Non-parametric Bayesian models for structured output prediction

Sébastien Bratières

King’s College

University of Cambridge

This dissertation is submitted for the degree of

Doctor of Philosophy

April 2017

Abstract

Structured output prediction is a machine learning tasks in which an input object is not just assigned a single class, as in classification, but multiple, interdependent labels. This means that the presence or value of a given label affects the other labels, for instance in text labelling problems, where output labels are applied to each word, and their interdependencies must be modelled.

Non-parametric Bayesian (NPB) techniques are probabilistic modelling techniques which have the interesting property of allowing model capacity to grow, in a controllable way, with data complexity, while maintaining the advantages of Bayesian modelling. In this thesis, we develop NPB algorithms to solve structured output problems.

We first study a map-reduce implementation of a stochastic inference method designed for the infinite hidden Markov model, applied to a computational linguistics task, part-of-speech tagging. We show that mainstream map-reduce frameworks do not easily support highly iterative algorithms.

The main contribution of this thesis consists in a conceptually novel discriminative model, GPstruct. It is motivated by labelling tasks, and combines attractive properties of conditional random fields (CRF), structured support vector machines, and Gaussian process (GP) classifiers. In probabilistic terms, GPstruct combines a CRF likelihood with a GP prior on factors; it can also be described as a Bayesian kernelized CRF.

To train this model, we develop a Markov chain Monte Carlo algorithm based on elliptical slice sampling and investigate its properties. We then validate it on real data experiments, and explore two topologies: sequence output with text labelling tasks, and grid output with semantic segmentation of images. The latter case poses scalability issues, which are addressed using likelihood approximations and an ensemble method which allows distributed inference and prediction.

The experimental validation demonstrates: (a) the model is flexible and its constituent parts are modular and easy to engineer; (b) predictive performance and, most crucially, the probabilistic calibration of predictions are better than or equal to that of competitor models, and (c) model hyperparameters can be learnt from data.

Declaration

This dissertation is the result of my own work and includes nothing which is the outcome of work done in collaboration except as declared in the Preface and specified in the text.

It is not substantially the same as any that I have submitted, or, is being concurrently submitted for a degree or diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text. I further state that no substantial part of my dissertation has already been submitted, or, is being concurrently submitted for any such degree, diploma or other qualification at the University of Cambridge or any other University or similar institution except as declared in the Preface and specified in the text

It does not exceed sixty-five thousand words in length, and contains no more than hundred fifty figures.

To Beatrice and Giulio

Acknowledgements

I would like to express my sincere gratitude to my supervisor Zoubin Ghahramani for his continuous support and his patient, steady and insightful guidance over the course of my research. Like many in our research group, I have found inspiration in Zoubin's ability to boil down complex machine learning notions to their simplest accurate explanation. Over the years, Zoubin made it a speciality to represent relationships between machine learning concepts with graph metaphors, and figure 3.2.1 in this thesis bears witness to this influence.

This thesis would not have been possible without the support, intellectual stimulation and friendships offered by my research group. In particular, I wish to thank Zoubin and Carl Rasmussen, my adviser, for creating, shaping and fostering such a brilliant group as ours; one that excels from a scientific point of view but also provides a vibrant and supportive environment. Thanks to Diane Hazell, our group administrator, for her unfailing help, and thanks to Sae and David Franklin, for the good time we had together. Thanks to Simon Lacoste-Julien, who sparked my interest for structured output prediction and statistical learning methods which lie outside the group's traditional scientific remit. I was fortunate to work with enthusiastic and knowledgeable co-authors: Jurgen, Andreas, Novi, Sebastian and Zoubin; Novi's drive and perseverance in particular allowed many of the ideas on GPstruct, the statistical model developed in this thesis, to come to fruition.

I chose to conduct my PhD research part-time, while working in industry, and am indebted to my two employers, Voice Insight and dawin gmbh, for they support, flexibility and trust.

"L'argent est le nerf de la guerre", "Money is the sinew of war", and accordingly, I would like to express my sincere gratitude to: Amazon for their generous sponsorship, through two Amazon Educational Grants towards computing expenses, as well as for the expert advice provided by the Amazon Elastic MapReduce engineering team; Yahoo! for the funding provided through the Key Scientific Challenges Award; King's College, the Department of Engineering and the Ford of Great Britain Trust for subsidising my conference trips; and my supervisor Zoubin Ghahramani for covering research and travel expenses from group resources on several occasions.

My parents, my sister, and my grand-parents, each according to their own character and life path, have been all of inspiration, encouragement and assistance at once.

Finally, and most importantly, I would like to thank my wife Vittoria, for encouraging me to return to Cambridge, for supporting me with patience, determination and loving advice throughout these years, in particular during the trying times, for keeping the home fires burning, and ultimately, for making the journey worthwhile.

Contents

List of Figures

List of acronyms

AEMR	Amazon Elastic Mapreduce
ANLPM	average negative log posterior marginal
ARD	automatic relevance determination
AWS	Amazon Web Services
BCRF	Bayesian conditional random field
BFGS	Broyden–Fletcher–Goldfarb–Shanno algorithm
CPU	central processing unit
CRF	conditional random field
ELBO	evidence lower bound
EM	expectation-maximisation
ESS	elliptical slice sampling
GP	Gaussian process
GPU	graphical processing unit
HDFS	Hadoop distributed file system
HDP	hierarchical Dirichlet process
HE	Hamming error
HMC	hybrid (or Hamiltonian) Monte Carlo
HMM	hidden Markov model
HPY	hierarchical Pitman-Yor
IBP	Indian buffet process
IHMM	infinite hidden Markov model
i.i.d.	independent and identically distributed
JVM	Java virtual machine
KCRF	kernel conditional random field
KL	Kullback-Leibler (divergence)
KS	Kolmogorov-Smirnov (statistical test)
LBFGS	limited-memory BFGS
LBMO	loss-based marginal optimisation
LDA	latent Dirichlet allocation
LV	latent variable

MAP	maximum a posteriori
MC	Monte Carlo
MCMC	Markov chain Monte Carlo
MH	Metropolis-Hastings
ML	maximum likelihood
MPI	message passing interface
MR	map-reduce
MRF	Markov random field
MVG	multivariate Gaussian
NLL	negative log likelihood
NLP	natural language processing
NP	noun phrase
NPB	non-parametric Bayesian
PL	pseudo-likelihood
POS	part-of-speech
PW	prior whitening
PY	Pitman-Yor
QQ	quantile-quantile (plot)
RKHS	reproducing kernel Hilbert space
SBC	stick-breaking construction
SD	surrogate data
SS	slice sampling
SVM	support vector machine
TRW	tree-reweighted belief propagation
UGM	undirected graphical model
VB	variational Bayes
VI	variational inference
WSJ	Wall Street Journal

Chapter 1

Introduction

1.1 Motivation

Machine learning is a young scientific discipline with high impact in the industry and sciences. Situated at the junction between computer science and statistics, it benefits from contributions from control theory, cognitive sciences, numerical computing, applied mathematics, and is fed with problems from artificial intelligence, engineering, data science, as well as “big data”.

A large swathe of interesting problems concerns objects with *structure*, such as images, signal, or text, rather than atomic objects (in the etymological sense of “indivisible”) such as continuous or discrete values, i.e. scalars or categories. In structured output prediction, we are interested in cases where the output, more specifically, has structure: the output consists of individual subcomponents, each of which is constrained by its neighbours, or even by the entire output.

We are motivated in particular by applications in natural language processing and computer vision. In natural language text, words are strongly interdependent due to grammatical agreement and linguistic fluency constraints. In computer vision, images are typically consumed as entire images, as opposed to pixel by pixel, and represents physical objects, therefore each pixel can be painted only in conjunction with its neighbours.

We will concern ourselves only with structured *classification* in this work, with discrete labels. Structured regression, where atoms have continuous labels, is a distinct problem, and uses models such as Gaussian Markov random fields or Gaussian process networks (Wilson et al. (2012), discussed in section 4.11). An application example for structured regression consists in modelling a network of temperature sensors, where we wish to model the temperature measured by one sensor as being correlated to the temperature measured by spatially close sensors.

This thesis examines *labelling* problems, where the input, for instance image or text, already exhibits structure, and where the output consists of a set of labels applied to the subcomponents of the input. Because there is structure in the input, it must be found in the output as well. Such problems are usually important sub-tasks in machine intelligence systems. For example, we will look at semantic segmentation of images, which consists in assigning, to each pixel of an image, a label describing which physical object it belongs to.

This is a very hard problem, because the algorithm needs to jointly satisfy a very large number of hard or soft constraints on all atoms, and because any brute-force, enumeration-based method fails due to the number of possible label combinations being exponential in the number of atoms.

One technique for structured labelling basically ignores the purported interdependence in the output, and simply performs local predictions on each atom. A common way to implement this approach is to take into account the local structure of the input, as opposed to enforcing it in the output. This is achieved by extracting features which represent not just the current input atom, but some context window around this atom, in some cases even the entire input object. With these features as input, a "local" classifier makes a decision about each output label, without considering their interdependencies. We call this approach *input-driven* structured prediction. For example, to label each amino-acid of a protein with the type of structure (helix, sheet, etc.) it will fold to in space — a problem known as protein secondary structure prediction — we can build a local multi-class support vector machine classifier which is fed data about a context window of a few amino-acids to the right, and a few amino-acids to the left of the current position (Wang et al., 2004).

In contrast, some fields and problems, such as natural language processing, call for a procedure which guarantees the coherence of the output, and work with probabilistic models which implement this. We qualify this approach as *model-driven.* It is the approach adopted in this thesis. A prominent example of a model built according to this principle is the conditional random field (Lafferty et al., 2001), in which labels interact with their neighbours, and which forms an important inspiration for the model we introduce in chapter 4, GPstruct. Of course, most practical implementations of model-driven structured prediction, including ours in this thesis, also make use of a feature template computed on context windows, to take advantage of information supplied by the neighbourhood of the current atom.

The model-driven approach is computationally hard. Beyond the exponential size of the output space, consider that learning, that is, adapting the parameters of a model so that predictions are optimal, must make repeated use of the "prediction routine" internally. Indeed, it calls it every time it evaluates the goodness of a particular choice of parameters. This problem alone is the object of many research efforts.

A traditional formulation of the prediction problem casts it as a score maximiser, and makes sure that the learning problem can be formulated as a constrained optimisation problem. Subsequently, efforts concentrate on gaining computational advantages from manipulating the optimisation problem by relaxing or adding constraints, working on the feasible set, introducing approximations, swapping the dual for the primal problem, or replacing the numerical optimisation with search. The structure of Nowozin and Lampert (2010), which gives an overview of the challenges and solutions associated with structured output prediction in computer vision, is established according to the type of reformulation of this very problem.

This thesis adopts a conceptually different approach by describing the problem in probabilistic terms, and by exploiting techniques from the subfield known as *probabilistic machine learning*. In this view, the central objects are probability distributions: observed input and output data are observations of random variables, while unobserved quantities are considered as latent, that is, initially distributed according to a prior distribution, and then constrained by the observed variables. This approach may be computationally more intensive than an optimisation-based formulation, but efficient approximations, some of which are discussed in chapter 5, can be applied to scale it to large data sets.

Its use is motivated by the number of advantages it brings. An important one is that it deals naturally with sparseness, that is, the undesirable property that even in large datasets, some characteristics which need to be learnt are exhibited only rarely. For instance, in film rating systems, most of the long tail of films will only have been rated by few people. Probabilistic models are able to easily and correctly combine evidence from different sources, and incorporate background information to evaluate rarely seen events.

These models produce results in the form of probabilities, which means that uncertainty in the prediction is accounted for naturally, and that going from probabilistic hypotheses to firm predictions can be informed by loss functions (or utility functions, for the optimists). Therefore, once the model has been learnt, changes to the loss function do not require the entire model to be learnt again (as is the case with many non-probabilistic methods), which could be costly in the case of large data sets.

Non-parametric Bayesian (NPB) models are a class of probabilistic models which exhibit one further advantage in this setting. Roughly speaking, one can view them as infinite-capacity models, i.e. as infinite extensions of parametric models. For instance, the Dirichlet process used for clustering can accommodate an unbounded number of clusters; the infinite hidden Markov model an unbounded number of discrete states; the Indian Buffet Process an unknown number of binary features per data point.

NPB models have recently received much attention, as they are more adaptive than parametric models, while retaining the desirable features of probabilistic modelling. Indeed, they allow the effective capacity to fluctuate towards its preferred value, with for example only one or two parameters to control its propensity to grow. Hence, these models eschew the issue of model averaging or model selection, which arises in parametric modelling: there we must compare how well models with different, but fixed, capacity fit the data. This particular aspect makes NPB models a particularly attractive choice for modelling, and justifies our focus in this thesis.

This introduction provided an informal overview and motivation for the topic of this dissertation. Further background on this dissertation's subject matter can be found in modern machine learning textbooks: on probabilistic machine learning, Bishop (2006) and Murphy (2012) are excellent texts; on inference in graphical models, a serious reference is Koller and Friedman (2009); on kernel machines, see Schölkopf and Smola (2002); Rasmussen and Williams (2006) provides a good introduction to Gaussian processes in machine learning.

1.2 Thesis structure

The thesis is roughly divided in two parts. The first, entirely contained in chapter 2, is more algorithmic in nature than the second part, contained in chapters 3 to 6, which is concerned with modelling. Each chapter has its own introduction and conclusion to motivate and contextualise the research, which is why this general introduction is kept short. In particular, chapter 3 functions as an introduction to the second part of the thesis.

Chapter 2 presents the application of the map-reduce technique (Dean and Ghemawat, 2004) for distributed computing to a Markov chain Monte Carlo algorithm used to train a Bayesian non-parametric model, the infinite hidden Markov model (IHMM; Beal et al. (2002)). The model is introduced as an answer to the issue of determining the state-space cardinality of sequence models like the hidden Markov model (HMM). The Dirichlet process (Antoniak, 1974) proves an essential building block to solve this problem, but only over a discrete base measure, which motivates the use of the hierarchical Dirichlet process. We then present our use case in natural language processing, unsupervised part-of-speech tagging. We give details of the method and algorithmic alternatives which surface when porting the training algorithm into the map-reduce framework. The main experimental insight from this work is the realisation that major map-reduce frameworks, such as Hadoop (White, 2009), which was used in our experiments, are not suitable to highly iterative algorithms. We show how, since our experiment, this finding became mainstream knowledge in the field, and how the map-reduce framework evolved into distributed paradigms which work on more general computational graphs.

The second part of the thesis is devoted to a non-parametric, kernelised model for structured output prediction, which we call GPstruct.

Chapter 3 presents the probabilistic model families which form the context of GPstruct. This model was conceived in response to the lack of a model exhibiting a list of desirable characteristics, so our presentation introduces these characteristics one by one: Bayesian inference, kernelisation, structured classification. This chapter, rather than merely exploring related work, attempts to clarify how pre-existing models can be inserted in a grid which schematises their characteristics, and to show that the last vertex of that grid remains unoccupied: filling this vertex is the role of GPstruct. The subsequent chapters of the thesis demonstrate how the model's characteristics come to bearing.

Chapter 4 introduces GPstruct formally, and describes an MCMC training algorithm based on elliptical slice sampling (Murray et al., 2010). In this chapter, we present an application of GPstruct to sequence modelling for natural language processing tasks, and show that it performs well in comparison to similar models. Analytical experiments probe different aspects of learning with the model: we evaluate the robustness of the learning algorithm against variations in the MCMC configuration, and test whether GPstruct works as well when moving from Bayesian to maximum a posteriori inference. In this chapter, we also contrast GPstruct with related models.

Chapter 5 tackles the challenges appearing when GPstruct is applied to large-scale data. For labelling tasks, image data sets obey the definition of "large-scale data", as they consist of a very high number of individual pixel positions which must be labelled individually. We describe three strategies which help overcome the scaling issues: (a) to overcome the intractability of the likelihood computation on a grid, we use a pseudo-likelihood approximation; (b) in addition, the learning problem is distributed over an ensemble of weak learners which are combined to produce predictions; (c) finally, the prediction process itself, which requires performing the marginal maximum a posteriori operation over a grid, is intractable, and is approximated. This chapter describes image semantic segmentation experiments on two image datasets. GPstruct is shown to perform well in terms of labelling accuracy and predictive probabilistic calibration.

Chapter 6 explores hyperparameter learning, which probabilistic models can achieve without grid cross-validation. To address the coupling between hyperparameters and latent variables in GPstruct inference, several reparameterisations are presented. A statistical test aimed at detecting MCMC implementation errors is derived from Geweke's "getting it right" setup (Geweke, 2004) and successfully applied to GPstruct code. Extensive hyperparameter learning experiments are carried out with both synthetic data and sequence data used in chapter 4. While labelling accuracy is not strongly improved, we show that hyperparameters are definitely learnt in the process by analysing their posterior distributions.

Material in section 2.3 was published as Bratières et al. (2010b). Sections 4.1 to 4.7 were published as Bratières et al. (2015). Chapter 5 was published as Bratières et al. (2014). All these works are the result of collaborations with my co-authors.

1.3 Notations and conventions

We write matrices with bold, uppercase letters, e.g. $\mathbf{X}$, vectors with bold, lowercase letters, e.g. $\mathbf{x}$, and scalars with italicised lowercase letters, e.g. x. The t-th element of vector $\mathbf{x}$ is written x_t, like a scalar.

Except where unpractical, running indices are lowercase letters, and the maximal value of an index is the corresponding uppercase letter: $t \in \{1...T\}$, $n \in \{1...N\}$.

We write $f : X \to Y$ to express the fact that function f has domain X and codomain Y ; when referring to individual variables $x \in X$, $y \in Y$, this can be written $f : x \mapsto y$.

We ignore the distinction between a random variable and the value it takes, and use the symbol $p(\cdot)$ to denote probability density or mass functions, according to the variable. $\mathcal{N}$ denotes the normal distribution. $y \sim \mathcal{N}(\mu, \sigma^2)$ is read "random variable y is distributed according to a normal distribution with mean μ and variance σ^2", and carries the same meaning as $p(y|\mu, \sigma^2) = \mathcal{N}(y|\mu, \sigma^2)$.

Throughout this thesis, $\log$ denotes the natural logarithm, i.e. in basis e.

Numbers are written according to standard ISO 80000-1, i.e. using a space for digit grouping, and an English-style point as decimal sign, while positive numbers smaller than 1 have a leading 0 before the decimal sign. Sometimes, large numbers are written in "e-notation" (scientific notation) for readability, so 7e4 is 7×10^4.

In citing references for specific techniques or concepts, modern, comprehensive descriptions are favoured over pinpointing the historical source of the idea. As a consequence, consolidated journal publications are preferred over the original conference paper, and for more classical material, textbook presentations are retained over articles. This is not to say, of course, that historical developments should be ignored. Despite the fact that machine learning is still a young discipline, many of its tools derive from applied mathematics in a broad sense, a much older established discipline. For this reason, studying the historical development of machine learning concepts is instructive; this motivates the importance of presentations such as Tanner and Wong (2010) and Schmidhuber (2015).

Chapter 2

Map-reduce inference for the infinite HMM

This chapter examines the application of big data methods, specifically map-reduce, to a probabilistic machine learning task. This research is best situated in the wider context of distributed methods for probabilistic modelling, specifically non-parametric Bayesian models.

"Big data", i.e. data too large to fit onto a single computer, due to the size of the data, or due to prohibitively long computation, calls for distributed algorithms. We will make the following difference between *parallel* and *distributed* in this thesis: parallel algorithms run on a multi-threaded (typically multi-core) processor and share memory[1] ; distributed algorithms run on separate processors, share no memory, but are connected by a network infrastructure.

Distributed algorithms are often more complex to develop than their batch or parallel counterpart, but probabilistic models will be more useful when trained on large datasets. This is especially true of the non-parametric family of models, to which the infinite HMM belongs, and whose capacity expands as more data is available for training.

The goal of the research presented in this chapter is to port a pre-existing algorithm, namely MCMC inference for the IHMM (introduced in section 2.1) on an unsupervised part-of-speech tagging task, to a map-reduce framework, and implement it on a mainstream distributed platform, Apache Hadoop running on Amazon Elastic MapReduce (introduced in section 2.2).

The broader research challenge is to invent scalable probabilistic inference tech-

[1]By "memory", we mean RAM, not disk storage. The distinction is on access latency and bandwidth; in a parallel setting, slave processes share access to a high-speed, high-throughput memory (RAM), while in a distributed setting, communication (over the network) is expensive, and disk accesses are slower than memory accesses.

niques, and to experimentally validate them on commodity computing clouds.

Before this project started, a parallel version of this algorithm had been implemented (Van Gael et al., 2009). With growing datasets, the parallel implementation was hitting its boundaries in terms of CPU power, while the non-parametric model seemed capable of accommodating more data. The promise that superior NLP performance was attainable motivated porting this specific algorithm to a distributed platform. The limiting factor here is computation, not data size.

This chapter is organised as follows.

Section 2.1 introduces sequence models and their use in an unsupervised setting akin to clustering, singles out the IHMM model, and describes how to apply it to part-of-speech tagging. Aspects of our research concerning distributed computing are analysed in section 2.2, which presents the motivation for a commodity cloud platform, our design strategy for the map-reduce implementation, and related challenges. Section 2.3 describes experimental settings and results. Section 2.4 discusses an important finding of this research, namely the need for map-reduce platforms which support highly iterative algorithms, and positions the research in trends we can name and identify only today. Section 2.5 offers a detailed literature review of distributed inference for probabilistic models, and section 2.6 concludes the chapter.

2.1 Sequence models and the IHMM

2.1.1 Hidden Markov models, state space cardinality, clustering, and non-parametric Bayesian models

The hidden Markov model (HMM) is an extremely popular model for probabilistic modelling of sequential dynamics. It works in discretised time, and distinguishes between a hidden state, which belongs to a discrete state space, and an observation which causally depends on the state. The observation could be continuous, discrete, a vector or matrix, as long as its emission probability $p(\text{observation}|\text{state})$ can be specified. The model dynamics are first-order[2] Markov, that is, the state at time t depends only on the state at time $t-1$. For this reason, the model needs a transition probability model $p(\text{state}_t|\text{state}_{t-1})$, as well as, to be complete, the distribution of the initial state.

When used in machine learning, the most important questions about the HMM are[3]:

(1) Given model parameters and an observation sequence, what is the probability of the observation sequence?

[2]We will restrict our discussion to first-order models in this thesis. Higher-order HMMs are of course possible and useful, depending on the task.

[3]cf. Rabiner (1989) for a classic tutorial overview, specifically its section C for the discussion of the following problems

(2) Given model parameters and an observation sequence, what is the most likely corresponding state sequence?

(3) Given training data in the form of observation sequences (states are hidden), how do we infer the model parameters which make the observation sequences most likely?

The answer to each of these question comes in the form of an algorithm.

Question (1) is answered by the forward-backward algorithm; (2) by the Viterbi algorithm[4], a dynamic programming algorithm; (3) by the Baum-Welch method, an instance of expectation-maximisation used to find the maximum likelihood estimates of the parameters.

When modelling a new real-world domain, one difficulty makes latent state inference on sequences hard: how many hidden states are there? For example, in speech recognition, how many context-dependent triphone HMM states do we use to model the phonetics of a language[5]? In activity recognition, among how many different activities should we discriminate? In part of speech tagging, how many parts of speech are there?

This problem is comparable to determining the number of clusters in a clustering application. Latent state inference can be viewed as assigning each observation to a cluster, with additional constraints or information provided by the Markov dynamics of the sequence.

Determining the number of hidden states can be formulated as a model selection issue in a probabilistic framework. A hard selection can be obtained by selecting the model with best score according to some criterion: this is the model posterior if we adopt a hierarchical Bayesian view, or it could be an information criterion like the Bayesian information criterion (Schwarz, 1978) or the factorised information criterion (Fujimaki and Hayashi, 2012). Alternatively, if the number of states itself is not of interest, it can remain latent, and marginalised out in a Bayesian model.

This last approach is the one adopted in the formulation of the infinite HMM (Beal et al., 2002), a variant of the HMM with unbounded discrete state space, in which the prior over states is given by a Dirichlet process. We describe this model and an MCMC parameter inference algorithm in the present chapter, in further sections. The task on which we apply the IHMM is part-of-speech tagging, which we describe now.

[4]assuming we are interested in the single best sequence (as opposed to other interpretations of the “most likely” state sequence)

[5]This is a controversial question not only for the application of HMM, but in linguistics in general, so the question is only exacerbated in the context of automatic speech recognition. A strand of research in speech recognition has tried to do away with settling on a definite list of phonemes, since it is not directly relevant to the determination of an orthographic form, trying to leave the choice “up to the data”; cf. Hannun et al. (2014) for a recent contribution on this question.

2.1.2 Part-of-speech tagging

Parts-of-speech (PoS) are grammatical categories of words, for instance verb, noun, or adverb. PoS tagging is an important pre-processing task inside many natural language processing pipelines.

Single words often do not correspond to a unique PoS, as exemplified in the sentence "I can can a can"[6]: the PoS of the word "can" depends strongly on neighbouring words, and neighbouring words' own PoS. Therefore, context-dependent tagging systems are needed. Early PoS tagging methods relied on rule and exceptions lists; a prototypical example is the transformation-based Brill tagger (Brill, 1995), which learns a set of rewrite rules from an annotated corpus; the Brill tagger has performed excellently for a long time despite its simplicity.

In contrast to rule-based systems, PoS tagging systems based on probabilistic models explicitly model the dependencies between single words and their possible tags, including neighbouring tags and words, sometimes inside a context window of several words. It is beneficial that the output is probabilistic, because the downstream processing pipeline might want to preserve and manipulate uncertainty on tags as subsequent operations deliver more information on the PoS.

A particularly fruitful model is the hidden Markov model, which is able to model dependencies of tags on their direct predecessor. More specifically, since there need not be a one-to-one correspondence between PoS tags and HMM states, the Markov dependence is between states. We will come across another popular model for PoS tagging, conditional random fields, in chapter 3.

Traditionally, HMM PoS tagging models have been trained by supervised learning, i.e. using large databases of PoS-annotated natural language text. Such corpora are expensive to produce, and especially scarce for new domains or low-resource languages. This state of affairs has motivated the use of unsupervised methods, which extract knowledge from unlabelled text, which is more easy to get by. Unsupervised HMM PoS tagging had met with some success in the years prior to the work reported in this chapter, cf. Johnson (2007); Gao and Johnson (2008); Griffiths and Goldwater (2007). A fundamental issue with the unsupervised approach is the identification of HMM states to PoS tags, as they appear in a tag list for instance. While Johnson (2007) and Gao and Johnson (2008) report that this mapping is relatively easy to carry out manually, Griffiths and Goldwater (2007) applies some amount of supervision by specifying the PoS tags of some common words. Even without this identification, though, the result of supervised learning can be used in NLP pipelines for instance as a partial language model, to assign scores to generated sentences (as in machine translation, natural language generation or speech recognition).

A methodological issue quickly comes up: how to evaluate the quality of the un-

[6]or the famous "Buffalo buffalo buffalo buffalo" sentence

supervised model (Vlachos, 2011)? Accuracy of prediction is made meaningless since HMM states cannot be compared to a reference labelling.

If we are ready to consider the clusterings induced by predictions on a test set, a solution to this issue is offered by the variation of information (VI) metric introduced by Meila (2007), which compares two clusterings. The scoring procedure compares the clustering induced from reference labels on a test set, with the clustering induced by some latent state assignment under the trained model (i.e. a sample of the latent state sequence under the posterior model). To make this clear, let us introduce some notation. We denote by $(y_1...y_T)$ the sequence of words (observables), and by $z_1...z_T$ the sequence of labels applied to them. A clustering k is a function defined over positions t in the sequence, which assigns a label to each position : $k(t) = z_t$ is the label applied to y_t under clustering k. Clustering k_{ref} assigns the reference labels as they appear in the test data. Clustering k_{pred} assigns as labels the HMM states obtained from a sample of the posterior model. We then define $p(k_{\text{ref}}(t) = k,\, k_{\text{pred}}(t) = k')$ from counts over the test data. We now have a joint probability distribution over couples of (reference, predicted) labels (k, k'), based on which we can define cross-entropies $H(\text{ref}|\text{pred})$ and $H(\text{pred}|\text{ref})$. Finally we define $VI = H(\text{ref}|\text{pred}) + H(\text{pred}|\text{ref})$. VI effectively compares the homogeneity and point distributions of two clusterings. Its interpretation and normalisation for comparison under different settings needs some thought, cf. the discussion in Meila (2007).

Other sensible ways of assessing a trained IHMM model for PoS tagging are extrinsic, e.g. assuming we have a language processing pipeline which requires only discrimination between PoS tags, but not their identity (as is the case for shallow parsing, for instance). We can then observe the impact (positive or negative) on a system metric when replacing a baseline PoS tagging component by the PoS tagger under study. This is the approach adopted in Van Gael et al. (2009).

All unsupervised HMM methods report issues with deciding on the number of HMM states. The infinite HMM provides a principled solution to this problem by assuming an unbounded pool of states, but actually only invoking a finite number of them for any finite data set. This ultimately motivates the choice of PoS tagging as an application for the work reported in this chapter.

2.1.3 Defining a non-parametric model for sequence observations

In this section, we make the parallel between clustering and latent state inference more concrete, by initially considering a Dirichlet mixture model used for unsupervised clustering, and then applying changes, first to model sequential data, and then to make the state inventory unbounded.

Observed data points are $y_n \in \mathcal{Y}$, $n \in \{1..N\}$, and assumed i.i.d. According to

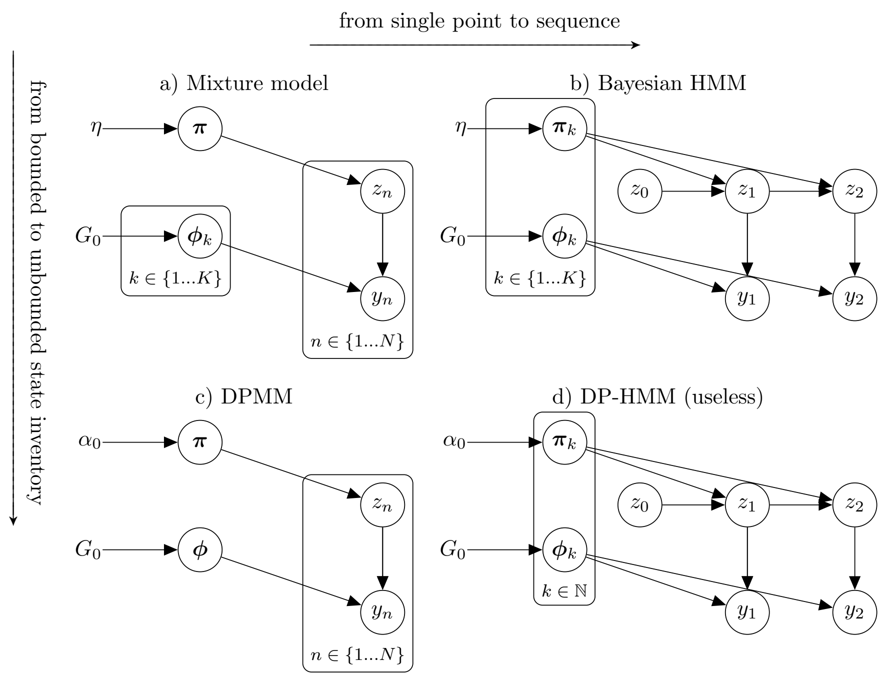


Figure 2.1.1: Various clustering and HMM models

the nature of the observables, $\mathcal{Y}$ could be a continuous space or a discrete space[7]. The output distribution (over $\mathcal{Y}$) is denoted by F, and is parameterised by $\boldsymbol{\phi}$. When y_n is discrete, F could be a categorical distribution (i.e. a multinomial distribution reduced to one draw), and $\boldsymbol{\phi}$ a vector of probabilities. $\boldsymbol{\phi}$ is given a prior G_0, in other words the support of G_0 is the parameter space for F.

In order for each state z to define a different emission probability, ϕ is indexed by z.

We can now present the generative model for the Dirichlet mixture model, in figure 2.1.1a. It is further defined by:

$$\begin{aligned}
\text{cluster membership prior } \boldsymbol{\pi} &\sim \text{Symmetric Dirichlet}(\eta) \\
\text{cluster membership } z_n &\sim \text{Categorical}(\boldsymbol{\pi}) \\
\text{cluster parameters } \boldsymbol{\phi}_k &\sim G_0 \\
\text{observed data } y_n &\sim F(\phi_{z_n})
\end{aligned} \tag{2.1.1}$$

G_0 serves as a prior over the parameters $\boldsymbol{\phi}_k$ of each cluster. Both the states z_n and the cluster index k are bounded for this model.

[7]For our part-of-speech tagging application, $\mathcal{Y}$ is the set of words, also cf. the notation introduced in section 2.2.3.

Recapitulating, we have:

$$\begin{aligned} \eta &\in \mathbb{R} \\ \boldsymbol{\pi} &\in \mathbb{R}^K \\ z_n, k &\in \{1..K\} \\ y_n &\in \mathcal{Y} \end{aligned} \tag{2.1.2}$$

Turning this model into an HMM means making the n-plate Bayesian network a dynamic Bayesian network (over t). Observed data now consists of sequences, each of the form $y_1, ...y_t, ...y_T$. We need to:

- define transition probability vectors (over destination states) for each origin state; therefore π is now inside the k-plate
- replicate the state and emission nodes over time steps; while each emission node y_t is connected to a hidden state node z_t in the same way as before, parameterised by $\boldsymbol{\phi}_k$, the hidden state is now parameterised by a transition distribution which is indexed by the previous state, instead of being unique

This yields the HMM illustrated in figure 2.1.1b, with the following definitions:

$$\begin{aligned} \text{row of transition matrix } \boldsymbol{\pi}_k &\sim \text{Symmetric Dirichlet}(\eta) \\ \text{state } z_t &\sim \text{Categorical}(\boldsymbol{\pi}_{z_{t-1}}) \\ \text{state parameters } \boldsymbol{\phi}_k &\sim G_0 \\ \text{observed data } y_t &\sim F(\phi_{z_t}) \end{aligned} \tag{2.1.3}$$

Our graphical model represents the case $T = 2$. We still have a finite state inventory, so $k \in \{1..K\}$ for this model, and $z_t \in \{1..K\}$. We assume that the initial state z_0 is fixed. Rows $\boldsymbol{\pi}_k$ of the transition matrix are (vectors of) transition probabilities for a given origin state k (the state corresponding to the row). $t \in \{1...T\}$ is the index inside a given sequence (the data may consist of several sequences of different length, for simplicity we do not account for this in our notation here, and consider only one sequence).

How to make the state inventory unbounded, i.e. how to make the model nonparametric ?

Starting from the mixture model in figure 2.1.1a, a known construction to make the number of states unbounded is to turn the Dirichlet prior on the cluster identity into a Dirichlet process prior, yielding a DP mixture model (Antoniak, 1974). Now the number of states K is unbounded, $\boldsymbol{\pi}$ is infinite-dimensional, and ϕ_k is defined for any

$k \in \mathbb{N}$. The corresponding model is illustrated in figure 2.1.1c and defined as:

$$\begin{aligned} \text{cluster membership prior } \boldsymbol{\pi} &\sim \text{SBC}(\alpha_0) \\ \text{cluster membership } z_n &\sim \text{Categorical}(\boldsymbol{\pi}) \\ \text{cluster parameters } \boldsymbol{\phi}_k &\sim G_0 \\ \text{observed data } y_n &\sim F(\boldsymbol{\phi}_{z_n}) \end{aligned} \tag{2.1.4}$$

SBC denotes the stick-breaking construction, which Ewens (1988) originally denoted by GEM after the names of Griffiths, Engen and McCloskey. We do not detail the SBC further and refer to Teh et al. (2006) for details.

$\boldsymbol{\pi}$ is now an infinite-dimensional vector. Yet for any realisation of the model with a finite number of data points N, only finite representations of $\boldsymbol{\pi}$ will be needed.

In the hope of obtaining an HMM model with unbounded states, we now apply the same changes to the HMM model in figure 2.1.1b, and obtain the model in figure 2.1.1d, which we might call the DP-HMM:

$$\begin{aligned} \text{row of transition matrix } \boldsymbol{\pi}_k &\sim \text{SBC}(\alpha_0) \\ \text{state } z_t &\sim \text{Categorical}(\boldsymbol{\pi}_{z_{t-1}}) \\ \text{state parameters } \boldsymbol{\phi}_k &\sim G_0 \\ \text{observed data } y_t &\sim F(\boldsymbol{\phi}_{z_t}) \end{aligned} \tag{2.1.5}$$

Upon closer inspection, however, we find that this model suffers from a fatal defect. For each origin state k, the DP prior indeed yields a new transition vector π_k and a parameter $\boldsymbol{\phi}_k$ for the emission probability F. However, with probability 1, each draw $\boldsymbol{\phi}_k$ from G_0, since it is a continuous distribution[8], will be different from any previous draw $\boldsymbol{\phi}_{k'}$, $k' < k$: almost surely, sampled states will not coincide. In other words, with transition probabilities drawn independently, and no coupling between them, there is no reason for the chain to preferentially revisit already existing states.

To apprehend the key issue, consider the following step of the stick-breaking construction of the DP: $\boldsymbol{\phi}_k \sim G_0$, i.e. for each draw from the DP, we need to sample a new collection $\boldsymbol{\phi}_k$ from the base measure G_0, with no guarantee that we will encounter already sampled states (again, unless G_0 is discrete, because then the chances could be non-zero, depending on the atom weights). To make sure we “recycle” states between source state distributions, we will make the base measure discrete: specifically, we will draw the base measure from another, higher-level DP, so that it is almost surely discrete.

In conclusion, while the DP is an interesting prior over clusters (for mixture models), it is not helpful as a prior for state-to-state transitions.

[8]The case in which G_0 is a fixed discrete distribution is contrived and not relevant here.

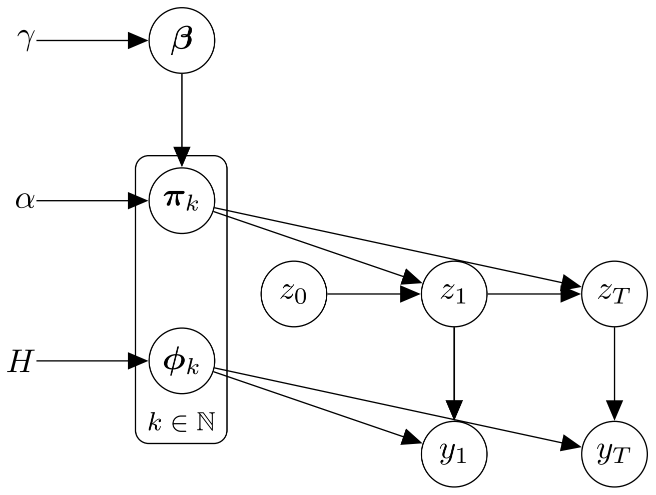


Figure 2.1.2: Graphical model of the IHMM. Rows $\boldsymbol{\pi}_k, k \in \mathbb{N}$ of the state transition matrix are given a HDP prior. Cf. section 2.1.4 for a full description.

2.1.4 The HDP-HMM

To solve the issue of the DP-HMM, we must make sure we "recycle" states between source state distributions; to ensure this, we will make the base measure discrete: specifically, we will draw the base measure G_0 from another, higher-level DP, so that G_0 is almost surely discrete. Beal et al. (2002) discussed this issue and proposed the hierarchical Dirichlet process HMM (HDP-HMM) to solve it.

A note on nomenclature: in this thesis, we will use the terms IHMM and HDP-HMM interchangeably. The original "IHMM paper" Beal et al. (2002) derived the IHMM by taking limits when $K \to +\infty$, and proposed an inference scheme constructed so that a parameter controls the propensity to create new states, and another parameter controls the probability of self-transitions. Later, Teh et al. (2006) (which uses the name "HDP-HMM") proposed a Gibbs sampling inference scheme.

G_0 is now a discrete distribution and transition probabilities π_k are coupled, as they are drawn from a DP with mean $\boldsymbol{\beta}$ (in doing so, we assimilate vectors with the categorical distribution they parameterise, e.g. $\boldsymbol{\beta}$ with Categorical($\boldsymbol{\beta}$)).

We now formally define the HDP-HMM when applied to the task of part-of-speech tagging. We introduce the notation and terminology which we use in the remainder of this chapter. The resulting model is illustrated in figure 2.1.2.

Let a sentence of length T be represented as a sequence of words (tokens, or observations in HMM terminology) $y_1, ...y_t...y_T$, with $y_t \in \mathcal{V}$, where $\mathcal{V}$ is the vocabulary, that is the set of observed words. We note card($\mathcal{V}$) $= V$, where V is the total number of unique words. V is assumed to be fixed and known; at prediction (test) time, unknown words can be dealt with via a back-off method.

In a generative view, tokens are generated from states (clusters) $z_1 ... z_T$ according to $p(y_t|z_t, \boldsymbol{\phi}_{z_t}) = \text{Categorical}(y_t|\boldsymbol{\phi}_{z_t})$, where ϕ_s is the parameter for the emission probability for state s; $\boldsymbol{\phi}_s \in \mathbb{R}^V$. In addition, we define dummy start and end states, and corresponding start and end tokens, which enclose each sentence.

State transitions are governed by a state transition matrix $\boldsymbol{\pi}$, of which each row $\boldsymbol{\pi}_k$ governs the transitions out of state k, so the probability of transitioning from state k at time t to state l at time $t+1$ is $p(z_{t+1} = l|z_t = k) = \pi_{k,l}$. The total number of states in usage for a given sample of the state sequences, including start and end states, is written K, so $\boldsymbol{\pi}_k^T \in \mathbb{R}^K$ and $\boldsymbol{\pi} \in \mathbb{R}^{K \times K}$.

Thus far, our description corresponds to the structure of an HMM. We now place priors on $\boldsymbol{\pi}_k$ and $\boldsymbol{\phi}_z$, so that the number of states is variable, and governed by a hierarchical Dirichlet process. In addition, note that $\forall k$, $\boldsymbol{\pi}_k$ should be of infinite size, since the number of states is not bounded. At this point of the exposition, in order to prepare for writing out the MCMC procedure, which, to be implemented, requires manipulating finite structures, and to keep a mathematically correct notation, we will maintain the notation $\boldsymbol{\pi}_k$ for the finite-dimensional vector, and denote by $\bar{\boldsymbol{\pi}}_k$ the infinite-dimensional vector (equivalently $\bar{\boldsymbol{\pi}}$ vs. $\boldsymbol{\pi}$, and $\bar{\boldsymbol{\beta}}$ vs. $\boldsymbol{\beta}$, which will be introduced below). Each sample of π_k uses only a finite number of states, since the corresponding data uses only a finite number of tokens. Consequently, we will call the finite-dimensional vector a "collapsed" version of its infinite-dimensional counterpart; it is obtained by ignoring elements corresponding to unused states, i.e. one of the (countably infinitely many) states which are not represented in the data.

The $\bar{\pi}_k$, like $\bar{\boldsymbol{\beta}}$, behave like probability distributions, so they sum to one, and it is essential to represent the probability mass which has been discarded in the process of collapsing (the "remaining mass"). For this reason, we append an extra element to each of the finite-dimensional counterparts: $\pi_{k,K+1}$ and β_{K+1} are set so that $\forall k$, $\sum_{j=1}^{j=K+1} \pi_{k,j} = 1$ and $\sum_{j=1}^{j=K+1} \beta_j = 1$.

