\mathbf{y}_*|\mathcal{D}) = \frac{1}{N_\mathbf{f}} \sum_\mathbf{f} p(\mathbf{y}_*|\mathbf{f})$ where $N_\mathbf{f}$ is the number of samples of $\mathbf{f}|\mathcal{D}$ available.

4 Relation to other models

Our proposed method builds upon a large body of existing models, none of which, however, exhibit all properties mentioned in section 2.3.

4.1 GP classification

Gaussian process models (or any regression models such as a linear regression) can be applied to classification problems. In a probabilistic approach to classification, the goal is to model posterior probabilities of an input point $\mathbf{x}$ belonging to one of $|\mathcal{L}|$ classes, $y \in \{1, \ldots, |\mathcal{L}|\}$. For binary classification (that is $|\mathcal{L}| = 2$), we can turn the output of a Gaussian process (in $\mathbb{R}$) into a class probability (in the interval $[0, 1]$) by using an appropriate non-linear activation function. The most commonly used such function is the logistic function $p(y = 1|f, \mathbf{x}) = \frac{\exp(f(\mathbf{x}))}{\exp(f(\mathbf{x}))+\exp(-f(\mathbf{x}))}$. The resulting classification model is called *GP binary classification* [5]. Let us now move from binary classification to multi-class classification. This is achieved by maintaining K regression models, each model being indexed by a latent function f_k. We use the vector notation $\mathbf{f} = (f_1 \ldots f_K)$ to index the collection of latent functions. The desired multi-class model is obtained by using a generalisation of the logistic to multiple variables, the softmax function: $p(y=k|\mathbf{f},\mathbf{x}) = \frac{\exp(f_k(\mathbf{x}))}{\sum_{k=1}^{K}\exp(f_k(\mathbf{x}))}$. The corresponding model is called *Gaussian process multi-class classification* (GPMC) [5]. Note that the above multiclass distribution is normalised over the set of possible output labels $\mathcal{L}$ (here $|\mathcal{Y}| = K$). Simply extending the multi-class model for a structured prediction case is computationally infeasible due to the sheer size of the label set $\mathcal{L}$. We provide a novel extension of Gaussian process for structured problems. Structured prediction itself has a long history of successful methods, which we discuss in subsequent sections.

4.2 From logistic to structured logistic

A natural way to cater for interdependencies between micro-labels at prediction time, is to define an MRF over $\mathbf{y}$, and to condition the resulting distribution on the input $\mathbf{x}$ (i.e. in a graphical model representation, inserting directed edges from input to output): we thus obtain a mixed graphical model, the *conditional random field* (CRF) [3], a very popular and successful model for structured prediction. The CRF defines a log-linear model for $p(\mathbf{y}|\mathbf{x})$: $p(\mathbf{y}|\mathbf{x},\mathbf{w}) = \frac{1}{Z(\mathbf{x},\mathbf{w})}\exp\left(\sum_c \langle \mathbf{w}_c, \phi(\mathbf{x}_c,\mathbf{y}_c)\rangle\right)$, for a weight vector $\mathbf{w}$, and a joint input-output feature representation $\phi(\mathbf{x},\mathbf{y})$. In the above, $Z(\mathbf{x},\mathbf{w}) = \sum_{\mathbf{y}'\in\mathcal{L}}\prod_c \exp(\langle \mathbf{w}_c, \phi(\mathbf{x}_c,\mathbf{y}_c)\rangle)$ is the normalising constant. As before, we use c to denote a clique. Instead of parameterizing the energy function, $E(\mathbf{x},\mathbf{y}) := -\langle \mathbf{w}_c, \phi(\mathbf{x}_c,\mathbf{y}_c)\rangle$, by means of a weight vector, GPstruct place a Gaussian process prior over energy functions, effectively side-stepping parameterization. Recent advances in CRF modelling by [19] also side-step linear parameterization of the energy function. Instead, a random forest is used to model the energy function, allowing higher order interactions.

4.3 Kernelizing structured logistic

[6] presents a kernelized variant of the CRF, the *kernel conditional random field* (KCRF), where a kernel is defined over clique templates. The kernelized version of the CRF is generally difficult to construct, to train, and have several hyperparameters that need to be set via cross-validation, therefore, have not been adopted as enthusiastically as regular CRF. *Structured support vector machines* [2], SVMstruct, and *maximum margin Markov networks* [1], $\mathrm{M}^3\mathrm{N}$ also model $p(\mathbf{y}|\mathbf{x})$ as a log-linear function. However, to learn $\mathbf{w}$, while traditional CRF learning maximizes the conditional log likelihood of the training data, both SVMstruct and $\mathrm{M}^3\mathrm{N}$ perform maximum margin training: learn $\mathbf{w}$ which predicts the correct outputs with a large margin compared to incorrect outputs (all other outputs except the correct ones). SVMstruct can be easily kernelized by means of the representer theorem. Our proposed GPstruct is also kernelized, with a practical advantage that kernel hyperparameters can be inferred from the data instead of requiring a cross-validation procedure.

4.4 Bayesian versus MAP inference

By Bayesian inference (as opposed to maximum likelihood ML or maximum a posteriori MAP inference), we mean the preservation of the uncertainty over LV, that is their representation, not as point-wise estimates, but as random variables and their probability distribution.

CRF parameters are usually estimated point-wise, e.g. often with an ML or MAP objective using gradient ascent or approximate likelihood techniques, cf. [12] for a review. An exception to this is *Bayesian conditional random field* (BCRF) proposed by [7]. Instead of point-wise parameter estimation, the BCRF approximates the posterior distribution of the CRF parameters as Gaussian distributions and learns them using expectation propagation [20]. GPstruct also follows a Bayesian inference procedure, and combines it with kernelization.

Despite the similarity in name, the model in [8] is more similar to the KCRF than to GPstruct. Like KCRF, this work tackles sequence labelling, while we purposefully formulate GPstruct to apply to any underlying MRF, even though we demonstrate its instantiation in sequences. More importantly, [8] take a MAP estimation of the LV, making the model, among others, unable to infer associated hyperparameters directly from the data. However, the point-wise MAP estimate comes with a benefit: sparsification, due to the applicability of the representer theorem. The LV appear as a weighted sum of kernel evaluations over the data. Two methods are applicable from here. The first, applied in [8, 4.2] and [6, 3], consists in greedily selecting the LV/ clique associated to the direction of steepest gradient, during the optimisation phase. The second method consists of applying the "Taskar trick" [1], and is concerned with the fact that in the LV expression obtained from the representer theorem, the sum runs over $\mathcal{Y}$, i.e. all possible macro-labels, which is exponentially large in $|\mathcal{L}|$. This trick consists in a rearrangement of terms inside the objective functions which allows a lower-dimensional reparameterization. These sparsification techniques are not accessible to us due to the use of a Bayesian representation; however alternative techniques may come from the GP sparsification literature.

4.5 Structured prediction via output kernels

All previously mentioned structured prediction methods explicitly model the output interdependency via MRF. A different strand of work aims at building an implicit model of output correlations via a kernel similarity measure [21, 22]. The twin Gaussian processes of [22] address structured continuous-output problems by forcing input kernels to be similar to output kernels. This objective reflects the assumption that similar inputs should produce similar outputs. The input and output are separately modelled by GPs with different kernels. Learning consists of minimising KL divergence.

5 Applications

In order to appreciate how the proposed model and learning scheme compare to existing techniques, we conducted benchmark experiments on a range of language processing tasks: segmentation, chunking, and named entity recognition, as well as on a video processing task, gesture segmentation, all involving a linear chain structure.

5.1 Text Processing Task

Our data and tasks comes from the CRF++ toolbox[1]. Four standard natural language processing tasks are available, cf. table 2. Noun phrase identification (`Base NP`) tags words occurring in noun phrases with B for beginning, I for a word inside a noun phrase, and O for other words. `Chunking` (i.e. shallow parsing) labels sentence constituents. The `Segmentation` task identifies words (the segments) in sequences of Chinese ideograms. Japanese named entity recognition (`Japanese NE`) labels several types of named entities (organisation, person, etc.) occurring in text.

The data was used in pre-processed form, with sparse binary features extracted for each word position in each sentence. Results were averaged over five experiments per task. Each experiment's training and test data was extracted from the full data set (sizes given in table 2) so that the training sets were always disjoint – except in the case of segmentation, a small-data set of 36 sentences overall, which was subjected to five random splits.