$\bar{\pi}_k$ is a sample from a Dirichlet process with concentration parameter α and base measure $\bar{\boldsymbol{\beta}}$ shared by all $\bar{\pi}_k$. $\bar{\boldsymbol{\beta}}$ is also generated by a Dirichlet process, and is modelled by the stick-breaking construction (Sethuraman, 1994) with parameter γ, i.e. $\forall i \in \mathbb{N}, \bar{\beta}_i = \zeta_i \prod_{j=1}^{j=i-1}(1-\zeta_j)$ with $\forall k \in \mathbb{N}, \zeta_k \overset{\text{iid}}{\sim} \text{Beta}(1, \gamma)$, which is also written $\bar{\boldsymbol{\beta}} \sim \text{SBC}(\gamma)$, cf equation 2.1.4.

The observation vectors also receive a prior, namely a symmetric Dirichlet distribution parameterised by $H \in \mathbb{R}_+$.

Summing up, we have

$$\begin{aligned}
y_t|z_t, \boldsymbol{\phi}_{z_t} &\sim \text{Categorical}(\boldsymbol{\phi}_{z_t}) \\
z_t|z_{t-1}, \bar{\boldsymbol{\pi}}_{z_{t-1}} &\sim \text{Categorical}(\bar{\boldsymbol{\pi}}_{z_t}) \\
\bar{\boldsymbol{\pi}}_k|\alpha, \bar{\boldsymbol{\beta}} &\overset{\text{iid}}{\sim} \text{DP}(\alpha, \bar{\boldsymbol{\beta}}) \quad \forall k \in \mathbb{N} \\
\bar{\boldsymbol{\beta}}|\gamma &\sim \text{SBC}(\gamma) \\
\boldsymbol{\phi}_k &\overset{\text{iid}}{\sim} \text{Dirichlet}(\underline{H}) \quad \forall k \in \mathbb{N}
\end{aligned} \tag{2.1.6}$$

where we have denoted by $\underline{H}$ the vector of size V with all elements set to H.

Note that there are two notation abuses in the line $\bar{\boldsymbol{\pi}}_k|\alpha, \bar{\boldsymbol{\beta}} \sim \text{DP}(\alpha, \bar{\boldsymbol{\beta}})$. The first is in $\text{DP}(\alpha, \bar{\boldsymbol{\beta}})$: we really mean the DP which has $\bar{\boldsymbol{\beta}}$ as its stick-weights sequence. The second is in writing $\bar{\boldsymbol{\pi}}_k|... \sim \text{DP}(...)$: $\bar{\boldsymbol{\pi}}_k$ is the stick-weights sequence obtained for the SBC for a draw from this DP; i.e. we completely ignore the positions $\boldsymbol{\phi}_k$ of the point-masses (which are shared anyway), only to concentrate on the weights.

We keep the hyperparameters α, γ and H fixed at all times (rather than giving them priors or estimating them using maximum likelihood).

2.2 Distributed computing aspects

After describing and motivating the IHMM model, we now delve into the aspects related to distributed computing and software engineering. We chose to avoid custom (closed) computing clusters, as all prior work had done, but deliberately exploited widely available commodity computing power.

2.2.1 Commodity computing infrastructure

In terms of deployment platform, this research targets commodity computing clouds for impact, visibility, reproducibility, and reusability. In particular, we are interested in clouds which are pre-configured for distributed computing, such as (in 2009, when this research was undertaken) Amazon's Elastic MapReduce (AEMR) platform, or Microsoft Azure with Microsoft HTC or Dryad (Isard et al., 2007). These clouds integrate storage, networking, computation, execution engine and language-level programming interface. The understanding of their economic model and advantages in 2009 is reflected in Armbrust et al. (2009).

At that time, distributed frameworks were in use in the industry and the open source community (Pig[9], Hadoop (White, 2009), BigTable (Google proprietary (Chang et al., 2006)) and HBase[10], Hive (Thusoo et al., 2009), Cassandra (Lakshman and

[9] `http://pig.apache.org/`
[10] `http://hbase.apache.org/`

Malik, 2010)[11]). In actual practice, in industry, they tended to be used mainly for non-statistical data manipulation: distributed grep's, inverted indexing, hashing and searching, sort, distributed aggregation. Distributed machine learning was very much confined to research.

Yet such clouds bring several advantages for research in distributed machine learning: they are publicly available, have a low upfront cost, use widely shared software configurations, and their hardware profiles are rather generic and reproducible. This summarises what we mean by "commodity" in this context, and is largely the opposite of custom computing clusters.

In particular, the requirement of reproducibility in computational sciences[12] has gained traction in recent years. Tools, methods, supporting procedures by scientific publishers, research institutions, funding agencies were developed to address this need. For instance, reproducibility in research is one of the major drivers of the development of the Python notebook system Jupyter[13], and it is indeed increasingly used to document scientific methods in supplementary material[14]. Often, software accompanying machine learning publications is available on Github. GitXiv was developed to coordinate the scientific preprint archive arXiv (largely used in the field of machine learning) with code repositories like Github and Bitbucket. Container technology such as Docker or Kubernetes, which was not widely available when this research was conducted, now allows even better archivability of the software configuration than what is achievable with commodity computing clouds.

The research focus on commodity cloud services proved to correctly anticipate future developments. AEMR, the deployment platform used in this research, was made public in April 2009, and was a forerunner in this area. It included a patched version of Apache Hadoop 0.18.3, a very early version. The Apache Hadoop Java open-source project[15] started around 2005 as a platform for map-reduce, and has since developed in many other directions, to the point that it can nowadays be considered an ecosystem for distributed computing. Its latest stable version[16] at the time of this writing is 2.8.0, and was released on 22nd March 2017.

Hadoop fostered the development of Apache Mahout[17], a Java package of map-reduce implementations of popular machine learning algorithms, which became an Apache top-level project on 4th May 2010[18]. The O'Reilly book "Mahout in Action"

[11]`http://cassandra.apache.org/`

[12]cf. Sandve et al. (2013) for a review of best practices, and Crick et al. (2015) for a suggested process

[13]earlier named IPython (Pérez and Granger, 2007)

[14]cf. `https://github.com/ipython/ipython/wiki/A-gallery-of-interesting-IPython-Notebooks#reproducible-academic-publications` for several examples of scientific publications with accompanying IPython (now Jupyter) notebooks and code.

[15]`http://hadoop.apache.org/`

[16]`http://hadoop.apache.org/releases.html`

[17]`http://mahout.apache.org/`

[18]`https://blogs.apache.org/foundation/entry/the_apache_software_foundation_announces4`

(Owen et al., 2011), written by the original authors of Mahout, was published on 9th October 2011. Mahout still exists, and has changed drastically in the meantime, giving up the map-reduce dependency altogether (cf. section 2.4.2). Our project uses the Mahout linear algebra libraries, but not its implementations of algorithms.

The dates we have just cited demonstrate that this research exploited very novel technology. Indeed, the founding papers for applications of machine learning to multicore platforms are arguably Dean and Ghemawat (2004) and Chu et al. (2007). To draw a parallel with technological developments at Microsoft, we can note that Microsoft Dryad (Isard et al., 2007), a distributed execution platform based on more complex computational graphs than map-reduce, started as an internal research project in 2006; it had an academic release on 13th July 2009[19], including the Dryad platform and DryadLINQ (Yu et al., 2008), which provided support to translate queries written in the LINQ query language into Dryad execution graphs. On 11th November 2013[20], however Microsoft withdrew its support to Dryad, favouring Apache Hadoop instead.

There now is a wide choice of managed distributed computing platforms, billed by compute time, from companies such as Databricks (Apache Spark), Google (Dataflow, Dataproc), Microsoft (Azure HDInsights running Apache Spark), Amazon Elastic MapReduce (now also running Apache Spark), Cloudera, MapR, Hortonworks.

Prior research in distributed probabilistic inference often did not target clouds or even private clusters or grids, but parallel multicore machines, sometimes running pseudo-distributed emulators. The few available deployments to clusters employed custom software. For instance, Nallapati et al. (2007) deployed on a cluster with 96 machines, using the PThread library and rsh for communication, Wolfe et al. (2008) ran a 32-machine cluster, (Doshi-Velez et al., 2009) deployed using Matlab Distributed Compute Engine. The research which comes closest to ours was (Wang et al., 2009), which used their own (Google Research) cluster with MapReduce, and compared with the Message Passing Interface (MPI), which is considered a standard for distributed computing in scientific fields which have a long tradition in high-performance computing such as climatology or theoretical chemistry), and is tailored less towards data partitioning and more towards computation partitioning. This shows that prior results, to be reproduced, required an essentially identically configured and tuned setup.

This section demonstrates the relevance, timeliness and significance, in the light of subsequent technological developments, of the research goal consisting of deploying on a commodity distributed computing cloud. In the rest of the present section 2.2, we detail the principles which guided our map-reduce development, as well as issues and best practices which emerged while implementing this research goal. We begin by recalling the basics of the map-reduce algorithm.

[19]`https://www.microsoft.com/en-us/download/details.aspx?id=52604`

[20]`http://www.zdnet.com/article/microsoft-drops-dryad-puts-its-big-data-bets-on-hadoop/`

2.2.2 Principle of map-reduce

We briefly outline the map-reduce algorithm[21].

Both mappers and reducers are functions which operate on key-value pairs, which we note $< K, V >$ here. A mapper (or *map* function) takes as input a single $< K_1, V_1 >$ pair, and outputs a single $< K_2, V_2 >$ pair. A reducer (or *reduce* function) takes as input a list of $< K_2, V_2 >$ pairs with the same key K_2, which is equivalent to defining their input as $< K_2, L_2 >$ where L_2 stands for a list of values. The reducer outputs a single $< K_3, V_3 >$ pair.

Here are the steps of the map-reduce algorithm:

Data preparation The input data is split up as $< K_1, V_1 >$ pairs, so that the data size of $< K_1, V_1 >$ and computation needed by the mapper to process $< K_1, V_1 >$ is orders of magnitude smaller than what can be processed on a mapper node. $< K_1, V_1 >$ pairs are sent to mappers, which are distributed on slave nodes.

Map The map function is applied iteratively by each mapper to the $< K_1, V_1 >$ pairs it has been assigned.

Shuffle $< K_2, V_2 >$ mapper outputs are sorted over the entire cluster according to key K_2, resulting in $< K_2, L_2 >$ pairs. These pairs are then assigned to reducers, which are also distributed on slave nodes. Each K_2 is assigned to exactly one reducer[22].

Reduce The reduce function is applied iteratively by each reducer to the $< K_2, L_2 >$ pairs it has been assigned. Each reducer may have several such pairs, and thus several K_2, to process.

Gather The reducer outputs are made available from the master node as a collection of $< K_3, V_3 >$.

We now illustrate the procedure on the word-count algorithm: given as input a collection of text documents, it outputs a dictionary of unique words associated to their count in the input corpus. Here, the mapper consumes $<$ document index, document $>$ pairs and outputs $< w, 1 >$ pairs (one for each word w; the number 1 here is nothing but a placeholder). These are shuffled, and reducers receive the pairs corresponding to a single word, so that given $< w, L >$ they only have to output $< w, \text{length}(L) >$, since L has exactly one element per word occurrence.

[21]More information can be found in very many places, starting with the original article Dean and Ghemawat (2004), countless web resources, the book White (2009).

[22]This, by the way, poses a restriction on the size of L_2

2.2.3 Map-reduce architecture for the PoS IHMM

The starting point for our map-reduce (MR) implementation is the blocked Gibbs sampling algorithm of the IHMM applied to PoS tagging. The derivation or properties of this algorithm are not the object of this thesis, so we refer to Beal et al. (2002) and Teh et al. (2006) for details of the algorithm and derivations, as well as to appendix A of this thesis, where we enumerate the steps of the Gibbs sampling algorithm and sketch each.

Most steps are for-loops and can be cast as MR jobs in a straightforward way, provided we solve issues regarding data sharing.

2.2.3.1 Data storage

Formalising the MR algorithm revolves around identifying data streams and sharing, and therefore requires identifying inputs, outputs and parameters at each step. This is done at the algorithm level in appendix A.

The question arises how to pass data from one MCMC iteration to the next. In Hadoop, data must be written to disk, and cannot stay in memory between MR jobs, because the Java virtual machine constitutes the execution framework for a single MR job, and is restarted after each. There are several option available for data storage in Hadoop (HDFS being the Hadoop distributed file system):

a) part-files: Mappers consume and reducers produce part-files, which remain local to the node after the reducer has finished. All data which must be mapped over by an MR job (as opposed to parameter data) is stored in this format. This concerns (notation from appendix A)

- sentences Σ,
- emission counts $\boldsymbol{f}$,
- transition counts $\boldsymbol{p}$,
- table counts $\boldsymbol{t}$,

Therefore, when a MR job needs to access data produced by a previous job, the data mapping of the former takes advantage of data locality.

b) regular HDFS files: Such files are accessible through the distributed file system from every worker node, and are be written once, and read several times. This format is adequate for parameters which are not mapped over by MR jobs, therefore we use it for hyperparameters $\boldsymbol{\pi}$, $\boldsymbol{\beta}$, $\boldsymbol{\phi}$, ν and the list of used states. This involves boilerplate serialisation code, as well as organising a directory structure per iteration.

2.2.3.2 Details of MR jobs

Analysing each Gibbs step from appendix A with these instruments, we can establish table 2.1, where the inputs and outputs of each MR job are formalised.

The table starts with the initialisation job MR0 which turns the initial text corpus into a set of `Sentence` objects, and is not part of the MCMC iterations. Each sentence from the training corpus has an identifier which is used as the key when the input to further MR jobs is Σ.

The table counts output from MR4 serves as input for the rebreak step, which is not a MR job. This is the only case in which data is reformatted, namely from part-file data format into a HDFS file.

MR jobs in table 2.1 fall into the following types:

map-only job (MR0, MR4, MR5, MRCleanupUnusedStates) This corresponds to an MR job which does not need a reducer. In that case the map's output stays on the slave nodes in part-file form, where it was generated, and is available for a subsequent map operation to run on.

summator reducer (MR1, MR2) This very basic and generic reducer will accept $< K_2, V_2 >$ with V_2 a numeric input, sum the numeric values, and return $< K_2, \sum V_2 >$. Often, the numeric input is simply 1, so the summator performs a count.

single reducer (MR1a, MR2a) Directing all the mapper output to a single reducer is possible: just set all K_2 to have a unique value, e.g. the numeric value 1. This might be necessary because the reducer needs to see all the intermediate data to run. This assumes that it is computationally efficient to require a single reducer to process the output of all mappers.

no-output reducer (MRMarkUsedStates) When the MR output of a reducer is not meant to be used as the MR input of a further mapper, the reducer may have no MR output (i.e. no $< K_3, V_3 >$ data), but instead write a piece of data directly to HDFS.

multiple reducer output types (mixed stream) (MR3) The need for V_3 to be of heterogeneous types may arise and requires workarounds to run with Hadoop[23]. Typically, different V_3 types are encapsulated into one single wrapper, or carrier, type. In a subsequent stage of processing, the wrapper is decapsulated according to the type of its content.

This issue does not arise in the single reducer case: since it is alone, the reducer can choose to output one main V_3 type as part-files, and write other output data directly to the distributed file system.

[23]This is due to map and reduce functions being implemented as Java functions. Java is statically typed, so that the input and output types are static throughout program execution.

2.2.3.3 Dependency diagram

Because the Gibbs steps do not depend on each other cyclically on purpose, one entire Gibbs iteration is realised by executing all these MR jobs in the order in which we have presented them so far. It proves useful for the design of the system to stay aware of the dependencies between MR jobs caused by input and output requirements. The best instrument for this purpose is a dependency graph (figure 2.2.1) of the MR jobs, in which we can represent the MR jobs, their dependencies, and the data streams. This diagram is read from top to bottom: the iteration input data (the (sentences, β) tuple) is found, in updated form, as the iteration output data. This is the only data passed between MCMC iterations. Note that the driver program, the one which is initially invoked, which contains the MCMC for-loop, and which calls all MR jobs in turn, runs on the master node.

2.2.4 MR job latency

As mentioned in the introduction to the present chapter, the reason why our task qualifies as "big data", and therefore the justification for turning to a MR framework, is the computing load, not the size of the data per se. Indeed all the source data (the Wall Street Journal corpus, cf. section 2.3.1 for a description), as well as intermediate data produced by single steps of the algorithm, fits on a single node. However, the computation needed for a single MCMC step on the full dataset (cf. section 2.3.1) on a 4-core machine is around 280 sec (cf. section 2.3 and table 2.2), making running the MCMC chain for several thousands of iterations, as seems needed by the model (cf. Van Gael et al. (2009)), prohibitive. We are therefore in a situation where we wish to accelerate a long computation by distributing it over several nodes.

A major contribution to the latency per MCMC iteration turned out to be MR job startup and teardown time (also cf. section 2.3). This did not appear initially because constant improvements on other aspects seemed to indicate that iteration duration could be reduced to an acceptable value.

Here is a summary of measures which were taken to optimise MR job latency.

- data exchanged between MR jobs within an iteration must be written to HDFS: The initial prototype was writing to AWS S3, generating extra latency and costs (S3 writes get billed, while cluster-internal HDFS writes are free). The latency impact turned out to be minor.

- JVM re-use: Hadoop by default restarts a JVM for each task, even when they run on the same node. It is possible to run tasks on the same node. However, JVM re-use is always among tasks of a single job. Tasks belonging to separate jobs (consequently also to separate iterations) are always executed on separate JVMs. Therefore this feature could not be exploited.

Table 2.1: Structure of map-reduce jobs

		key	value	remark
MR0	K1, V1	sentence key	corpus line	
	K2, V2	sentence key	sentence	
	K3, V3			map-only job
MR1	K1, V1	sentence key	sentence	
	K2, V2	(from, to) pair	1	summator reducer
	K3, V3	(from, to) pair	#transitions	
MR2	K1, V1	sentence key		
	K2, V2	(state, token)	1	summator reducer
	K3, V3	(state, token)	#emissions	
MR1a	K1, V1	(from, to) pair	#transitions	this is equivalent to passing the entire sparse transition count matrix
	K2, V2	1	(from, to, #transitions)	single reducer
	K3, V3	-	-	reducer output written directly to HDFS (π)
MR2a	K1, V1	(state, token) pair	#emissions	this is equivalent to passing the entire sparse emission count matrix
	K2, V2	1	(state, token, #emissions)	single reducer
	K3, V3	-	-	reducer output written directly to HDFS (ϕ)
MR3	K1, V1	sentence key	sentence	
	K2, V2	1 or sentence key	float resp. sentence with auxiliary variables	mixed stream
	K3, V3	sentence key	sentence with auxiliary variables	
MR4	K1, V1	(state, token) pair	#emissions	this is equivalent to passing the entire sparse emission count matrix
	K2, V2	(from, to) pair	#tables	this is equivalent to passing the entire sparse table count matrix
	K3, V3			map-only job
MR5	K1, V1	sentence key	sentence with auxiliary variables	
	K2, V2	sentence key	sentence with resampled states	
	K3, V3			map-only job
MRMarkUsedStates	K1, V1	sentence key	sentence	
	K2, V2	1	used state	
	K3, V3	-	-	reducer output written directly to HDFS (list of used states)
MRCleanupUnusedStates	K1, V1	sentence key	sentence	
	K2, V2	sentence key	sentence with renumbered states	
	K3, V3			map-only job

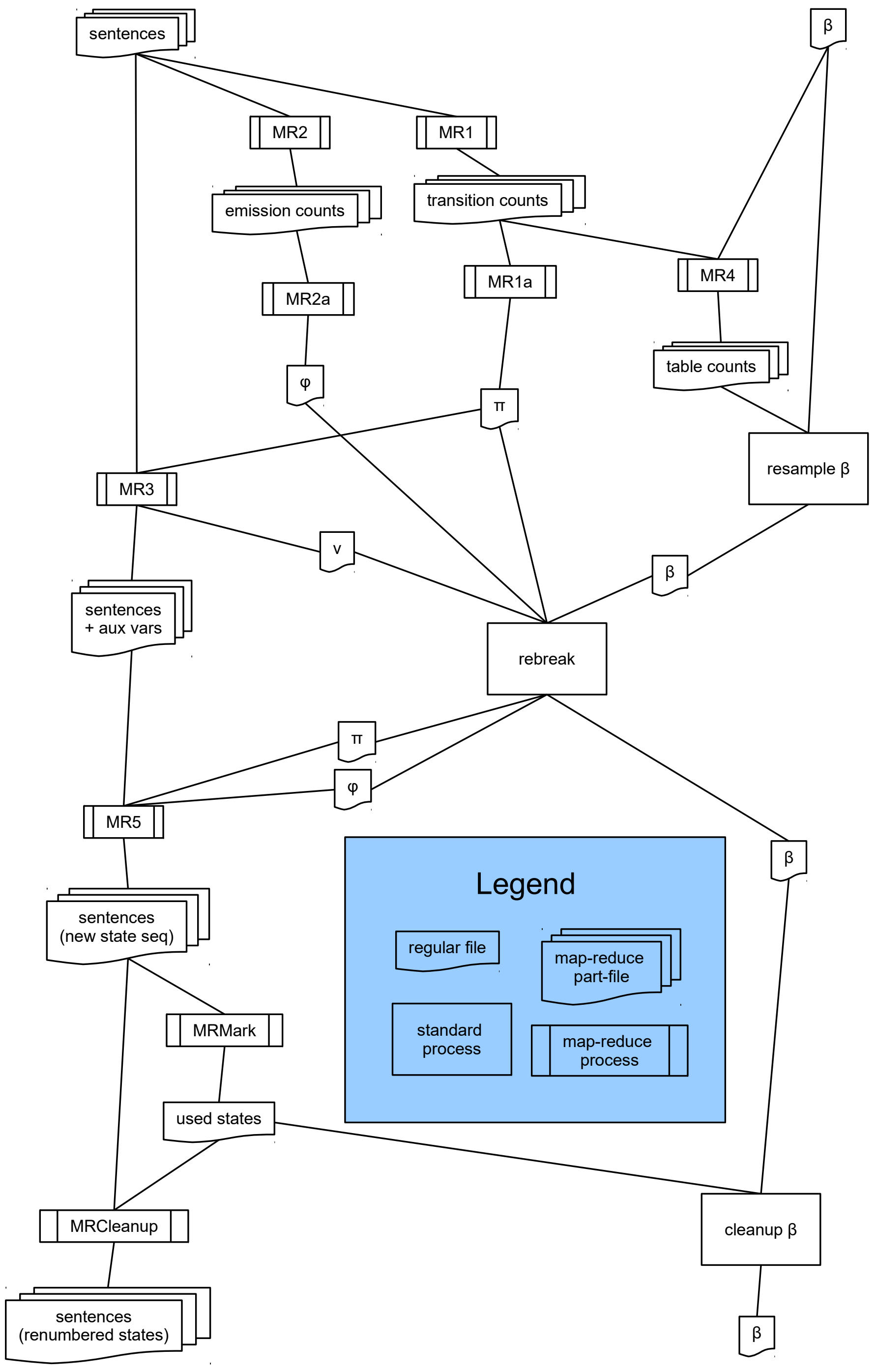


Figure 2.2.1: Dependency diagram for one MCMC iteration. "Regular files" are HDFS files. "Standard processes" are small tasks executed in-memory on the master node, possibly as Java for-loops. There is a sequence of in-place modifications of the sentences (represented in vertical alignment) which traverses the following jobs: MR3, MR5, MRCleanupUsedStates.

- JobControl parallelisation: this Hadoop option allows defining dependency graphs of MR jobs. Implementing the entire iteration represented in figure 2.2.1 as a dependency graph, as opposed to MR jobs called in program sequence, hardly improved latency.
- map-only jobs: an initial implementation used an IdentityReducer which would simply reproduce $< K_2, V_2 >$ into $< K_3, V_3 >$, but we found the reducer could be left out entirely. This improved the map-only MR jobs.
- set number of mappers per node: by default Hadoop starts several mappers per node, as a preventive measure against mapper failure. This parameter was set to one mapper per node.
- re-use Writable objects: the Writable object constructor is called several times per map and reduce task. Instead of reconstructing the object, it can be reused, and its properties modified. This measure, found on a Hadoop performance tuning blog, did not improve latency much.

Despite these measures, MR job startup and teardown latency remained long, on the order of 10 to 20 sec per job[24]. Therefore, it seems opportune to re-examine the maximalist approach adopted so far, which consisted of formulating all possible steps as MR jobs, resulting in nine such jobs per iteration, as seen in table 2.1.

To better understand their relevance, one should consider the cardinality of the key-value data set on which each job operates.

Six MR jobs map over sentences, i.e. over the full training corpus. This data set has the largest cardinality of all MR input data sets in our setup, on the order of 1e6 for the text corpus we use in experiments in section 2.3. MR2a maps over the vocabulary (cardinality of order 1e5); MR1a and MR4 map over realised (from, to) transition pairs (cardinality at most the square of the total current number of states, i.e. on the order of $100^2 = 10\,000$).

Given that MR job latency is a limiting factor, it is not justified to implement small-cardinality Gibbs steps as MR jobs. To cut down on the number MR jobs, all but MR5 were turned into for-loops executed on the master node, as they are computationally light. MR5 is both compute-intensive (it is a dynamic program) and data-intensive (it maps over tokens), and consumes the largest part of total iteration computation time.

In our experiments, we implemented both variants: all steps as MR jobs and only MR5 as an MR job (cf. section 2.3.2)

[24]This can be seen from table 2.2, in cases with very little data ($N = 1000$), so that the iteration time is almost only dedicated to MR job startup and teardown. *hadoop-1-** jobs have nine MR jobs per iteration, *hadoop-2-** jobs have one MR job per iteration.

2.2.5 Use of a reference, non-distributed implementation

A successful model for turning a single-core algorithm into a map-reduce one was given by the then early-stage Mahout project, in which several algorithms were programmed on Hadoop. Our work confirmed that a particular design rule used in the Mahout project proved to be of primary importance: starting from a **reference, non-distributed implementation of the algorithm**[25] .

Indeed, the typical path for distributing a machine learning algorithm, as described by Gonzalez (2014), starts with a batch implementation, then maybe a sequential or online learning version, followed by a parallel version, and eventually a distributed version. At each step, the program often needs to be rewritten, new software architecture issues are solved, and testing, debugging, and optimisation take place.

While the answer of Gonzalez (2014) consists of new supporting technology, our argument in this section is that improvements are brought about by systematically starting from a reference implementation, then encapsulating algorithmic sections into independent functions, which can be tested, typically with unit and functional tests; and finally writing the map-reduce driver program and individual jobs, where each job reuses procedures from the reference implementation.

This approach brings the following advantages.

In maintenance terms, rewriting a specific algorithmic function, for performance or correctness, is made simple as each is packaged in a modular form. To validate and tune input/ output format, readers/ writers, and their compatibility across MR jobs, development and debugging on the reference implementation does not incur network latencies on the cluster. Evolving the program continuously from the original implementation reduces the likelihood of introducing bugs.

Many of these advantages are rooted in the fact that **debugging, monitoring and logging on the map-reduce cluster is tedious**. Hadoop 0.18.3 offered a rather complete web interface for a running job to read text logs on slave nodes, per task. However, the development "loop" time, i.e. the time between modifying source code, compiling and deploying the application, running a session, and obtaining logs, lied at about 2 or 3 minutes per cycle. This is long compared to development on a single processor, where such time is usually reduced to seconds. (Despite major advancements since this research was carried out, debugging on a cluster remains much more complex and time-consuming than on a single machine). Causes for this state of affairs are numerous: for instance each worker's Hadoop configuration still requires effort (such as passphrase-less SSH on localhost, customising logging into the distributed filesystem HDFS), while the reference implementation instead is a standard

[25]The reference implementation must be in the same language (Java in our case) as the map-reduce implementation, to avoid bugs introduced while porting from one language to the other. In our case, the research work started by porting the Matlab implementation used for Van Gael et al. (2008) to Java, using Mahout linear algebra primitives. This phase proved easy.

Java program with a `main` method, which supported all debugging functions of the IDE (integrated development environment) (Eclipse), such as step-by-step execution or variable inspection.

These remarks relate to debugging code written by the algorithm developer, not platform or library bugs. As a matter of fact, our research project was affected by two **system-level AEMR bugs** which we uncovered. The first bug consisted of gradually slowing down each iteration's execution time, and manifested itself as a memory leak diagnosed by an Amazon system engineer. This is disturbing because JVMs are restarted between MR jobs, and was attributed to the high number of intermediate files produced when all Gibbs steps are implemented as MR jobs. The second bug consisted of a sudden crash of the master node after several thousand iterations, leaving only a short error message. This problem had rarely been encountered before, and had no resolution known to the Amazon team which accompanied the project. Both bugs were unsolved, and blocking for a large-scale deployment, when this project ended. These two issues demonstrate the challenges linked to working with cutting-edge systems such as AEMR and Hadoop.

2.3 Experiments

The core motivation behind these IHMM PoS tagging experiments was to run a Bayesian non-parametric method on a large amount of data, in a distributed setting, in the hope of observing continuously increasing NLP performance as more data was added, along the lines of n-gram language model results using very large corpora (Brants et al., 2007).

During the experimentation phase, an unexpected hurdle was met, however, in the form of large startup overheads for each Gibbs sampling step. This meant that running the MCMC chain for a time long enough to obtain NLP-relevant results was practically out of reach with the established Hadoop infrastructure. As a consequence, the focus of experimental evaluation shifted from NLP performance to the scaling behaviour of the IHMM MCMC inference algorithm[26].

2.3.1 Algorithm and data

The implemented algorithm, as described in the above section, was the same in all settings. In particular, the initial value for K was fixed to 100, a value to which the state cardinality converges as reported in Van Gael et al. (2009).

In all our experiments, the datasets were derived from the Wall Street Journal (WSJ) part of the Penn Treebank, which is one of the standard corpora used in NLP

[26]This section was adapted from, and has been previously published as, Bratières et al. (2010b).

research. It consists of 1 million tokens of financial newswire text and it has been labelled manually with PoS tags.

We extracted subsets of the WSJ dataset of sizes 1e3, 1e4 and 1e5 tokens (words). Together with the full dataset of 1e6 tokens and a dataset with all sentences duplicated 10 times (1e7 tokens) we are covering a large range. Note that the dataset with 1e7 is not interesting from a NLP point of view as it consists of duplicated data; nonetheless the computational analysis remains valid.

As a sanity check, we evaluated the output of our distributed implementation on the 1e5 subset of the WSJ and the performance in terms of VI (as discussed in section 2.1.2) was 4.5 bits, roughly equivalent to the ones achieved by the parallel implementation of Van Gael et al. (2009). It must be noted that these scores are not strictly comparable due to differences in the dataset size.

To investigate the scaling behaviour of the training algorithm, the performance indicator used in our result reports below consists of the duration of a Gibbs sampling iteration (in seconds), averaged over 10 Gibbs sampling iterations.

2.3.2 Configurations

parallel is an implementation of the IHMM in .Net which uses multithreading on a 4-core 2.4 GHz machine with 8GB of RAM. The implementations stems from Van Gael et al. (2009).

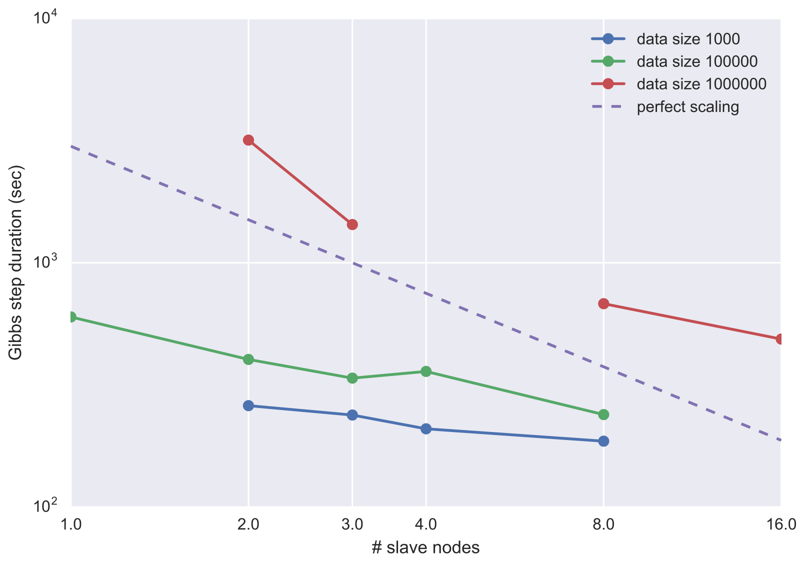
hadoop-1 is an implementation of the IHMM on Hadoop, where each step of the Gibbs iteration, as enumerated in appendix A and table 2.1, is implemented as map-reduce. “Each step” means each operation which scales with the number of data points N, the number of states K or the size of the vocabulary V. There are nine such jobs per iteration, and MR0 is run only once after deployment.

hadoop-2 is like *hadoop-1*, except that only the most CPU-intensive step, namely the dynamic program in MR5, was implemented as map-reduce. The other steps were programmed as for-loops and carried out on the master node. *hadoop-2* runs are duplicates of each other and illustrate the variance of the measurements.

hadoop-3 is exactly like *hadoop-2* from the software point of view, but used different hardware, as described below.

The Hadoop experiments were implemented in Java using the Hadoop map-reduce library. They ran on Amazon’s Elastic MapReduce computing cloud. For *hadoop-1* experiments, they ran on clusters of different sizes: each cluster had one master node (the job tracker, in map-reduce terminology) and one or several slave nodes: 1, 2, 3, 4, 8 or 16 depending on the experiment. For *hadoop-2* experiments, the cluster size was kept constant: one master and one slave node. For both *hadoop-1* and *hadoop-2*, we used nodes of the Amazon “small” type (marked as S in table 2.2), i.e. 32-bit platforms with one CPU equivalent to a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor, 1.7

Figure 2.3.1: *hadoop-1* experiments for selected data set sizes, with varying numbers of slave nodes (x-axis logarithmic). If total computational cost of an experiment scaled linearly with the number of nodes (which is the optimum when distributing computation over a cluster), the lines would run parallel to the dashed line. However, they are more horizontal; this means that the Gibbs step duration does not decrease as much as desired.



GB of memory, with 160 GB storage. For *hadoop-3*, to discover the cluster size needed to outperform the parallel setting, we resorted to Amazon “extra large” nodes (marked as XL in table 2.2), 64-bit platforms with 8 virtual cores, each equivalent to 2.5 times the reference 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor, 7 GB of memory, 1690 GB of storage.

In our experiments, the reducers perform very fast tasks, if any at all, while the bulk of the computation is devoted to the mappers. When there are any reducers (i.e. only in *hadoop-1*), they are distributed on the slaves nodes just like the mappers.

2.3.3 Results

Figures 2.3.1, 2.3.2 and 2.3.3 represent the duration (in seconds) of a single Gibbs iteration across different settings against the range of data set sizes described above, in section 2.3.1. The raw data used for the plots is in table 2.2. In the plots, each point marker represents an experiment with a given software and hardware setting, and a given data set size. The results matrix is sparse due to the fact that not all combinations (setting, data set size) were run during the original experiments. Given the dissatisfying results of the experiment, as explained below, no further effort was

Figure 2.3.2: All *hadoop-2-{a...f}* experiments are duplicates of one another. This demonstrates that there is little variability between runs.

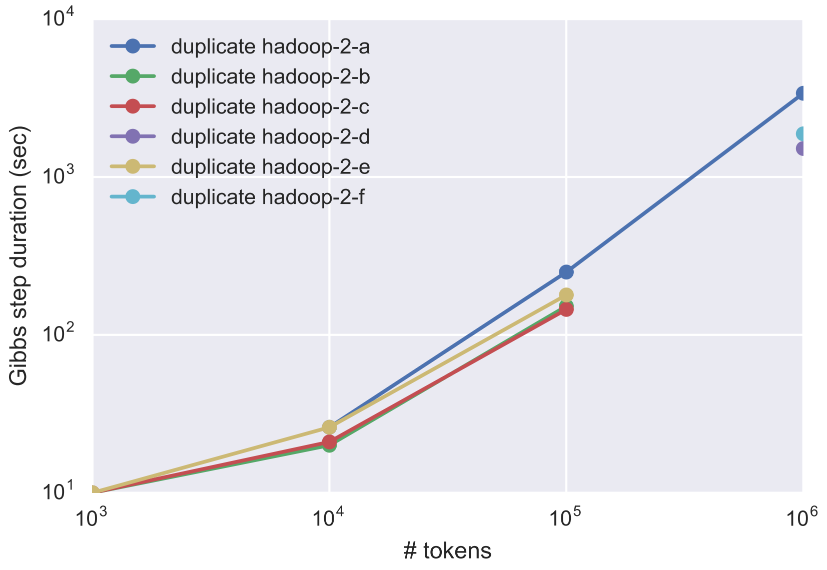
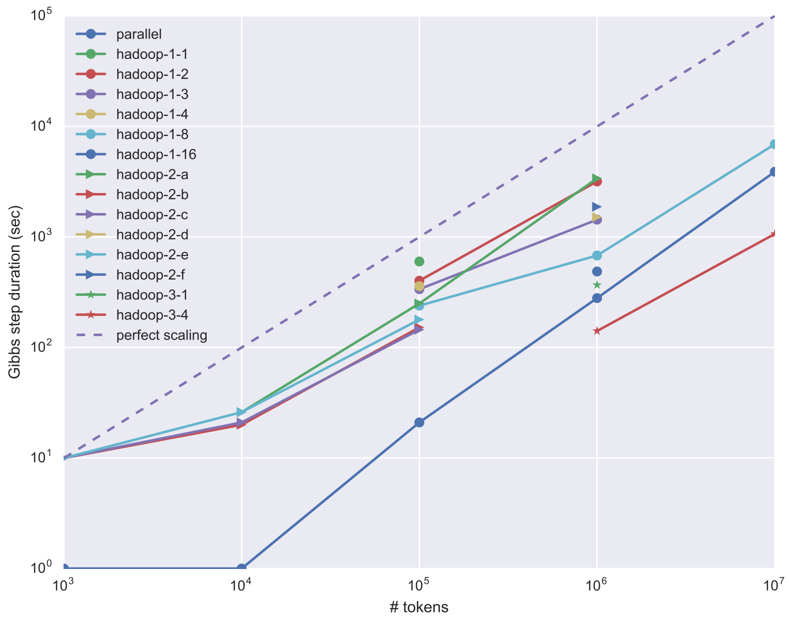


Table 2.2: Duration of a single Gibbs sampling iteration, in seconds, according to the experimental configuration used.

data size	parallel	hadoop-1-1	hadoop-1-2	hadoop-1-3	hadoop-1-4	hadoop-1-8	hadoop-1-16	hadoop-2-a	hadoop-2-b	hadoop-2-c	hadoop-2-d	hadoop-2-e	hadoop-2-f	hadoop-3-1	hadoop-3-4
1 000	1		260	238	209	186			10	10		10			
10 000	1							26	20	21		26			
100 000	21	600	402	337	359	239		250	152	145		179			
1 000 000	280		3179	1434		680	487	3390			1515		1877	368	141
10 000 000	3893					6910									1066
#slave nodes	n/a	1	2	3	4	8	16	1	1	1	1	1	1	1	4
AEMR machine	n/a	S	S	S	S	S	S	S	S	S	S	S	S	XL	XL

Figure 2.3.3: All experiments in the same plot. The general scaling trend follows that of the parallel setup. Increasing data size even further than what was tested here, we expect that *hadoop-1* and *hadoop-2* setups will become faster than the *parallel* setup. However, the point where e.g. lines for experiments *hadoop-1-8* and *parallel* intersect seems several orders of magnitude above present experiments. The *hadoop-3* experiments demonstrate a setup which beats the *parallel* setup, using distribution and more powerful slave nodes.



invested into filling the gaps.

log-log representation implies that an increase in computing cost linear with the number of data points should be reflected in a line. This is indeed the general trend of all experiments. The initially flatter slope of some curves reflects the overhead of parts of the algorithm, or parts of the distribution process, which do not scale linearly with the data size.

The *parallel* implementation scales almost perfectly with data size, but cannot accommodate data sizes beyond those shown, i.e. from 10 million data points onwards, because it is entirely memory-based (and therefore incurs no disk I/O costs). It is therefore necessary, for large datasets, to resort to the distributed implementations.

hadoop-1's performance does not scale well with the number of nodes. This implementation, where each Gibbs iteration contains 9 map-reduce jobs, is apparently badly suited to a Hadoop implementation because of the heavy overhead each map-reduce job incurs: about 20 seconds (with almost no data to move) notwithstanding the number of nodes it runs on.

hadoop-2's performance is much better than that of *hadoop-1*, because it contains only one map-reduce job, and does not suffer from the large overhead effect. It therefore scales well with the amount of data.

The isolated *hadoop-3* experiments used hardware which is comparable with the *parallel* experiment, and demonstrate a speed-up when running from a cluster of 4 machines (experiment *hadoop-3-4*).

Overall, none of the *hadoop-1* and *hadoop-2* implementations were faster, in absolute terms, than *parallel* for the data sizes demonstrated here. It is true that *parallel* experiments used a more powerful CPU than the *hadoop-1* and *-2* experiments, but the bulk of the performance gap is due to the overhead incurred by individual map-reduce jobs.

Adopting a broader perspective, this is also the main takeaway from these experiments: despite demonstrating some scaling speed-up, the overhead due to each MR job does not scale, and prevents running MCMC chains of length typical of machine learning applications, with many thousands of iterations. We now discuss this issue more thoroughly.

2.4 Iterative map-reduce

The initial goal of this project was to reproduce the experiments of Van Gael et al. (2009) on a distributed map-reduce platform, and then observe near-linear acceleration as the number of workers was increased. An important result of our research shows that this was not the case with typical platforms such as Hadoop and Amazon Elastic MapReduce.

The crucial issue, not widely acknowledged at the time of our experiments, proved to be running map-reduce iteratively. In this section, we offer insights into how this issue came to be understood in the years since, and how it lies at the root of the progressive demise of map-reduce in favour of distributed computing frameworks which support not only iterative flow control, but general computational dependency graphs.

It is legitimate to wonder "if we were to restart this project today, what should change?". A fresh look on continuation plans made at the time (involving the Twister platform), and an analysis of existing options today (section 2.4.2) leads us to propose Apache Spark as a platform of choice today[27].

2.4.1 Understanding of the issue in the community

Iterative map-reduce is challenging. The core issue is that map-reduce (and the Hadoop implementation is no exception) has no provision for loops; operators are limited to map, shuffle, and reduce. Iteration must be controlled by an external driver program which in turn calls the map-reduce framework.

This entails a number of performance-degrading consequences:

- job startup costs ("tens of seconds", confirms Lin (2012); we found a value of 10 to 20 sec): the machine processes embedding map and reduce operations are not assumed to pre-exist the map-reduce job, and are restarted from scratch. In Hadoop, this implies restarting the JVM, loading libraries, configuring the jobs.
- data repartitioning: in our case, all the data fits on every worker node, and we are exempt of this issue, but in general, data needs to be repartitioned and transmitted to the mapper or reducer. In iterative situations, in many cases, the data assigned to a mapper stays the same across iterations.
- job synchronicity: the slowest reducer ("straggler") determines the duration of the entire map-reduce job. In an iterative setting, this induces poor cluster usage since the faster nodes are idle while the straggler finishes. This situation persists in synchronous iterative algorithms such as ours, but not in asynchronous settings, such as asynchronous MCMC (cf. section 2.5).