Baselines We compared GPstruct to CRF and SVMstruct. The CRF implementation[2] used LBFGS optimisation. In nested cross-validation, the L_2 regularisation parameter ranged over integer powers from 10^{-8} to 1. Prediction in the CRF and GPstruct minimised Hamming loss (cf. section 3.1). The SVMstruct[3] used a linear kernel, for comparison with the CRF. The regularisation parameter in nested cross-validation ranged over integer powers from 10^{-3} to 10^{2}.

Computing The CRF package is MEX-compiled Matlab, while the SVMstruct system is coded in C++. Our Matlab implementation of GPstruct used MEX functions from the UGM toolbox[4] for likelihood (implementing the forward-backward algorithm). To illustrate runtimes, a 10 hour job on a single core of a 12-core Hex i7-3930K 3.20 GHz machine can accommodate CRF/ SVMstruct learning and prediction, including nested cross-validation over the parameter grid mentioned above, for one experiment, for one task. In the same computing time, GPstruct can perform 100 000 iterations for one experiment for the chunking or segmentation task (the fastest), including hyperparameter sampling (50 000 resp. 80 000 iterations for `Base NP` resp. `Japanese NE`). Getting a precise runtime comparison of CRF, SVMstruct and GPstruct code

[1]by Taku Kudo `http://crfpp.googlecode.com/svn/trunk/doc/index.html`

[2]by Mark Schmidt `http://www.di.ens.fr/~mschmidt/Software/crfChain.html`

[3]by Thorsten Joachims `http://www.cs.cornell.edu/people/tj/svm_light/svm_hmm.html`

[4]also by Mark Schmidt `http://www.di.ens.fr/~mschmidt/Software/UGM.html`

	`Base NP`	`Chunking`	`Segmentation`	`Japanese NE`
number of categories	3	14	2	17
number of features	6,438	29,764	1,386	102,799
size training/ test set (sentences)	150 / 150	50 / 50	20 / 16	50 / 50
SVMstruct	5.91±0.43	9.79±0.97	16.21±2.21	5.64±0.82
CRF	5.92±0.23	**8.29±0.76**	14.98±1.11	**5.11±0.65**
GPstruct ($h_p = 1$)	**4.81±0.47**	8.76±1.08	14.87±1.79	5.82±0.83
GPstruct (prior whitening)	5.06±0.38	8.57±1.20	**14.77**±**1.78**	5.65±0.80

Table 2: Experimental results on text processing task. Error rate across 5 experiments (mean ± one standard deviation). GPstruct experiments on 250 000 ESS steps (i.e. $\mathbf{f}$ samples), using the $\mathbf{f}_*$ MAP scheme, linear kernel, sampling hyperparamers every 1 000 steps (prior whitening) or fixing $h_p = 1$, thinning at 1:1 000.

is not straightforward since implementation languages differ. Having said that, our GPstruct experiments were roughly a factor of two slower than the baselines including grid search.

Kernel hyperparameter learning The GP prior over $\mathbf{f}$ is parameterized by its mean (zero in our case), and the kernel function, which may possess hyperparameters. To explore the effect of kernel hyperparameter learning, we introduce a multiplicative hyperparameter h_p in front of the pairwise kernel, and give it a scaled Gamma hyperprior : $h_p/10^{-4} \sim \text{Gamma}(1, 2)$.

MCMC sampling of the hyperparameter is performed using the prior whitening technique [18], which is easy to implement. Surrogate data modelling [18] is tailored to GP prior LV models, and is reported to give better results; however, it requires an approximation of the posterior variance for the structured softmax case. While it is possible to derive such approximations, we could not observe any performance gain in our experiments so far.

Experimentally, ESS (which samples $\mathbf{f}|\mathcal{D}$) needs to be run over many more iterations than hyperparameter sampling (sampling from the hyperparameter posterior $\mathbf{h}|\mathcal{D}$). We therefore sample from the hyperparameter once every 1 000 ESS steps. Kernel learning is possible as well with GPstruct, but a few exploratory experiments using polynomial and squared exponential kernels on the binary-valued text datasets did not improve the performance.

Results and interpretation Our experimental results are summarised in table 2. GPstruct is generally comparable to the CRF, and slightly better than SVMstruct. Our choice of hyperprior does not seem to fit the Base NP task, where hyperparameter sampling turns out to be worse than keeping h_p fixed at 1.

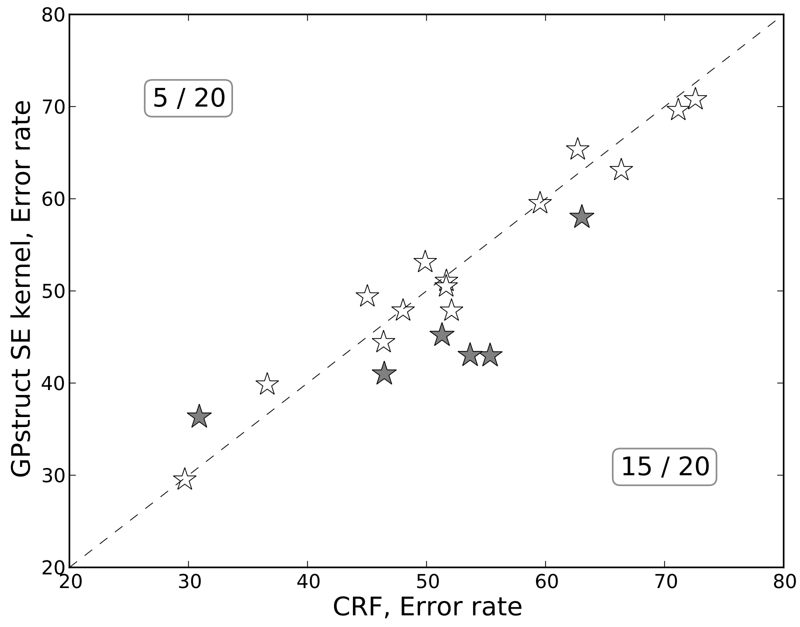

Figure 3: Error rate cross plot of the 20 gesture video sessions. The axes correspond to error rate of GPstruct with SE kernel and CRF, the diagonal line shows equal performance. The shadowed stars are those with at least 5% performance difference.

5.2 Video Processing Task

In a second set of experiments, we apply CRF and GPstruct to the ChaLearn gesture recognition dataset[5]. The data in this case consists of video sequences of an actor performing certain gestures. Each video frame is labelled with a gesture. The videos have an average length of 86 frames, and maximum length 305 frames. The dataset has 20 sessions of 47 videos each. The label space size varies for different sessions, between 9 and 13. For each session, we use 10 videos to train a chain CRF or GPstruct and the rest as test data. At each frame of the video we extract HOG/HOF [23] descriptors and construct a codebook of 30 visual words using a k-means clustering algorithm. Frames are represented by normalized histograms of visual words occurrence, resulting in 30 feature dimensions. A squared exponential kernel, $\exp(-\frac{1}{\gamma}||\mathbf{x}_t - \mathbf{x}_{t'}||^2)$, was used. The kernel hyperparameter γ is given a Gamma$(1, 2)$ prior and is initially set to the median pairwise distance.

Results The experimental results summarize as follows: averaged across

[5]`https://sites.google.com/a/chalearn.org/gesturechallenge/`

all 20 sessions, the error rates were 52.12 ± 11.73 for the CRF, 51.91 ± 11.02 for GPstruct linear kernel, and 50.42 ± 11.24 for GPstruct SE kernel. Since each session effectively represents one specific learning task, we compare the pairwise performances across 20 sessions between GPstruct SE kernel and CRF in figure 3. GPstruct outperforms the CRF baseline by more than 5% in five cases, while it underperforms it in one case. The performance between GPstruct linear kernel and CRF are comparable and we did not include details here due to space constraints.

5.3 Practical insights

We will open-source our GPstruct code on MLOSS[6] to expose the GPstruct model more widely and encourage experimentation.

Kernel matrix positive-definiteness To preserve numerical stability of the Cholesky operation, diagonal jitter of 10^{-4} is added to the kernel matrices. Depending on the hyperprior, some hyperparameter samples may make the kernel matrices badly conditioned: this is best prevented by rejecting such a proposal by simulating a very low likelihood value.

How many $\mathbf{f}$ samples? All subplots in figure 4 plot the error rate of some configuration against the number of $\mathbf{f}$ samples generated (i.e. iterations of the ESS procedure). For all our tasks, the error rate improves until 100 000 iterations, which shows heuristically that sampling histories of at least this length are needed to attain equilibrium for these problems.