Algorithms which can be distributed with Hadoop map-reduce include those with tens or at most hundreds of iterations, such as expectation-maximisation, Lloyd's algorithm for k-means, iterative graph algorithms like PageRank, optimisation algorithms such as gradient descent. With MCMC algorithms, easily running into the

[27]It is not clear that moving back to a parallel setting and using GPUs would help, given that (a) the HDP-HMM training algorithm is inherently sequential, and that (b) the length of matrices we manipulate is at most the number of states (on the order of a few hundred at most), which is small for the current GPU generation.

thousands of iterations, the problem of incompressible per-iteration computational cost is exacerbated.

We published our findings, not yet widely acknowledged in the distributed machine learning community, in (Bratières et al., 2010a,b). In the following years, these findings were echoed in the literature, to the point that they became commonplace in discussions about Hadoop and map-reduce. Researchers and practitioners ran into the same issues time and again, and agreed with our findings. To present the gradual understanding of the issue in the community in context, let us go through a couple of examples of such statements, in chronological order.

- Panda et al. (2009), section 6 "Engineering issues": *[...] we encountered several unanticipated challenges. First, because MapReduce was not intended to be used for highly iterative procedures like tree learning, we found that MapReduce start up and tear down costs were primary performance bottlenecks.*[28]
- Bekkerman et al. (2012), section 12.3.1, page 246: *[...] MapReduce is suitable for non-iterative algorithms [...]*
- Agarwal et al. (2012), section 2: *However, the [MapReduce] abstraction is rather ill-suited for machine learning algorithms [...], because it does not easily allow iterative algorithms, such as typical optimization algorithms [...]*
- Rosen et al. (2013), abstract: *It is now widely recognized that while MapReduce is highly scalable, it suffers from a critical weakness for machine learning: it does not support iteration.*
- Sculley et al. (2015), section 5, "Abstraction debts": *Indeed, one of the few areas of broad agreement in recent years appears to be that Map-Reduce is a poor abstraction for iterative ML algorithms.*

The progression of certainty in these statements is telling of the increasing awareness of the issues surrounding iterative map-reduce.

2.4.2 Fixing iterative map-reduce: today's perspective

Answers to the problem, with special attention to Hadoop, came in different guises.

At the first level, one can configure Hadoop as best as possible to cope; this was our approach, as illustrated above in section 2.2.5.

At the second level, one can try to turn the problem into an non-iterative algorithm, an approach advocated by Lin (2012). We did not follow this route as we were interested in a specific MCMC algorithm for IHMM training.

[28]To work around the issue during the course of their project, the authors applied extra task scheduling tricks : jobs are started before they are needed, and stay idle until they are required. While this shortens the runtime of each individual job, this technique does not reduce the runtime of the entire loop, as it merely shifts workload in time.

At the third level, one can port the code to another framework than Hadoop, which inevitably lacks the traction, maturity, community expertise around Hadoop. We can divide such frameworks in frameworks which maintain the map-reduce framework, on the one hand, and frameworks based on more general or different computational concepts, on the other hand. The former are becoming increasingly marginal, while the latter started developing in the footsteps of map-reduce and Hadoop, and are still expanding in scope, performance and supported platforms (e.g. nowadays GPUs are becoming a relevant target platform). We now examine each of these two categories in turn.

2.4.2.1 Adapted map-reduce frameworks

HaLoop (Bu et al., 2010) modifies the Hadoop code to introduce an iteration control to the Hadoop master node. It avoids data transfer latency by leaving data on a node across iterations when possible, by caching. It distinguishes data which changes across iterations from data which does not. Peregrine[29], a Java open-source project which ran until 2012 before being abandoned, was addressing the same issues, with design ideas close to Pregel (cf. below).

iHadoop (Elnikety et al., 2011) addresses the task scheduling issue of iterative map-reduce (synchronicity exposes to straggler latency) by introducing asynchronous iterations and tentative execution, where possible: nodes which are done with their computation start the next iteration before all nodes are finished. This measure does not apply when global parameters must be updated before the next iteration.

iMapReduce (Zhang et al., 2012) and i^{2}MapReduce (Zhang et al., 2016b), deal with this same issue. While iMapReduce targets use cases with one-to-one or one-to-all mapper-to-reducer correspondence, i^{2}MapReduce lifts this restriction. Both cache loop-invariant data and exploit data locality.

The system described by Rosen et al. (2013) introduces loops as fundamental language constructs. Loops contain chains of map-reduce and mapper-only[30] operators, where the first operator is constrained to take output generated by the last operator of the chain, to make iterations possible. An optimiser deals with partitioning the data and shuffling intermediate results.

At the time of our project, we had set our eyes on Twister (Ekanayake et al., 2010) [31], a memory-stream-based map-reduce framework under development in 2009-2010 at the University of Indiana, as part of a PhD thesis (development has ceased since). To support iterative constructs, job restarts are avoided, and data is kept alive in memory between iterations. Like HaLoop, task scheduling favours data locality, and caches

[29]`http://peregrine_mapreduce.bitbucket.org`

[30]named "sequential" in the paper

[31]The authors, in a smart move in the light of later developments documented section 2.4, had secured the domain name `http://www.iterativemapreduce.org/` in November 2009 already.

static data. Twister is much more lightweight than Hadoop. It offers no distributed file system (data broadcasting and collection are command-line operations), no failure recovery, no web-based monitoring interface. It never had a broad user base, and was much less documented and tested than Hadoop. However, at the time, we had direct access to Twister's developers, it ran on Amazon EC2 (which meant we could export at least some of our experience), it is a Java application (which meant we could re-use the reference implementation), it had demonstrated high performance in tests, due to its simplicity it would have been easier to fix platform errors than with Hadoop, and finally, it seemed deliberately designed for iterative map-reduce algorithms.

I was awarded an Amazon AWS Education grant of EC2 credits to implement my project on Twister, but the project was abandoned, considering that this would have made more of a distributed computing and software engineering project than a machine learning one, and was therefore ill-adapted to the purpose of this thesis.

2.4.2.2 Beyond map-reduce

In the second category, several platforms for distributed machine learning, based on different computational concepts than map-reduce, have appeared.

For instance, Pregel (Malewicz et al., 2010) is a message-passing, graph computing operator. It implements the bulk-synchronous parallel model (Valiant, 1990). In this model, the graph topology is central; computations consist of aggregating vertex-incoming messages, and distributing outgoing messages. Implementations like Giraph or Apache Spark's GraphX work optimally when the graph structure is kept in memory. The popular injunction to the programmer working with graph-centric platforms is "think like a vertex!" — underlining that they place the burden of formulating the algorithm of interest as message-passing on the programmer. Clearly, certain classes of algorithms can be more easily formulated as vertex-oriented message-passing algorithms than others. While implementing iterative map-reduce on Pregel is no doubt possible (by seeing mappers and reducers as vertices, and taking care of the messages they need to pass to each other), it is not a natural programming model.

It is worth mentioning that as of this writing, Hadoop has outgrown its 2010 description of a map-reduce implementation. It is now a complete stack, with important pieces such as its file system HDFS, the resource management and scheduling platform Yarn, and a large ecosystem of other libraries developed under the aegis of the Apache Foundation. Similarly, Mahout has evolved from a collection of machine learning algorithms using Hadoop as its map-reduce backend, to deprecating the map-reduce concept[32] in 2014 and offering an R-like domain-specific language running under a variety of backends (Apache Spark, H2O, Apache Flink).

Frameworks centring on a distributed implementation of computational graphs,

[32] `https://issues.apache.org/jira/browse/MAHOUT-1510GoodbyeMapReduce`

like Microsoft Dryad (Isard et al., 2007), or the open-source Apache Spark (Zaharia et al., 2010) appeared after map-reduce had registered some successes, but also had started to show shortcomings, specifically for iterative tasks as outlined above, and after the need for more elaborate distributed frameworks had become apparent.

These frameworks function in similar ways: the programmer starts by describing the computations to perform on the data in a high-level idiom, which may be a general purpose programming language, like Scala (Spark) or a .Net language (Dryad), or a domain-specific language (LINQ for DryadLINQ (Yu et al., 2008)). An optimisation engine casts the execution jobs as a directed acyclic graph, and takes care of distributing the computation, which can be configured by the programmer to some extent. Task-specific operations are supplied as user-defined functions.

The dominant platform for distributed machine learning now is Apache Spark (Zaharia et al., 2010), which originated in 2009 at the University of California at Berkeley and was open-sourced in 2010. Spark uses immutable data containers as base objects[33], which should fit in memory for optimal execution. It offers operators for data transformations (map, filter, sample, union, group) and aggregation actions (reduce and variants). It also incorporates a graph computation library, GraphX, mentioned above. Spark has proven very popular and enduring, has seen a large community grow around it, has matured to its version 1.0 in 2014 and 2.0 in 2016, and is probably the platform we would choose today if we were to re-implement these experiments. In addition, Spark has been adopted as a managed service by many commercial cloud offerings.

2.5 Review of the state of the art for distributed probabilistic inference

We present a review of the state of the art of distributed inference for probabilistic models, with special consideration to NPB models and MCMC methods. This section situates our research project within a larger context, and illustrates the research landscape both before and after our research was conducted. We will first discuss schemes in which a single MCMC step should be distributed, before moving on to approaches which run several MCMC chains in parallel. The last part of our discussion centres around clustering models, both parametric and Dirichlet-process-based, and finally the IHMM.

Many ideas for distributed inference in probabilistic models are actually generic to distributed computing in general. This is the case of the *synchronous vs. asynchronous* distinction, common to inference schemes which can be formulated iteratively

[33]named “resilient distributed datasets” (RDD) in Spark parlance; these are wrapped by DataFrames in Spark 2.0, which support streaming data, and appear like RDD with unbounded rows

(sampling approaches, variational approaches, exact or approximate message passing approaches).

For example, consider an MCMC chain in which every MCMC step is computationally demanding and needs to be distributed. This could be the case of Gibbs sampling in a mixture model with K clusters and N data points. Typically, we will want to map over the dataset, because N is large, and the dataset may not fit on one single worker node; but in general terms, the advisable distributed architecture is dictated by the specifics of the task, including characteristics of the data, and more specifically the dimension or parameter which can be iterated or mapped over.

The most natural algorithm in this example consists of iterating two Gibbs steps: in the first step, the algorithm maps over data points in worker nodes, and updates the cluster assignments; in the second step, it collects the cluster parameters on the master node based on current assignments. At the end of the iteration, the cluster parameters are synchronised, which is why the scheme is characterised as synchronous. As a drawback, the master node must wait for the iteration's slowest node each time, which introduces a bottleneck. In addition, at larger scales, the central aggregation step may become infeasible on a single machine.

Asynchronous approximate methods were developed to counter these effects. Their approximation can often be characterised as treating a local subgraph's statistics, parameters, or messages as if they were the entire model's, so that algorithms do not maintain a central copy of the parameters, but decentralised copies. In block-synchronous schemes, blocks stay synchronous but the whole model does not. In window-synchronous schemes for MCMC, synchronicity of parameters is preserved "in the long run" over several iterations, by allowing a forward or backward lag between samples from different blocks, with regular synchronisation steps. While asynchronous schemes pose a convergence issue in general, in the MCMC case this surfaces as the fact that the stationary distribution is no longer the true posterior. Despite this, (Asuncion et al., 2008; Newman et al., 2007) showed that, at least in certain cases, statistics of interest converge rapidly to a robust estimate.

As mentioned, the synchronous vs. asynchronous distinction carries over outside of MCMC inference. Synchronous and asynchronous approximate belief propagation is distributed over very large undirected graphs in (Gonzalez et al., 2009), or factor graphs in (Stern et al., 2009). Variational expectation-maximisation is also amenable to distributed inference for mixture models (Kowalczyk and Vlassis, 2005), and for LDA (Nallapati et al., 2007). The E step is a natural candidate for parallelisation, and Wolfe et al. (2008) investigated how to distribute an M step which would exceed the capacity of one compute node.

To conclude the discussion of the case where a single MCMC step needs to be distributed, some approaches[34] are generic to Metropolis-Hastings, in contrast to meth-

[34]see Green et al. (2015), section 2.7, for a review

ods which distribute over a model-specific parameter. Brockwell (2006) suggests tentatively precomputing the next Metropolis-Hastings step without waiting for the current step to terminate, by tentatively branching on the outcome of the accept/ reject step; this procedure can be conducted several steps in the future by maintaining a tree of outcomes. Calderhead (2014) suggests drawing several proposals instead of just one, considering the index of the replica as an auxiliary variable. These approaches take advantage of a parallel architecture to accelerate single MCMC chains, but since they waste computation, they do not reduce the computational cost of each sample.

On the other hand, if the single MCMC step is fast enough, we can naïvely run *parallel independent MCMC chains* on the full data, and combine their samples for prediction; this approach is very natural, and requires that the chains mix well independently. Similar approaches, inspired by particle filtering, motivated Strid (2008); Brockwell (2006); Wood and Griffiths (2007).

Another strategy aims at increasing the number of samples produced per unit of time on the entire distributed platform. Consensus Monte Carlo (Scott et al., 2016)[35] and other methods cited by Green et al. (2015) suggest rewriting the posterior

$$p(\text{parameters}|\text{data}) = \prod_{r=1}^{R} p_r(\text{data}_r|\text{parameters})p(\text{parameters})^{1/R} \tag{2.5.1}$$

where data $= \bigcup_{r=1}^{R} \text{data}_r$ is a partitioning of the data, and we split the prior over the partitioning. Chain r runs on data$_r$, each p_r is estimated separately and combined as a factor. A similar idea is explored by VanDerwerken and Schmidler (2013): instead of partitioning the data, the parameter space is partitioned. The chains each explore a portion of the parameter space; this raises the practical difficulty of partitioning the space in the first place.

We now change our point of view and discuss distributed inference for specific models, starting with *clustering models.*

Indeed, distributed inference for latent Dirichlet allocation (LDA) and its nonparametric counterpart, hierarchical Dirichlet Process mixture modelling, had seen considerable interest before our research. Wang et al. (2009) implemented a synchronous approach for LDA, while (Asuncion et al., 2008; Newman et al., 2007) had synchronous and asynchronous schemes. Asuncion et al. (2008) worked on asynchronous MCMC for hierarchical Dirichlet Processes. The most compelling work on distributed inference for NPB models before 2009 was Huang and Renals (2007): language modelling requires large text corpora to cover as many word combinations as possible, and is therefore an application of interest for distributed machine learning; Huang and Renals (2007) showed how a hierarchical Pitman-Yor language model trained over a computing cluster outperforms traditional language modelling techniques (n-grams

[35] first published as Scott et al. (2013)

with accurate back-off schemes).

The period since 2009 is covered by Qiu et al. (2014), which in its section 2 reviews the different implementations proposed in the meantime[36]: they vary in inference method, be it variational learning, or synchronous and asynchronous MCMC, and in the underlying backend, be it multi-core CPUs, map-reduce, MPI, GraphLab or GPUs.

A related line of research (Lovell et al. (2012); Williamson et al. (2013); Dubey et al. (2014); Williamson et al. (2015)) focuses on parallel MCMC implementations of DP-based non-parametric Bayesian models, building on an auxiliary variable formulation (and accompanying independence properties) of DP models. Williamson et al. (2013) introduces the auxiliary variable formulation used in the subsequent papers. The key remark is that a DP mixture model is a Dirichlet mixture of DP models. Using this equivalence, the mixture assignments in the latter model can be made an auxiliary variable. Hence, in a Gibbs sampling algorithm, conditional on the mixture assignments, one part of the computation is local to data assigned to that mixture; the other part of the computation is global and requires resampling the mixture assignments (and potentially moving around the data corresponding to newly assigned data points, in case the entire dataset does not fit on each worker). The cited references extend this construction to HDP, PY and HPY processes. An issue with this approach, analysed in Gal and Ghahramani (2014), is that the allocation of data to worker nodes is unbalanced and may make the algorithm impractical. (In fact, the issue is more general and applies to distributed inference overall, suggesting several questions: how must data and computation be mapped onto the cluster topology? Instead of the naïve "uniformly at random" scheme, should data move adaptively across the cluster to form similarity blocks which enhance the approximation? Are there smarter distribution schemes?)

Finally, we return to the IHMM.

To our knowledge, there have not been other efforts to parallelise the IHMM since the research described in this chapter. Interacting particle Gibbs MCMC (Rainforth et al., 2016), a variant of particle Gibbs MCMC used in Tripuraneni et al. (2015) as a building block for IHMM inference, can, in essence, run different particles on parallel workers; yet it is hard to imagine applying this to the IHMM, as it would require parallelising the state sampling step of each sequence. In applications with many sequences like ours, the computational cost of a single such step vanishes when compared to the overall computational cost, therefore parallelising it would presumably bring more overhead than savings.

There have been recent developments in inference methods for the IHMM since the beam sampling approach of Van Gael et al. (2008), but these have not been distrib-

[36]It ignores Zhai et al. (2012), which is based on variational inference. Note that by picking Apache Mahout's very basic inbuilt LDA implementation, it chooses a weak opponent in its benchmark; the choice is presumably motivated by the fact that it is another map-reduce VB algorithm.

uted. Rather than attempting to review them here, we will refer the interested reader to the engaging discussion in Tripuraneni et al. (2015), section 1 and appendix C, for sampling schemes, and Zhang et al. (2016a) for variational schemes.

2.6 Conclusion

This work's main contributions are the following:

- the Gibbs sampling training algorithm for an important non-parametric model, the infinite HMM, was adapted to the map-reduce distributed computing paradigm
- in particular, a Java implementation based on Mahout and Hadoop was produced
- the distributed implementation was compared experimentally to a parallel implementation
- the lessons learnt during development, deployment and debugging were analysed
- the current state of the art on iterative map-reduce and distributed MCMC methods was summarised, addressing the issues encountered in this project

Can we characterise to which class of problems the distributed inference strategy exposed in this chapter is applicable?

We distribute single Gibbs sampling steps within each MCMC iteration. In particular, the most computationally intensive step, referred to as MR5 above, which consists of sampling a new state sequence, is data-parallel. Therefore, while the order of Gibbs sampling operations and MCMC both computationally sequential due to their dependence on previous computations, the first level at which we can distribute computation is within single Gibbs steps. Our strategy is therefore applicable to problems in which individual MCMC iterations are computational bottlenecks, but where the mixing behaviour is good, that is, where we do not require on the order of millions of MCMC iterations. Future work could explore other strategies reviewed in section 2.5, such as running slow MCMC steps on a single node each, but drawing several new posterior samples (e.g. by using different random seeds) starting from a select previous sample; this procedure would be beneficial once the chain has mixed, to obtain more posterior sample than from running a simple chain.

To our knowledge, no other work has attempted to apply the IHMM to large-scale datasets on a distributed platform since our work. This research question thus remains unanswered. It is worth considering a follow-up project in which the IHMM training procedure would run on Apache Spark on a public cloud, running widespread operating systems and software, on publicly available large-scale text data. Our Java implementation could be partially re-used.

For this follow-up project for the IHMM, should the application still be PoS tagging? If so, considering the increase in computational capacity of cloud platforms, we would consider moving from a corpus with one million tokens such as the WSJ to e.g. English Wikipedia (currently 3 billions of words[37]).

Without looking into alternative applications at this point, let us reformulate this question in two ways: firstly, is it fair to consider the IHMM clusters as PoS tags at all? Secondly, assuming that IHMM-based, or more broadly, HMM-based models are viable for PoS induction, as the NLP community seems to do, is the large-scale application of the IHMM promising for this purpose?

From a fundamental point of view, it is surprising that while PoS tags are in large part just syntactic classes (determinants, prepositions, modal verbs) which determine long-range structure, current unsupervised PoS tagging models are limited to very local dependency structures such as the bigram conditionals found in HMM. That is, syntax beyond the immediate neighbours is not taken into account, and certainly no hierarchical (parse tree) information is. This prompts the suggestion that IHMM states in our experiments might not be best thought of as PoS tags. Indeed, they are probably better characterised as clusters of words with similar distributional properties based on first-order Markov dependencies. In fact, an early language modelling paper, Brown et al. (1992), which used several methods to cluster words into groups based on their distributional properties, found many different types of classes and mixes of classes: semantic, topical, syntactic, genre-related, co-occurrence, natural classes ("January February etc.", "daily monthly quarterly periodic etc."). Interestingly, a benchmark (Christodoulopoulos et al., 2010) shows this model to perform better than others, when used to bootstrap a prototype-based unsupervised PoS induction model. Considering the types of clusters which are found by the Brown et al. (1992) model, the IHMM could be considered a class-based language modelling device, with a Markov constraint on class transitions, and with in-class emission probabilities defined (in our current model) by a multinomial observation probability. These remarks all put the use of IHMM for unsupervised PoS learning into question.

Considering the second question, the same benchmark (Christodoulopoulos et al., 2010) suggests that the IHMM would not fare particularly well among available methods[38] on the standard corpus for PoS tagging, the Wall Street Journal corpus.

All this seems to discourage from conserving the PoS tagging task for a rerun of this project; and to turn to other tasks based on sequence observations, such as activity recognition or change point detection, for example. On the other hand, it

[37]`https://en.wikipedia.org/wiki/Wikipedia:Size_comparisons`

[38]This study did not run the IHMM code, unfortunately, cf. its footnote 2. However, results reported on the IHMM do not seem to top the benchmark.

might well be that the competing methods, all of which are parametric, have seen their inductive bias engineered so as to perform well on currently accessible corpus sizes, while non-parametric models still have potential on larger corpora, at scales where the performance of parametric models saturates.

Chapter 3

Discriminative models for structured output prediction

3.1 From generative to discriminative modelling

The HMM is an example of a *generative model*, i.e. it models the full joint distribution $p(x, y)$, where x is generically some input, and y is the output. Generative models have the advantage that, because they model the input, they can "impute" missing data naturally, and can be used for simulation, by sampling from the joint distribution.

The remainder of this thesis is concerned exclusively with prediction and supervised training, with models of the form $p(y|x)$. This type of model is called a *conditional model*, as it is conditional on the input; it is also called a *discriminative model* because it discriminates between outputs, and, in the case of classification, can be thought of as modelling the boundary between classes. A generative model would require $p(x)$ so that we can produce $p(x, y) = p(y|x)p(x)$. Yet for prediction, the part $p(x)$ of the model is useless. This follows from decision theory, when considering the choice of a decision rule given some observed input x: if we adopt the Bayesian approach, we will want to minimise the posterior expected loss $\mathbb{E}_{y|x}[\text{loss(predicting } y \text{ under state of nature } x)]$[1]. As a consequence, we only need access to $p(y|x)$.

At this point, to provide some context, we can attempt to give a rather general catalogue of predictors commonly used in supervised machine learning.

1. The most general type of predictor directly maps $x \mapsto y$, such as linear binary discriminant functions (of the form $x \mapsto \text{sign}(w^T x)$, with $w \in \mathbb{R}$ a weight

[1]This is further motivated by the fact that it corresponds to minimising the Bayes risk $\mathbb{E}_y \mathbb{E}_{x|y}[\text{loss(predicting } y \text{ under state of nature } x)]$, and that Bayes decision rules are admissible. Cf. discussions in Berger (1985), chapters 1 and 4, and Lehmann and Casella (1998).

vector) or rule-based systems, such as association rule learning.

2. Often, the decision is based on the maximisation of a *score function* $s(x, y)$, so that the predictor takes the form $x \mapsto \arg\max_y s(x, y)$; this covers many non-probabilistic approaches to machine learning, such as linear predictors and their kernelised variants, e.g. support vector machines, or voting schemes, e.g. k-nearest-neighbours.

3. Probabilistic approaches typically separate *inference*, i.e. obtaining a *conditional model* for $p(y|x)$, from *decision*, which is the stage where the predictor or decision rule comes into play, and which seeks to minimise the expected loss. This separation is motivated by decision theory, and takes into account the loss function. Often enough, 0-1 loss is used here, so that the predictor becomes the maximum a posteriori decision rule, $x \mapsto \arg\max_y p(y|x)$.
   Discriminative models are therefore the simplest class of probabilistic models.

4. Among probabilistic approaches, we find the *generative models*: they represent $p(y|x)$ as $\frac{p(x,y)}{\sum_{y'} p(x,y')}$, and therefore model the joint distribution $p(x, y)$. As a consequence, they also model $p(x)$.

Dispensing of a density model for x, as discriminative models do, gives a decisive engineering advantage: we can represent x under the most diverse and most relevant aspects, as a rich set of features; it would be hard to specify a distribution for these features. When x is composed of text, for instance, features can be as diverse as: capitalisation, location in a semantic net such as WordNet, membership of application-specific word lists, morphological and inflectional features, features of surrounding words, position in a sentence, or even the output of some other machine learning model, such as a word embedding or the hidden layer representation obtained from a deep neural network.

As an alternative to maximum likelihood training, a popular parameter estimation method for generative models is maximising the *conditional* likelihood $p(y|x)$ over the training set. This parameter estimation method ignores the $p(x)$ term of the distribution, and hence does not appear to maximise the correct likelihood term. Yet it can be defended as an alternative to maximum likelihood parameter estimation by arguing that the asymptotic consistency guarantees offered by e.g. maximum likelihood or MAP training (with the use of the plug-in ML or MAP decision rule) break down in case of model misspecification, i.e. when the family of models under consideration does not contain the true model. This approach is common in speech recognition (Chou and Juang, 2003, chapters 1 to 3), machine translation (Andrés-Ferrer et al., 2008) and many other applied fields, and often called "discriminative training of a generative model"[2].

[2]These fields, specially speech recognition, have seen a flourish of alternative training methods which

Minka (2005) takes issue with this nomenclature and remarks that this procedure actually consists of maximum likelihood or MAP training of a model which is derived from the generative model. Building on this result, Lasserre and Bishop (2007); Lasserre (2008) propose a method to construct hybrids of discriminative and generative models, leading to better predictive performance than either type in isolation.

The debate “discriminative or generative?” received the beginning of an answer with Ng and Jordan (2001), which produced the intuitively appealing conclusion that generative models would fare better in a small-data setting, while discriminative models would obtain higher accuracy with large datasets. This conclusion, however, was put in question by later research (Xue and Titterington, 2008), which does not find a clear cut regime change. Further theoretical research in the comparison and hybrid models (Liang and Jordan, 2008; Xue and Titterington, 2009; Bouchard, 2007), as well as many comparisons of discriminative and generative methods on a given problem, are witness to the continuing interest in this question. A recent incarnation is found in generative adversarial networks (Goodfellow et al., 2014), a neural network method consisting of an adversarial configuration of a generative and a discriminative model, the latter working on the output of the former (here, however, the discriminative problem is auxiliary to the main task).

3.2 A roadmap

While the previous section motivated conditional models from general considerations, we would now like to discuss specifics of such models. Rather than presenting an unordered catalogue of models, we have developed a synthetic, if partial, diagrammatic view of models, their characteristics and relations, which we present here, and use as a roadmap in the remainder of this chapter.

Figure 3.2.1 offers a panorama of discriminative models for the progressively more complex tasks of regression (with output $y \in \mathbb{R}$), binary classification ($y \in \{+1, -1\}$), multi-class classification ($y \in \mathcal{R} = \{1, 2, ...R\}$), and finally structured output prediction ($\mathbf{y} \in \mathcal{Y}$, a structured set, for instance $\mathcal{Y} = \mathcal{R}^T$ for sequence labelling problems).

Each vertex of the graph could typically be illustrated by several models; we will not comment on or name them all here, but instead provide references to a few representative models.

This diagram is necessarily schematic: for instance, by only distinguishing “(fully) Bayesian vs. not Bayesian”, we ignore learning principles: “not Bayesian” covers MAP and ML, but also max-margin learning. In addition, while the “kernel” and “structured” axes refer to properties of the model, the “Bayesian” axis refers to the train-

try to get over the model misspecification hurdle: minimum Bayes risk, maximum mutual information, minimum phone or word error, to name but a few.

ing and inference procedure in probabilistic models, as discussed in section 3.4. As a matter of fact, in the machine learning literature the distinction between model and algorithms, i.e. the training procedure, is often blurred, unfortunately: "CRF" often denotes the CRF model (which we introduce section 3.7) trained using maximum likelihood parameter estimation, implemented with some variant of gradient descent. Similarly, the term "SVMstruct training" often implicitly refers not only to the SVMstruct model, but to the 1-slack, margin rescaling, linear kernel, cutting plane algorithm described in Joachims et al. (2009).

3.3 Illustrating the roadmap: linear regression

We will now illustrate how to move around this roadmap with an example based on linear regression:

- input feature vectors[3] are written $\mathbf{x} \in \mathbb{R}^D$
- outputs are written $y \in \mathbb{R}$
- training inputs $\{\mathbf{x}_1, \dots \mathbf{x}_N\}$ are stacked (in transpose form) in matrix $\mathbf{X} \in \mathbb{R}^{N \times D}$
- training outputs are stacked in $\mathbf{y} \in \mathbb{R}^N$
- test data is denoted by $\mathbf{x}_*, y_*$

We adopt the following model: the output y is a noisy version of a latent variable, with independent Gaussian noise. The latent variable is obtained from a linear model parameterised by a weight vector $\mathbf{w} \in \mathbb{R}^D$: it is $\mathbf{x}^T\mathbf{w}$.

Thus

$$y \sim \mathcal{N}(f, \beta^{-1}) \tag{3.3.1}$$

where we note the inverse variance (a.k.a. precision) β. In this way, $p(y|\mathbf{x}, \mathbf{w}) = \mathcal{N}(y|\mathbf{x}^T\mathbf{w}, \beta^{-1})$.

To build a full Bayesian probabilistic model, we define a prior over the weights, e.g.:

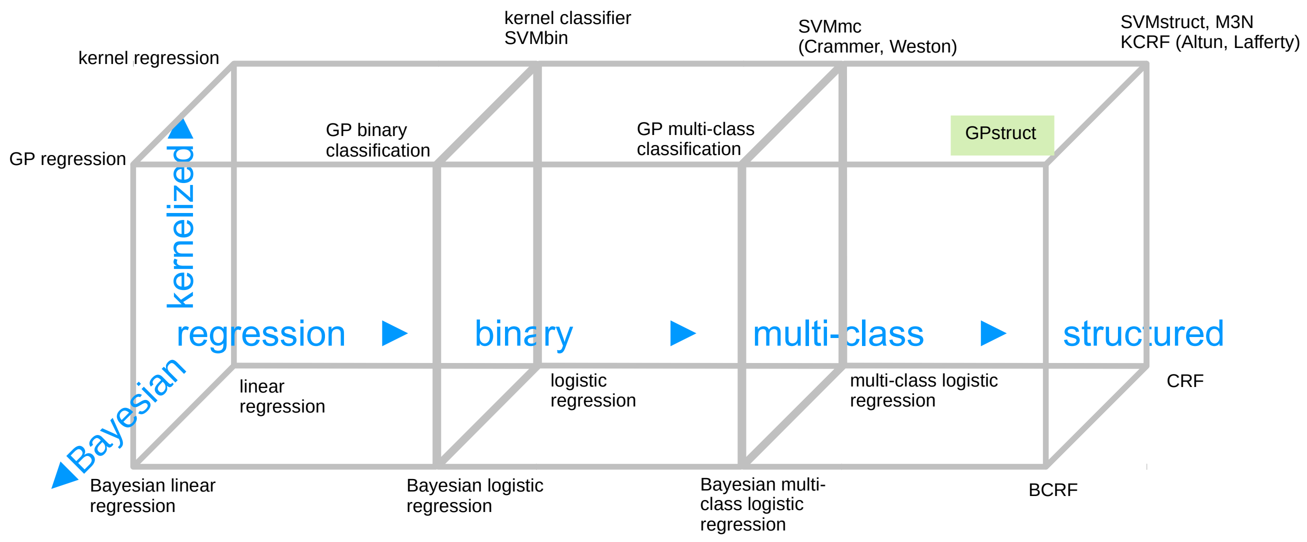
$$\mathbf{w} \sim \mathcal{N}(0, \boldsymbol{\Sigma}) \tag{3.3.2}$$

with the covariance matrix $\boldsymbol{\Sigma}$ symmetric positive definite of shape (D, D).

We wish to obtain the predictive distribution for new, unseen test points. Applying the rules of probability, we can write

$$p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \int p(y_*|\mathbf{x}_*, \cancel{\mathbf{X}, \mathbf{y}}, \mathbf{w})p(\mathbf{w}|\cancel{\mathbf{x}_*}, \mathbf{X}, \mathbf{y})\, d\mathbf{w} \tag{3.3.3}$$

[3] We represent inputs directly as feature vectors here, instead of making the feature extraction function, usually written ϕ, explicit; this simplifies notation. Otherwise we would have to write $\boldsymbol{\phi}(x) \in \mathbb{R}^D$ instead of $\mathbf{x}$.



GP	Gaussian process	Rasmussen and Williams (2006)
GPbin	GP binary classification	Rasmussen and Williams (2006)
GPmc	GP multi-class classification	Williams and Barber (1998)
CRF	conditional random field	Lafferty et al. (2001), and Sutton (2012) for a review
KCRF	kernelised CRF	Lafferty et al. (2004), Altun et al. (2004a)
BCRF	Bayesian CRF	Qi et al. (2005)
M3N	maximum-margin Markov network	Taskar et al. (2004)
SVM	support vector machine	Schölkopf and Smola (2002)
SVMbin	binary SVM	Schölkopf and Smola (2002)
SVMmc	multi-class SVM	Weston and Watkins (1999), Crammer and Singer (2001)
SVMstruct	structured SVM	Tsochantaridis et al. (2005)
GPstruct	structured GP classification	Bratières et al. (2015)

Figure 3.2.1: An overview of some discriminative models in supervised learning. (Author names in the figure are included to distinguish several variants of a model)

where terms cancel because of the conditional independence properties of our model.

How to compute this integral? As a first approximation, we could apply the maximum a posteriori (MAP) learning principle: we approximate the integral by its value in the point $\mathbf{w}_{MAP}$:

$$p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) \approxeq p(y_*|\mathbf{x}_*, \mathbf{w}_{MAP}) \tag{3.3.4}$$
$$= \mathcal{N}(y_*|f_{*MAP} = \mathbf{x}_*^T \mathbf{w}_{MAP}, \beta^{-1}) \tag{3.3.5}$$

To obtain $\mathbf{w}_{MAP}$, we must conduct the following optimisation:

$$\mathbf{w}_{MAP} = \arg\max_{\mathbf{w}} p(\mathbf{w}|\mathbf{y}, \mathbf{X}) \tag{3.3.6}$$
$$= \arg\max_{\mathbf{w}} p(\mathbf{y}|\mathbf{w}, \mathbf{X}) p(\mathbf{w}) \tag{3.3.7}$$
$$= \arg\max_{\mathbf{w}} \beta(\mathbf{y} - \mathbf{X}\mathbf{w})^T (\mathbf{y} - \mathbf{X}\mathbf{w}) + \mathbf{w}^T \mathbf{\Sigma}^{-1} \mathbf{w} \tag{3.3.8}$$

where in the last line we have substituted the log probabilities obtained from equations 3.3.1 and 3.3.2. Setting the gradient to 0 yields

$$\mathbf{w}_{MAP} = (\mathbf{X}^T\mathbf{X} + \beta^{-1}\mathbf{\Sigma}^{-1})^{-1}\mathbf{X}^T\mathbf{y} \tag{3.3.9}$$

so the predictive distribution is

$$p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) = \mathcal{N}(y_*|f_{*MAP} = \mathbf{x}_*^T(\mathbf{X}^T\mathbf{X} + \beta^{-1}\mathbf{\Sigma}^{-1})^{-1}\mathbf{X}^T\mathbf{y}, \beta^{-1}) \tag{3.3.10}$$

This solution is equivalent to a linear regression model with squared loss and an L_2 (a.k.a. *ridge*) regulariser.

3.4 Making the model Bayesian

The "**Bayesian**" axis indicates the passage from

1. a MAP or more generally a point-wise parameter estimate, as the output of the training step

2. to a fully Bayesian training procedure which preserves the uncertainty over the parameters, learns a posterior distribution over them (having specified a prior), and where prediction consists of marginalising out the parameters of the model

To illustrate the difference with the previous section, we now calculate the fully Bayesian solution. Instead of approximating $\mathbf{w}$ to a point estimate, we calculate its posterior distribution:

$$p(\mathbf{w}|\mathbf{X},\mathbf{y}) \propto p(\mathbf{y}|\mathbf{w},\mathbf{X})p(\mathbf{w}) \tag{3.4.1}$$
$$= \mathcal{N}\left(\mathbf{w}|(\mathbf{\Sigma}^{-1}+\beta\mathbf{X}^T\mathbf{X})^{-1}\beta\mathbf{X}^T\mathbf{y}, (\mathbf{\Sigma}^{-1}+\beta\mathbf{X}^T\mathbf{X})^{-1}\right) \tag{3.4.2}$$

To obtain this, we apply the classical formulae for joint Gaussian distributions[4].

Now $p(y_*|\mathbf{x}_*,\mathbf{X},\mathbf{y})$ is obtained as the convolution of two Gaussians in equation 3.3.3, hence

$$p(y_*|\mathbf{x}_*,\mathbf{X},\mathbf{y}) \sim \mathcal{N}(\mathbf{x}_*^T(\mathbf{\Sigma}^{-1}+\beta\mathbf{X}^T\mathbf{X})^{-1}\beta\mathbf{X}^T\mathbf{y}, \tag{3.4.3}$$
$$\beta^{-1}+\mathbf{x}_*^T(\mathbf{\Sigma}^{-1}+\beta\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_*) \tag{3.4.4}$$

By comparing with equation 3.3.10, we see that the predictive mean is the same as in the MAP case, but the variance is larger by the second term, which reflects the uncertainty in $\mathbf{w}$. The Bayesian solution has a correct estimation of the uncertainty in the predictive distribution, which is important for applications where uncertainty at one stage influences actions or decisions at a later stage, such as active learning or information maximisation settings.

This very simple example illustrates the passage from point-wise (maximum likelihood or maximum a posteriori) estimates of latent variables to a fully Bayesian treatment of the uncertainty.

3.5 Making the model kernelised

Continuing with the same example, we now illustrate **kernelisation**. So far, computing the predictive distribution in equation 3.4.4 requires the inversion of a $D \times D$ matrix which requires $O(D^3)$ operations, or equivalently solutions to a system of D linear equations with D unknowns. We replace this operation by an $O(N^3)$ inversion, which can be less costly if the dimensionality of the output space is very high (as in the case of sparse binary features).

To achieve this, we work first on the predictive mean and apply the Woodbury inversion lemma in its variant $(A+BB^T)^{-1}B = A^{-1}B(I+B^TA^{-1}B)^{-1}$ (Pedersen et al. (2008), equation 150):

[4]cf. for example Bishop (2006), section 2.3.2, or Murphy (2012), section 4.3.1

$$\mathbf{x}_*^T(\boldsymbol{\Sigma}^{-1} + \beta\mathbf{X}^T\mathbf{X})^{-1}\beta\mathbf{X}^T\mathbf{y} \tag{3.5.1}$$
$$= \mathbf{x}_*^T(\beta^{-1}\boldsymbol{\Sigma}^{-1} + \mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{3.5.2}$$
$$= \mathbf{x}_*^T\beta\boldsymbol{\Sigma}\mathbf{X}^T(\mathbf{I}_N + \mathbf{X}\beta\boldsymbol{\Sigma}\mathbf{X}^T)^{-1}\mathbf{y} \tag{3.5.3}$$
$$= \mathbf{x}_*^T\boldsymbol{\Sigma}\mathbf{X}^T(\beta^{-1}\mathbf{I}_N + \mathbf{X}\boldsymbol{\Sigma}\mathbf{X}^T)^{-1}\mathbf{y} \tag{3.5.4}$$

In the same way, applying the Woodbury lemma in its variant $(A+CBC^T)^{-1} = A^{-1} - A^{-1}C(B^{-1}+C^TA^{-1}C)^{-1}C^TA^{-1}$ (Pedersen et al. (2008), equation 145), we can rewrite the predictive variance:

$$\beta^{-1} + \mathbf{x}_*^T(\boldsymbol{\Sigma}^{-1} + \beta\mathbf{X}^T\mathbf{X})^{-1}\mathbf{x}_* \tag{3.5.5}$$
$$= \beta^{-1} + \mathbf{x}_*^T\left[\boldsymbol{\Sigma} + \boldsymbol{\Sigma}\mathbf{X}^T(\beta^{-1}\mathbf{I}_N + \mathbf{X}\boldsymbol{\Sigma}\mathbf{X}^T)^{-1}\mathbf{X}\boldsymbol{\Sigma}\right]\mathbf{x}_* \tag{3.5.6}$$

Now, note that $\mathbf{x}_*$, $\mathbf{X}$ and $\boldsymbol{\Sigma}$ appear only in dot products involving $\boldsymbol{\Sigma}$, for instance $\mathbf{x}_*^T\boldsymbol{\Sigma}\mathbf{x}_* = \langle\mathbf{x}_*, \mathbf{x}_*\rangle_{\boldsymbol{\Sigma}}$. That is to say, to express the solution, we only need to know the value of the equivalent kernel, defined by $k(\mathbf{x}, \mathbf{x}') = \langle\mathbf{x}, \mathbf{x}'\rangle_{\boldsymbol{\Sigma}}$, over the training and test inputs $\{\mathbf{x}_1, \ldots \mathbf{x}_N\} \cup \{\mathbf{x}_*\}$. These values are entirely contained in the Gram matrix of k over this set of points. We can write:

$$\begin{pmatrix} \mathbf{X} \\ \mathbf{x}_*^T \end{pmatrix} \boldsymbol{\Sigma} \begin{pmatrix} \mathbf{X} \\ \mathbf{x}_*^T \end{pmatrix}^T = \begin{pmatrix} \mathbf{X}\boldsymbol{\Sigma}\mathbf{X}^T & \mathbf{x}_*^T\boldsymbol{\Sigma}\mathbf{X}^T \\ \mathbf{X}\boldsymbol{\Sigma}\mathbf{x}_* & \mathbf{x}_*^T\boldsymbol{\Sigma}\mathbf{x}_* \end{pmatrix} = \begin{pmatrix} \mathbf{K} & \mathbf{k}_*^T \\ \mathbf{k}_* & k_{**} \end{pmatrix} \tag{3.5.7}$$

Using this definition, the predictive distribution takes the form

$$p(y_*|\mathbf{x}_*, \mathbf{X}, \mathbf{y}) \sim \mathcal{N}(\mathbf{k}_*^T(\beta^{-1}\mathbf{I}_N + K)^{-1}\mathbf{y}, \tag{3.5.8}$$
$$\beta^{-1} + k_{**} + \mathbf{k}_*^T(\beta^{-1}\mathbf{I}_N + K)^{-1}\mathbf{k}_*) \tag{3.5.9}$$

What have we gained by performing this transformation? Not much if we keep the equivalent kernel defined using $\boldsymbol{\Sigma}$ in equation 3.5.7, and whose rank is at most D. Much more if we start from the kernel as a first-class construct, use it as a dot product, and no longer look to the input space, as will now become clear.

In practice, suppose the feature vectors $\mathbf{x} \in \mathbb{R}^D$, with which we have been working so far, are the output of a feature map $\phi : \Omega \to \mathbb{R}^D, \omega \mapsto \mathbf{x}$. Now, ω, the original input object, might be an image, a string, a tree, a video, the representation of a website user, a product. Kernels have been devised for a wide variety of such input spaces Ω, for instance the TF-IDF (term frequency-inverse document frequency) kernel for documents, spectrum kernels for strings, pyramid match kernels for images, or even more generally Fisher kernels defined from probabilistic models[5].

In the linear regression example, we found out that we can work with kernels instead of feature maps; a kernel is a representation of similarity, and *implicitly* defines a feature map. This enables us to use any (positive definite) kernel which is suitable for the input space Ω we wish to work with. We can inject a great deal of engineering in making a kernel which adequately expresses similarity in the input space. Some such kernels might be equivalent to very high, or even infinite-dimensional feature maps. For instance the squared exponential kernel

$$k(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{|\mathbf{x} - \mathbf{x}'|^2}{\sigma^2}\right) \tag{3.5.10}$$

is equivalent to an infinite-dimensional feature representation, while the polynomial kernel

$$k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}' + c\rangle^d, c \in \mathbb{R} \tag{3.5.11}$$

is equivalent to a feature representation containing all the monomials of degree d of elements of $\mathbf{x}, \mathbf{x}'$.

If we simply use a linear kernel

$$k(\mathbf{x}, \mathbf{x}') = \langle \mathbf{x}, \mathbf{x}'\rangle \tag{3.5.12}$$

we are using the original input feature representation $\mathbf{x}$, and this is just Bayesian linear regression, so it is like not using any kernel at all.

More generally, the theory of reproducing kernel Hilbert spaces tells us, in the form of the Moore-Aronszajn lemma[6], that any positive definite kernel (a.k.a. Mercer kernel), defines a (possibly very high- or even infinite-dimensional) feature representation, and its corresponding feature space.

This means that the kernelised formulation can port the problem into such a feature space (termed a reproducing kernel Hilbert space), solve it as a linear problem there, and therefore express a non-linear solution in the original space. This manipulation (replacing the dot products appearing in an algorithm by kernel values) is called the "*kernel trick*", and can be applied to a range of algorithms which can be expressed exclusively in terms of dot products[7].

An alternative kernel-based view of linear regression introduces Gaussian stochastic processes[8]; GP regression appears in figure 3.2.1 on the same vertex as ker-

[5]cf. Murphy (2012), chapter 14, or Schölkopf and Smola (2002), chapter 13.

[6]cf. for example Rasmussen and Williams (2006), theorem 6.1

[7]We will not introduce kernels and the theory of reproducing kernel Hilbert spaces any further in this thesis; several excellent introductions are available, for instance Schölkopf and Smola (2002), chapters 1 and 2. For a didactic presentation of the kernel trick, cf. for example Murphy (2012), section 14.4.

[8]We will not further introduce Gaussian processes in this thesis. Seeger (2004), Bishop (2006), section 6.4, Murphy (2012), chapter 15 provide excellent introductions. Rasmussen and Williams (2006) is an in-

nelised Bayesian linear regression for this reason. We will not derive this approach here, but refer to Rasmussen and Williams (2006), chapter 2. The Bayesian treatment of GP regression with independent noise will yield the same predictive distribution as we obtained in equation 3.5.9.

3.6 Making the model structured

We have described how to move along the "kernelisation" and "Bayesian" edge of the diagram in figure 3.2.1. How do we move along the "structure" edge, and turn the regression models into classification, and then structured classification models, as defined in section 1.1 ?

The first step in that direction consists of turning a regression model into a binary classification model (with output $y \in \{+1, -1\}$). This is typically achieved by directing the noise-free regression output (in $\mathbb{R}$) through a "squeezing" function $\sigma : \mathbb{R} \to [0, 1]$ (usually a *logistic* function $a \mapsto \frac{1}{1+\exp(-a)}$, or another sigmoid function), which will turn it into a scalar in the interval $[0, 1]$. This scalar can then be interpreted as the probability of belonging to the positive class. In this way, the model becomes *linear logistic regression*[9]*:*

$$p(y = +1|\mathbf{x}) = \sigma(\mathbf{x}^T\mathbf{w}) \tag{3.6.1}$$

Moving one notch to the right in the diagram, multi-class classification with R classes can be achieved by modelling each class probability using a separate regression model, now using the multi-class extension of the logistic, called *softmax:*

$$p(y = r|\mathbf{x}) = \frac{\exp\left(\mathbf{x}^T\mathbf{w}_y\right)}{\sum_{y'=1}^{R} \exp\left(\mathbf{x}^T\mathbf{w}_{y'}\right)}, \forall r \in \mathcal{R} \tag{3.6.2}$$

Note that this parameterisation of R probability masses by R scalars is overcomplete: one could use only $R-1$ scalars (and therefore weights $\mathbf{w}_r$) for $p(y = r|\mathbf{x})$, $r \leq R-1$, and then assign $p(y = R|\mathbf{x}) = 1 - \sum_{r=1}^{R-1} p(y = r|\mathbf{x})$. However, the former parameterisation has the advantage of being symmetric: all classes play the same role.

These steps can be applied to Gaussian processes to go from regression to binary and multi-class classification (Williams and Barber (1998)). An important difference to the parametric models described here is that the dot product $\mathbf{x}^T\mathbf{w}$ is replaced, applying the kernel trick, by a GP-distributed latent variable. In multi-class classification, we need R such latent variables, and therefore the question arises how to describe the covariance between them, the simplest solution being to consider them independent (cf. section 4.3). Another difficulty, present in GP binary classification

depth textbook on the topic.

[9]which, despite the name, is a model used for classification

already, arises from the form of the likelihood; while in GP regression with Gaussian noise (i.e. Gaussian likelihood), the solution is analytic, but as soon as the likelihood takes another form, computing the posterior and predictive distributions requires resorting to approximations; these are well explored by now, cf. Kuss and Rasmussen (2005, 2006); Nickisch (2008), and Rasmussen and Williams (2006), chapter 3, for a textbook presentation.

The passage to structured models is achieved by moving to a *structured softmax* likelihood; the resulting model could naïvely be written:

$$p(\mathbf{y}|\mathbf{x}) = \frac{\exp\left(\mathbf{x}^T \mathbf{w}_{\mathbf{y}}\right)}{\sum_{\mathbf{y}' \in \mathcal{Y}} \exp\left(\mathbf{x}^T \mathbf{w}_{\mathbf{y}'}\right)} \tag{3.6.3}$$

This presentation, however, is unrealistic: it assumes that we can index, as before, the weight parameters $\mathbf{w}$ with $\mathbf{y}$. However, $\mathcal{Y}$ is now of size exponential in the number of atoms in $\mathbf{y}$, which makes the entire problem intractable, were we to use it as an index.

We now introduce conditional random fields, which were developed to address this issue.

3.7 The CRF model and extensions

The central idea of the CRF model (Lafferty et al., 2001) is a change of parameterisation with respect to the logistic regression model. Instead of indexing the weights $\mathbf{w}_y$ with y, we will rewrite the vector product $\mathbf{x}^T\mathbf{w}_y$ in the form $\phi(\mathbf{x}, y)^T\mathbf{w}$. To achieve this, we can stack all $\mathbf{w}_y, \forall y$, into a large vector $\mathbf{w} \in \mathbb{R}^{RD}$, and on the other hand, build a vector of the same length RD consisting of only zeros, except for the elements which will be multiplied, in the dot product, with the desired $\mathbf{w}_y$ inside $\mathbf{w}$: in this position we place $\mathbf{x}$. The result can be written with a Kronecker product $\otimes$ and $e_y \in \{0,1\}^R$, the unit vector containing a single one in the position corresponding to y:

$$p(y|\mathbf{x}) \propto \exp\left(\left(e_y \otimes \mathbf{x}\right)^T \mathbf{w}\right) \tag{3.7.1}$$

This is now in the desired form $\phi(\mathbf{x}, y)^T\mathbf{w}$.

As noted, we cannot directly generalise this to the structured setting, because manipulating $e_{\mathbf{y}}$ with $\mathbf{y} \in \mathcal{Y} = \mathcal{R}^T$ is unwieldy: it would entail $\mathbf{w} \in \mathbb{R}^{R^T \times D}$.

Instead, the CRF parameterisation makes an assumption on the dependency structure of components of $\mathbf{y} = (y_t)_t$: it assumes they are modelled by a Markov random field (MRF). In addition, it posits that each of the y_t depends on $\mathbf{x}$. This results in the graphical model represented in figure 3.7.1. In the MRF part of the model, the y_t are conditionally independent of each other given $\mathbf{x}$ and their neighbours.

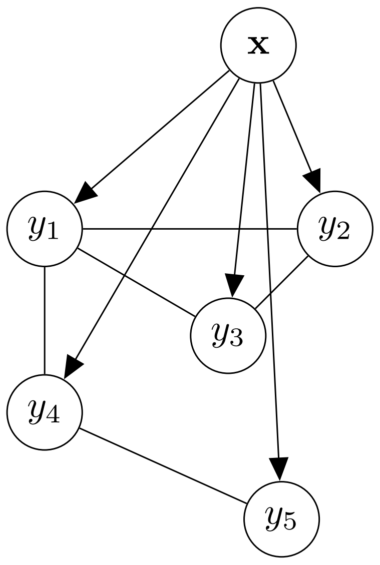


Figure 3.7.1: Graphical model for a CRF as a mixed (i.e. directed and undirected) graph, also called a partially directed graph. The MRF part specifies the dependencies inside $\mathbf{y}$. All y_t nodes depend on $\mathbf{x}$, which may have structure, but it is ignored here.

The Hammersley-Clifford theorem[10] applies in this graph, and tells us that $p(\mathbf{y}|\mathbf{x})$ factorises over the maximal cliques[11] of the graph:

$$p(\mathbf{y}|\mathbf{x}) = \frac{1}{Z(\mathbf{x})} \prod_{c \in \text{max cliques}} \exp\left(\boldsymbol{\phi}_c(\mathbf{x}, \mathbf{y}_c)^T \mathbf{w}_c\right), \tag{3.7.2}$$

where the $\boldsymbol{\phi}$ are functions called clique potentials, which parameterise $p(\mathbf{y}|\mathbf{x})$.

The normalisation function is $Z(\mathbf{x}) = \sum_{\mathbf{y}' \in \mathcal{Y}} \exp\left(\sum_c \boldsymbol{\phi}_c(\mathbf{x}, \mathbf{y}'_c)^T \mathbf{w}_c\right)$.

We can make four remarks on equation 3.7.2.

Firstly, we now have a parameterisation which decomposes over cliques, and requires much fewer parameters than the naïve attempt in equation 3.6.3.

Secondly, in contrast to the parameterisation used so far in this chapter, we have made the feature extraction function explicit, hence used $\boldsymbol{\phi}(\mathbf{x})$ instead of $\mathbf{x}$, and moved the dependency on $\mathbf{y}$ into the feature extraction function, which therefore becomes an input-output feature extraction function $\boldsymbol{\phi}(\mathbf{x}, \mathbf{y})$.

Thirdly, the formulation in equation 3.7.1 makes apparent that logistic regression is a special case of the CRF, corresponding to a single-clique CRF, and a single-node

[10]The Hammersley-Clifford theorem can be summarised as follows: a positive distribution satisfies the independence properties encoded in a graph if and only if it decomposes over maximal cliques of the graph as a product of factors. Cf. Hammersley and Clifford (1971) for the original formulation, and Koller and Friedman (2009), theorem 4.2, for a didactic presentation.

[11]In a graph consisting of vertices and edges, a clique is a set of vertices such that every element of the set is connected by an edge to every other element. A maximal clique is a clique which cannot be made larger by including further vertices.

$\mathbf{y}$, i.e. $T = 1$.

Finally, while the exponential clique potential seems to follow from the use of the softmax function in equation 3.6.2, it is interesting to note that they have an independent motivation as a maximum entropy, or log-linear, model (Della Pietra et al., 1997).

The CRF has been popular and successful, particularly in sequence labelling problems in NLP. The typical training procedure is maximum likelihood or maximum a posteriori parameter estimation. In the latter case, a Gaussian prior on $\mathbf{w}$ is often used, so that the optimisation problem becomes:

$$\arg\min_{\mathbf{w}} \log p(\mathbf{w}) \prod_{n=1}^{N} p(\mathbf{x}_n, \mathbf{y}_n | \mathbf{w}) \tag{3.7.3}$$

$$= \arg\min_{\mathbf{w}} \left(\frac{1}{2\sigma} \|\mathbf{w}\|^2 + \sum_{n=1}^{N} \left(\log \left(\sum_{\mathbf{y}' \in \mathcal{Y}} \exp \left(\boldsymbol{\phi}(\mathbf{x}_n, \mathbf{y}')^T \mathbf{w} \right) \right) - \boldsymbol{\phi}(\mathbf{x}_n, \mathbf{y}_n)^T \mathbf{w} \right) \right) \tag{3.7.4}$$

(we have stacked the $\boldsymbol{\phi}_c$ and $\mathbf{w}_c$ to simplify the notation). The objective function is convex[12]. In Lafferty et al. (2001), the optimisation is performed using a form of improved iterative scaling; nowadays, on small data sets quasi-Newton methods are used, while stochastic gradient descent is preferred on large data sets.

3.7.1 Variants of the CRF

A close variant of the CRF is the **structured perceptron** (Collins, 2002), motivated by sequence labelling problems. It can be characterised as follows: the likelihood model $p(\mathbf{y}|\mathbf{x}, \mathbf{w})$ is comparable to the CRF (but goes under the name "maximum entropy tagger"), however the parameter estimation procedure is not maximum likelihood as usually assumed for the CRF. Instead, training uses the so-called *perceptron algorithm*, which consists of iterative updates[13] to the parameter vector $\mathbf{w}$: initialising $\mathbf{w}^0 = 0$, for epoch e from 1 to E, for each training example n from 1 to N,

1. predict the labelling

$$\hat{\mathbf{y}}_n^e = \arg\max_{\mathbf{y}} p(\mathbf{y}|\mathbf{x}_n, \mathbf{w}^{e-1}) = \arg\max_{\mathbf{y}} \boldsymbol{\phi}(\mathbf{x}_n, \mathbf{y})^T \mathbf{w}^{e-1} \tag{3.7.5}$$

   based on current parameters

[12]This follows from standard convexity results, using Boyd and Vandenberghe (2004), section 3.10, including convexity of log-sum-exp, section 3.1.5.

[13]The number of epochs E has to be estimated using a subset of the training data set apart for validation (a.k.a. development data set).

2. if $\hat{\mathbf{y}}_n^e \neq \mathbf{y}_n$ (i.e. the prediction results in an incorrect labelling), update the parameter vector:

$$\mathbf{w}^e = \mathbf{w}^{e-1} + \eta \left(\boldsymbol{\phi}(\mathbf{x}_n, \mathbf{y}_n) - \boldsymbol{\phi}(\mathbf{x}_n, \hat{\mathbf{y}}_n^e) \right) \tag{3.7.6}$$

   (where η is a learning rate)

This training procedure amounts to a noisy gradient version of CRF maximum likelihood training with learning rate η, where the term $\boldsymbol{\phi}(\mathbf{x}_n, \hat{\mathbf{y}}_n^e)$ approximates the expected feature vector term $\mathbb{E}_{\mathbf{y} \sim p(\mathbf{y}|\mathbf{x}_n, \mathbf{w})} \boldsymbol{\phi}(\mathbf{x}_n, \mathbf{y})$ which appears in the gradient of the CRF log-likelihood $\nabla_{\mathbf{w}} \log p(\mathbf{y}_n | \mathbf{x}_n, \mathbf{w})$. The update is therefore based on a local gradient update which may have convergence issues.

As with linear regression above, we would now like to look at Bayesian and kernelised variants of the CRF. The prize question, and the one which motivates the rest of this thesis, is then: just like GP regression was both a Bayesian and kernelised method, but now moving to structured output prediction, can we find a model that is both Bayesian and kernelised, filling the last vertex on figure 3.2.1?

We begin by looking at the **Bayesian CRF** (Qi et al., 2005): Rather than using a point estimate for $\mathbf{w}$, this paper pursues a Bayesian training (and prediction) procedure as explained above in the case of linear regression. In the Bayesian CRF, the posterior is approximated by a Gaussian distribution using power expectation propagation, a generalisation of expectation propagation and fractional belief propagation (which all carry out some form of minimisation of an α-divergence) to arbitrary α.

As before, we can also kernelise the CRF. This gives rise to the **kernel CRF** described in Lafferty et al. (2004), and in a different form, with reference to Gaussian processes, in Altun et al. (2004a)[14]. These two papers are important for the model developed in the next chapter, so we discuss them now. We refer to both as KCRF, in agreement with Perez-Cruz et al. (2007), which also discusses them.

In the KCRF, instances of $\boldsymbol{\phi}_c(\mathbf{x}, \mathbf{y}_c)^T \mathbf{w}_c$ are replaced by $f(\mathbf{x}, \mathbf{y}, c)$. Learning is formulated as a regularised empirical risk minimisation problem of the form

$$\arg\min_{\mathbf{f}} - \sum_n \left[\sum_c f(\mathbf{x}_n, \mathbf{y}_n, c) - \log \sum_{\mathbf{y}'} \exp \left(\sum_c f(\mathbf{x}_n, \mathbf{y}', c) \right) \right] + \lambda \|f\|_{\mathbf{K}} \tag{3.7.7}$$

[14]Note that Altun et al. (2004a) contains a pervasive error: in every statement of the objective function in dual terms, the regulariser $\frac{1}{2}\mathbf{u}^T K^{-1} \mathbf{u}$ is turned into $\alpha^T \mathbf{K} \alpha$, forgetting the $\frac{1}{2}$ term. This error is present in eq. 6 (both terms), 8, 12, 15 (where the gradient contains an unwanted factor 2), as well as in the technical report version (Altun and Hofmann, 2003) eq. 5, 6, 9, 21 (and presumably in the gradients eq. 17 and 22, though it is not clear what γ is there). The correct derivation, however, follows from Altun et al. (2004a) eq. 5: $u^{\text{map}}(\mathbf{x}_i, y) = \sum_{j,y'} \alpha_{(j,y')} K_{(i,y),(j,y')} = \alpha \mathbf{K} e_{(i,y)} = e_{(i,y)}^T \mathbf{K} \alpha$ (with e and α column vectors), so that by stacking the nm terms, $\mathbf{u}^{\text{map}} = \mathbb{I}_{nm \cdot nm} \mathbf{K} \alpha = \mathbf{K} \alpha$, and the regularising term in the objective can be rewritten as $\frac{1}{2} \mathbf{u}^{\text{map}T} K^{-1} \mathbf{u}^{\text{map}} = \frac{1}{2} \alpha^T \mathbf{K} \mathbf{K}^{-1} \mathbf{K} \alpha = \frac{1}{2} \alpha^T \mathbf{K} \alpha$. It is unclear whether this error was carried over to the implementation and affected experimental results.

where (simplifying the notation by removing dependencies) n runs over training structures, c runs over cliques in the structure corresponding to n, y' represents any labelling compatible with c, and $\|\cdot\|_K$ is the RKHS norm of kernel k, whose Gram matrix is $\mathbf{K}$. The problem is equivalent to maximum a posteriori parameter estimation with a Gaussian prior (with covariance matrix $\mathbf{K}$).

The representer theorem[15] states that a minimiser of the cost function of equation 3.7.7 has the form

$$f(\cdot) = \sum_n \sum_c \sum_{\mathbf{y}'} \alpha(n, c, \mathbf{y}') k((n, c, \mathbf{y}'), \cdot) \qquad (3.7.8)$$

We have cause for concern at this point: this multiple summation contains the intractable element $\sum_{\mathbf{y}' \in \mathcal{Y}_n}$ (the set of labellings on data point n depends on n, for instance if data consists of sequences, they might have different length). It would be desirable that the solution exhibit some amount of sparsity, i.e. the fact that most α terms are zero, making computation easier. We are not in the lucky case of binary classification with support vector machines, however, where the hinge loss induces sparsity automatically; we have the log loss here.

Can we obtain a sparser solution nonetheless? Following a line of thought also present in the import vector machine (Zhu and Hastie, 2005), both Lafferty et al. (2004) and Altun et al. (2004a) seek approximate solutions $\hat{f} \in \text{span}(\mathcal{A})$, where $\mathcal{A}$ is a (hopefully small) set of functions $k((n, c, \mathbf{y}'), \cdot)$. Starting with $f = 0$, i.e. $\mathcal{A} = \emptyset$, the learning algorithm greedily adds the $k((n, c, \mathbf{y}'), \cdot)$ which gives the direction of steepest (maximal) gradient of the objective function. Lafferty et al. (2004), section 3, refers to this as "clique selection", because each $k((n, c, \mathbf{y}'), \cdot)$ corresponds to one clique, while Altun et al. (2004a), section 4.2, calls this "column generation", referring to columns of the Gram matrix $\mathbf{K}$ of the Mercer kernel k.

A related strategy is employed in the Conditional Graphical Model[16] of Perez-Cruz et al. (2007). A surrogate loss (Bartlett et al., 2006) of the logistic/ softmax loss is introduced, and brings advantages in terms of computation: gradient computation decouples over cliques.

Of the models discussed so far, Altun et al. (2004a)'s model comes closest to our desiderata. One originality of this paper resides in the route it takes, starting with an approach which seems paradoxical. The paper's steps can be schematised as follows (equations numbers refer to Altun et al. (2004a)):

- approach sequence labelling as a multi-class classification problem (boldly ignoring the cardinality of the output space for a moment, like in the naïve at-

[15]For a modern statement and proof, cf. Schölkopf and Smola (2002), theorem 4.2, page 90. Lafferty et al. (2004), with good notation and a clear conceptual basis, proves the representer theorem for the specific case of the KCRF.

[16]Somewhat of a misnomer, since the probabilistic model (used for prediction, for instance) really is still the CRF; the difference is in the objective function for training, which now contains a surrogate loss.

tempt presented in section 3.6 of this thesis)

- formulate a probabilistic model in which latent variables u (which play the role of f above) are GP-distributed
- note that marginalising u is intractable, so formulate the MAP objective (eq. 4)
- apply the representer theorem, obtain a dual objective which contains intractable sums (eq. 6);
- remark that the sparseness in the corresponding SVM solution is due to the hinge loss, as opposed to the logistic loss here
- formulate (section 3.2) a kernel from feature maps, inspired by the HMM, respecting the design assumption that y_t is only affected by y_{t-1} and y_{t+1}
- recognise that this simplifies the objective function sufficiently (eq. 12), because arbitrary labellings $\mathbf{y}'$ (the labellings absent from the training data) need only be considered clique-wise, not sequence-wise, which reduces the cardinality of the index set notably
- notice that some latent variables (specialised to micro-labels) are responsible for unary or binary effects (as seen in the ad hoc design of the linear map $\mathbf{\Lambda}$ eq. 9, 10, and discussed in Altun and Hofmann (2003), below eq. 14)

3.7.2 Maximum margin extensions of the CRF

Another perspective is given by abandoning ML, MAP or the (mathematically) equivalent empirical risk minimisation learning principles in favour of maximum-margin learning.

This is the basic choice which motivates the definition of support vector methods for regression, classification, multi-class classification (Weston and Watkins, 1998; Crammer and Singer, 2001), up to structured classification with the SVMstruct models (Tsochantaridis et al., 2004, 2005; Altun et al., 2003)[17].

We now discuss **max-margin Markov networks (Taskar et al., 2004)** (M3N), which, like Lafferty et al. (2004) and Altun et al. (2004a), are very close to the model we are looking for.

M3N's starting point is a maximum margin formulation of a CRF model.

In the structured output context, adopting the maximum margin learning principle simultaneously imposes its own losses[18].

[17]Schmidt (2009) offers a didactic presentation, comparative discussion and derivation of the SVMstruct "family of models" from a probabilistic formulation (using log-loss empirical risk minimisation).

[18]It is interesting to note that these losses cannot be formulated as likelihoods, cf. Sollich (2002) for the binary classification case.

This can be seen when moving from the separable to the non-separable data case, and answering the question: "how to penalise constraint violations?". The alternatives are typically margin-scaled and slack-scaled loss, which correspond to respectively a margin rescaling or slack rescaling formulation of the penalties[19]. Of the two, M3N use margin rescaling (Taskar et al. (2004), equation 4). The margin-scaled loss, $\max_{\mathbf{y}\in\mathcal{Y}} \ell(\mathbf{y}_n, \mathbf{y}) + \mathbf{w}^T \left(\boldsymbol{\phi}(\mathbf{x}_n, \mathbf{y}) - \boldsymbol{\phi}(\mathbf{x}_n, \mathbf{y}_n)\right)$, where $\ell(\mathbf{y}^*, \mathbf{y})$ denotes the loss incurred by predicting $\mathbf{y}$ while the truth is $\mathbf{y}^*$, is also know as *structured hinge loss*. The resulting optimisation $\max_{\mathbf{y}\in\mathcal{Y}}(...)$ is an example of *loss-augmented inference*, since the loss $\ell(\mathbf{y}_n, \mathbf{y})$ appears in the objective function.

The structured hinge loss is a generalisation of the hinge loss

$$\max\left(0, 1 - y_n \mathbf{w}^T \psi(x_n)\right) \tag{3.7.9}$$

used in binary classification. To understand this, observe the maximand in the structured hinge loss: when $y = y_n \in \mathcal{Y}$, it is 0, and so the loss rewrites as $\max\left(0, \max_{y\in\mathcal{Y}\setminus y_n} ...\right)$: we see the "hinge" appear. Further, noting that $\mathcal{Y}$ is reduced to $\{-1, +1\}$, and setting $\phi(x_n, y) = \frac{1}{2} y \psi(x_n)$, we recover the hinge loss. In addition, like the hinge loss, it is a convex surrogate loss to $\ell(y_n, y)$ (cf. e.g. Murphy (2012), equations 19.105 sq.).

We have described the sparsity-inducing training methods employed by Lafferty et al. (2004) and Altun et al. (2004a) above. Taskar et al. (2004) has a different approach, in its section 7, to the problem of the sum over all possible labellings appearing in the solution. By exploiting the fact that, conditional on $\mathbf{x}_n$, the α in the solution can be interpreted as unnormalised probabilities, and the fact that the loss function they use decomposes over cliques, they reinterpret the sums appearing in their formulation of the dual problem (their equation 7) as expectations of the loss, which can be expressed analytically using the marginals of the clique-related factors. This insight, termed the "Taskar trick" in analogy to the "kernel trick" applied in kernel machines[20], generates extra constraints to ensure the marginals are valid marginals of the MRF. For Markov networks which can be triangulated[21], this results in a quadratic program with a number of constraints which is polynomial in the number of nodes; relaxations are introduced for networks which cannot be triangulated.

[19]cf. Keshet (2014), equations 10 and 11 for a formulation in terms of losses; cf. Murphy (2012) equation 19.101 for the equivalence between the loss and constraint formulations of the objective.

[20]cf. p. 69

[21]i.e. whose graph can be made chordal, i.e. so that every minimal loop is of length 3; cf. Koller and Friedman (2009), definition 2.24, and theorem 9.8

3.8 Conclusion

Starting from regression models, and moving on to classification models, we have reviewed members of the discriminative probabilistic (i.e. conditional) family. Focusing on structured classification models, we went through the MAP branch, for which a prominent representative is the CRF, as well as the max-margin branch, epitomised by the SVM, and represented here by the max-margin Markov network.

What appears clearly in such a review is the lack of a model endowed with all the desirable properties exhibited separately by each of the existing models: to be kernelised and non-parametric, fully Bayesian (i.e. preserve uncertainty between the learning and the prediction stage), and apply to structured, i.e. multi-label multi-class, data. The goal of the next chapter is really to complete the roadmap in figure 3.2.1 with such a model.

Chapter 4

GPstruct for sequence labelling

Chapter 3 gave a reasoned overview of a family of discriminative models in increasing order of complexity, going from linear regression to CRF models, explaining their relations and advantages, and generally setting the stage for the present chapter.

We are now in a position to present GPstruct, a conceptually novel model which fills the gap pointed out at the end of the last chapter; it fits into the last vertex of figure 3.2.1 : it is kernelised, fully Bayesian and performs structured classification. More particularly, as discussed in section 1.1, our motivating task is labelling: atoms in the input are individually labelled with output categories, so that the output structure maps directly to the input structure.

After defining the modelling assumptions underlying GPstruct, we instantiate it to sequence data, and present an MCMC inference algorithm. We then show experimentally, on natural language processing tasks, that GPstruct performs well in comparison to other models adapted to the same tasks. Several analytical experiments shed light on the configuration of the MCMC training algorithm and on the performance of the MAP variant of GPstruct, which is similar to a KCRF. We then compare GPstruct to related models, and conclude the chapter with ideas for further developments.

Sections 4.1 to 4.7 were published in Bratières et al. (2015), and are the result of collaboration with my co-authors.

4.1 Model formulation

We define GPstruct straight away, to clarify the exposition. Assume data consists of observation-label (or input-output) tuples, which we will write

$$\mathcal{D} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \dots, (\mathbf{x}^{(N)}, \mathbf{y}^{(N)})\} \tag{4.1.1}$$

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. $\mathbf{y}$, which we would like to predict, is a structured 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.

GPstruct consists of two parts:

1. a conditional log-linear likelihood which specifies the dependency structure in the output $\mathbf{y}$, as a Markov random field, given latent variables (LV) which mediate the influence of the input on the output. This effectively results in a CRF.
2. a (prior) distribution for the latent variables, which is given by an appropriate Gaussian process

The likelihood model specifies the distribution of the output labels given the input and the LV. It is defined per *clique* of the CRF defined over the output $\mathbf{y}$. The likelihood model is:

$$p(\mathbf{y}|\mathbf{x}, \mathbf{f}) = \frac{\exp\left(\sum_c f(c, \mathbf{x}_c, \mathbf{y}_c)\right)}{\sum_{\mathbf{y}' \in \mathcal{Y}} \exp\left(\sum_c f(c, \mathbf{x}_c, \mathbf{y}'_c)\right)} \tag{4.1.2}$$

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 $\mathbf{y}_c$. We call the distribution (4.1.2) *structured softmax*, in analogy to the softmax (a.k.a. multinomial logistic) likelihood used in multinomial logistic regression, equation 3.6.2.

The prior distribution of the LV $f(c, \mathbf{x}_c, \mathbf{y}_c)$ is given by a Gaussian process (GP) with covariance function (i.e. Mercer kernel) $k((c, \mathbf{x}_c, \mathbf{y}_c), (c', \mathbf{x}_{c'}, \mathbf{y}_{c'}))$: denoting by $\mathbf{f}$ the collection of all LV of the form $f(c, \mathbf{x}_c, \mathbf{y}_c)$,

$$\mathbf{f} \sim \mathcal{GP}(0, k(\cdot, \cdot)) \tag{4.1.3}$$

In summary, the GPstruct is a probabilistic model in which the likelihood is given by a structured softmax, with a conditional Markov random field modelling output interdependencies; the CRF's latent variables, one per factor, are given a GP prior. It can be viewed in different ways: as a variant of a multi-class GP classifier, using a joint input-output kernel, and specialised for structured output; or as a GP indexed by factors (or cliques) in a factor graph; or as a kernelised, Bayesian log-linear model.

The GPstruct model has the following appealing statistical properties, which motivate its design:

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

Non-parametric: the number of LV grows with the size of the data. For sequences (section 4.2), 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: uncertainty estimates ("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.

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

This chapter is structured as follows. Now that we have given a schematic description of the model, we will present design choices to be made to apply it to sequence labelling problems. We are then in a position to describe an MCMC sampling algorithm for training. Experiments on real datasets, and a benchmark comparison to similar structured output prediction techniques are then presented. Practical aspects of inference and prediction are discussed. This chapter closes with limitations of the model and possible remedies.

Chapter 5 presents an application of GPstruct to grid structures in a large-scale application, which requires several adjustments to the inference procedure. Because GPstruct is a Bayesian model, it is interesting to investigate whether hyperparameters can be learnt during inference alongside latent variables; this is the object of Chapter 6.

4.2 Parameterisation for sequence problems

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 sequence of length T. The input is represented as feature vectors of length D, i.e. $\mathcal{X} = \mathbb{R}^D$. The micro-labels of the output all stem from a common set, i.e. $\mathcal{Y} = \prod_{t=1,\ldots,T} \mathcal{Y}_t$ with $\forall t$, $\mathcal{Y}_t = \mathcal{R}$, with $|\mathcal{R}| = R$. 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,

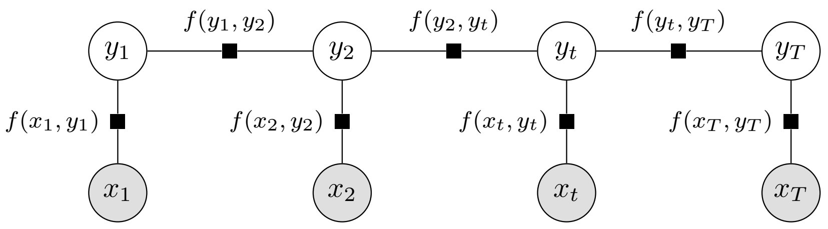


Figure 4.2.1: Factor graph for sequence prediction with two clique types, unary location-dependent cliques $\tilde{c}_t$ and binary location-independent cliques $\tilde{\tilde{c}}$. Input nodes are always treated as observed.

and micro-labels just "labels", without risk of confusion.

In our experiments below, we tackle text data: the input generally consists of sentences (chains of words), and the output of corresponding chains of task-specific labels. A common natural language processing task is noun phrase recognition: here the (3-class) micro-labels are B, to label the beginning of a noun phrase segment, I, to label the inside of a segment, and O, for all other tokens.

We will now expose further design decisions involved in instantiating GPstruct to a sequence CRF. Among the available choices, we point out the options we selected for our experiments.

To start with, the sequence CRF we work with is according to the factor graph in figure 4.2.1: there are two clique types (or *clique templates* (Lafferty et al., 2004)): binary cliques (y_t, y_{t+1}), and unary[1] cliques $(\mathbf{x}_t, y_t)$. If we removed binary factors, we would be carrying out micro-label-wise prediction, and micro-labels would not be interdependent (this type of prediction can be qualified as structured in the weak sense, cf. section 1.1). We could add factors (y_t, y_{t+2}) and (y_t, y_{t-2}) for longer-range dependencies.

A priori, there is one LV per $(c, \mathbf{x}_c, \mathbf{y}_c)$ tuple, 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 our treatment of sequential GPstruct, we distinguish each individual unary clique based both on position t and node values, but tie all binary cliques independently of their position (but distinguish on their nodes values), so that we can denote them by $\tilde{c}_t$ resp. $\tilde{\tilde{c}}$. This is illustrated in figure 4.2.1 . By distinguishing each unary clique, we obtain a non-parametric model, where the number of unary LV grows with the data. Plausible alternatives are: tie binary LV except for the edge positions $t = 1$ or $t = T$, where the task may dictate a special behaviour; tie unary LV with identical values of (x_t, y_t), effectively ignoring the position (this introduces some bookkeeping since the values of x_t are usually from a high-dimensional feature space; there may

[1]These are unary because in our supervised learning setting, we always assume $\mathbf{x}$ to be given.

be no identical x_t values in the data in fact).

How many LV do we need to store? We have 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 equation 4.1.2, 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 $\mathbf{y}$. Concerning the inputs, we do not need to define LV for ranges of x_t values, since both $\mathbf{x}$ and x_t are always assumed observed.

Let $T^{(n)}$ denote the length of $\mathbf{y}^{(n)}$ (which need not be constant across label sequences). In our chosen parameterisation, there is one unary LV for each position t, and each value y_t, so there are $\sum_n T^{(n)} \times R$ unary LV. There is one binary LV per (y_t, y_{t+1}) tuple; and so there are $|\mathcal{Y}_t| \times |\mathcal{Y}_{t+1}| = R^2$ binary LV. The number of unary LV usually dominates the number of binary LV.

4.3 Kernel function specification

The Gaussian process prior defines a multivariate Gaussian density over any subset of the LV, with a covariance function given by the positive definite kernel (Mercer kernel) k. For a sequence CRF depicted in figure 4.2.1, we need to specify a kernel over arbitrary pairs cliques, either binary (y_t, y_{t+1}) or unary $(\mathbf{x}_t, y_t)$:

$$\begin{aligned} &k((c, \mathbf{x}_c, \mathbf{y}_c), (c', \mathbf{x}_{c'}, \mathbf{y}_{c'})) = \\ &\qquad h_u \mathbb{I}[c, c' \text{are unary}]\ k_u((t, \mathbf{x}_t, y_t), (t', \mathbf{x}_{t'}, y_{t'})) \\ &\qquad + h_b \mathbb{I}[c, c' \text{are binary}]\ k_b((y_t, y_{t+1}), (y_{t'}, y_{t'+1})) \end{aligned} \tag{4.3.1}$$

In the above, we make use of Iverson's bracket notation: $\mathbb{I}[P] = 1$ when proposition P is true and 0 otherwise. The positions of c, c' are denoted by t, t', and $\mathbf{x}_t, \mathbf{x}_{t'}, y_t, y_{t'}$ are the corresponding input resp. output node values. In this chapter, we will set $h_u = h_b = 1$, but will attempt to learn these values in chapter 6.

The unary kernel $k_u((t, \mathbf{x}_t, y_t), (t', \mathbf{x}_{t'}, y_{t'}))$ corresponds to a dot product between features associated with unary cliques $(\mathbf{x}_t, y_t)$ and $(\mathbf{x}_{t'}, y_{t'})$. We make the common assumption that the joint feature map $\phi(\mathbf{x}_t, y_t)$ is an outer product $\varphi(y_t) \otimes \psi(\mathbf{x}_t)$. With this assumption, the unary kernel is a product:

$$k_u((t, \mathbf{x}_t, y_t), (t', \mathbf{x}_{t'}, y_{t'})) = k_y(y_t, y_{t'}) k_x(\mathbf{x}_t, \mathbf{x}_{t'}). \tag{4.3.2}$$

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:

$$k_{se}(\mathbf{x}_t, \mathbf{x}_{t'}) = \exp\left(-\frac{1}{\sigma^2} ||\mathbf{x}_t - \mathbf{x}_{t'}||^2\right) \tag{4.3.3}$$

where σ^2 is a kernel hyperparameter. k_y is an "output-only" kernel. Here we choose a label-identity output kernel $k_y(y_t, y_{t'}) = \mathbb{I}[y_t = y_{t'}]$ corresponding to the standard winner-takes-all multi-class classification (Tsochantaridis et al., 2005). The final form of our kernels for unary resp. binary cliques is now:

$$k_u((t, \mathbf{x}_t, y_t), (t', \mathbf{x}_{t'}, y_{t'})) = \mathbb{I}[y_t = y_{t'}] k_x(\mathbf{x}_t, \mathbf{x}_{t'}) \tag{4.3.4}$$

$$k_b((y_t, y_{t+1}), (y_{t'}, y_{t'+1})) = \mathbb{I}[y_t = y_{t'} \wedge y_{t+1} = y_{t'+1}] \tag{4.3.5}$$

The above kernel has also been explored in Altun et al. (2004a).

With the proper ordering[2] of LV, the Gram matrix has a block-diagonal structure:

$$K = cov[\mathbf{f}] = \begin{pmatrix} K_{\text{unary}} & 0 \\ 0 & K_{\text{binary}} \end{pmatrix} = \begin{pmatrix} K_x & 0 & \dots & 0 \\ 0 & \ddots & & \vdots \\ \vdots & & K_x & 0 \\ 0 & \dots & 0 & K_{\text{binary}} \end{pmatrix} \tag{4.3.6}$$

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

To summarise, designing an instance of a GPstruct model requires three types of *model design* decisions: the choice of the CRF shape, mainly dictated by the task and the output data structure; parameter tying; kernel design.

4.4 Inference procedures

4.4.1 Predictive distribution

Given an unseen test point $\mathbf{x}_*$, and the LV $\mathbf{f}_*$ corresponding to $\mathbf{x}_*$, we wish to predict label $\hat{\mathbf{y}}_*$ with lowest loss. Given $\mathbf{x}_*$ and $\mathbf{f}_*$, the underlying CRF is fully specified. For sequences, if we adopted 0/1 loss $\ell(\mathbf{y}, \hat{\mathbf{y}}) = \mathbb{I}[\mathbf{y} = \hat{\mathbf{y}}]$, we would predict the jointly most probable output sequence obtained from the Viterbi procedure. The 0/1 loss ignores error locality, and penalises an error in a single label in the sequence as much as an error in all labels. A loss which fixes this defect, and which is widely used for sequence labelling, is Hamming (micro-label-wise) loss $\ell(\mathbf{y}, \hat{\mathbf{y}}) = \sum_t \mathbb{I}[\mathbf{y}_t = \hat{\mathbf{y}}_t]$: to minimise this loss, the prediction decouples into maximising the marginal for each micro-label: $y_{*t} = \arg\max_{\hat{y}_t} p(\hat{y}_t|\mathbf{f})$.

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. Nickisch (2008, section

[2]unaries before binaries, and unaries in the following sequence: iterate over positions t, keeping y_t constant, and iterate over micro-labels y_t in the "outer for-loop", so as to obtain: $f(\tilde{c}_{t=1}, x_t, y_t = 1) \dots f(\tilde{c}_{t=T}, x_t, y_t = 1) \dots f(\tilde{c}_{t=1}, x_t, y_t = R) \dots f(\tilde{c}_{t=T}, x_t, y_t = R)$

2) for a derivation; this simply stems from the expression for a conditional distribution of variables given a Gaussian joint distribution):

$$\mathbf{f}_*|\mathbf{f} \sim N(K_*^T K^{-1}\mathbf{f}, K_{**} - K_*^T K^{-1} K_*) \tag{4.4.1}$$

where matrices K, K_*, K_{**} have their element (i, j) equal to $k(\mathbf{f}_i, \mathbf{f}_j)$ resp. $k(\mathbf{f}_i, \mathbf{f}_{*i})$ resp. $k(\mathbf{f}_{*i}, \mathbf{f}_{*j})$, with k the kernel described section 4.3, and assimilating k to the GP covariance function.

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

$$p(\mathbf{y}_*|\mathbf{f}) = \int p(\mathbf{y}_*|\mathbf{f}_*)p(\mathbf{f}_*|\mathbf{f})\, d\mathbf{f}_* \tag{4.4.2}$$

$$\approx \frac{1}{N_{\mathbf{f}_*|\mathbf{f}}} \sum_{\mathbf{f}_*|\mathbf{f}} p(\mathbf{y}_*|\mathbf{f}_*) \tag{4.4.3}$$

where $N_{\mathbf{f}*|\mathbf{f}}$ is the number of samples from $\mathbf{f}_*|\mathbf{f}$. Instead of sampling, an approximation at this stage consists of reducing $p(\mathbf{f}_*|\mathbf{f})$ to a point mass at its mode, $\mathbf{f}_*^{MAP}$, so that $p(\mathbf{y}_*|\mathbf{f}) \approx p(\mathbf{y}_*|\mathbf{f}_*^{MAP})$. This approximation, which we refer to as "$\mathbf{f}_*$ MAP scheme" below, was retained as it did not seem to impact accuracy, as discussed in section 4.7.

4.4.2 Sampling from the posterior distribution

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 (4.1.2). We suggest using elliptical slice sampling (ESS) (Murray et al., 2010), an efficient MCMC procedure for 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, a sum over $\mathcal{Y}$, of size exponential in the number of micro-labels R. 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 $O(R^2T)$.

ESS requires the following operations: computing, storing, and multiplying the Cholesky decomposition of the kernel matrix K by any vector. Equation 4.3.6 showed that it has diagonal block structure, so that these operations can be performed on a reduced form of K which contains only one exemplar of K_x, as well as K_{binary}. Despite this, the large size of the K_x is a limiting factor to our implementation.