How many $\mathbf{f}_*|\mathbf{f}$ samples? $\mathbf{f}_*$ need not be sampled for every $\mathbf{f}$ sample which is generated; to save computing time, we can *thin* and e.g. sample $\mathbf{f}_*|\mathbf{f}$ only every 10th $\mathbf{f}$ sample, disregarding the other nine samples entirely. Our exploratory experiments show the following: high thinning rates, such as 1:4 000, seem to have very limited impact on the error rate, cf. figure 4 (middle). Similarly, how many samples $\mathbf{f}_*|\mathbf{f}$ do we need? Do we need any at all, or could we use only the mean of the predictive posterior? This would save computing the predictive variance, which involves a Cholesky matrix inversion, and is called "$\mathbf{f}_*$ MAP" here. Figure 4 (bottom) answers this: sampling more often does not decrease the error rate. These findings are very valuable in practice, and seem to indicate that the predictive posterior is peaked, while the posterior is rather flat, and requires a long MCMC exploration path to be adequately sampled from. Computing time is dictated by the ESS sampling procedure, so performance improvement efforts should clearly aim at obtaining decorrelated posterior samples.

6 Conclusions and future work

As a model, GPstruct possesses many desirable properties, discussed in detail above. Our experiments with sequential data yielded encouraging results: we

[6]`www.mloss.org`

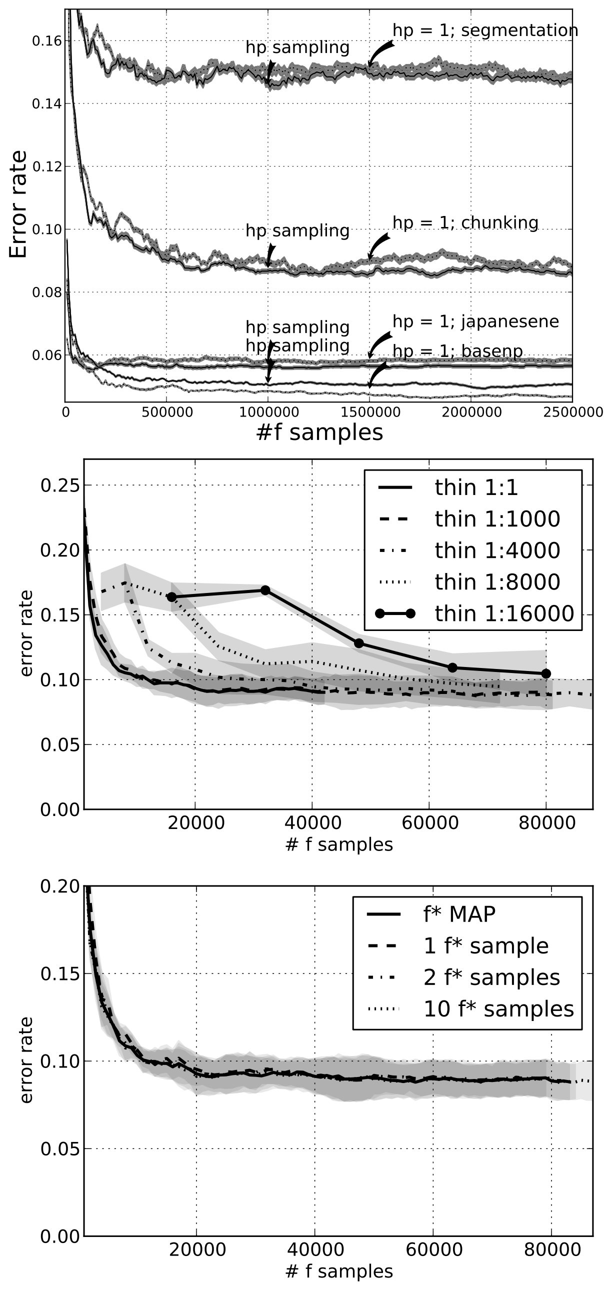

Figure 4: top: Effect of sampling hyperparameters every 1 000 steps versus fixing $h_p = 1$, over the full history of $\mathbf{f}$ samples. $\mathbf{f}_*$ MAP scheme, thinning at 1:1 000. **middle**: Effect of thinning, i.e. sampling $\mathbf{f}_*|\mathbf{f}$ more rarely than every $\mathbf{f}$ sample. `Chunking` task, $\mathbf{f}_*$ MAP scheme, $h_p = 1$. **bottom**: Effect of number of $\mathbf{f}_*|\mathbf{f}$ samples for each $\mathbf{f}$ sample. `Chunking` task, thinning at 1:1 000, $h_p = 1$.