Algorithm 4.1 *Training and Prediction* of GPstruct

```
Input Training dataset $\mathcal{D} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \dots, (\mathbf{x}^{(N)}, \mathbf{y}^{(N)})\}$ and a test sequence $\mathbf{x}_*$
Input Unary $k_u(\cdot,\cdot)$ and binary $k_b(\cdot,\cdot)$ kernel functions with their hyperparameters
Compute the block-diagonal Gram matrix $K$ of the training dataset as in equation (4.3.6)
Compute Cholesky factorisation on $(K + 10^{-4}\mathbf{I}_{\sum_n T^{(n)} \times |\mathcal{L}| + |\mathcal{L}|^2})$ (cf. page 91 for a description of the jitter parameter)
Sample posterior distribution $\mathbf{f}|\mathcal{D}$ using elliptical slice sampling (ESS) with forwards-backwards procedure to compute likelihood $p(\mathbf{y}^{(n)}|\mathbf{f}, \mathbf{x}^{(n)})$
Sample predictive latent function $\mathbf{f}_*$ of test sequence $\mathbf{x}_*$ via equation (4.4.1)
Compute $p(\mathbf{y}_*|\mathbf{f}_*, \mathbf{x}_*)$ using forwards-backwards procedure
Perform averaging over $\mathbf{f}_*$ to obtain $p(\mathbf{y}_*|\mathbf{f})$ as in equation (4.4.2)
Perform averaging over $\mathbf{f}$ to obtain $p(\mathbf{y}_*|\mathbf{x}_*, \mathcal{D})$ as in equation (4.4.4)
Output predictive distribution $p(\mathbf{y}_*|\mathbf{x}_*, \mathcal{D})$
```

ESS yields samples of the posterior. To perform prediction, it is necessary to introduce one more level of Bayesian model averaging: continuing from (4.4.2),

$$p(\mathbf{y}_*|\mathcal{D}) = \int p(\mathbf{y}_*|\mathbf{f}) p(\mathbf{f}|\mathcal{D})\, d\mathbf{f} \tag{4.4.4}$$

$$\approx \frac{1}{N_\mathbf{f}} \sum_\mathbf{f} p(\mathbf{y}_*|\mathbf{f}) \tag{4.4.5}$$

where $N_\mathbf{f}$ is the number of samples of $\mathbf{f}|\mathcal{D}$ available.

The entire sequence of operations is recapitulated in algorithm 4.1.

4.5 Experiments: text processing tasks

We evaluate sequential GPstruct on standard natural language processing tasks consisting of sequence labelling. The tasks, corresponding datasets and relevant feature definitions are provided by CRF++[3]. All methods we compare receive the same input features.

`Base NP` consists of identifying noun phrases (NP): tokens in English sentences are labelled with one of three labels: B to signal the beginning token of an NP, I (for *inside*) for a follow-up token, and O for any other token.

`Chunking` is a shallow parsing task in which tokenised English text is labelled according to its syntactic constituents, using the B/I/O system, but with B and I duplicated for each of several syntactic classes.

`Segmentation` tags sequences of Chinese ideograms with B/I tags to group them into words.

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

`Japanese NE` (named entity recognition) labels several types of named entities (organisations, persons, etc.) occurring in Japanese text.

Feature extraction using CRF++ yielded sparse binary feature vectors, which were considered as input data $\mathbf{x}$. Typical features for CRF NLP applications were used, see for instance Sutton (2012), section 2.5 "Feature engineering".

4.5.1 Train vs. test split

Except for the segmentation task, the data sets are large enough (cf. table 4.1) to be split in five; each subset is used as training set for an experiment, with the following subset as test set. In this way, five experiments can be run with no overlap in training or test sets. The segmentation data set (36 sentences) was subjected to 5-fold cross-validation.

Note that throughout this thesis, cross-validation is used exclusively to obtain a better estimate of the generalisation error, but not to perform any model selection.

4.5.2 Baselines

GPstruct was compared, using the same input features, to a maximum-likelihood trained CRF, and to a max-margin trained SVMstruct. The CRF implementation[4] 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 4.4.1). The SVMstruct[5] toolbox does not support non-linear kernels yet, so linear kernels were used for both SVMstruct and GPstruct; in the case of GPstruct, this means that k_x is the ordinary dot product. The SVM-struct regularisation parameter ranged over integer powers from 10^{-3} to 10^2 in nested cross-validation.

4.5.3 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[6] 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 fold, 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), or 50 000 resp. 80 000 iterations for `Base NP` resp. `Japanese NE`. Getting a precise runtime comparison of

[4]by Mark Schmidt `http://www.di.ens.fr/~mschmidt/Software/crfChain.html`
[5]by Thorsten Joachims `http://www.cs.cornell.edu/people/tj/svm_light/svm_hmm.html`
[6]by Mark Schmidt `http://www.di.ens.fr/~mschmidt/Software/UGM.html`

	`Base NP`	`Chunking`	`Segmentation`	`Japanese NE`
number of categories $\|\mathcal{L}\|$	3	14	2	17
number of features D	6438	29 764	1386	102 799
size training/ test set (sentences)	150 / 150	50 / 50	20 / 16	50 / 50
number of tokens in training set $\sum_n T^{(n)}$	3739.8	1155.8	942.0	1315.4
SVMstruct	5.91±0.19	9.79±0.43	**16.21±0.99**	**5.64±0.37**
CRF	5.92±0.10	**8.29±0.34**	**14.98±0.50**	**5.11±0.29**
GPstruct	**4.81±0.21**	**8.76±0.48**	**14.87±0.80**	5.82±0.37

Table 4.1: Experimental results on text processing task. In the top part of the table, data sets are described; token numbers are averaged over the training sets. In the bottom part of the table, Hamming error rate across 5 experiments (cf. section 4.5.1) are reported (mean $\pm$ standard error). GPstruct experiments on 250 000 ESS steps (i.e. $\mathbf{f}$ samples), using the $\mathbf{f}_*$ MAP scheme, linear kernel, thinning at 1:1000. The best result and those results not significantly worse than it are highlighted in boldface. We used a paired t-test with 99% confidence level as reference.

CRF, SVMstruct and GPstruct code 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.

4.5.4 Results and interpretation

Our experimental results are summarised in table 4.1. GPstruct is generally comparable to the CRF, and slightly better than SVMstruct.

4.6 Experiments: video processing task

We now apply CRF and GPstruct to the ChaLearn gesture recognition dataset[7]. The dataset consists of 20 video recording sessions of an actor (one actor per session) performing 100 instances of scripted gestures, and returning to a rest position between each gesture. Gestures are taken from a lexicon which is session-specific, and is of size 8 to 15. Each video frame is labelled with a gesture.

Each session contains a variable number of videos. The videos have an average length of 86 frames, and maximum length 305 frames.

For each session, we use a prescribed training set of 10 videos (covering all gestures) to train a chain CRF or GPstruct. At each frame of the video we extract histogram of gradient/ histogram of optical flow (HOG/HOF) (Laptev et al., 2008) descriptors and construct a codebook of 30 visual words using a k-means clustering algorithm. Frames are represented by normalised histograms of visual words occurrence, resulting in 30 feature dimensions. A squared exponential kernel (equation

[7]`https://www.kaggle.com/c/GestureChallenge/data`

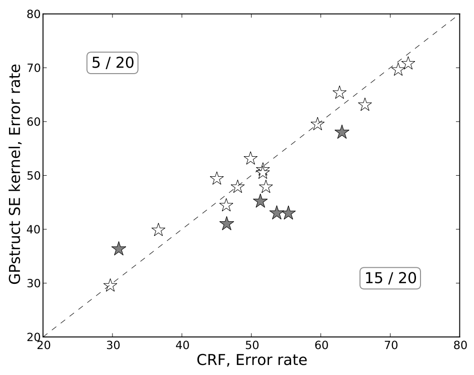


Figure 4.6.1: Error rate cross plot of the 20 gesture video sessions. The axes correspond to error rate of GPstruct with SE kernel and CRF, the diagonal line shows equal performance. The shadowed stars are those with at least 5% performance difference.

4.3.3) was used[8].

Results The experimental results are presented in the following table, in which we show the Hamming error rates averaged across 20 sessions (mean $\pm$ one standard deviation).

	Error
CRF	52.21±11.73
GPstruct linear kernel	51.91±11.02
GPstruct SE kernel	50.42±11.09

Considering the high standard deviation, we can only conclude that GPstruct's performance is roughly comparable to the one of the CRF.

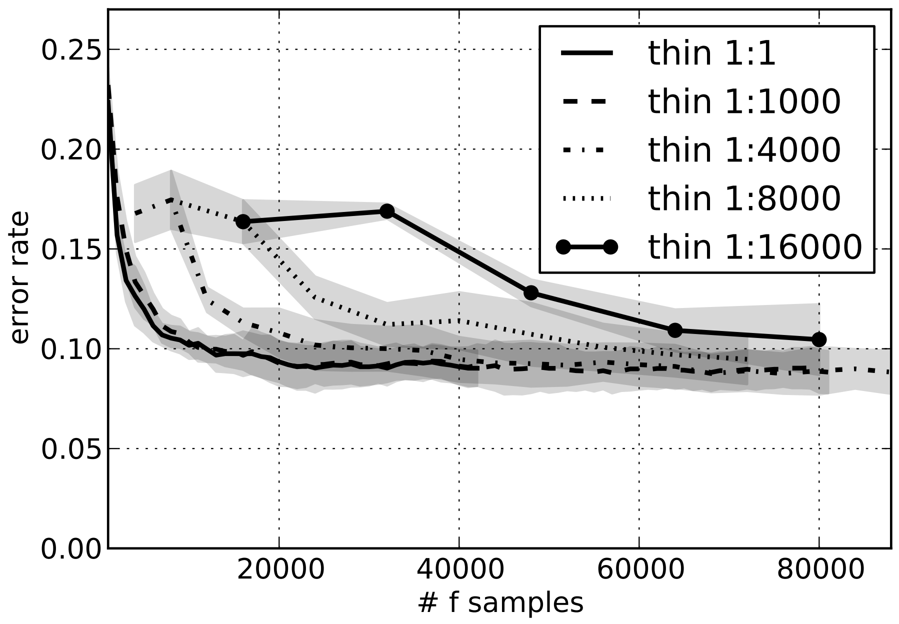
Since each session effectively represents one specific learning task, we compare the pairwise performances across 20 sessions between GPstruct SE kernel and CRF in figure 4.6.1. GPstruct outperforms the CRF baseline by more than 5% in five cases, while it underperforms it in one case. GPstruct SE kernel performs better than CRF.

4.7 Practical issues

Code Our code is available as an open-source Python package, which we hope will expose the GPstruct model more widely and encourage experimentation[9].

[8]Note that this experiment used hyperparameter learning using the prior whitening scheme described in algorithm 6.2. Only the squared exponential kernel hyperparameter σ^2 is learnt. It is given a scaled Gamma prior so that $\sigma^2/10^{-4} \sim \mathrm{Gamma}(1, 2)$ and is initially set to the median pairwise distance.

[9]`https://github.com/sebastien-bratieres/pygpstruct`



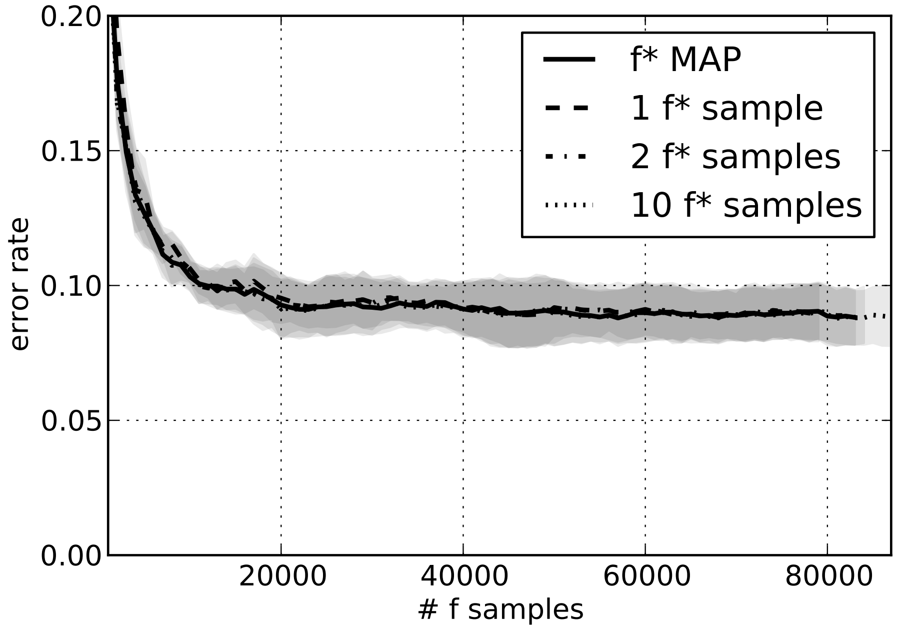


Figure 4.7.1: **Top**: Effect of thinning, i.e. sampling $\mathbf{f}_*|\mathbf{f}$ more rarely than every $\mathbf{f}$ sample. `Chunking` task, $\mathbf{f}_*$ MAP scheme, $h_b = 1$. **Bottom**: Effect of number of $\mathbf{f}_*|\mathbf{f}$ samples for each $\mathbf{f}$ sample. `Chunking` task, thinning at 1:1 000, $h_b = 1$.

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 ill-conditioned: this is best prevented by rejecting such a proposal by simulating a very low likelihood value.

How many $\mathbf{f}$ samples? The MCMC trace plots in figure 4.7.1 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 often to sample $\mathbf{f}_*|\mathbf{f}$? $\mathbf{f}_*$ 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 1:4000, seem to have very limited impact on the error rate, cf. figure 4.7.1 (top).

How many $\mathbf{f}_*|\mathbf{f}$ samples? 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 4.7.1 (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.

4.8 Cross-chain MCMC variance

To gain an understanding of the order of magnitude of variation between different random initialisations of the ESS chain, we have performed the following experiment.

We selected the segmentation task, and for each fold, ran 20 ESS chains to sample $\mathbf{f}|\mathcal{D}$. Due to the computational cost of the experiment, we have not conducted more experiments, and will have to extrapolate to other tasks and configurations.

We measured the following two metrics, which both decompose over test examples and sequence positions t, and make use of the predicted micro-labels $\hat{\mathbf{y}}_{*t}$:

- the Hamming error, used so far in this chapter:

$$HE = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{n \in \mathcal{D}_{\text{test}}} \frac{1}{T^{(n)}} \sum_{t=1}^{t=T^{(n)}} \mathbb{I}[\hat{\mathbf{y}}_{*t} = \mathbf{y}_t^{(n)}] \tag{4.8.1}$$

- the average negative log posterior marginals, which we will use in chapter 5 and 6:

$$ANLPM = \frac{1}{|\mathcal{D}_{\text{test}}|} \sum_{n \in \mathcal{D}_{\text{test}}} \frac{1}{T^{(n)}} \sum_{t=1}^{t=T^{(n)}} - \log p(\hat{\mathbf{y}}_{*t}|\mathcal{D}) \tag{4.8.2}$$

For each MCMC step (up to 250 000), we took the mean value, over the 5 data folds, of each metric. (In results in section 4.5.4, we report this value for HE for the final MCMC step.) We then compute the standard deviation of this value over all of the 20 MCMC chains. In figure 4.8.1, we plot both the mean value and the standard deviation of either metric.

At the end of the chain, the standard deviations are:

- around 3e-3 for HE, i.e. around 1% of the mean value
- around 4e-4 for ANLPM, i.e. around 1‰ of the mean value

We conclude that overall, in the results we report, the cross-chain variation due to random initialisation is negligible, as it is much less than the variation due to the train vs. test data split.

4.9 Comparisons with existing models

GPstruct shares common ground with the models described in section 3.7.

An important shared modelling assumption is the use of the MRF to specify the output structure and likelihood. MRFs are pervasive in the structured output prediction literature (Bakir et al., 2007; Nowozin and Lampert, 2010; Nowozin et al., 2014), specially in NLP (with the CRF (Sutton, 2012)) and computer vision (Blake et al., 2011). MRF posit Markov dependencies (not necessarily first order) in output atoms, and they make it hard to include long-range dependencies (patch or object information in vision, or grammatical dependencies in language), or additional sources of statistical knowledge such as n-gram information. As a consequence, GPstruct, just like the other models we have reviewed, has to use MRF inference repeatedly during training, and therefore faces the known issues and depends on solutions from the MRF literature: computational complexity of training, especially in non-tree-shaped MRF, techniques applicable to submodular potentials, approximate likelihoods, and others[10]. We will put such techniques to use in the next chapter.

Further shared issues are the exponential cardinality of the output label set, which one could really consider as the fundamental curse of structured prediction, and which goes hand in hand with the known problem of (a) local vs. global potential normalisation; (b) scale issues which occur almost immediately.

[10]Cf. Nowozin and Lampert (2010) for a review.

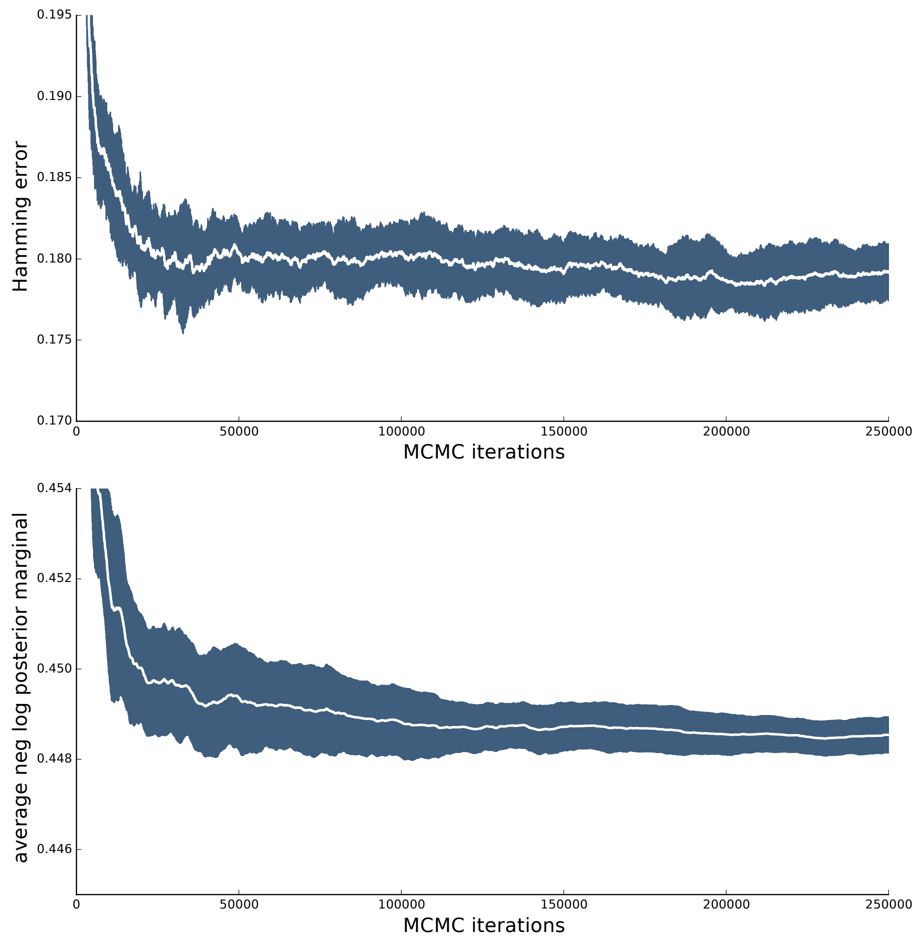


Figure 4.8.1: Mean and variance plots over 20 MCMC chains for HE and ANLPM metrics.

Let us now compare GPstruct with individual models.

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. In contrast, GPstruct replaces the dot product by $f(c, \mathbf{x}_c, \mathbf{y}_c)$, a non-parametric function of its arguments, and gives this function a GP prior.

In contrast to most of the GP literature, the GPstruct's Gaussian processes is indexed over MRF cliques, at the first level, and labels or label tuples, not just outputs. Furthermore, due to the parameterisation step mentioned in section 3.7, GPstruct uses input-output kernels, as opposed to output kernels in GP multi-class classification.

Comparing GPstruct to the KCRF model of Altun et al. (2004a), we can make the following points:

- In Altun et al. (2004a), binary parameters are absorbed in the unary parameters, so that the final parameter count corresponds to the number of unary parameters. In the presentation of this chapter, binary latent variables are kept separate.
- The GPstruct formulation starts from parameterisation and parameter tying. As a consequence, GPstruct easily supports different CRF shapes, likelihoods, kernels, tying schemes. Some of these will be demonstrated in the following chapter.
- In contrast, Altun et al. (2004a) uses a formulation that starts from kernels over entire label sequences, and takes advantage of the decomposition of $\mathbf{u}$ and α over cliques (Altun et al., 2004b). This makes it more difficult to formulate model variants such as those just cited for GPstruct.
- In particular, Altun et al. (2004a) has not been applied to other tasks than sequence labelling. Altun (2005) (section 4, page 85) leaves applications to more general graphs than chains as future work, using approximate inference schemes. The next chapter of this thesis fills that gap for GPstruct, by applying it to grid graphs using approximate likelihoods.

Compared to all the KCRF models, GPstruct's main advantage is the use of Bayesian inference instead of MAP or max-margin parameter estimation of the latent variables. This opens the door to principled model selection or comparative model criticism, and hyperparameter learning without grid search or cross-validation (cf. chapter 6). In addition, it contributes to making posterior probability estimates produced by GPstruct better than those of other models. This should stay true even in the face of attempts at probabilistic interpretations of SVM (cf. section 4.12), or attempts to have max-margin methods produce probabilistic output, which is known to be hard to calibrate (Stoyanov et al., 2011; Stoyanov and Eisner, 2012; Finley and Joachims, 2008; Yuille and He, 2012; Uřičář et al., 2013).

Now taking a broader perspective on MRF-based models for structured output prediction, note that MRF parameters (governed by a GP in GPstruct) could really be supplied by any function approximator. Jancsary et al. (2012) suggests using a random forest to model the energy function of a Gaussian MRF, allowing higher-order interaction. Do and Artières (2010) take the energy function from a neural network. Chen et al. (2015) also uses an neural network on top of an MRF, and jointly learns the MRF parameters and the neural network parameters.

Several sequence structured classification problems, especially in natural language processing, have seen great experimental success thanks to recurrent neural networks (RNN). Sequence labelling problems can be tackled by *synchronous* RNNs in which one output is produced each time one input is consumed. Sequence generation problems like machine translation can be modelled by *asynchronous* RNN, in which all input tokens are consumed, while a hidden representation of the input sequence is accumulated, and in a second stage, all output tokens are generated. The latter approach is referred to as "sequence-to-sequence" paradigm (Sutskever et al., 2014). How do such models compare to GPstruct? They generally require larger data sets to start performing well, but are capable of automatic feature extraction in the input, while (at least in the embodiment presented in this thesis) requires hand-crafted features. To adapt to a new problem, however, the neural network topology needs to be adapted, and sometimes the effort gained from automatic feature extraction is spent on network structure search, an approach which gains prominence in industrial applications. The major drawback of RNNs, however, lies in their inability to enforce or explicitly model inter-label constraints like CRFs, which the GPstruct models in its MRF component. For this reason, in several modern applications RNNs are topped with some graphical model to model label constraints (e.g. Ma and Hovy (2016)).

GPstruct builds on a probabilistic approach, with a prior parameterised by a kernel function, and a likelihood governed by the MRF topology. Alternatively, one can combine direct loss optimisation with the benefits brought by neural networks, not least the availability of reliable open-source libraries. Belanger and McCallum (2016); Belanger et al. (2017); Belanger (2017) generalise the energy function (which is equivalent to operating on the loss) from factor graphs to autoregressive energy functions, and relax discrete output spaces into continuous spaces, so that the energy function can be differentiated all the way through to inputs and parameters; in this architecture, complicated constraints between output labels can be implemented in the energy function, which GPstruct achieves both in the kernel function over output labels, and in the MRF distribution over output labels. Building on this work, Tu and Gimpel (2018) learn inference networks to bypass combinatorial optimisation or argmax operations, while Gygli et al. (2017) directly learn to estimate the loss function using gradient-based inference.

4.10 MAP variant of GPstruct

How to probe whether using a fully Bayesian parameter estimation procedure provides an advantage with respect to a MAP procedure ? This question arises in the comparison between GPstruct and KCRF schemes, for example.

In this section, we will compare the following two experimental configurations:

- (Bayesian method) predictions are obtained as described in section 4.4.1:

$$p(\mathbf{y}|\mathcal{D}) = \int p(\mathbf{y}|\mathbf{f}^*)p(\mathbf{f}^*|\mathbf{f})p(\mathbf{f}|\mathcal{D})\, d\mathbf{f}^*\, d\mathbf{f} \tag{4.10.1}$$

$$\approx \frac{1}{N_{\mathbf{f}}} \sum_{\mathbf{f}\sim p(\mathbf{f}|\mathcal{D})} p(\mathbf{y}|\mathbf{f}^*_{MAP}) \tag{4.10.2}$$

  with $\mathbf{f}^*_{MAP}$ obtained from a sample $\mathbf{f}$. This is "Bayesian" in that it uses MCMC samples from $\mathbf{f}|\mathcal{D}$. Note that it uses a single MAP value for $\mathbf{f}^*$, however, as we found out experimentally, early on, that this was sufficient.

- (MAP method) predictions are obtained as

$$p(\mathbf{y}|\mathcal{D}) = \int p(\mathbf{y}|\mathbf{f}^*)p(\mathbf{f}^*|\mathbf{f})p(\mathbf{f}|\mathcal{D})\, d\mathbf{f}^*\, d\mathbf{f} \tag{4.10.3}$$

$$\approx p(\mathbf{y}|\mathbf{f}^*_{MAP}) \tag{4.10.4}$$

  with $\mathbf{f}^*_{MAP}$ obtained from $\mathbf{f}_{MAP} = \arg\max_{\mathbf{f}} p(\mathbf{f}|\mathcal{D})$. This is obtained as follows:

$$\mathbf{f}_{MAP} = \arg\max_{\mathbf{f}} \log p(\mathbf{f}|\mathcal{D}) \tag{4.10.5}$$

$$= \arg\max_{\mathbf{f}} (\log p(\mathbf{f}) + \log p(\mathcal{D}|\mathbf{f})) \tag{4.10.6}$$

$$= \arg\max_{\mathbf{f}} \left(-\frac{1}{2}\mathbf{f}^T K^{-1}\mathbf{f} + \sum_n \log p(\mathbf{y}^{(n)}|\mathbf{x}^{(n)}, \mathbf{f}) \right) \tag{4.10.7}$$

We obtain the maximum of this concave function by setting the derivative to zero. The derivative of the second term, which sums over $\mathcal{D}$ with index n, with respect to the element f_c of $\mathbf{f}$ corresponding to clique c, is:

$$\frac{\partial \log p(\mathbf{y}^{(n)}|\mathbf{x}^{(n)}, \mathbf{f})}{\partial f_c} = \#c \in (\mathbf{x}^{(n)}, \mathbf{y}^{(n)}) - \mathbb{E}_{\tilde{\mathbf{y}}\sim p(\tilde{\mathbf{y}}|\mathbf{x}^{(n)},\mathbf{f})}[\#c \in (\mathbf{x}^{(n)}, \tilde{\mathbf{y}})] \tag{4.10.8}$$

where $\#a \in A$ (read "count a in A) is the number of instances of pattern a in set A.

Thus for each training sequence n, the first term is a count on the training data. The second term, the expectation, is computed as follows:

- when c is a unary clique $(\mathbf{x}, y_t)$: $\mathbb{E}[\#c] = p(y_t|\mathbf{x}, \mathbf{f})$ is the marginal obtained from the forward-backward algorithm (to be computed for each position t in the sequence)
- when c is a binary clique (y_t, y_{t+1}): $\mathbb{E}[\#c] = \sum_t p((y_t, y_{t+1})|\mathbf{x}, \mathbf{f})$ is the two-slice marginal, which can also be obtained exactly from the forward-backward algorithm

This result, which in its more general form can be formulated as: "at the maximum likelihood estimate, the empirical sufficient statistics are equal to the expected sufficient statistics", can be extended to exponential families in general, and is a consequence of the convex duality between maximum likelihood and maximum entropy problems in exponential families.

4.10.1 Experiments

We run the NLP tasks described above with the MAP and MCMC methods, and report the metrics HE and ANLPM defined section 4.8. We use a linear kernel for k_x, and $h_u = h_b = 1$. The gradient ascent procedure to estimate $\mathbf{f}_{MAP}$ uses the off-the-shelf BFGS implementation from the *scipy.optimize* library, with $\mathbf{f}$ initialised by a draw from the prior.

Results are presented in table 4.2.

On the `Japanese NE` task, GPstruct MAP does better than GPstruct Bayesian, and mostly so on the `Segmentation` task. On the `Base NP` task, GPstruct Bayesian does consistently better than MAP. Results on the `Chunking` task are mixed.

As it is, this experiment does not give a clear-cut answer. We have not reproduced these experiments using hyperparameter learning (cf. chapter 6), which for the MAP scheme would imply a costly cross-validation procedure; we believe results would be similar. As we discuss in section 4.12, we believe that the datasets and tasks we are using, despite being standard benchmarks for structured prediction output, cannot discriminate between these two rather good models. Different datasets and tasks might be more revealing.

4.11 Further GP models with structure

We now discuss several more distantly related statistical models, mainly to give the reader a complete overview.

GP regression networks (Wilson et al., 2012) tackle structured regression, i.e. the problem of regression over a fixed-size vector output $\mathbf{y}$. The input x may belong to an arbitrary set. The inductive bias realised by GP regression networks can be summarised as follows: we wish to model input-dependent signal and noise correlations

	MCMC ANLPM	MAP ANLPM	MCMC HE	MAP HE	MCMC HE | last $\mathbf{f}$
task= segmentation fold=0	0.369	*0.365	*0.155	0.166	0.193
task= segmentation fold=1	0.331	*0.315	0.134	*0.132	0.173
task= segmentation fold=2	0.354	*0.345	0.154	*0.152	0.193
task= segmentation fold=3	0.391	*0.379	*0.155	0.157	0.224
task= segmentation fold=4	0.356	*0.341	*0.143	0.146	0.219
task= basenp fold=0	*0.151	0.163	*0.044	0.057	0.074
task= basenp fold=1	*0.162	0.167	*0.048	0.051	0.086
task= basenp fold=2	*0.161	0.164	*0.048	0.054	0.072
task= basenp fold=3	*0.150	0.155	*0.046	0.052	0.072
task= basenp fold=4	*0.156	0.166	*0.050	0.056	0.080
task= chunking fold=0	0.329	*0.312	*0.067	0.080	0.163
task= chunking fold=1	0.361	*0.350	*0.082	0.086	0.174
task= chunking fold=2	*0.379	0.382	0.100	*0.096	0.184
task= chunking fold=3	0.342	*0.333	*0.086	0.089	0.167
task= chunking fold=4	*0.383	0.400	*0.094	0.101	0.178
task= japanesene fold=0	0.300	*0.276	0.069	*0.064	0.074
task= japanesene fold=1	0.266	*0.261	0.061	*0.058	0.068
task= japanesene fold=2	0.263	*0.240	0.051	*0.050	0.062
task= japanesene fold=3	0.314	*0.277	*0.059	*0.059	0.072
task= japanesene fold=4	0.224	*0.203	0.049	*0.047	0.054

Table 4.2: GPstruct Bayesian vs. MAP. Each row represents one data split for one given task. Every column represents one metric. Meaningful columns to compare: MCMC ANLPM vs MAP ANLPM, MCMC HE vs MAP HE. The * marks the better (lower) of the two scores. The column “MCMC HE | last $\mathbf{f}$” is the HE when computed using the last sample from $\mathbf{f}|\mathcal{D}$ as a point-wise estimate, and is given for information. Folds are displayed separately to illustrate the variance across folds.

between dimensions of the output. The output is modelled as a noisy mixture of GPs, in which the mixing weights $\mathbf{W}$ are GP-distributed:

$$\mathbf{y}(x) = \mathbf{W}(x)[\mathbf{f}(x) + \sigma_f \boldsymbol{\epsilon}] + \sigma_y \mathbf{z} \tag{4.11.1}$$

where $\boldsymbol{\epsilon}$ and $\mathbf{z}$ are independent Gaussian noise processes, and both $\mathbf{f}$ and $\mathbf{W}$ are independent Gaussian processes. In this fashion, the output is modelled as a spatially (i.e. w.r.t. x) adaptive combination of GP latent variables. Variational inference methods for this model were developed by Nguyen and Bonilla (2014).

Still for structured regression, in cases where we can build kernels over both the input and the output spaces, an interesting learning technique consists of building an implicit model of output correlations via a **kernel similarity measure** (Weston et al., 2002). The twin Gaussian processes of Bo and Sminchisescu (2010) 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 as a measure of discrepancy.

GP latent random fields[11] (Zhong et al., 2010) are a version of the GP latent variable model (Lawrence, 2005) augmented with discriminative information, as an improvement over the discriminative GP latent variable model of Urtasun and Darrell (2007). The input consists of points in $\mathbb{R}^d$, which all belong to some class. The aim of the model is to obtain a good representation of the input in latent space, with the help of the class membership information. The model places a Gaussian Markov random field prior on each dimension of the latent representation (the connectivity of the random field being imposed by the class membership). The likelihood $p(\text{observation}|\text{latents})$ consists of a Gaussian distribution with covariance matrix equal to the Gram matrix of a given kernel over the latent variables.

An interesting non-parametric extension of the SVM, parallel to GPstruct, is the **infinite SVM** (Zhu et al., 2011), which is based on a DP mixture of SVMs, which contains aspects of both Bayesian inference in modelling the DP mixture, and of kernel machines in the SVM component. In Zhu et al. (2014), this model was recast in the RegBayes framework[12], which we do not detail further here. It was extended to structured prediction and applied to speech recognition problems in Yang et al. (2014). These models' non-parametric character is not induced by GP as in GPstruct. Instead, they are based on DP models, and perform point-wise parameter estimation.

[11]Despite the similar name, GP random fields (Moore and Russell, 2015) are a method for scaling GP computation; instead of partitioning the GP input space, and learning separate GPs for each partition, a random field of (parameters of) GPs is built over the input space.

[12]We ought to remark that several works by Jun Zhu and collaborators are relevant to the study of the generative-discriminative interface, including a form of the HDP-HMM (Zhang et al., 2014).

4.11.1 Different definitions of “structure” for GP structured regression

The GPstruct model for structured classification, as presented in this chapter, is one of possible non-parametric GP models which exhibit “structure”. Here are suggestions for constructing different models for structured regression, which emerged during our research, but which we did not explore further. A complete review on vector-valued (multivariate) Gaussian processes is provided by Alvarez et al. (2011).

GPstruct uses a softmax link function, and as a consequence, it defines both an output structure (governed by the MRF) and a latent variable structure over the factors of the MRF (governed by the GP and the structure of its kernel). In the following models, we are discussing only the latent variable structure.

GPstruct as defined here has the global property that its Gram matrix $\mathbf{K}$ is sparse, with zeroes encoding marginal independence of the variables – a sensible design decision in the case of GPs, because deleting one variable will change K only locally, not globally, but will change K^{-1} globally, and also because it is hard to specify the inverse covariance of two points.

While GPstruct encodes structure in the CRF, we could define a Gaussian MRF which captures interdependencies between variables. Missing edges in a Gaussian MRF correspond to conditional independence of variables, and are encoded as zeroes in the inverse covariance matrix of the Gaussian distribution over the variables[13]. Formally, we could define this model as follows: $\mathbf{x}$ is of arbitrary shape, while output $\mathbf{y} \in \mathbb{R}^T$. Precision matrix $\mathbf{W} \in \mathbb{R}^{T \times T}$ specifies the inverse covariance of a Gaussian MRF over the latent variables f_t. We now define

$$\begin{aligned} \mathbf{y} &= \mathbf{f}(\mathbf{x}) + \epsilon \\ \forall t,\ y_t | f_t &\sim \mathcal{N}(0, \sigma^2) \\ f &\sim GP(0, k) \end{aligned} \tag{4.11.2}$$

where k is assumed to be a separable kernel, $\text{cov}(f_t(\mathbf{x}), f_{t'}(\mathbf{x}')) = k((t, \mathbf{x}), (t', \mathbf{x}')) = k_{\mathbf{W}}(t, t') k_x(\mathbf{x}, \mathbf{x}')$, where $k_{\mathbf{W}}$ has Gram matrix $\mathbf{W}^{-1}$. In this model however, nothing connects the output y_t to local features inside $\mathbf{x}$; if this is a modelling goal, it can be introduced by appending a multiplicative factor $k_{\text{local}}(\phi(t, \mathbf{x}), \phi(t', \mathbf{x}'))$ to the kernel.

We could also assign the structure to the noise (the residuals), with a model where

$$\begin{aligned} \mathbf{y} &= \mathbf{f}(\mathbf{x}) + \epsilon \\ \mathbf{y} | \mathbf{f}(\mathbf{x}) &\sim \mathcal{N}(\mathbf{f}(\mathbf{x}), \mathbf{W}^{-1}) \end{aligned} \tag{4.11.3}$$

A model of this type is explored by Rakitsch and Lippert (2013).

If we replace the Gaussian MRF by a Gaussian bidirected graphical model (de An-

[13]Sparseness in the inverse covariance matrix is a sufficient condition for conditional independence

drade e Silva and Ghahramani, 2006), missing edges encode marginal independence between variables, and appear as zeroes in the covariance matrix. This is not usually considered "structured", but could correspond to specific modelling needs.

Still assuming vector-valued latent variables $\mathbf{f}(\mathbf{x})$, we can impose structure over their components $f_t(\mathbf{x})$ by defining a Gaussian directed graphical model: let μ be the mean vector of $\mathbf{f}(\mathbf{x})$, and $\mathbf{g}(\mathbf{x}) = \mathbf{f}(\mathbf{x}) - \mu$ the centered version of the variables, then such a model can be expressed as $\mathbf{g} = \mathbf{W}\mathbf{g} + \mathbf{S}\mathbf{z}$, where $\mathbf{W}$ is the regression weight matrix (which can be restricted to be lower triangular, without loss of generality, provided the variables are ordered according to topological ordering), $\mathbf{S}$ is the diagonal matrix of noise covariances, and $\mathbf{z} \sim N(0, \mathbf{I})$ is the noise. Structure is built into the regression weights $\mathbf{W}$. The covariance matrix across values of t is then (cf. derivation in

Murphy (2012, section 10.2.5) and Bishop (2006, section 8.1)) $\text{cov}(\mathbf{x}, \mathbf{x}) = (\mathbf{I} - \mathbf{W})^{-1}\mathbf{S}\mathbf{S}^T(\mathbf{I} - \mathbf{W})^{-1T}$.

4.12 Conclusion

In designing GPstruct, two main benefits were key: the model can be engineered nicely, and it produces calibrated probabilistic predictions. We comment on how these benefits were (or were not) obtained, and pursue by pointing out some limitations of our work.

The model engineering advantage means it is easy for humans to add knowledge to the model, in the form of kernel design, form of the likelihood, feature extraction, dependencies and types of factors in the CRF governing the output structure. This chapter was dedicated to sequence models, and in the next chapter we adapt GPstruct to grid models and approximate likelihoods. It would be relevant to apply GPstruct to further tasks, for instance to parsing with probabilistic context-free grammars, which is regularly used as a benchmark for structured output prediction (e.g. Taskar (2004)).

Calibrated probabilistic predictions are important when the mere "best" prediction is not sufficient, but we need "error bars", an estimation of uncertainty. This is the case when a machine learning components is part of a pipeline, as is typically the case in NLP, and its output must be fed into further components while preserving and quantifying uncertainty in the predictions. An example of such a situation occurs in speech understanding: the exact words used in an utterance and output by the speech recognition engine are not crucial, but the semantic extraction engine has to marginalise over different word sequences which yield the same semantic representation, which requires summing over probabilities of recognised word sequences. Another important scenario is active or reinforcement learning, where exploration depends on expected information gain.

In such contexts, relevant probabilistic information includes the posterior probability of a prediction $p(\mathbf{y}|x, \mathcal{D})$, as well as marginal probabilities of micro-labels $p(\mathbf{y}_t|x, \mathcal{D})$, both for the best and for other predictions. In addition, probabilistic answers to queries, i.e. the probability mass assigned by the posterior to an event, is often important. For instance, in printed character recognition, we might need a probabilistic answer to the question “does this character sequence represent a bank account number?”, ignoring (i.e. marginalising over) the exact sequence of characters.

For classifiers which do not naturally produce probabilistic predictions, there are attempts to retrofit the model to output such information, e.g. probabilistic interpretations of SVMs[14]. An issue that arises frequently with such schemes is the missing calibration: said bluntly, there is no reason why a number which looks like a probability should be a probability. This motivates further efforts to calibrate retrofitted probabilistic output, as in Kuleshov and Liang (2015).

GPstruct, instead, should produce better calibrated probabilistic output by design; in the next chapter, comparing GPstruct to other models, we confirm this experimentally.

What are advantages of a kernelised Bayesian model compared to mainstream structured prediction approaches, such as SVMstruct and CRF? One common benefit lies in the domain knowledge which can be infused in the MRF structure (higher-order factors, network topology). Kernelised models have kernel design as an additional handle, whose potential we have only started to explore in this work; for instance one can define informative kernel (similarity) values between output labels, instead of an identity matrix. As a kernelised Bayesian model, GPstruct has some robustness against approximations, unlike other models which rely on optimisation. For instance, CRFs trained with pseudo-likelihood (cf. chapter 5 for a discussion) are known to perform poorly; SVMstruct has difficulties when exact energy minimisation is not available, as the cutting plan algorithm fails to find the worst margin violation.

We are aware of several limitations of the work presented in this chapter.

Experimental results in this chapter and the next, while showing a slight superiority of GPstruct beyond the state of the art, are not as strong as we expected. The experiments in section 4.5 and 4.6 did not give GPstruct a large winning margin, and results in section 4.10 are disappointing. In chapter 6, a related issue will be pointed out, and discussed in section 6.5: the interpretation of results is quite sensitive to the choice of metric.

We argue that different tasks and datasets are needed to establish under what circumstances GPstruct’s inductive bias is helpful. On NLP and video tasks, the methods we have tested are all very good, and seem to be hitting a ceiling in terms of performance. Our models may be overfitting, that is, fitting residual noise on these tasks. Arguably, synthetic tasks could be designed to display where GPstruct is at an advant-

[14]This issue has been worked on many times, mostly unsatisfactorily; cf. Platt (1999) for an early reference, and Franc et al. (2011) for a recent approach.

age, but would not necessarily illuminate real-world applicability for GPstruct. More credibly, new tasks should come from domains where probabilistic output makes a difference, as discussed above: active learning[15], reinforcement learning, Bayesian optimisation problems, and tasks with a downstream requirement that uncertainty on results is preserved.

A Bayesian statistician would find that our MCMC analysis can be improved. We have characterised cross-chain variance in section 4.8 and attempted to give practical guidance on settings of training configuration settings (hyperparameters, really) in section 4.7, but our analysis was largely experimental and heuristic. This thesis work could be furthered by a complete MCMC convergence analysis, taking into account effective sample size and convergence criteria. A more complete study would shed light on stopping criteria and thinning rates, in connection with desired levels of performance.

Finally, we have not explored GP sparsification techniques in our work[16]. The Kronecker structure of the particular kernel we are working with could be exploited to reduce memory footprint, probably at the expense of computation, so that a trade-off has to be considered. The Gram matrix can also be approximately factorised, and it remains to be seen what impact this would have on performance.

[15]One scheme to create such tasks consists of casting a supervised learning task into an active learning task, as in Blundell et al. (2015).

[16]cf. section 5.6.1 for related work on sparsification in variational inference schemes

Chapter 5

Scaling up GPstruct: pixel grid labelling

Having convinced ourselves that GPstruct is a useful model, the natural next step is to scale up the datasets in can handle. The most obvious limiting factors are the number of iterations needed in ESS, and the storage and computation requirements connected to the kernel matrix. In addition, GPstruct is expected to be "engineerable", in the sense that several pieces can be replaced by variants; we would like to see how it accommodates a new CRF topology, and a new expression for the likelihood.

Semantic image segmentation is a labelling task over pixels grids which poses all these challenges. It is much harder than text labelling tasks, because the number of pixels in a typical image is vastly larger than the number of words in a sentence. The task typically consists of separating zones of the image depicting different objects, such as persons, cars, the sky, buildings, and so forth. The number of object classes to be identified varies from application to application. Labels are often applied at pixel level, which distinguishes this task from other computer vision task such as object identification, where the output could come as a single image label, or such as joint object localisation and identification, where the output is a label jointly with a bounding box.

It is manifest that there are strong inter-pixel label agreement constraints due to the spatial continuity of physical objects. This makes it a typical application for structured prediction algorithms, which, to label the current pixel, combine constraints from neighbouring labels with information from a pixel patch centered at the current pixel.

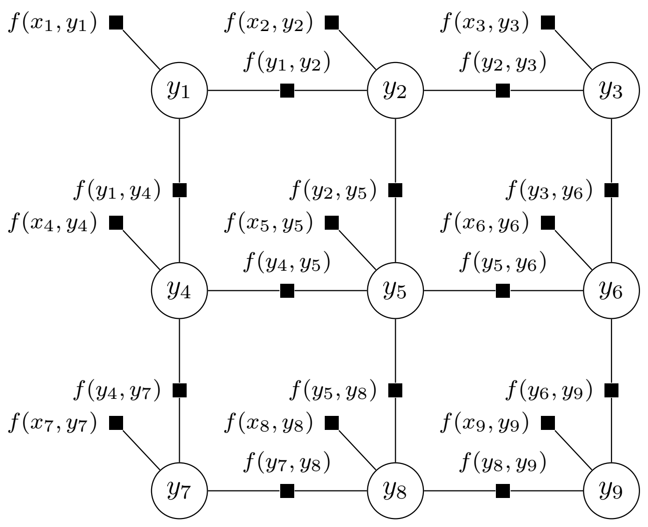


Figure 5.1.1: Grid factor graph with pairwise factors. There is one unary factor per pixel (the observed variable nodes for the x_t have been left out to not crowd the picture), and one pairwise factor.

5.1 Grid parameterisation

We now describe the parameterisation used to apply GPstruct to grid labelling tasks, and introduce our notation along the way.

We are given images which are grids (or lattices, in graphical model parlance) of pixels, and which we represent as grids $\mathbf{x}$ of feature vectors $\mathbf{x}_t$. A semantic segmentation of an image is represented by $\mathbf{y} = (y_t)_{t\in\{1..T\}}$, with $y_t \in \mathcal{R}$, the set of pixel-level semantic labels, and $|\mathcal{R}| = R$. Following the notation introduced in the previous chapter, we continue to note $t \in \{1...T\}$ the positions in $\mathbf{x}$ and $\mathbf{y}$. In contrast to the previous chapter, however, we assume that all images have the same shape, and hence total number of pixels T.

As before, training data is denoted by $\mathcal{D} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), ...(\mathbf{x}^{(N_{train})}, \mathbf{y}^{(N_{train})})\}$.

A CRF topology adapted to a grid output, similarly to the sequence CRF we worked with in the previous chapter, consists of unary factors for each pixel, and binary (edge) factors representing the potential between two neighbouring pixel labels, cf. figure 5.1.1. Unary factors depend on the pixel label y_t and on the corresponding input pixel feature vector $\mathbf{x}_t$. The neighbourhood for binary factors consists of the four closest pixels in the North, West, South, East positions (except for edge and corner pixels, which have fewer neighbours). We note $\mathcal{N}(t)$ the index set of the neighbour positions of a given position t.

An alternative topology could include binary factors for diagonal pixels in addition to these, or even larger neighbourhoods, or higher-order factors.

5.2 Approximations for scaling

Learning in GPstruct consists of obtaining

$$p(\mathbf{f}|\mathcal{D}) \propto p(\mathcal{D}|\mathbf{f})p(\mathbf{f}) \tag{5.2.1}$$

where the likelihood is given by the CRF model, and the prior is given by the GP prior over clique factors. In the approach introduced in the previous chapter, we represent this distribution as a collection of posterior samples obtained through an elliptical slice sampling MCMC algorithm.

Prediction of label $\mathbf{y}^*$ for test input image $\mathbf{x}^*$ consists of minimising an expected loss. As before, we use **per-pixel Hamming loss** rather than 0-1 loss, which more closely resembles losses which decompose over the image pixels than global losses. Our probabilistic problem for prediction is therefore the maximum posterior marginal inference (Marroquin et al., 1987) problem

$$\forall t,\ \arg\max_{y_t^*} p(y_t^*|\mathbf{x}^*, \mathcal{D}) \tag{5.2.2}$$

for which we marginalise

$$\begin{aligned} p(y_t^*|\mathbf{x}^*, \mathcal{D}) &= \int_{\mathbf{f}^*} p(y_t^*|\mathbf{x}^*, \mathbf{f}^*)p(\mathbf{f}^*|\mathbf{x}^*, \mathcal{D})\, d\mathbf{f}^* & (5.2.3)\\ &= \int_{\mathbf{f}^*} p(y_t^*|\mathbf{x}^*, \mathbf{f}^*) \int_{\mathbf{f}} p(\mathbf{f}^*|\mathbf{x}^*, \mathbf{f})p(\mathbf{f}|\mathcal{D})\, d\mathbf{f}\, d\mathbf{f}^* & (5.2.4)\end{aligned}$$

The first term is again the likelihood given by the CRF model, the second term is a multivariate Gaussian, and the third is the posterior obtained during training. We approximate the integrals by Monte Carlo sums using samples of the relevant distributions.

In the sequence case, the partition function in the likelihood $p(\mathbf{y}|\mathbf{f})$ could be obtained exactly using the forwards-backwards algorithm. In the grid case, we no longer have a tree-shaped CRF, which would allow the application of linear-time algorithms. An exact solution would be given by the junction tree algorithm, whose runtime, however, is exponential in the treewidth of the graph. For an $n \times n$ grid, the treewidth is n, therefore, for even small pixel grids, the junction tree algorithm is intractable.

To address the intractability of the likelihood normaliser, we use a surrogate likelihood, the **pseudo-likelihood** (Besag, 1975) (PL), as a drop-in for the true likelihood in the ESS procedure. The PL approximation can be explained in simple terms: we can write the likelihood in any MRF model as a product of likelihoods by repeated applications of the chain rule of probability:

$$p(\mathbf{y}) = \prod_t p(y_t | \{y_u | u < t\}) \tag{5.2.5}$$

Here, each factor depends on the "previous" variables in index order. The pseudo-likelihood consists of approximating each factor by its prediction given all other variables[1]:

$$p(y_t | \{y_u | u < t\}) \approx p(y_t | \{y_u | u \neq t\}) \tag{5.2.6}$$

Applied to an undirected graphical model, conditioning on all other variables is equivalent to conditioning on the Markov blanket, which consists of the neighbouring variables:

$$p(y_t | \{y_u | u \neq t\}) = p(y_t | \{y_u | u \in \mathcal{N}(t)\}) = p(y_t | y_{\mathcal{N}(t)}) \tag{5.2.7}$$

with the slight abuse of notation $y_{\mathcal{N}(t)} = \{y_u | u \in \mathcal{N}(t)\}$. Hence

$$p(\mathbf{y}) \approx \prod_t p(y_t | y_{\mathcal{N}(t)}) \tag{5.2.8}$$

The global normalisation necessary for the exact likelihood computation is replaced with a local, neighbourhood-based computation, which makes it tractable.

The full-data likelihood used in equation 5.2.1 is therefore approximated as

$$p(\mathcal{D} | \mathbf{f}) \approx \prod_n \prod_t p(y_t^{(n)} | y_{\mathcal{N}(t)}^{(n)}, \mathbf{f}) \tag{5.2.9}$$

The maximum PL estimator (as an estimator for model parameters) is known to be a consistent estimator in special cases such as the Boltzmann machine (Besag, 1975), but here we use it to approximate the intractable likelihood function itself. The use of PL in MCMC schemes for Bayesian parameter learning in Markov random fields dates back to Wang et al. (2000).

A detailed study of MCMC for Bayesian learning in non-trivial undirected graphical models is given in Murray and Ghahramani (2004), which conjectures that for general undirected models, there are no tractable MCMC methods that give the correct equilibrium distribution over parameters. Their pragmatic solution is to explore a variety of approximations for the normalising constant $Z(\mathbf{x}, \mathbf{f})$. The method of Murray et al. (2006) is not exactly suited to elliptical slice sampling, since it works on the Metropolis-Hastings acceptance rate. However, earlier, Parise and Welling (2005) have concluded that for fully observed MRFs (as in our case), PL is recommended over

[1]This is the only approximate step in pseudo-likelihood. Any different motivation for PL is therefore necessarily equivalent.

perfect sampling due to the computational burden of the latter, which is not balanced by a corresponding performance gain. In this research, we show empirically that PL works well when used as a likelihood approximation in the GPstruct model.

This approximation does not solve all scaling issues posed by learning on images, however. Consider a data set with 1000 square images of dimension 100×100 pixels. During MCMC, a single evaluation of the PL requires the evaluation of 1e7 local probabilities $p(y_t|y_{\mathcal{N}(t)})$, cf. equation 5.2.9, which is very high. In addition, the number of latent variables necessary to parameterise a binary segmentation output, such as foreground-background segmentation, is 2e7, and the input kernel matrix is square that size (4e14). For these reasons, we need to cut down on the number of parameters.

We achieve this by formulating an **ensemble method** as in Nowozin and Lampert (2010, section 4.5), in which each weak learner is trained on only a small subset of pixel positions $\mathcal{S} \subset \{1...T\}$. The corresponding subset PL is

$$p_{PL}(\mathcal{D}|\mathbf{f}, \mathcal{S}) = \prod_n \prod_{t \in \mathcal{S}} p(y_t^{(n)}|y_{\mathcal{N}(t)}^{(n)}, \mathbf{f}) \tag{5.2.10}$$

To produce diversified weak learners, a necessity for ensemble methods, we train them on disjoint subsets $\mathcal{S}_{n_{WL}}$ of pixels, while keeping $|\mathcal{S}|$ fixed. To clarify, we do not cover all pixel positions in this fashion, i.e. $\bigcap_{n_{WL}} \mathcal{S}_{n_{WL}} \subsetneq \{1...T\}$. This approach to subset-based ensemble learning is related to the Bayesian committee machine (Tresp, 2000)[2].

Much evidence shows that ensemble learners can exceed the performance of simple models. Examples of ensemble methods are bagging, boosting, random forests and their variants. The bagging algorithm (Breiman, 1996), which we apply, trains each weak learner from bootstrap data and combines individual predictions by uniform averaging or voting over class labels. We use the non-parametric bootstrap of Fushiki et al. (2005) to construct the predictive distribution from Monte Carlo samples.

At prediction time, for each t, we need to compute $p(y_t^*|\mathbf{x}^*, \mathbf{f}^*)$. The marginal predictive likelihood is again intractable due to the normaliser. We address this last issue by applying **tree-reweighted (TRW) belief propagation** (Wainwright and Jordan, 2008) as an approximation. TRW yields a tractable upper bound on the log partition function, which might give inconsistent marginal predictive likelihoods $p(y_t^*|\mathbf{x}^*, \mathbf{f}^*)$, in the sense that no joint distribution would yield those marginals. Despite this inconsistency, practically, TRW-based inference delivers state-of-the-art predictive performance (see for example Domke (2013)).

Note that we cannot use TRW instead of PL to obtain an approximation of the likelihood, because our PL is calculated over a subset of labels $\mathcal{S}$: this is natural to do with PL, but cannot be obtained with TRW; also, TRW gives an upper bound on the partition function, and hence a lower bound on the likelihood, but not an approxim-

[2]It would be interesting to try and match it with Dillon and Lebanon (2010)

ation, which instead is needed for MCMC training. Conversely, PL cannot be used instead of TRW for prediction because it is only a likelihood approximation, not a marginal MAP inference method.

5.3 Algorithm

We now restate the resulting algorithm for clarity.

1. (**Distributed stage**) for each weak learner $n_{WL} \in \{1..N_{WL}\}$

   (a) Generate bootstrap data $\mathcal{D}_{n_{WL}} \subset \mathcal{D}$ from the empirical distribution $p(\mathbf{x}, \mathbf{y}) = \frac{1}{N_{train}} \sum_{n=1}^{N_{train}} \delta(\mathbf{x} - \mathbf{x}^{(n)}) \delta(\mathbf{y} - \mathbf{y}^{(n)})$. Draw a pixel set $\mathcal{S}_{n_{WL}}$.

   (b) (**Training**) Based on the subset of pixel positions $\mathcal{S}_{n_{WL}} \subset \{1...T\}$, perform training by ESS using the PL expression from equation 5.2.10 applied to $\mathcal{D}_{n_{WL}}$, resulting in MCMC samples $\tilde{\mathbf{f}}_{n_{WL}}$.

   (c) (**Partial prediction**) For each test image $\mathbf{x}^*$, for each sample $\tilde{\mathbf{f}}_{n_{WL}}$, obtain samples $\tilde{\mathbf{f}}^*_{n_{WL}}$ from the multivariate Gaussian distribution $\mathbf{f}^*_{n_{WL}} | \tilde{\mathbf{f}}_{n_{WL}}$. For each sample $\tilde{\mathbf{f}}^*_{n_{WL}}$, obtain the predictive distribution $p(\mathbf{y}^* | \mathbf{x}^*, \tilde{\mathbf{f}}^*_{n_{WL}})$ using TRW. Using a Monte Carlo estimate, aggregate these in $p(\mathbf{y}^* | \mathbf{x}^*, \mathcal{D}_{n_{WL}}) \approx \frac{1}{N_{\text{predictive}}} \sum_{\tilde{\mathbf{f}}^*_{n_{WL}}} p(\mathbf{y}^* | \mathbf{x}^*, \tilde{\mathbf{f}}^*_{n_{WL}})$.

2. (**Aggregation stage**) Compute the complete predictive distribution as a uniform average $p(\mathbf{y}^* | \mathbf{x}^*, \mathcal{D}) \approx \frac{1}{N_{WL}} \sum_{n_{WL}} p(\mathbf{y}^* | \mathbf{x}^*, \mathcal{D}_{n_{WL}})$.

Note that we do not share binary latent variables in our algorithm; this is a reasonable idea, but requires synchronising the weak learners, which partly defeats distribution.

5.3.1 Algorithm complexity

The algorithm complexity can be computed separately for the distributed stage, where it scales linearly with the number of weak learners N_{WL}, and for the aggregation stage.

Learning occurs in each weak learner in parallel. For each weak learner n_{WL}, a one-time setup phase involves computing the kernel matrix from the features, and vector-multiplying by the Cholesky decomposition: with $|\mathcal{S}| \times |\mathcal{D}_{n_{WL}}| = N_{pixel/WL}$ subsampled pixels in the whole of $\mathcal{D}_{n_{WL}}$, the kernel computation takes $O(M^2)$, and a Cholesky factorisation takes $O(M^3)$, where $M = RN_{pixel/WL} + R^2$; however in practice, due to the repeat block-diagonal structure of the kernel matrix due to our particular choice of kernel, cf. equation 4.3.6, we work with $\mathbf{K}_x$ of length $N_{pixel/WL}$. Space requirements for learning are dominated by the storage of the Cholesky matrix. The ESS runtime is dominated by likelihood computations; ESS being a slice sampling algorithm, the number of likelihood computations in each MCMC step is not fixed. Each PL computations scales as $O(N_{pixel/WL})$. In practice the number of required MCMC iterations seems adequate when each WL runs for 12 hours.

Partial prediction (1c in section 5.3) is carried out on each weak learner, for each test image in sequence. Runtime is dominated by the TRW computation for each image, for each sample $\tilde{\mathbf{f}}_{n_{WL}}$ obtained by ESS. Space requirements are dominated by storing the marginals for each such sample, and by the train-test and test-test kernel matrices: for a weak learner n_{WL} with T pixels in each test image, these matrices contain $N_{pixel/WL}T$ and T^2 elements respectively, but they need not be present in memory simultaneously, since partial predictions are independent for different images. GPstruct has this in common with other kernel methods that at test time, it needs to evaluate, store and vector-multiply by the Cholesky decomposition of these matrices, and its prediction runtime is therefore linear with the test dataset size.

Aggregate prediction (stage 2 in section 5.3) is carried out for each image independently and is the fastest stage, consisting of averaging the marginals over weak learners.

5.4 Experimental setup

We assess the performance of these approximation techniques applied to the GPstruct model on a multi-class image segmentation task using two datasets. We compare GPstruct to a number of other techniques.

Stanford Background Dataset (Gould et al., 2009)[3] This dataset consists of 715 photographs of outdoor scenes, resised to 50×150 pixels. Each pixel in the image is labelled with one of 8 classes, i.e. {sky, tree, road, grass, water, building, mountain, foreground object}. We keep 80% of the data for the training set (572 images), and 20% for the test set (143 images). This split is repeated over 5 folds. The dataset was chosen over more challenging datasets such as the MSRC 23-class dataset[4] for comparison purposes: Domke (2013) uses it, and makes its code available.

LabelMeFacade Image Database (Fröhlich et al., 2010)[5] To check that results generalise, we conducted further experiments with this other, similar dataset. By convention, it is pre-split in 100 images for training and 845 images for testing. The original images are of different sizes and are resised to 50×150 pixels. Each pixel in the image is labelled with one of 9 classes, i.e. {building, car, door, pavement, road, sky, vegetation, window, unlabelled}.

For both datasets, we used the output of a random forest algorithm (Nowozin et al., 2011) as feature vectors. The feature vectors are prediction probabilities for each of

[3] `http://dags.stanford.edu/projects/scenedataset.html`
[4] `https://www.microsoft.com/en-us/research/project/image-understanding/`
[5] `https://github.com/cvjena/labelmefacade`

the pixel class labels.

We compare the performance of GPstruct to that of other models suited to the same task. Our aim is to assess the contribution of different aspects of our proposed model: the non-parametric property of the latent variables, the choice of learning technique (MCMC vs. margin optimisation), the "structured output" property of GPstruct obtained by factors on the inter-pixel edges, and the improvement brought by bagging.

`CRF PL` is a CRF model trained with pseudo-likelihood.

`CRF LBMO` (for loss-based marginal optimisation) is the model described in Domke (2013), and is considered to be state-of-the-art for vision CRF applications. While traditionally, CRF parameter learning optimises the likelihood, Domke (2013) suggests fitting parameters based on the quality of prediction of a given marginal inference algorithm, obtained by TRW or mean-field (we use TRW in our experiments), using truncated univariate logistic loss. Domke (2013) outperforms likelihood-based learning methods such as PL on difficult problems where the model being fit is approximate in nature, such as image denoising and image segmentation tasks.

`independent` is a variant of `CRF LBMO` based on the same training procedure, but where prediction ignores pairwise edges, preserving only unary features. This helps appreciating the contribution of the edge factors.

`CRF LBMO bag` applies the same bagging procedure as the `GPstruct` model to the `CRF LBMO` model, i.e. weak learners are trained on the training set, and their predictions combined to obtain an overall prediction. Since `CRF LBMO` is CRF-based, it produces probabilistic predictions, so that combining predictions consists of averaging marginals produced by each weak learner.

We train these CRF-based models with a regularisation parameter of 10^{-4} as in Domke (2013). Splitting regularisation parameters into unary and pairwise parameters, and giving the pairwise parameter a smaller value did not improve performance in our experiments. We use a data independent pairwise Potts factor. Our implementation was done in Matlab, and we use J. Domke's toolbox[6] for the CRF models.

The GPstruct model uses a squared exponential kernel between the pixels, with kernel width σ^2 set to $1/(\text{number of features}) = 1/(\text{number of classes})$.

We train $N_{WL} = 50$ weak learners in total. Each weak learner is trained on $N_{pixel/WL} = 5000$ pixel positions uniformly chosen in the training set images, irrespective of the training set size. The runtime of the training algorithm is proportional to N_{WL}. To explore the trade-off between predictive quality and runtime, we will carry out predictions using only a random subset of these 50 weak learners.

Computations for GPstruct were distributed on an Amazon Web Services cluster using MIT's Starcluster[7]. Each weak learner trained and issued partial predictions

[6] `http://users.cecs.anu.edu.au/~jdomke/JGMT/`
[7] `http://star.mit.edu/cluster`

(steps 1a through 1c in section 5.3) on a separate slave node, and the final aggregation step (step 2) was carried out on the master node.

A sample visualisation of marginals and predicted labels for all methods is in figure 5.4.1.

5.4.1 Runtimes

Training runtimes on the full training data set are around $360\,000$ sec for `CRF PL`, $61\,000$ sec for `CRF LBMO` and `independent`. The ensemble method we describe here allows trading performance against runtime, since we can choose how many weak learners to train. Each weak learner of `GPstruct` trains for around 12h ($43\,000$ sec), the same applies to each weak learner of `CRF LBMO bag`. `GPstruct` outperforms the other non-bagging methods with just around 5 weak learners (equivalent runtime $215\,000$ sec). It has equal runtime to `CRF LBMO bag` and outperforms it from around 15 weak learners.

How many images can be processed if no bagging is used? For a single image of shape $50 \times 150 = 7500$ pixels, we have as many unary latent variables. The limiting factor is the storage of the kernel matrix for $\mathbf{K}_x$ in equation 4.3.6, square of length $O(\sum_n T^{(n)})$. Assuming the largest reasonable memory requirement is 1.5 GB per CPU, we can store $\frac{1.5\,GB}{4\,byte} \approx 4e8$ single-precision floating numbers, for a kernel matrix length of 2e4. That is, we can store the kernel matrix for about 3 images. These values are in line with the discussion in Hensman et al. (2013).

5.5 Experimental research questions and results

5.5.1 What is the predictive performance of GPstruct?

We measured the per-pixel Hamming error of the predictions of each algorithm. For the Stanford Background dataset, we evaluated the error on a test set of 143 images, while varying the size of the training data set over the following values: $N_{train} \in \{25, 50, 100, 200, 300, 572\}$. We experiment with bagging configurations involving either $N_{WL} = 15$ or 50 weak learners. For the LabelMeFacade dataset, we vary the size of the training data set over $N_{train} \in \{1, 5, 10, 25, 50, 100\}$.

Results are in tables 5.1 and 5.2. For ease of interpretation, Stanford Background results are also plotted in figure 5.5.1.

`GPstruct` outperforms the other models consistently, even with only about 15 weak learners. Note that in all cases, each `GPstruct` weak learner ever only learns from $N_{pixel/WL} = 5000$ pixels. The decrease in error rate when increasing N_{train} is therefore only due to GPstruct being exposed to a larger variety of images. More independent training images wins over more dependent pixels from the same images; this corroborates Shotton et al. (2011) and Nowozin et al. (2011).

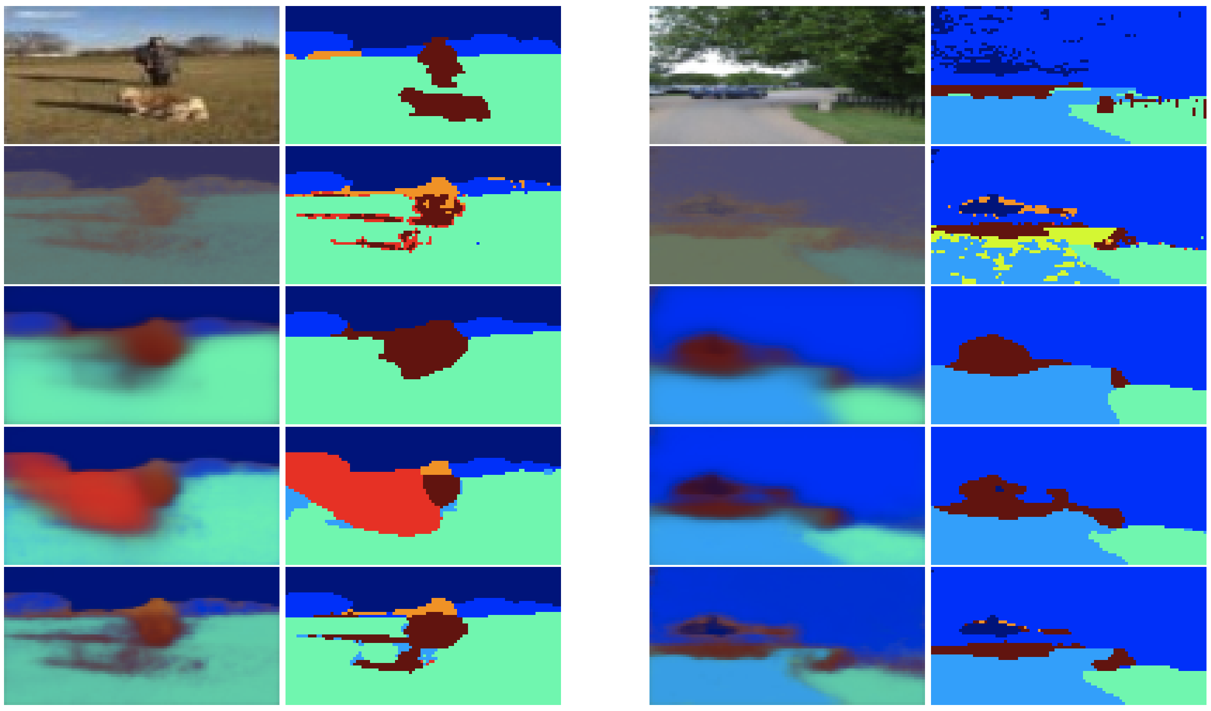


Figure 5.4.1: Semantic segmentation task. Example marginals (brightness level encodes certainty) and predicted labels from the Stanford Background Dataset. All methods use the same image features. **First row**: input image and true labels, **second row**: marginals and predicted labels of `independent`, **third row**: of `CRF PL`, **fourth row**: of `CRF LBMO`, **fifth row**: of `GPstruct`. The `independent` model performs reasonably well in predicting per-pixel segmentation, but makes rather noisy predictions, whereas `CRF PL` puts more emphasis on pairwise factors resulting in large same-label patches in predictions. `GPstruct` combines the good per-pixel segmentation of `independent` and smoothness of `CRF PL`.

`CRF PL` performs weakly, which is consistent with previous studies (Domke, 2013) which found CRF trained with PL suffers from model mis-specification and places too much emphasis on the pairwise factors. However, interestingly, PL performs well when used as a likelihood approximation for ESS in the GPstruct model.

5.5.2 Are GPstruct's predictions probabilistically calibrated?

We evaluate the average negative log posterior marginal, defined in section 4.10. A plot against N_{WL} is in figure 5.5.2.

The experimental setup is that of the Stanford Background Dataset, with a training set of $N_{train} = 50$ images, and $N_{WL} \in \{1, 2, ...50\}$. In order to reduce the variance when using very few weak learners, we averaged over repeats of the experiments; the number of repeats was set to $\max(5, 50/N_{WL})$, so that for $N_{WL} \in \{1..10\}$, for

Table 5.1: Error rate performance on test set of 143 images when training set size varies, $N_{train} \in \{25, 50, 100, 200, 300, 572\}$. `WL` denotes weak learner. Results are averaged over 5 folds. The best result and those results that are not significantly worse than it are highlighted in **boldface**. We used a paired Wilcoxon test with 95% confidence level as reference. Bayesian posterior inference of GPstruct generalises well at all training set sizes.

Stanford Background Dataset

N_{train}	25	50	100	200	300	572 (all)
`CRF PL`	57.14 ± 6.92	58.40 ± 6.46	47.19 ± 3.67	44.92 ± 2.05	60.00 ± 1.51	65.00 ± 1.33
`CRF LBMO`	30.60 ± 1.21	27.03 ± 0.67	26.09 ± 0.67	$\mathbf{25.31 \pm 0.65}$	$\mathbf{24.91 \pm 0.63}$	$\mathbf{24.78 \pm 0.61}$
`CRF LBMO bag 15 WL`	29.56 ± 0.77	25.68 ± 0.61	25.24 ± 0.54	$\mathbf{24.76 \pm 0.58}$	$\mathbf{24.50 \pm 0.61}$	$\mathbf{24.63 \pm 0.57}$
`CRF LBMO bag 50 WL`	29.56 ± 0.77	25.65 ± 0.61	25.20 ± 0.54	$\mathbf{24.73 \pm 0.58}$	$\mathbf{24.49 \pm 0.61}$	$\mathbf{24.61 \pm 0.57}$
`GPstruct 15 WL`	$\mathbf{26.61 \pm 0.65}$	$\mathbf{24.94 \pm 0.68}$	24.82 ± 0.63	$\mathbf{24.60 \pm 0.67}$	$\mathbf{24.56 \pm 0.72}$	$\mathbf{24.55 \pm 0.70}$
`GPstruct 50 WL`	$\mathbf{26.56 \pm 0.64}$	$\mathbf{24.90 \pm 0.67}$	$\mathbf{24.75 \pm 0.63}$	$\mathbf{24.51 \pm 0.68}$	$\mathbf{24.50 \pm 0.72}$	$\mathbf{24.53 \pm 0.69}$

Table 5.2: Error rate performance on test set of 845 images when training samples size varies, $N_{train} \in \{1, 5, 10, 25, 50, 100\}$. Bayesian posterior inference of GPstruct generalises well even with small training set sizes.

LabelMeFacade Image Database

N_{train}	1	5	10	25	50	100 (all)
`CRF LBMO`	55.99	34.84	32.11	28.37	28.01	27.81
`GPstruct`	34.05	30.23	28.57	27.86	27.84	27.80

Table 5.3: Error rate performance on the Stanford Background Dataset on a test set of 143 images, when fixing the training set size to $N_{train} = 50$ images, and using 15 weak learners. Results are averaged over 5 folds. The number of subsampled pixels is varied. The last column corresponds to the standard case in table 5.1 above, with $N_{train} = 50$ training images. The performance decreases only slightly when training from fewer pixels.

Varying Subsampled Pixel Number

$N_{pixel/WL}$	1250	2500	5000
`GPstruct 15 WL`	25.23±0.68	25.18±0.70	24.94±0.68

instance, 5 repeats were used. In all cases, results are averaged over 5 folds of the complete dataset, using $N_{test} = 143$ test images in each fold.

5.5.3 Is GPstruct's performance just due to bagging?

As can be seen in table 5.1 and figure 5.5.3 (from $N_{WL} = 10$ weak learners onwards), bagging also has a positive effect on the performance of `CRF LBMO`. However this effect is insufficient to bring it to par with `GPstruct`. This is also seen in the ANLPM metric in figure 5.5.2.

Therefore bagging alone does not explain why `GPstruct` performs better than `CRF LBMO`.

5.5.4 What is the influence of varying the number of sampled pixels?

The main experiment was carried out with $N_{pixel/WL} = 5000$ pixels sampled at random from the training images. Results of an analytic experiment in which we vary this number are shown in table 5.3. GPstruct's performance seems robust against sampling from fewer pixels, which illustrates its robustness in the small-data regime.

On a related note, due to pixel subsampling, there are fewer pseudo-likelihood terms than there are original likelihood terms: the former is the number of subsampled pixels on each image, while the latter is the number of pixels on the original image. We could have upscaled the pseudo-likelihood contribution to reverse this effect, but did not; despite this, our results are very good overall. We attribute this effect to the improvement provided by power-likelihood techniques (Antoniano-Villalobos and Walker, 2013; Walker and Hjort, 2001) in the presence of model mis-specification. In speech recognition, this same technique is applied routinely in the form of a language model scaling factor, cf. for example Gales and Young (2008), chapter 2, footnote 2.

5.5.5 What is the impact of PL and TRW approximations?

In order to assess the effect of approximating the likelihood with PL, and the predictive marginals with TRW, we conducted a small-scale experiment using the standard GPstruct, in which exact likelihood and prediction are tractable. For this, the Stanford Background Dataset images were resised to 10 by 10 pixels, and the multi-class segmentation problem was turned into a foreground-background (2-class) segmentation task (the foreground label is available in the original dataset). We perform 4 shuffles of the data, and in each select 100 images for training, and 100 for testing. Error rates are averaged over these 4 shuffles and plotted against ESS iterations in figure 5.5.4. The exact inference algorithms are implemented in libDAI (Mooij, 2010).

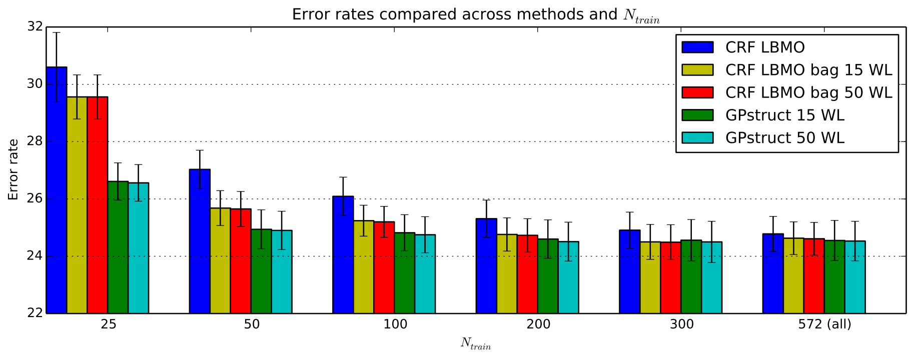


Figure 5.5.1: Stanford Background Dataset: plot corresponding to table 5.1 (`CRF PL`, whose results are much worse than the rest, is not represented).

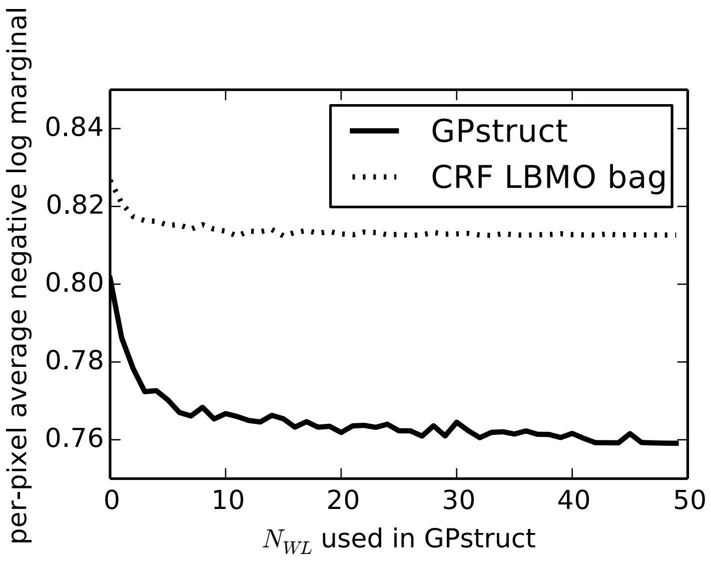


Figure 5.5.2: Quality of the posteriors, measured by the ANLPM metric. The experimental configuration is described in section 5.5.2.

This figure shows that the effect of approximating the prediction function has virtually no effect in the error rate (curves with exact and TRW predictions overlap). The approximation in parameter estimation using PL appears to be robust in GPstruct, leaving only a 1% gap in absolute accuracy with the exact likelihood. We can therefore conclude that the approximations we use, while efficient enough to make GPstruct scalable, are still robust enough to have a small impact on performance.

5.6 Scaling further

While the model design properties of GPstruct are convincing, the ESS training algorithm does not scale well. This motivated the work described in this chapter, with the help of chiefly three techniques: a pseudo-likelihood approximation of the likelihood, a TRW approximation for prediction, and a bootstrapping technique to distribute training.

In addition to using subsets of pixels to train weak learners, GP sparsification techniques (Snelson and Ghahramani, 2005b) applied inside each weak learner should allow further scale gains. GP sparsification techniques aim at reducing the time complexity of GP methods from $O(N^3)$ to $O(NM^2)$, where N is the number of training data points, and M is a user-selected number of inducing points. Indeed, almost all sparse GP methods exploit a conditional independence assumption between training and test sets, given a set of inducing points (see Quiñonero-Candela and Rasmussen (2005) for a unifying review of such methods).

Recent progress in sparsification, based on stochastic variational inference, has led to methods that can potentially process millions of data points (Hensman et al., 2013). Applying these methods inside each weak learner would result in hybrid methods that combine sampling and variational methods (see for example Welling et al. (2008)). A difficulty stems from the fact that these methods assume that the likelihood factors over the data (cf. e.g. Hensman et al. (2013), section 2, above equation 2), which the MRF likelihood used in GPstruct does not. Approaches to this problem are presented below, section 5.6.1.

More work on the kernel computation and storage could also bring benefits. Low rank approximations of the kernel are possible; in case we were using a linear kernel, in our applications, the Gram matrix would be of rank at most the number of features, which is low, and therefore the factorisation could be accurate. A further avenue of research could be on-demand computation of the kernel, to trade storage for computation.

If the size of the test images turned into a bottleneck, we could apply the ensemble approach described for training here to prediction, by distributing the stage in which we sample from $\mathbf{f}^*_{n_{WL}}|\tilde{\mathbf{f}}_{n_{WL}}$: this implies subdividing the latent variable set in test images, and sampling the posterior LV in parallel processes, before collecting them to

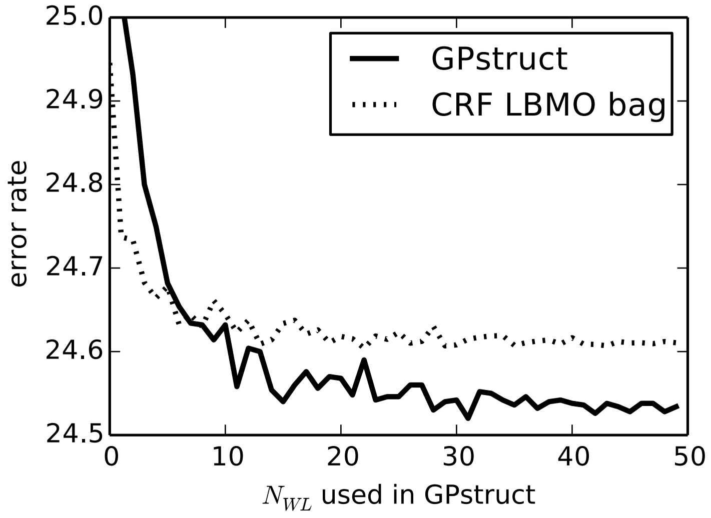


Figure 5.5.3: Speed-accuracy trade-off for GPstruct and CRF LBMO bag. The pixel-wise Hamming error is measured. The experimental configuration is the same as in section 5.5.2.

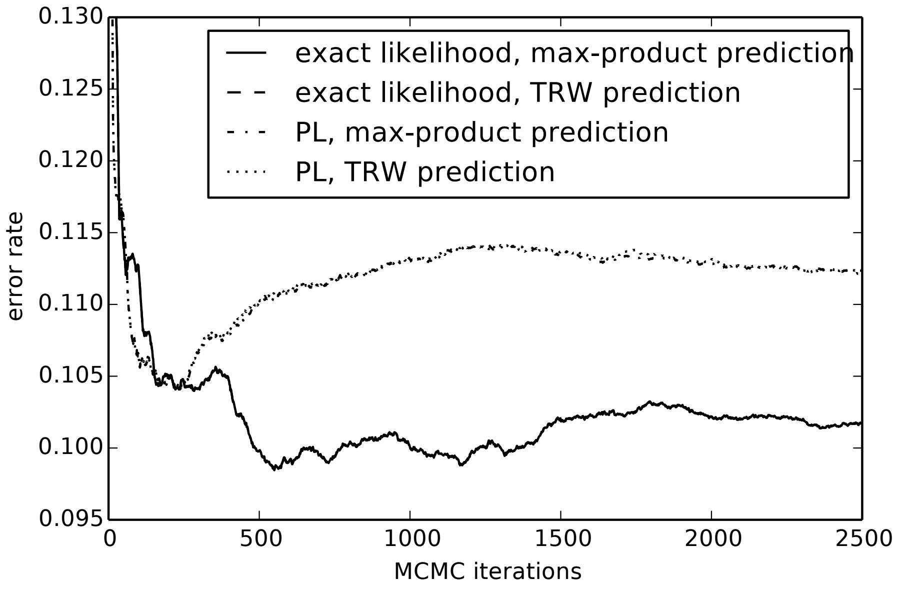


Figure 5.5.4: Effect of approximations in the standard GPstruct. All combinations of exact likelihood vs. PL and max-product prediction vs. TRW prediction are explored. The prediction approximation has virtually no effect on performance (curves with exact max-product and TRW prediction overlap), and the likelihood approximation proves very robust.

compute the predictive distribution using TRW. This technique would allow higher resolution images, but a balance must be found between the size of pixel subsets per weak learner, and the computation overhead. Further inspiration to accelerate prediction could come from compaction techniques such as Snelson and Ghahramani (2005a) or Balan et al. (2015).

5.6.1 Variational inference for GPstruct

After the publication of Bratières et al. (2015), two separate strands of work proposed variational inference techniques for training a sequence GPstruct model, the first in Srijith et al. (2016)[8], and the second in Galliani et al. (2017).

Both essentially aim at approximating the latent variable posterior with a tractable approximate (variational) distribution q; this is achieved by constructing a lower bound (a.k.a. ELBO, for "evidence lower bound") to the marginal log-likelihood, which takes the form

$$\log p(\mathbf{y}) \geqq -\mathrm{KL}[q(\mathbf{f})||p(\mathbf{f})] + \mathbb{E}_{q(\mathbf{f})}[\log p(\mathbf{y}|\mathbf{f})] \tag{5.6.1}$$

The second term of the right-hand side is referred to as the expected log-likelihood, and is usually the most difficult term to approximate.

We now analyse the techniques employed by both papers.

To apply a variational Bayes method, **Srijith et al. (2016)** approximates the sequence softmax likelihood, which does not factorise over positions t in a sequence, by several types of pseudo-likelihoods, which do factorise.

In the previous chapter, in the grid structure case, we were considering, as the neighbourhood, the local unary factor, as well as the binary factors connecting to neighbouring pixels. While still restricting to a single local unary factor, Srijith et al. (2016) allow more breadth for the binary factors, by considering, in principle, any shape of neighbourhood defined on the output sequence. The experiments cover three such shapes: the factor connecting to the previous (i.e. left) atom, the factors connecting to both the left and right atoms, and as the third shape, the same factors, with factors connecting the current atom to the -2 and +2 atoms (these 2nd order Markov factors are not present in a 1st order Markov chain, as we considered in chapter 4).

Like in our PL approximate likelihood, in Srijith et al. (2016) each binary factor appears several times, once in each "dependency set" to which it belongs, where one such dependency set is defined with respect to the current position t in the sequence. This implies that at prediction time, each factor will be estimated several times, in different dependency sets, and an iterative heuristic is introduced to converge to one

[8]Srijith et al. (2016) is a consolidated version of Srijith et al. (2014a) and Srijith et al. (2014b), which report on intermediate stages of the work.

single predicted value. In contrast, in this chapter, we use TRW for prediction, not PL, and are not confronted with this issue.