achieve performance comparable to CRF and exceed SVMstruct in text processing tasks, and exceed CRF in a video processing task. While GPstruct is theoretically attractive and empirically promising, we have clearly only touched the surface of the model's possibilities. An important limitation preventing the application to larger data sets is the size of the kernel matrix K, square in the number of LV. One promising direction is an ensemble learning approach in which weak learners can be trained on subsets of the LV constrained by the underlying MRF (thus with quadratically smaller K), and their predictions combined, by Bayesian model combination, into a strong learner.

References

[1] Ben Taskar, Carlos Guestrin, and Daphne Koller. Max-margin Markov networks. In *NIPS*, 2004.

[2] Ioannis Tsochantaridis, Thorsten Joachims, Thomas Hofmann, and Yasemin Altun. Large margin methods for structured and interdependent output variables. *JMLR*, 6:1453–1484, 2005.

[3] John D. Lafferty, Andrew McCallum, and Fernando C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[4] Carl E. Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning.* MIT Press, 2006.

[5] Christopher K. I. Williams and David Barber. Bayesian Classification With Gaussian Processes. *IEEE Trans. Pattern Anal. Mach. Intelligence*, 20(12):1342–1351, 1998.

[6] John Lafferty, Xiaojin Zhu, and Yan Liu. Kernel conditional random fields: representation and clique selection. In *ICML*, 2004.

[7] Yuan Qi, Martin Szummer, and Thomas P. Minka. Bayesian conditional random fields. In *AISTATS*, 2005.

[8] Yasemin Altun, Thomas Hofmann, and Alexander J. Smola. Gaussian process classification for segmenting and annotating sequences. In *ICML*, 2004.

[9] Jason Weston and Chris Watkins. Support vector machines for multi-class pattern recognition. In *Proceedings of the Seventh European Symposium On Artificial Neural Networks*, 1999.

[10] Koby Crammer and Yoram Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *JMLR*, 2:265–292, 2001.

[11] Fernando Perez-Cruz, Massimiliano Pontil, and Zoubin Ghahramani. Conditional graphical models. In *Predicting Structured Data*, pages 265–282. MIT Press, 2007.

[12] Charles A. Sutton and Andrew McCallum. An introduction to conditional random fields. *Foundations and Trends in Machine Learning*, 4(4):267–373, 2012.

[13] Sebastian Nowozin and Christoph H. Lampert. Structured learning and prediction in computer vision. *Foundations and Trends in Computer Graphics and Vision*, 2011.

[14] Ben Taskar, Dan Klein, Michael Collins, Daphne Koller, and Christopher Manning. Max-margin parsing. In *Conference on Empirical Methods on Natural Language Processing (EMNLP)*, 2004.

[15] Bernhard Schölkopf and Alexander J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, 2001.

[16] Kevin P. Murphy, Yair Weiss, and Michael I. Jordan. Loopy belief propagation for approximate inference: An empirical study. In *UAI*, 1999.

[17] Hannes Nickisch. Approximations for Binary Gaussian Process Classification. *JMLR*, 2008.

[18] Iain Murray, Ryan P. Adams, and David J. C. MacKay. Elliptical slice sampling. *JMLR - Proceedings Track*, 9:541–548, 2010.

[19] Jeremy Jancsary, Sebastian Nowozin, Toby Sharp, and Carsten Rother. Regression tree fields - an efficient, non-parametric approach to image labeling problems. In *CVPR*, 2012.

[20] Thomas P. Minka. *A family of algorithms for approximate bayesian inference*. PhD thesis, MIT, 2001.

[21] Jason Weston, Olivier Chapelle, André Elisseeff, Bernhard Schölkopf, and Vladimir Vapnik. Kernel dependency estimation. In *NIPS*, 2002.

[22] Liefeng Bo and Cristian Sminchisescu. Twin gaussian processes for structured prediction. *IJCV*, 2010.

[23] Ivan Laptev, Marcin Marszalek, Cordelia Schmid, and Benjamin Rozenfeld. Learning realistic human actions from movies. In *CVPR*, 2008.

Acknowledgments

The authors would like to thank Simon Lacoste-Julien, Viktoriia Sharmanska, and Chao Chen for discussions.