The following technical measures are implemented in the derivation of the variational lower bound for the expected log-likelihood, to proceed from Srijith et al. (2016)'s equation 8 to equation 9:

- the PL approximation allows the expectation to traverse the $\sum_t$, where t is the position in a sequence
- with Jensen's inequality, the expectation with respect to the variational distribution traverses the log in $\log Z$
- the moment-generating function of a multivariate Gaussian is used to obtain an analytic expression for terms of the type $\mathbb{E}_q[\exp f]$, where f is a single latent variable

The ELBO is optimised by gradient ascent under positive semi-definiteness constraints. To obtain an estimate of kernel hyperparameters, variational expectation maximisation is applied.

The experimental setup uses the same tasks and datasets as Bratières et al. (2015) (sequence tasks); the proposed inference method has similar accuracy but faster runtimes than ESS (no results are given concerning posterior calibration).

Galliani et al. (2017)'s approach does not rest on an approximation to the likelihood. Instead, it applies the variational inference method of Dezfouli and Bonilla (2015) to the GPstruct sequence model. Following Dezfouli and Bonilla (2015), the posterior approximating family is the mixture of Gaussians family. The KL term in the variational lower bound can be lower-bounded analytically. The expected log-likelihood term undergoes a stochastic approximation, which is improved using control variates. On this basis, a conjugate gradient descent optimiser can be used, and supplied with the approximate gradients.

In a second step, an inducing point sparsification based on Titsias (2009) allows reducing the computational complexity.

Experiments are conducted using both the sequence softmax likelihood, and on the "one-neighbour" pseudo-likelihood. Performance is comparable to training with the ESS algorithm, which shows that the approximations which were introduced cause no harm, and training time is reduced, so that medium-scale experiments, based on the original CRF++ data set, of which Bratières et al. (2015) used a subset, can be run.

Both reported methods are important improvements on the original GPstruct training method for sequences. Two comments come to mind. With such approximate schemes, it would be interesting to explore the computation versus accuracy trade-off. Also, in both cases, performance is on par with ESS-trained GPstruct, but also SVMstruct and CRF, as if the analysed tasks did not allow designating a clear winner. This prompts the remark that new, possibly harder or different tasks are needed, on which the accuracy of different methods would vary, as discussed in sections 4.12 and 6.5.

5.7 Conclusion

The GPstruct model has appealing properties which distinguish it among the structured prediction models, but it does not scale well due to both its $O(M^2)$ memory and $O(M^3)$ computation complexities, where M is the number of atoms to be labelled. This chapter's task, image labelling, constitutes a challenging test case for GP-based methods in general.

The main contribution of this research is a distributed ensemble method in which weak GPstruct learners produce partial probabilistic predictions based on subsets of latent variables, which can be aggregated for a high-accuracy final prediction. Each individual weak learner benefits from the GPstruct properties: it is kernelised, non-parametric and performs Bayesian inference.

The resulting method is shown to match and sometimes outperform state-of-the-art methods, and to scale very well.

GPstruct does share fundamental modelling issues with MRF-based models.

For instance, long-range dependencies are ignored, while in reality, objects cover not just a few, but very many pixels. Further, pixels are not a natural atomic representation of images (as tokens are in text), as they are tied to a particular scale of the scene, while image properties are independent of the scale[9]. For good performance in pixel-level labelling tasks, the computer vision community often uses edge features: this would amount to introducing a new type of pairwise factor in our model which connects neighbouring labels y_t and $y_{t'}$, but is also parameterised by edge features based on x_t and $x_{t'}$: $c(x_t, x_{t'}, y_t, y_{t'})$. For instance, these factors can encode directionality or colour/ intensity gradients, and can even by non-symmetric (i.e. inverting t and t' yields a different potential).

Using Bayesian model selection and hyperparameter learning, it can be "left to the data" whether to make use of these model design options. In the next chapter, we examine a particular case of these techniques: hyperparameter inference for GPstruct kernel hyperparameters.

[9]Nowozin et al. (2011) has some insights on the performance impact of varying image resolution.

Chapter 6

Hyperparameter inference

6.1 Motivation

In the previous chapters, we used arbitrarily set values for several covariance parameters, such as h_u and h_b, the scaling factors for the unary and binary kernel components, or the length-scale σ^2 in a squared exponential kernel. Our experiments demonstrate that these values provide decent results, but can we be sure they are set to their optimal value? They certainly influence the learning ability of our model and its predictive capability. Are we sure the values we chose are right? Will different datasets not call for different values of these parameters?

In addition, several choices exist in the model design, many of which we are not exploring in this thesis, for instance: which kernel type is best? Should we incorporate third or higher-order factors in the CRF design? Should we apply different kernels to different parts of the feature vector? The former type of questions relates to the choice of so-called *kernel hyperparameters,* and the latter to *model design.* In many ways, though, these questions can be addressed jointly. In this chapter, we will look at the treatment of kernel hyperparameters, but much if not all of the discussion carries over to model design options as well.

In statistical machine learning, the traditional answer to these problems is a form of systematic search called cross-validation. The data set is split into training set, test set and validation set. Given a choice of hyperparameters θ, a model M_θ is trained on the training set, and its performance according to some metric is evaluated on the validation set. In this way, the best model M_{θ^*} can be retained based on its performance on the validation set. To obtain an estimate of the performance of M_{θ^*} on unseen data, it is assessed on the test set.

In cross-validation, the search for the best hyperparameters proceeds by exploring a grid of values (often assuming continuous hyperparameters). Because the number of different models (each of them to be trained) is exponential in the number of grid values and of hyperparameters, this method is hard to apply to cases with three or more hyperparameters.

Therefore, improved methods attempt to guide search by focusing on regions of interest in hyperparameter space.

Reinforcement learning approaches consider this problem as an information acquisition process, with actions consisting of training a model and querying its performance, and feedback consisting of the validation set performance, as an estimate of the generalisation error. In other words, we seek to maximise a metric (the validation set performance), under the computational cost incurred by training the model and computing its performance. The field of Bayesian optimisation (Shahriari et al., 2016) was born out of a probabilistic perspective on this problem statement. Typically, the search is aided by (a) assuming that the variations of the function to be maximised (the validation set performance) are bounded, and (b) balancing the gain from locally maximising function in a known, good neighbourhood (exploitation) with information to be obtained by a totally new data point in an unexplored region (exploration).

In line with the probabilistic modelling approach used in this thesis, we can also offer a Bayesian treatment of the problem. We reflect our uncertainty in the best value of the hyperparameters θ (or indeed in model design) by turning them from fixed values into random variables in our probabilistic model. As random variables, we place priors $p(\theta)$ on them, thereby creating a *hierarchical probabilistic model.* After training the hierarchical model, we will obtain a posterior distribution of the hyperparameters $p(\theta|\mathcal{D})$, which we expect to be influenced by data and look different from the hyperpriors[1].

The change in the probabilistic model is illustrated in figure 6.1.1.

Before introducing hyperparameters, the prediction problem could be summarised as as follows ($\mathbf{y}*$ is the predicted output, $\mathbf{f}$ collects all the latent variables, θ are the (fixed) hyperparameters):

$$p(\mathbf{y}*|\mathcal{D},\theta) = \int_{\mathbf{f}} p(\mathbf{y}*|\mathbf{f})p(\mathbf{f}|\mathcal{D},\theta)d\mathbf{f}$$

where $p(\mathbf{f}|\mathcal{D},\theta)$, the posterior latent variables, are obtained from $p(\mathbf{f}|\mathcal{D},\theta) \sim p(\mathcal{D}|\mathbf{f})p(\mathbf{f}|\theta)$, where we recognise the likelihood and the latent variable prior. If we now decide that the hyperparameters θ are no longer fixed, but are random variables, with hyperprior $p(\theta|\eta)$, the prediction problem requires marginalising out both $\mathbf{f}$ and θ, which is often impossible analytically in practice.

As an alternative, approximating $p(\theta|\mathcal{D},\mathbf{x})$ point-wise, using a maximum likelihood estimate[2] (an approach know as empirical Bayes or type II maximum likelihood) is probably the most popular method in GP applications. This especially applies when

[1] We confirm this experimentally in section 6.4.3

[2] It could be turned into a MAP estimate by adjoining a prior or regulariser to the likelihood.

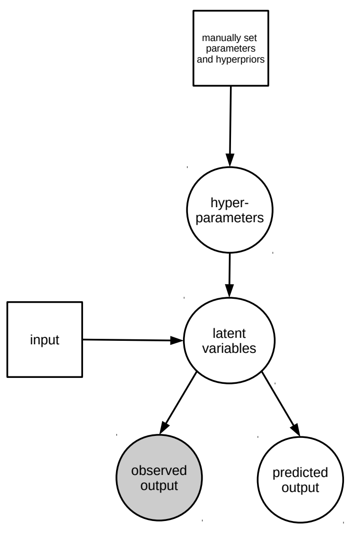


Figure 6.1.1: Sketch of probabilistic model with latent variables and hyperparameters

gradients of the (log) marginal likelihood can be computed analytically or approximated well, since gradient methods can come into play.

If we are prepared to place an explicit, possibly vague, hyperprior on the hyperparameters, we can treat the entire system homogeneously (following the argument "hyperparameters are just another type of latent variables"), and apply learning methods from the Bayesian toolkit: variational inference will approximate the latent variables' distributions by a parameterised family of distribution, and find the closest matching distribution by fitting (maximising) a lower bound on the likelihood; sampling methods will approximate the latent variables' distribution by a finite but arbitrarily high number of samples.

In this chapter we will apply the latter family of techniques, as sampling techniques are usually a first point of call when a new inference algorithm is needed: they are easy to derive from the probabilistic graphical model, as they require no analytic approximations, and will, though maybe at the cost of high runtime, give posterior samples eventually. A further motivation stems from the fact that the inference technique we have developed so far for GPstruct is an MCMC sampling technique, therefore developing additional sampling steps to sample not only from $\mathbf{f}|\mathcal{D}$, but also from $\theta|\mathcal{D}$, is very intuitive.

6.2 Bayesian hyperparameter inference for GPstruct

So far, we are obtaining the posterior over latent variables $\mathbf{f}|\mathcal{D}, \theta$ using a sampling approach, elliptical slice sampling. Sampling methods applied to the problem of obtaining the posterior over hyperparameters, $\theta|\mathbf{f}, \mathcal{D}$, are a natural continuation, and we will detail them in this section. Important references for this section are Murray and Adams (2010); Filippone et al. (2013); Yu and Meng (2011).

There have been attempts at marginalising hyperparameters using deterministic (variational) approximations of the posterior, which rely on quadrature for integration, see for instance Rue et al. (2009). Because these methods are difficult to use in the case of several hyperparameters, we will not investigate them further here.

Returning to sampling approaches, what Monte Carlo transition operators could we use?

The Metropolis Hastings sampler, with a proposal distribution of the form $p(\theta'|\theta) = N(\theta'|\theta, \alpha I)$, simulates a random walk.

Hamiltonian Monte Carlo (Neal, 2011) is a method based in a physical analogy: a point particle, characterised by position and momentum, moves without friction in a potential energy field. The movement of the particle is described by the partial differential equations of Hamiltonian dynamics. This method avoids the random walk behaviour of Metropolis-Hastings, and is therefore well suited to problems in which components of the parameter vector are highly correlated. It has several tunable parameters: the mass matrix M (which is often chosen to be of the form αI), the size of a leapfrog step ϵ, and the number of leapfrog steps L.

Slice sampling (Neal, 2003) is an auxiliary variable MCMC method which offers the advantage, in practice, of requiring no parameter to be tuned. (This applies in practice because slice sampling parameters like the stepping-out size make little difference to efficiency and are usually not tuned).

Variants, as well as other, more specialised methods exist; some are compared in Filippone et al. (2013).

All is not said with the choice of the transition operator. Indeed, sampling hyperparameters is made difficult in practice by the strong coupling between the hyperparameters and the latent variables: $p(\theta|\mathbf{f})$ is sharply peaked. Thus it is fruitful to work on a reparameterisation of the dependence between θ and $\mathbf{f}$ to reduce the coupling. Several methods have been developed for this, and can be combined with any of the transition operators above.

6.2.1 Canonical (forward) parameterisation

The canonical parameterisation for the connection between $\mathbf{f}$ and θ stems directly from the hierarchical model: $\mathbf{f}|\theta \sim N(0, K)$. It is called "sufficient augmentation[3]" in Yu and Meng (2011), because it can be viewed as a MCMC data augmentation scheme (in the sense that the MCMC procedure builds upon the auxiliary variable $\mathbf{f}$), and because $\mathbf{f}$ is a sufficient statistic for θ. Sampling a new θ' with this parameterisation boils down to applying the transition operator of our choice to the previous state θ, using the following distribution:

$$p(\theta|\mathbf{f}, \mathcal{D}) \propto p(\mathcal{D}, \mathbf{f}|\theta)p(\theta) = p(\mathcal{D}|\mathbf{f})p(\mathbf{f}|\theta)p(\theta) = \mathcal{L}(\mathbf{f})\mathcal{N}(\mathbf{f}|0, K_\theta)p(\theta) \tag{6.2.1}$$

The corresponding algorithm 6.1 is very straightforward, and can be compared to the progressively more complex alternatives which we will now discuss. We refer to this algorithm as simply "slice sampling" (and abbreviate SS) in the following. This is the parameterisation we will end up using preferentially.

Algorithm 6.1 Transition operator for θ using the forward parameterisation

```
**Input:** $\theta, \mathbf{f}$
**Output:** $\theta'$
sample $\theta'$ from $p(\theta|\mathbf{f}, \mathcal{D}) \propto \mathcal{L}(\mathbf{f})\mathcal{N}(\mathbf{f}|0, K_\theta)p(\theta)$
**Return** $\theta'$
```

6.2.2 Whitening the prior

As a first alternative, consider a white noise variable $\nu \sim N(0, I)$. We can rewrite $\mathbf{f}|\theta \sim N(0, K_\theta)$ in terms of ν using $\mathbf{f} = L_{K_\theta}\nu$, where $_{K_\theta}$ is the lower Cholesky decomposition of K_θ. Therefore starting from $\mathbf{f}$, whose prior distribution we know to be $\mathbf{f}|\theta \sim N(0, K_\theta)$, we can obtain a white noise variable $\nu \sim N(0, I)$ by positing $\nu = L_{K_\theta}^{-1}\mathbf{f}$.

This suggests the following procedure (algorithm 6.2): starting from θ and $\mathbf{f}$, obtain a new sample of $\theta|\nu, \mathcal{D}$; for this, apply the transition operator of our choice using $p(\theta|\nu, \mathcal{D}) \propto \mathcal{L}(\mathbf{f})p(\theta)$ (rederived below). At this point, to obtain a new sample of the posterior latent variables $\mathbf{f}|\theta, \mathcal{D}$, instead of applying our transition operator to $\mathbf{f}$ using K_θ (which yields a weakly coupled sample $\mathbf{f}'$), we will compute $\nu = L_{K_\theta}^{-1}\mathbf{f}$, and then "unwhiten" ν by taking as our new sample $\mathbf{f}' = L_{K_{\theta'}}\nu$, where $K_{\theta'}$ is computed from the new sample θ', not θ.

This scheme is called "whitening the prior" by Murray and Adams (2010), and "ancillary augmentation" by Yu and Meng (2011), because now, the data augmentation

[3]Cf. Yu and Meng (2011, section 1, page 533) for a discussion of the "sufficiency" vs. "ancillarity" terminology.

scheme rests on the auxiliary variable ν, which is an ancillary statistic of θ (in other words, $p(\nu)$ is free of θ, or ν is unconditionally independent of θ).

The sampling distribution is derived as follows:

$$\begin{aligned}
p(\theta|\nu,\mathcal{D}) &\propto p(\theta,\nu,\mathcal{D}) \\
&= p(\theta,\mathbf{f}=L_{K_\theta}\nu,\mathcal{D}) \left|\frac{\partial \mathbf{f}}{\partial \nu}\right| \\
&= p(\mathcal{D}|\mathbf{f},\theta)p(\mathbf{f}|\theta)p(\theta)\left|L_{K_\theta}\right| \\
&= \mathcal{L}(\mathbf{f}=L_{K_\theta}\nu)\mathcal{N}(\mathbf{f}=L_{K_\theta}\nu|0,K_\theta)p(\theta)\left|L_{K_\theta}\right| \\
&= \mathcal{L}(\mathbf{f}=L_{K_\theta}\nu)\frac{1}{\left|2\pi L_{K_\theta}L_{K_\theta}^T\right|^{\frac{1}{2}}}e^{-\frac{1}{2}\mathbf{f}^\mathbf{T}L_{K_\theta}^{-T}L_{K_\theta}^{-1}\mathbf{f}}p(\theta)\left|L_{K_\theta}\right| \qquad (6.2.2) \\
&= \mathcal{L}(\mathbf{f}=L_{K_\theta}\nu)\frac{1}{\left|2\pi\mathbb{I}\right|^{\frac{1}{2}}}e^{-\frac{1}{2}\nu^T\nu}p(\theta) \\
&= \mathcal{L}(\mathbf{f}=L_{K_\theta}\nu)\mathcal{N}(\nu|0,\mathbb{I})p(\theta) \\
&= \mathcal{L}(\mathbf{f}=L_{K_\theta}\nu)p(\nu)p(\theta) \\
&\propto \mathcal{L}(\mathbf{f}=L_{K_\theta}\nu)p(\theta)
\end{aligned}$$

We refer to this algorithm as “prior whitening” (PW) in the following.

Algorithm 6.2 Transition operator for θ using prior whitening

```
**Input:** $\theta, \mathbf{f}$
**Output:** $\theta', \mathbf{f}'$
compute $K_\theta$ and $L_{K_\theta}$
sample $\theta'$ from $p(\theta|\nu,\mathcal{D}) \propto \mathcal{L}(\mathbf{f})p(\theta)$ (cf. text)
compute $\nu = L_{K_\theta}^{-1}\mathbf{f}$
compute new $\mathbf{f}' = L_{K_{\theta'}}\nu$
**Return** $\theta', \mathbf{f}'$
```

6.2.3 Surrogate data method

A further alternative, presented in Murray and Adams (2010) and known as the “surrogate data” parameterisation, goes one step further (algorithm 6.3). The sampling distribution is derived as follows[4]:

$$\begin{aligned}
p(\theta|\eta, \mathbf{g}, \mathcal{D}) &\propto p(\theta, \eta, \mathbf{g}, \mathcal{D}) \\
&= p(\theta, \mathbf{f} = L_{R_\theta}\eta + m_{\theta,\mathbf{g}}, \mathbf{g}, \mathcal{D}) \left|\frac{\partial \mathbf{f}}{\partial \eta}\right| \\
&= p(\mathcal{D}|\theta, \mathbf{f}, \mathbf{g}) p(\mathbf{f}|\mathbf{g}, \theta) p(\mathbf{g}|\theta) p(\theta) \left|L_{R_\theta}\right| \\
&= \mathcal{L}(\mathbf{f} = L_{R_\theta}\eta + m_{\theta,\mathbf{g}}) \mathcal{N}(\mathbf{f}|m_{\theta,\mathbf{g}}, R_\theta) \mathcal{N}(\mathbf{g}|0, K_\theta + S_\theta) p(\theta) \left|L_{R_\theta}\right| \qquad (6.2.3) \\
&= \mathcal{L}(\mathbf{f} = L_{R_\theta}\eta + m_{\theta,\mathbf{g}}) \frac{1}{\left|L_{R_\theta}\right|} \mathcal{N}(\eta|0, \mathbb{I}) \mathcal{N}(\mathbf{g}|0, K_\theta + S_\theta) p(\theta) \left|L_{R_\theta}\right| \\
&\propto \mathcal{L}(\mathbf{f} = L_{R_\theta}\eta + m_{\theta,\mathbf{g}}) \mathcal{N}(\mathbf{g}|0, K_\theta + S_\theta) p(\theta)
\end{aligned}$$

We refer to this algorithm as “surrogate data” (SD) in the following.

Algorithm 6.3 Transition operator for θ using the surrogate data method from Murray and Adams (2010)

```
Input: $\theta, \mathbf{f}$
Output: $\theta', \mathbf{f}'$
Parameters: auxiliary noise covariance $S_\theta$
compute $K_\theta$, $L_{K_\theta}$
draw surrogate data vector $\mathbf{g}|\mathbf{f}, \theta \sim \mathcal{N}(\mathbf{f}, S_\theta)$
sample $\theta'$ from $p(\theta|\eta, \mathbf{g}, \mathcal{D}) \propto \mathcal{L}(\mathbf{f})\mathcal{N}(\mathbf{g}|0, K_\theta + S_\theta)$ (cf. text)
compute $\eta = L_{R_\theta}^{-1}(\mathbf{f} - m_{\theta,\mathbf{g}})$ with $R_\theta = (K_\theta^{-1} + S_\theta^{-1})^{-1}$
compute new $\mathbf{f}' = L_{R_{\theta'}}\eta + m_{\theta',\mathbf{g}}$
Return $\theta', \mathbf{f}'$
```

6.2.4 Further variants

Further variants are possible. Murray and Adams (2010) compares different parameterisations using slice sampling as the MCMC transition kernel. Filippone et al. (2013) compares different parameterisations with their own choice of MCMC transition kernel. One scheme they mention stems from Yu and Meng (2011), which suggests alternating the canonical and prior whitening parameterisations under the name of Ancillarity-Sufficiency Interleaving Strategy. Another scheme mentioned in Filippone et al. (2013) stems from Knorr-Held and Rue (2002), which suggests a Metropolis-Hastings strategy for jointly sampling $\mathbf{f}$ and θ. The joint proposal then undergoes an accept-reject decision. This addresses the main issue of the standard MH strategy above, where the accept-reject decision concerns $\mathbf{f}$ and θ separately, so that it is hard to find an acceptable $\mathbf{f}$ after accepting θ. To obtain a proposal distribution for $\mathbf{f}$, their strategy requires a Laplace approximation to $p(\mathbf{f}|\mathcal{D}, \theta')$ inside the MH procedure; this introduces an extra parameter, and requires a series of Cholesky factorisations.

[4]This expands on Murray and Adams (2010)’s equation 11 and uses their notation and definitions.

To conclude our discussion of design choices for hyperparameter sampling, we must remark that its computation runtime is orders of magnitude larger than that of latent variable sampling. We are assured that in infinite time, any of the proposed MCMC operators will explore the entire posterior hyperparameter space; however, to explore this space faster, it makes sense to interleave fast updates of $\mathbf{f}|\mathcal{D},\theta$ before one expensive update of $\theta|\mathbf{f},\mathcal{D}$ (or possibly jointly $\theta,\mathbf{f}|\mathcal{D}$), which will be based on the last sample $\mathbf{f}'$. This improvement, inspired by Murray and Adams (2010), will reduce the correlation between subsequent samples of the hyperparameters at little cost.

6.3 Geweke's "Getting it right" tests

Among inference methods for probabilistic models, MCMC algorithms and several other stochastic methods stand out because they offer the theoretical guarantee that sampling for "long enough" yields samples from the target distribution, so that unbiased estimators of quantities of interest can be constructed from MCMC samples.

However, their manual implementation is delicate and error-prone. Errors can occur at several levels.

Once a family of MCMC algorithm has been chosen, for instance Gibbs sampling, slice sampling, or simulated annealing, the method must be applied to the probabilistic model of interest. This requires determining which quantities (typically conditionals) have to be computed, in what order, within which algorithmic control flow of acceptance or other conditions, and iterations. Errors can be introduced already at that stage, as demonstrated by Geweke (2004) on a synthetic example in his section 3 (error examples 2 and 5), and on a real example, in his section 4.

Not unlike their variational counterparts, their derivation often requires pages of calculations, in which mathematical (algebra) errors can occur, such as sign errors, drawing from the wrong distribution, etc.

The computer implementation sometimes does not correspond to the pseudo-code for various reasons. Real-life examples include errors in function calls, wrong choice of a library function, mistake in parameters, overlooked defaults, or wrong overloaded function choice.

Finally, despite falling into the same category, errors in the last level of implementation, namely numerical or library bugs, occur but they are very rare.

What makes MCMC implementations hard to debug is that, judging from statistics obtained from the MCMC chain, such as likelihood plots and others, such bugs are confounded by convergence issues. Correctness is a necessary, not sufficient, condition for convergence on the correct stationary distribution. An incorrect algorithm may even perform well on benchmarks. Convergence itself is hard to assess, and there is an ample body of work on MCMC convergence checks, both in the form of literature (Gelman et al., 2009, section 11.6) and implementations (cf. the CODA package in R, Plummer et al. (2006), or convergence checking in PyMC (Salvatier et al., 2016)). There is not much work on correctness checks, however, and we could extend this criticism to stochastic machine learning in general[5].

In response to this lack, this section describes the correctness test which was applied to GPstruct inference, both in the case of latent variables (elliptical slice sampling) and hyperparameters. Our approach builds on the seminal paper by John Geweke (Geweke, 2004), who set out to identify MCMC implementation errors in his own papers of the previous decade. We will briefly describe his proposed method, before describing our adaptation of it.

For this section, we assume a two-level generative model: parameters θ, distributed according to prior $p(\theta)$, generate observed data x i.i.d. according to $p(x|\theta)$. We will usually assume our forward implementations of the prior $p(\theta)$, and the likelihood (or conditional) $p(x|\theta)$, to be correct, as they are simple and usually consist of sampling from a parametric distribution. In contrast, the posterior $p(\theta|x)$ is implemented as an MCMC procedure and needs to be tested for correctness.

Geweke's suggested procedure is simple: we will produce joint samples of (θ, x) in two different ways, take real-valued functions of the samples, and apply a goodness-of-fit test to check whether their function values stem from the same distribution. The test can be a Kolmogorov-Smirnov (KS) test for continuous distributions, or a χ^2-test for discrete distribution. We will assert that we have uncovered an implementation error if the statistic rejects the null hypothesis.

More specifically, the suggested procedure is:

- define a number of L_2-integrable scalar test functions $(\theta, x) \mapsto g(\theta, x) \in \mathbb{R}$
- obtain samples (θ^m, x^m) according to the *marginal-conditional* procedure (the first of two procedures, effectively a forward sampling procedure), which amounts to iterating
  - sample θ^m from $p(\theta)$
  - sample x^m from $p(x|\theta^m)$
- obtain samples (θ^m, x^m) according to the *successive-conditional* procedure (the second of the two procedures, which involves iterating between forward and backward sampling), which initialises θ^0 from $p(\theta)$ and then iterates (cf. figure 6.3.1)
  - sample x^m from $p(x|\theta^m)$
  - sample θ^{m+1} from $p(\theta|x^m)$

[5]cf. the useful Schaul et al. (2013) for a counterexample

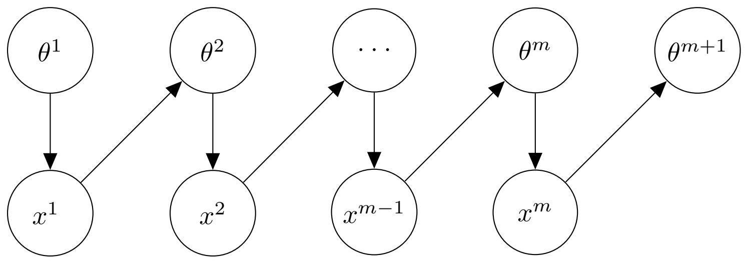


Figure 6.3.1: Graphical model illustrating the successive-conditional procedure.

- for each m, each test function g, and each of the two procedures, compute the value of $g(\theta, x)$

At this point, the most intuitive and popular decision tool is, for each g, a PP-plot (for probability-probability) or QQ-plot (for quantile-quantile) to compare the two distributions visually[6]. The visual inspection of the plot, and of the successive-conditional distribution plot, may already hint at the cause of the error.

For cases where a single test result per statistic g is desired (as opposed to a plot which needs to be analysed)[7], a goodness-of-fit test can be conducted between the two samples, as we describe in our adapted procedure below.

6.3.1 Our variant of the test

We adapt the Geweke procedure slightly, by making use of the property that the empirical distribution of θ will approach the marginal distribution of θ (with unobserved x, since we are marginalising over x), that is the prior $p(\theta)$. This is clearly the case in the marginal-conditional procedure.

There are two ways of proving this in the successive-conditional procedure, in addition to the proof in Geweke (2004).

First, we can consider x as an auxiliary variable added to the model containing only the θ node, for the sake of introducing an MCMC transition kernel making use of x.

A different, inductive argument consists of viewing the model as in figure 6.3.1, under the assumptions that $p(\theta^m) = \pi(\theta^m)$ with π the prior over θ, that $p(x^m|\theta^m) = L(\theta^m; x^m)$ is given by the likelihood function, and that

$$p(\theta^{m+1}|x^m) = \frac{L(\theta^{m+1}; x^m)\pi(\theta^{m+1})}{p(x^m)} \tag{6.3.1}$$

[6]cf. a typical use case in Grosse and Duvenaud (2014)

[7]Note that Geweke's original intent seems to have been combining both statistics into a single one (cf. Geweke (2004), eq. 6), which converges to $N(0, 1)$ in probability, and presumably to conduct a normality test. His BACC software does not seem to be available any more.

is the posterior. It results that the marginal distribution of θ^{m+1} is the distribution of θ^m. Indeed

$$\begin{aligned} p(\theta^{m+1}) &= \int p(\theta^{m+1}|x^m)p(x^m)\,dx^m \\ &= \int \frac{L(\theta^{m+1};x^m)\pi(\theta^{m+1})}{p(x^m)}p(x^m)\,dx^m \\ &= \pi(\theta^{m+1})\int L(\theta^{m+1};x^m)\,dx^m \\ &= \pi(\theta^{m+1}) \end{aligned} \tag{6.3.2}$$

Now, the prior over θ is supposed to be known analytically as $\pi(\cdot)$, so as a test, we will simply compare θ samples from the successive-conditional procedure to the analytic prior.

We deal with continuous distributions in this chapter, and would like to apply the Kolmogorov-Smirnov (KS) test. A caveat is in order here: the computation of the two-sample KS quantiles relies on two conditions:

1. the samples need to be independent[8], which is an issue when the reference distribution's parameters are estimated from the data; in our setup, however, this condition is verified

2. the samples need to be uncorrelated[9]. The successive-conditional sample clearly does not meet this condition, since it is produced from an MCMC procedure.

To approximately meet the second condition, and reduce the autocorrelation almost down to 0, we suggest thinning to the effective sample size before applying the KS test. Ignoring this precaution practically results in more frequently rejecting the null hypothesis[10]; when debugging MCMC samplers like here, this implies more often signalling a faulty implementation; therefore, it only makes the test more demanding, not lenient, which makes it a minor ailment.

We found several publications and tutorials guilty of this methodological oversight (Heaukulani et al. (2014, appendix D), Ardia (2008, section 3.2)), although some statistical literature points out this very problem and suggests solutions: computationally intensive bootstrap methods (Olea and Pawlowsky-Glahn, 2009), or modelling the correlation (Weiss, 1978).

6.3.2 Experiment: testing elliptical slice sampling implementation

We now switch notation back to GPstruct notation, using **f**,**y** instead of θ, x.

[8]cf. Babu and Feigelson (2006) for a discussion
[9]cf. Olea and Pawlowsky-Glahn (2009) for a discussion
[10]cf. Olea and Pawlowsky-Glahn (2009, eq. 7 and 8)

We wish to check the correctness of our elliptical slice sampler applied to GPstruct. This test will enable us to probe not only the MCMC code, but also the code computing the likelihood.

Since $\mathbf{f} \sim N(0, K)$, each of its elements' marginal prior is $f_i \sim N(0, K_{ii})$: we can apply the Kolmogorov-Smirnov test for each f_i, under the null hypothesis H_0 that each thinned sample $(f_i^{(n)})_n$ obtained from the Geweke procedure stems from this marginal prior. If the MCMC code is bug-free (and the test carried out correctly) the rate of H_0 rejection asymptotically equals the level of significance assigned to the test (we will use the usual $\alpha = 5\%$).

In our experiments, to test GPstruct code in realistic conditions, rather than isolating the MCMC transition operator, we designed a simplified, synthetic data structure consisting of length 1 chains, i.e. with a single label $y \in \{1...5\}$. There are 5 data points, one for each value of the label. The features are a one-hot encoding of the micro-label, therefore they allow perfect prediction. There are no binary latent variables, but only unary variables, and the covariance function over the unary latent variables is identity[11]. The Geweke procedure produces 2000 samples from $\mathbf{f}|\mathcal{D}$, by alternately applying the MCMC transition operator to latent variables $\mathbf{f}|\mathcal{D}$, and forward sampling $\mathcal{D}$.

We consider the sample sequence obtained at each position i in $\mathbf{f}$ separately. According to the position i, the effective sample size is roughly between 250 and 400. Each sequence $(f_i^{(n)})_n$ is thinned to its effective sample size, in order to obtain approximately uncorrelated sequences. The resulting thinned sample is compared, using the one-sample KS test at a significance threshold of $\alpha = 5\%$, to the true marginal prior which we set to $f_i \sim N(0, 1)$.

Results

When averaging over 5 random seeds, we find a rejection rate of 5%. This corroborates that the elliptical slice sampling MCMC implementation in our code is correct.

6.3.3 Experiment: testing hyperparameter sampling implementations

We use the same synthetic data as above. This time, the kernel is the exponential ARD kernel

$$k_{\text{exponential ARD}}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}\sum_i \frac{(x_i - x_i')^2}{(\exp \psi_i)^2}\right) \tag{6.3.3}$$

. The hyperparameters are stochastic, and their prior is either

- uniform: $\psi_i \sim U(\log 10^{-3}, \log 10^2)$, or

[11]From a theoretical point of view, we could have simplified down to a binary classification task with only one data point, but more code coverage could be achieved by the described synthetic data setup.

- Gaussian: $\psi_i \sim N(0, \log 10)$

The steps of the Geweke successive-conditional procedure are:

- from ARD hyperparameters θ, for a set of data points $\mathcal{D}_x$, generate $\mathbf{f} \sim N(0, K_\theta)$ and $\mathbf{y} \sim \text{softmax}(\exp(\mathbf{f}(x, \mathbf{y})))$.
- using likelihood function based on new data set $\mathcal{D}_x, \mathcal{D}_y$, and hyperparameter prior, apply MCMC transition operator to θ: obtain θ'

The transition operator uses one of the three parameterisations presented in section 6.2: forward parameterisation, prior whitening, or surrogate data.

We run the Geweke procedure for 20 000 steps, across 32 random seeds. As above, we thin each sequence of hyperparameters, and carry out the KS test each time. For each configuration (MCMC parameterisation, hyperprior uniform or Gaussian), we average over random seeds.

Results

This test allowed us to catch mistakes in the implementations, several of which remain undetectable if one relies only on MCMC monitoring tools[12]. Such mistakes would systematically exhibit very high reject proportions (above 90%). In addition, a very helpful tool for diagnosing such mistakes is the cumulative distribution plot of the prior and of the empirical distribution. Figure 6.3.2 shows such plots in cases where the KS test rejects or does not reject the null hypothesis. In practice, we have found the KS test to be relatively conservative.

In the case of incorrect implementations of the MCMC transition operator, we have observed a systematically different shape of the empirical distribution: for instance, it could be shifted, squeezed, or asymmetric, as can be seen in figure 6.3.2.

When applying the experiment to the corrected and final version of the source code, we obtain the following results.

% KS rejections	SS	PW	SD
uniform prior	6%	3%	7%
Gaussian prior	4%	6%	6%

[12]One such mistake, found in our Python code, turned out to originate in Murray and Adams (2010)'s accompanying Matlab code. Specifically, their implementation of the surrogate data method in the file `update_theta_aux_surr.m` does not use the expression for $p(\theta|\eta, \mathbf{g}, \mathcal{D})$ derived in Murray and Adams (2010) equation 11 and in our equation 6.2.3, but the alternative $p(\theta|\eta, \mathbf{g}, \mathcal{D}) \propto p(\mathcal{D}|\theta, \mathbf{f}, \mathbf{g})p(\mathbf{g}|\mathbf{f}, \theta)p(\mathbf{f}|\theta)p(\theta) \left|L_{R_\theta}\right|$, implemented in source code in lines 124 to 147. In line 146, the expression for `Lg_f`, corresponding to $\log p(\mathbf{g}|\mathbf{f}, \theta) = \log \mathcal{N}(\mathbf{g}|\mathbf{f}, S_\theta)$, is missing a 1/2 in the second additive term. The Matlab code was downloaded from `http://homepages.inf.ed.ac.uk/imurray2/pub/10hypers/surr_code.tar.gz` on 5/10/2016. The author believes this is indeed a bug in the source code (Iain Murray, personal communication).

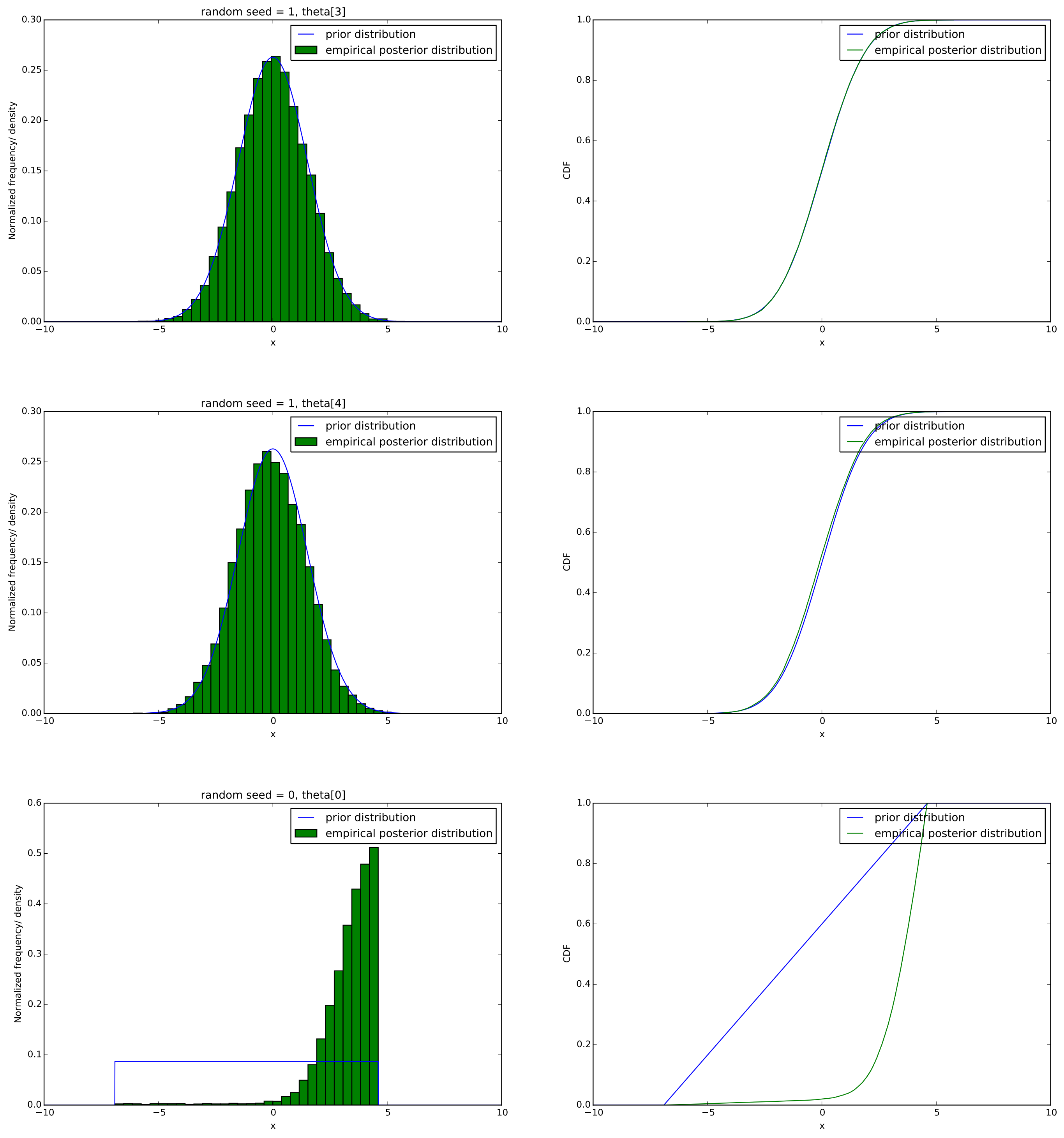


Figure 6.3.2: Diagnostic plots of Geweke experiment runs resulting in different results for the KS test. From top to bottom: not rejected, rejected but correct implementation, rejected due to faulty implementation. Each row is for a specific hyperparameter. The **left** plot shows the prior or empirical posterior distribution, the **right** plot shows the (empirical) cumulative distribution function.

These reject proportions are around the expected 5%, corresponding to the KS test significance α. This supports the assertion that the implementation used in our main experiments, cf. section 6.4.3, is correct.

6.3.4 Hyperparameter learning: synthetic data experiment

We implemented the three parameterisations of section 6.2 with slice sampling as the MC transition operator.

Two tests helped corroborate that our implementation was sound: the Geweke-Kolmogorov test described above, and a test on a synthetic task, which we present now. Moving on to real data in section 6.4, we will apply the different hyperparameter learning methods to the NLP tasks used in chapter 4, and monitor the results.

The data used for our synthetic task consists of single-atom structures, as above: one input node, one output node. The output label y takes its values in $\{1, 2, 3, 4, 5\}$, while the input, $\mathbf{x} \in \mathbb{R}^{10}$, consists of two parts: $x_1 ... x_5$ are a one-hot encoding of the output (we will call $x_1 ... x_5$ "signal features"), and $x_6 ... x_{10} \overset{iid}{\sim} U(0, w)$ are irrelevant noise features. The "weight" w controls the magnitude of the noise feature values with respect to the signal feature values (binary).

Input similarity is determined by an exponential automatic relevance determination (ARD) kernel (Rasmussen and Williams, 2006, section 5.1) of the form

$$k_{\text{exponential ARD}}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}\sum_i \frac{(x_i - x_i')^2}{(\exp \psi_i)^2}\right) \tag{6.3.4}$$

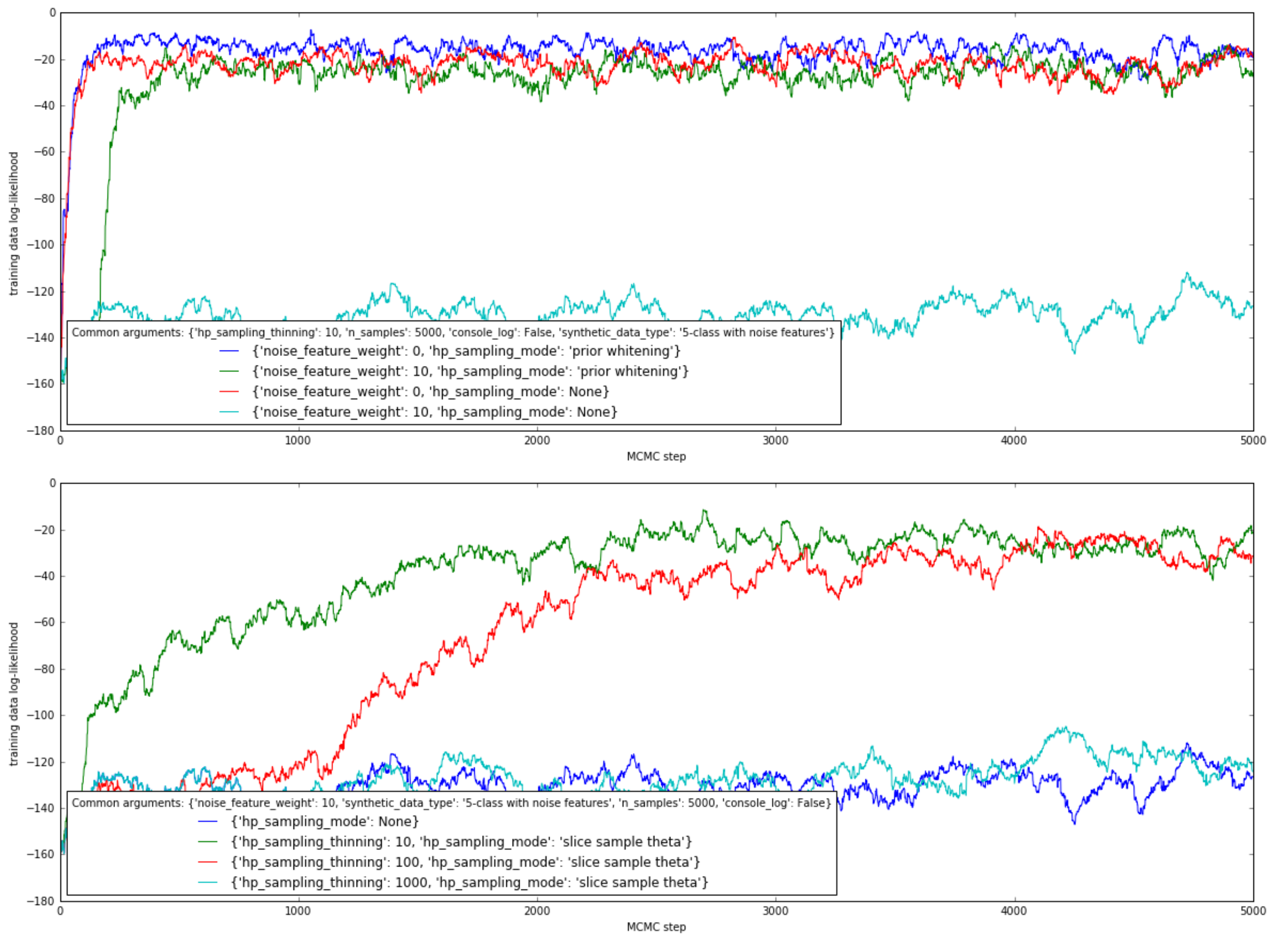
. The hyperparameters are given a uniform prior $\psi_i \sim U(\log 10^{-3}, \log 10^2)$, and are initialised at 0 (this is also their default value when they are kept fixed).

Our hope is that, by learning kernel hyperparameters, the hyperparameters ψ_i, $i \in \{6...10\}$ corresponding to noise features will grow large so as to cancel out the effect of noise terms in $k_{\text{exponential ARD}}(\mathbf{x}, \mathbf{x}')$.

In our first experiment, we visualise the MCMC likelihood plot (figure 6.3.3, top plot) obtained when modelling this synthetic data with GPstruct as described, with and without hyperparameter learning (using prior whitening), with $w = 0$ or $w = 10$. In another experiment using slice sampling, we investigate varying the hyperparameter update rate while keeping $w = 10$.

The experiments show several points.

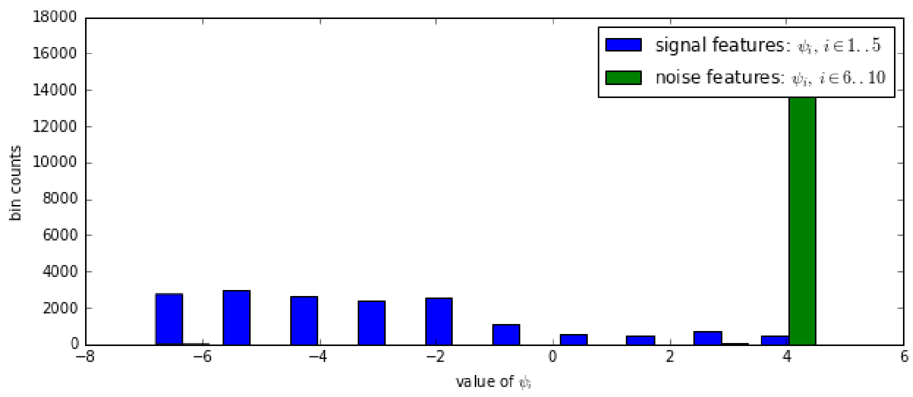
- noise features with $w = 10$ make learning hard or impossible (figure 6.3.3, top plot, line "hp fixed, $w = 10$")
- their effect can be mostly removed by learning the ARD hyperparameters (figure 6.3.3, top plot, line "PW, $w = 10$")



Common arguments: {'hp_sampling_thinning': 10, 'n_samples': 5000, 'console_log': False, 'synthetic_data_type': '5-class with noise features'}
{'noise_feature_weight': 0, 'hp_sampling_mode': 'prior whitening'}
{'noise_feature_weight': 10, 'hp_sampling_mode': 'prior whitening'}
{'noise_feature_weight': 0, 'hp_sampling_mode': None}
{'noise_feature_weight': 10, 'hp_sampling_mode': None}

Common arguments: {'noise_feature_weight': 10, 'synthetic_data_type': '5-class with noise features', 'n_samples': 5000, 'console_log': False}
{'hp_sampling_mode': None}
{'hp_sampling_thinning': 10, 'hp_sampling_mode': 'slice sample theta'}
{'hp_sampling_thinning': 100, 'hp_sampling_mode': 'slice sample theta'}
{'hp_sampling_thinning': 1000, 'hp_sampling_mode': 'slice sample theta'}

Figure 6.3.3: Does hyperparameter learning help ignoring the noise features? Top: Experiments with prior whitening vs. no hyperparameter sampling, showing influence of noise feature weight. Bottom: experiments with slice sampling, noise feature weight $w = 10$, showing the influence of the hyperparameter update rate.



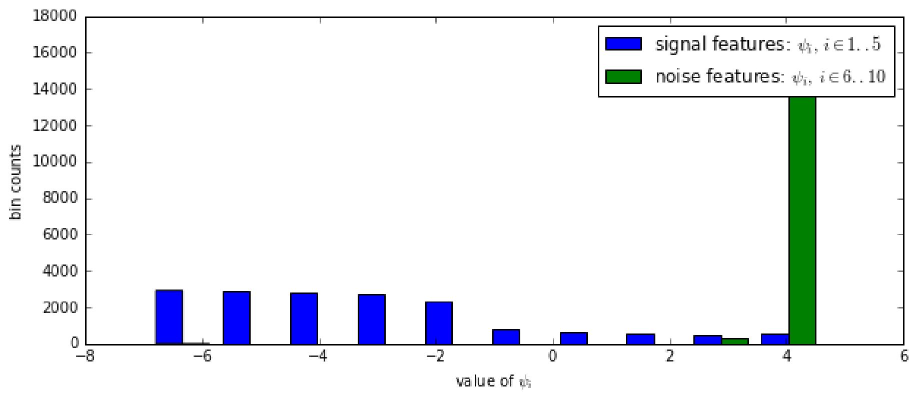


Figure 6.3.4: Histogram of sample path for the log ARD variances $\psi_1 ... \psi_5$ (signal features, all collected in one set, in blue on the plot), and $\psi_6 ... \psi_{10}$ (noise features, collected in another set, in green on the plot). The configurations considered are: top, PW; bottom, SS. In both cases, noise feature weight $w = 10$, and hyperparameter update rate is 1 every 10 ESS steps. The values of the hyperparameter were binned into 10 bins of equal width over their range.

- updating the ARD hyperparameters every 10 or 100 latent variable updates still produces good results (figure 6.3.3, bottom plot), while doing it every 1000 updates is more noticeably inefficient (though it would presumably reach the same levels of likelihood had we run the MCMC chain for order of magnitudes longer)

As an additional diagnostic tool, we can plot the sample path histograms obtained by collecting values of the noise hyperparameters from the MCMC chain, and compare them to signal hyperparameters. The histograms can be found in figure 6.3.4 . While the signal hyperparameters adopt small values, which causes their respective contribution in the sum to be large, the noise hyperparameters adopt large values, making their contribution vanishingly small.

In this synthetic data case, hyperparameter learning has demonstrated its usefulness, and we now hope to apply it to real data.

6.4 Experiments on NLP tasks

6.4.1 Hyperparameter update runtime

What is the computational cost of hyperparameter updates with each of the three methods?

We need to measure the runtimes of updates, to make sure the trade-off between performance gain and computational expense remains manageable. We compared timings for each of the three methods, on the following experimental configuration: `Base NP` task, 50 000 MCMC iterations, hyperparameter updates every 100 latent variable updates, full data set, several folds of the data. Each of SS, PW, SD routines ran 4000 times. This experiment took around 2e6 CPU-seconds to run (1e5 CPU-seconds per fold). Every fold, we report median runtimes and standard deviations of the hyperparameters update step. The runtimes are generally variable because unlike a Metropolis-Hastings update, which is not iterative in nature, slice sampling may iterate an unknown number of times before finishing "on slice" again. These are the results.

parameterisation	runtime in sec
SS (6.1)	102 ± 50
PW (6.2)	117 ± 52
SD (6.3)	607 ± 208

After exploratory experiments in which the surrogate data or prior whitening methods did not appear superior, and considering the runtime of the tasks *per se*, we decided to carry out the main experiments with the SS method, and occasionally PW, but discarding SD.

6.4.2 Applying ARD kernels to the NLP tasks

In addition to their feature selection properties, ARD kernels (Rasmussen and Williams (2006, section 5.1) and MacKay (1996)) lend themselves to compare *structured* feature vectors, which are obtained by stacking feature vectors of different origins. This is the case of the NLP experiments, with feature vectors comprising e.g. gazetteer features; word identity features; features applied to left neighbour, to right neighbour, etc.

A naïve approach would attempt to assign an ARD hyperparameter to every features; in the NLP tasks, they run into the tens or hundreds of thousands of features[13]. Learning such a high number of hyperparameters is challenging, if only because of a data sparseness issue: any hyperparameter change has very little impact, and their

[13]cf. task description in section 4.5.4

posteriors will probably be remarkably flat. In addition, in the particular case of multivariate slice sampling MCMC methods, we need to make coordinate-wise moves, i.e. we will cycle through all the dimensions of the hyperparameter vector to complete a hyperparameter update, each time computing the training data Gram matrix, and its Cholesky factorisation, anew.

How do we take advantage of the feature vector make-up to reduce the number of hyperparameters? The feature vector is obtained by concatenation of subvectors of different nature. We may exploit its internal structure by assigning the same ARD variance to a contiguous block of features of similar nature, using the assumption that similar features should be scaled similarly.

To discover these blocks post-hoc, without access to the feature extraction functions, a simple, heuristic method is to plot the empirical variance of each feature (an example of such a plot is figure 6.4.1), and to identify visually similar blocks. In our experiments we arbitrarily decided to use 6 ARD hyperparameters, and so identified 6 blocks in this fashion. This method is crude, but efficient.

The ARD kernels we will use thus take the form:

$$k_{\text{linear ARD}}(\mathbf{x}, \mathbf{x}') = \sum_i \frac{x_i x_i'}{(\exp \psi_{B(i)})^2} \tag{6.4.1}$$

$$k_{\text{exponential ARD}}(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{1}{2}\sum_i \frac{(x_i - x_i')^2}{(\exp \psi_{B(i)})^2}\right) \tag{6.4.2}$$

where $B(i)$ is the index of the feature block to which feature i belongs, and the ψ_b are the ARD kernel hyperparameters. The particular form of the hyperparameters allows the ψ_b to be interpreted as length-scale hyperparameters.

Setting the values of the ψ controls the weight given to each group of features. For squared exponential ARD kernels, there exists a rule of thumb in the computer vision community for a “decent” default value (cf. for instance Lampert (2009), section 2.7.1): set the denominator so that the exponent becomes -1 for the median of $(x_i - x_i')^2$, which is easy to evaluate on the training set. It is not clear what motivates this rule of thumb. We use it in some of our experiments as a reasonable default value, cf. table 6.1 for details.

6.4.3 Experimental configuration and results

In section 4.5, we described the experimental configurations for latent variable inference on the NLP tasks. We now complement these with new configuration parameters for the hyperparameter inference experiments we conducted.

Kernel type We used either the linear kernel, the linear ARD kernel, or the squared exponential ARD kernel.

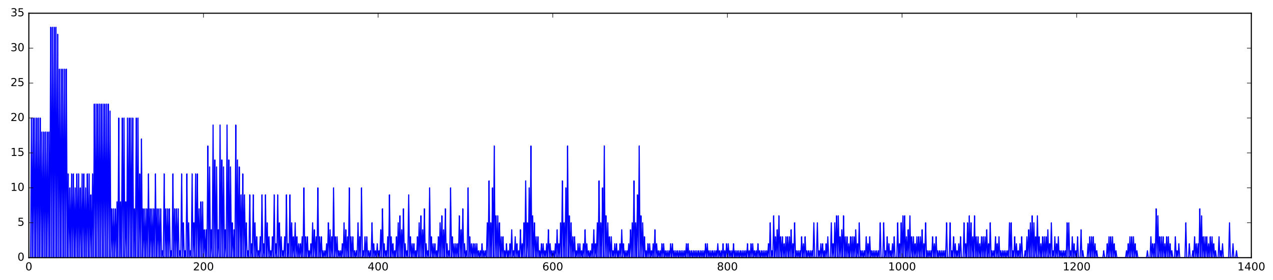


Figure 6.4.1: Plot of values of $\mathbf{diag}(X^T X)$ where X is the feature matrix. If we decide to try and identify 5 boundary points between features of similar covariance, we can choose the following values: 252, 522, 732, 852, 1186.

Hyperparameter set, initial values In the case of ARD kernels, the 6 ARD hyperparameters obtained as in section 6.4.2 could vary. Their initial value is indicated in each experiment description. In addition, the log binary scaling hyperparameter $\log h_b$ (defined in equation 4.3.1) was optionally allowed to vary. Its initial value was always 0. The initial values also serve as fixed values in case the hyperparameters are not allowed to vary, i.e. they are not sampled. The jitter hyperparameter is never sampled in our experiments, though it could be in theory, and is set to $\log 10^{-4}$.

Hyperparameter priors We used Gaussian priors, in the descriptions *broad* means $N(0, \sigma^2 = 0.7)$, while *narrow* means $N(0, \sigma^2 = 0.3)$. When a uniform prior is used, it is $\log h_b \sim U(\log 10^{-3}, \log 10^2)$.

Hyperparameter updates We used the methods described in section 6.2. We updated the hyperparameters either every 100 or 1 000 latent variable updates.

The experiments we conducted are exhaustively described in table 6.1, along with pointers to the corresponding MCMC traces, and partial result interpretations.

Considering that the results are not clear-cut, a legitimate question to ask is: “In our tasks, will the choice of hyperparameters determine performance at all? Or is the posterior distribution of hyperparameters so flat that any value is as good as any other, and variations in performance are spurious?” We set up a simple experiment to answer these questions. We keep the hyperparameters fixed, but test several values in a range resulting from an educated guess. The MCMC runs are depicted in figure 6.4.7.

In both cases, we observe that $\psi_b = 1$ gives better results than $\psi_b = 0$; in addition, in the case of SE ARD, this is much better than $\psi_b = -1$. We also observe that, in the case of linear ARD, which gives better HE and ANLPM than exponential ARD anyway, HE and ANLPM are not consistent. We hypothesise this shows that with already excellent performance metrics, any improvement in one metric will be detrimental to

kernel	common parameters	varying parameters	configuration	final Hamming error	final ANLPM	interpretation
linear	Hyperparameter set: just $\log h_b$. Hyperprior $\log h_b \sim U(\log 10^{-3}, \log 10^2)$. Initial value $\log h_b = 0$. Sampling hyperparameters every 1 000 latent variable updates. (figure 6.4.2)	Task: basenp, segmentation, chunking, japanesene. MCMC method: none, SS or PW	task: basenp, no hp sampling	0.047	0.156	Hyperparameter learning brings no performance benefit.
			task: basenp, method: PW	0.048	0.157	
			task: basenp, method: SS	0.049	0.158	
			task: segmentation, no hp sampling	0.148	0.360	
			task: segmentation, method: PW	0.150	0.361	
			task: segmentation, method: SS	0.150	0.360	
			task: chunking, no hp sampling	0.086	0.359	
			task: chunking, method: PW	0.089	0.366	
			task: chunking, method: SS	0.086	0.365	
			task: japanesene, no hp sampling	0.058	0.273	
			task: japanesene, method: PW	0.059	0.299	
			task: japanesene, method: SS	0.058	0.290	
linear ARD	Task: segmentation. Sampling hyperparameters every 1 000 latent variable updates. Sampling method: SS. Initial values $\log h_b = 0$, $\psi_{1..6} = 0$. (figure 6.4.4)	Hyperparameter set $\log h_b$, $\psi_{1..6}$ or just ψ_b. Hyperprior.	$\psi_{1..6}$, prior broad	0.152	0.358	Hyperparameter learning seems to do better than the "bad" choice $\psi_{1..6} = 0$, but worse than the "good" choice $\psi_{1..6} = 1$. (NB: experiments with different initial values should be comparable, but they're not; it is not clear whether to attribute this to the chain not mixing long enough, or to cross-chain variance.)
			$\log h_b$, $\psi_{1..6}$, prior broad	0.152	0.358	
			$\psi_{1..6}$, prior broad	0.150	0.363	
			$\log h_b$, $\psi_{1..6}$, prior broad	0.152	0.360	
	As above, but initial values $\log h_b = 0$, $\psi_{1..6} = 1$. (figure 6.4.5)	Hyperparameter set $\log h_b$, $\psi_{1..6}$ or just $\psi_{1..6}$. Hyperprior.	$\psi_{1..6}$, prior broad	0.155	0.360	
			$\log h_b$, $\psi_{1..6}$, prior broad	0.153	0.358	
			$\psi_{1..6}$, prior broad	0.153	0.356	
			$\log h_b$, $\psi_{1..6}$, prior broad	0.154	0.358	
	Task: segmentation. Hyperparameters fixed. Initial value for $\log h_b = 0$. (figure 6.4.4 and 6.4.5)	Initial value for $\psi_{1..6}$.	$\psi_{1..6} = 1$	0.154	0.352	
			$\psi_{1..6} = 0$	0.149	0.362	
expo-nential ARD	Task: segmentation. Sampling hyperparameters every 100 latent variable updates. Initial values: $\log h_b = 0$, $\psi_{1..6} = \frac{1}{2}\log(2 * 7) = 1.32$. Sampling method: SS. (figure 6.4.6)	Hyperparameter set $\log h_b$, $\psi_{1..6}$ or just $\psi_{1..6}$, and hyperprior. Hyperparameters fixed, vary random seed (as an indicator of cross-chain MCMC variance)	$\psi_{1..6}$, prior broad	0.180	0.443	Hyperparameter learning works; the best results are obtained from specifying a broad hyperprior and including $\log h_b$ among the sampled hyperparameters.
			$\log h_b$, $\psi_{1..6}$, prior broad. (Hyperparameter posteriors figure 6.4.8 and 6.4.9)	0.182	0.443	
			$\psi_{1..6}$, prior narrow	0.181	0.446	
			$\log h_b$, $\psi_{1..6}$, prior narrow	0.182	0.447	
			seed 0	0.181	0.449	
			seed 1	0.180	0.449	
			seed 2	0.177	0.448	
			seed 3	0.178	0.448	
	Task: segmentation. Hyperparameters fixed. Initial value for $\log h_b = 0$. (figure 6.4.7)	Initial value for $\psi_{1..6}$.	$\psi_{1..6} = -1$	0.181	0.449	Analysis experiment: Hyperparameter values do make a difference in performance.
			$\psi_{1..6} = 0$	0.219	0.489	
			$\psi_{1..6} = 1$	0.320	0.596	

Table 6.1: Summary of NLP hyperparameter learning experiments

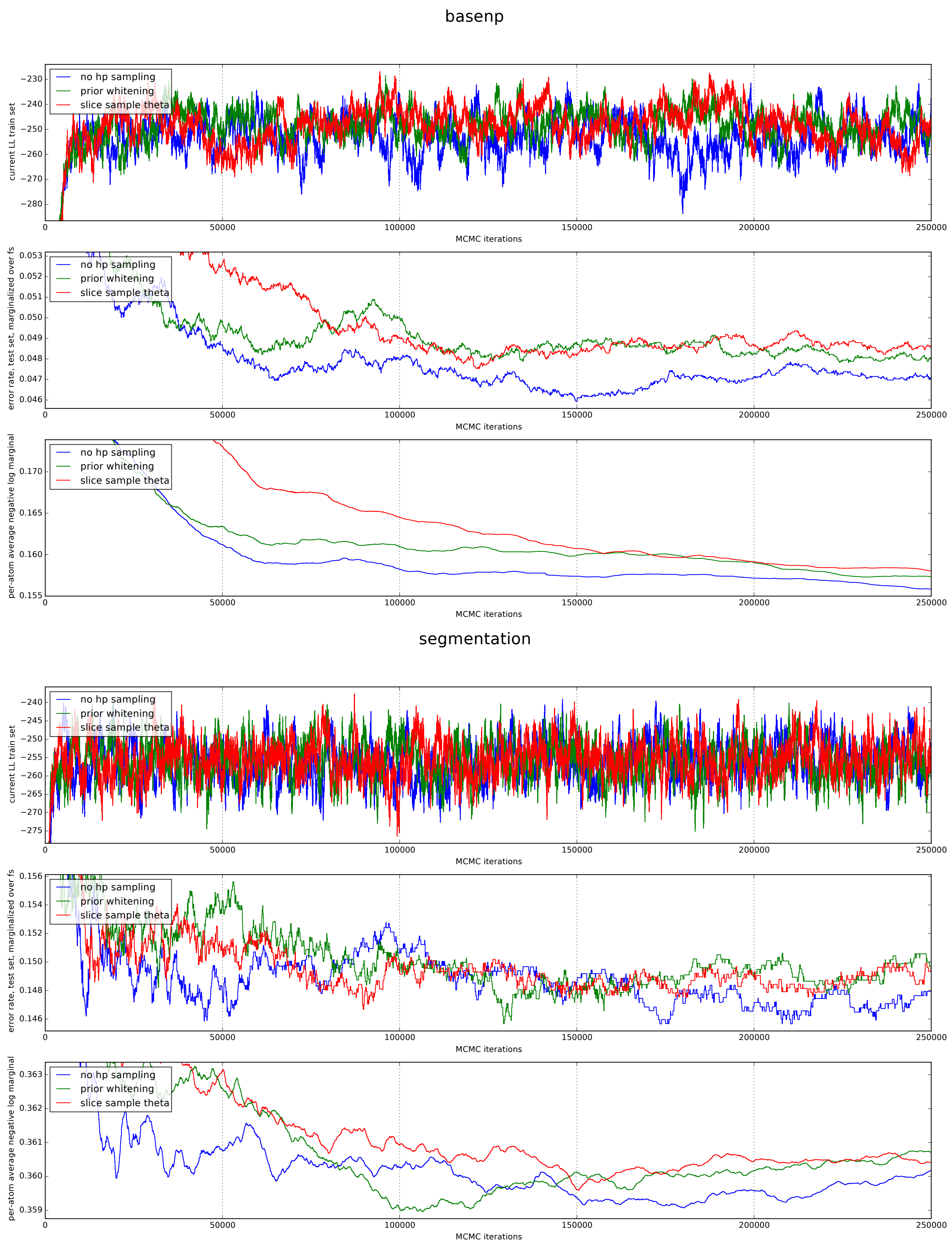


Figure 6.4.2: Experiment: learning the linear kernel's binary scaling hyperparameter h_b (represented as $\log h_b$). Hyperparameter set: just $\log h_b$. Hyperprior $\log h_b \sim U(\log 10^{-3}, \log 10^{2})$. Initial value $\log h_b = 0$. Sampling hyperparameters every 1 000 latent variable updates. Different configurations: different NLP tasks; hyperparameter updates using slice sampling or prior whitening.

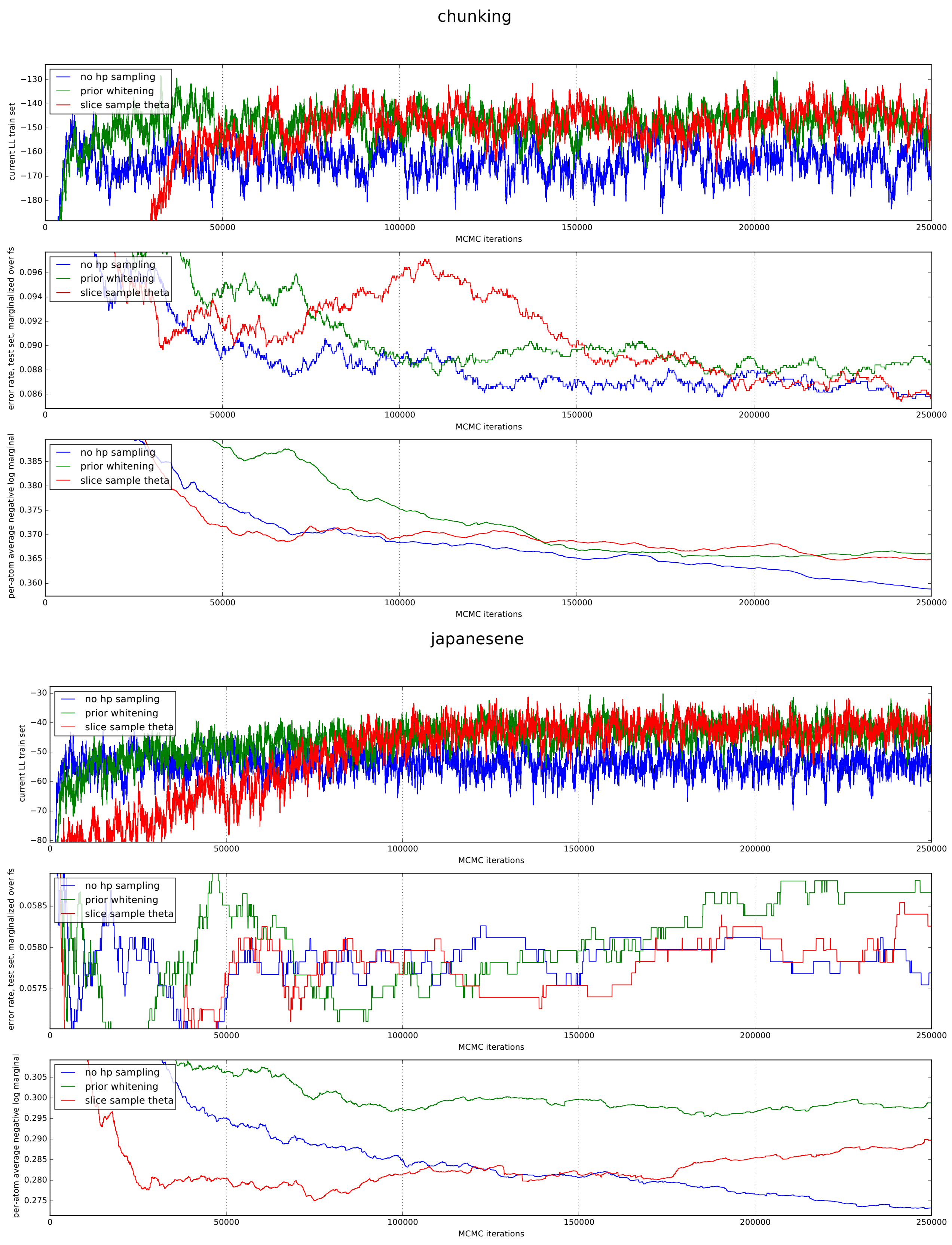


Figure 6.4.3: Continued from figure 6.4.2

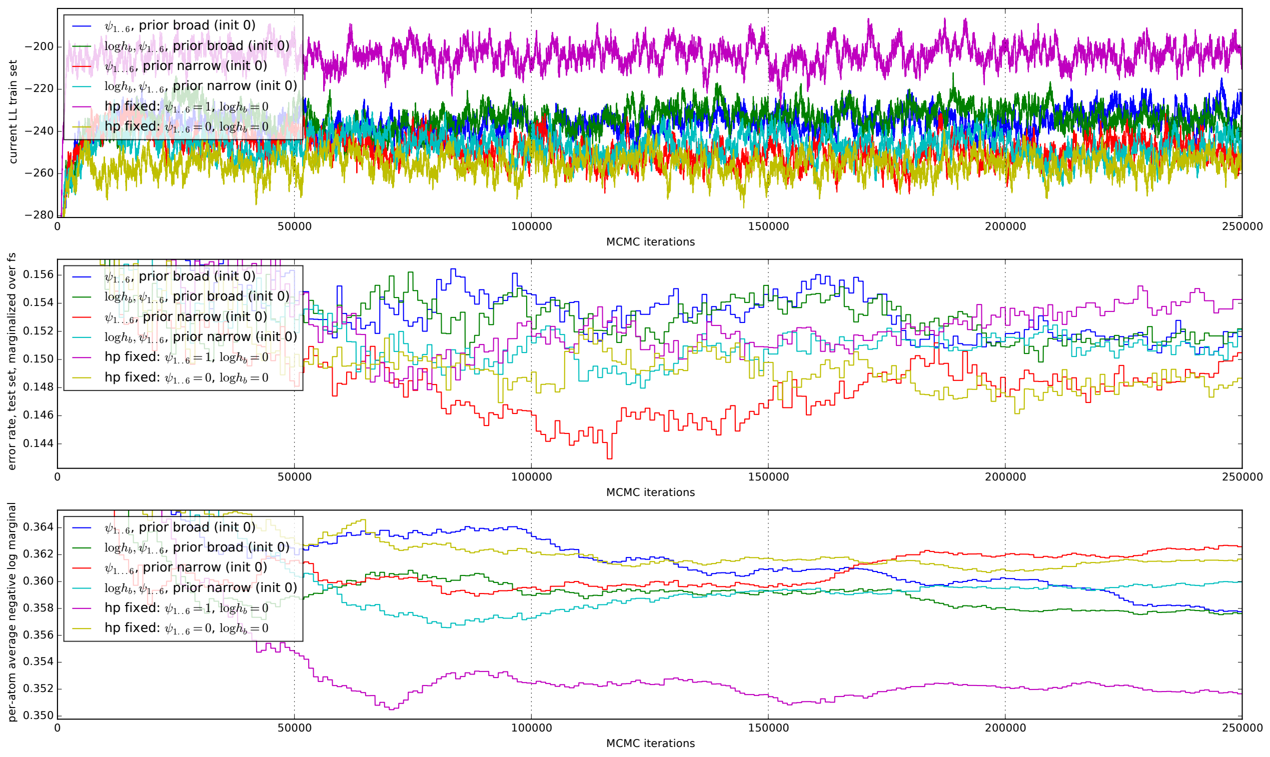


Figure 6.4.4: Experiment: linear ARD kernel. Task: segmentation. Sampling hyperparameters every 1 000 latent variable updates. Sampling method: SS. Initial values $\log h_b = 0$, $\psi_{1..6} = 0$

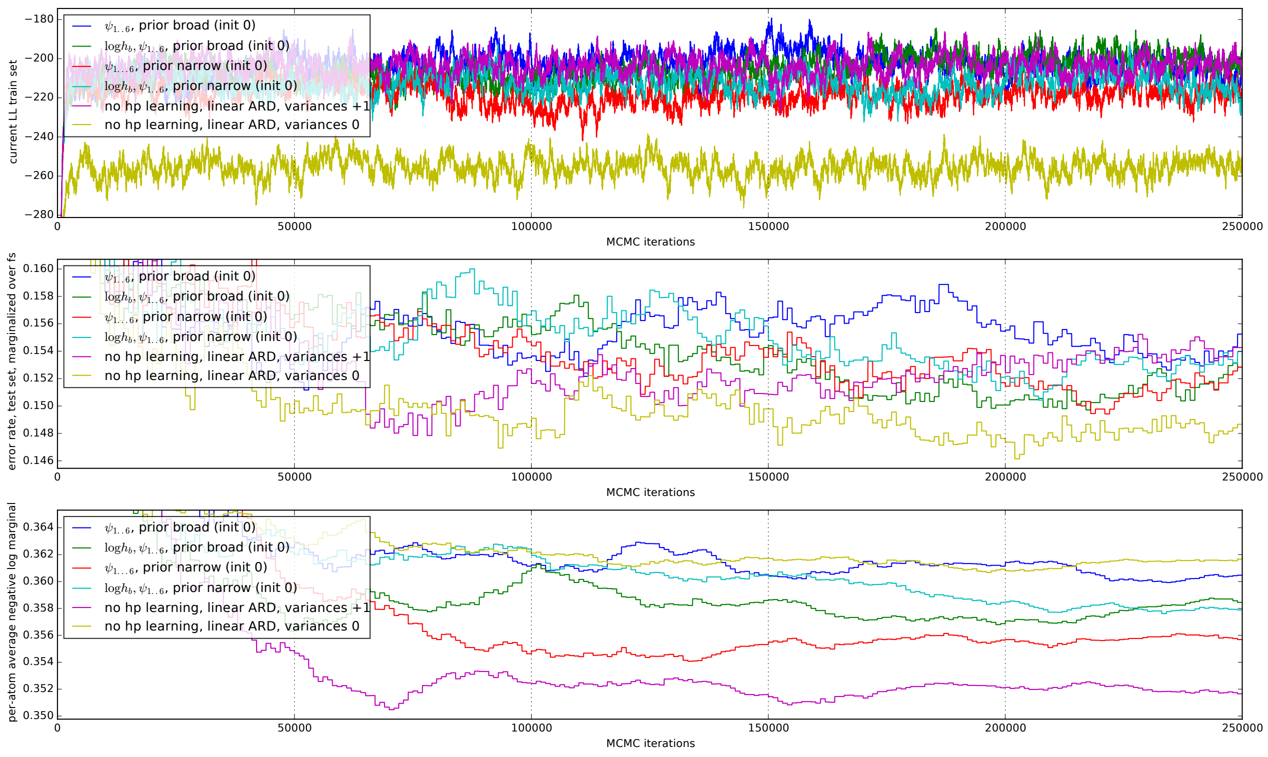


Figure 6.4.5: Experiment: linear ARD kernel. Task: segmentation. Sampling hyper-parameters every 1 000 latent variable updates. Sampling method: SS. Initial values $\log h_b = 1$, $\psi_{1..6} = 0$

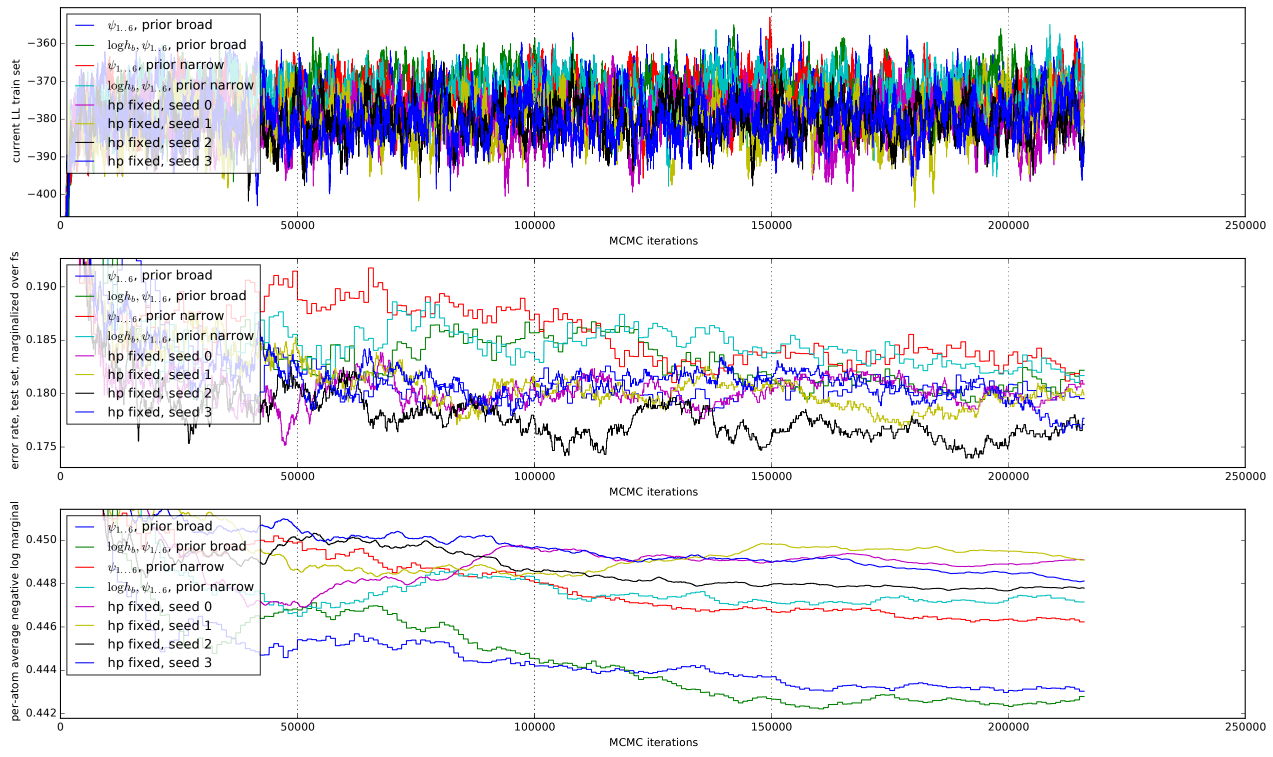


Figure 6.4.6: Experiment: squared exponential ARD kernel, with 6 variances assigned each to one block, with block boundaries as identified visually above. Task: segmentation. Different configurations: learning binary scaling hyperparameter h_b and/or ARD hyperparameters ψ_b. Hyperprior (on log hyperparameters): broad means $N(0, \sigma^2 = 0.7)$, while narrow means $N(0, \sigma^2 = 0.3)$. Hyperparameter update every 100 MCMC steps. Initial values: $h_b = 0$, $\psi_b = \frac{1}{2}\log(2 * 7)$ for ARD variances, following rule of thumb for squared exponential variances.

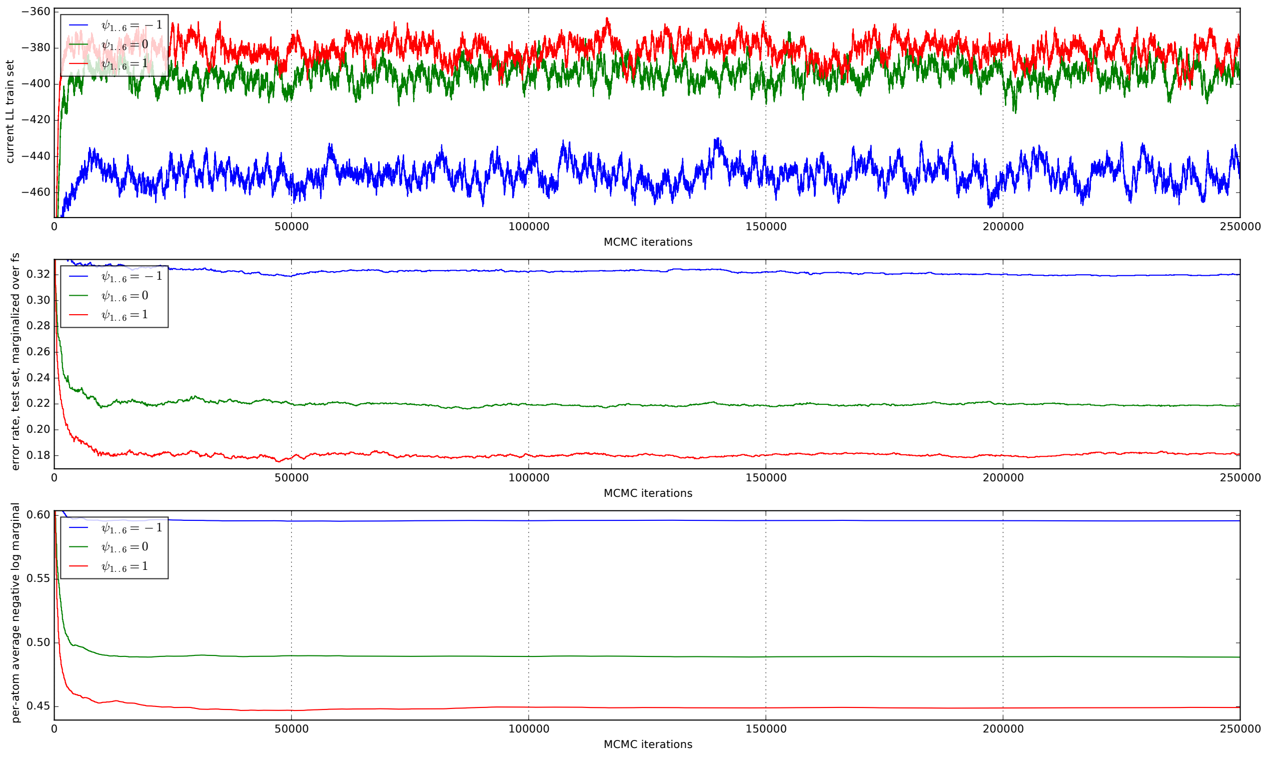


Figure 6.4.7: Task: segmentation. Hyperparameters fixed (different values, cf. legend). Initial value for $\log h_b = 0$.

the other.

We see that overall, the linear and linear ARD kernels are much better suited to the task than the exponential ARD kernel. This is natural considering that the data are represented as sparse binary feature vectors, for which the dot product is more meaningful than the Euclidean distance. It seems that, as a consequence, while hardly any improvements can be obtained in the linear and linear ARD experiments, there is some room for improvement in the exponential ARD experiments, which hyperparameter inference manages to exploit.

The default values selected for hyperparameters before conducting these experiments seem to have been in the correct range. An optimistic view of our results would assert that, despite knowing hardly anything about the hyperparameters, the hyperparameter inference procedure managed to obtain competitive results.

The interpretation of results is affected by MCMC cross-chain variance. As a yardstick, we use the cross-chain estimates obtained in section 4.8. It must be noted, however, that these values were obtained when sampling $\mathbf{f}|\mathcal{D}$, not $\mathbf{f}, \theta|\mathcal{D}$. The cross-chain variance for the latter model is presumably higher, and depends on the sampling method used, although we have not explored this further experimentally.

An insightful result obtained as a by-product from the MCMC runs is the shape of the hyperparameter posterior, $p(\theta|\mathcal{D})$. We plot a histogram of the samples of each hyperparameter and compare it to the hyperparameter prior (defined analytically). This is done for the exponential ARD experiments in figure 6.4.8. We note that the results are remarkably similar across the 5 folds, in the sense that the hyperparameter posteriors are comparable across folds. This motivates aggregating the posteriors for all folds, per hyperparameter, as is done in figure 6.4.9 (top plot). We also plot the result for a linear ARD experiment[14]. In some cases (e.g. for exponential ARD, ψ_2 and ψ_5) posteriors sit at the tail of the Gaussian hyperprior, suggesting that a broader hyperprior would still have received support from the evidence.

The most important observation from these plots, however, is the following: In all cases, we remark that the posteriors are visually distinct from the hyperprior; this indicates that the evidence from the data influenced the posterior. This is despite modest improvements in the performance metrics, and suggests that our choice of tasks is moderately suited to revealing the advantage of hyperparameter inference. That is to say, not much improvement is visible in the task metrics, despite clear signs that hyperparameter inference is working.

[14]Here we note that some posteriors, like those for $\psi_{4..6}$, seem particular: they are one-sided on the positive side of the real line. It is not clear why this is the case, and whether it has any bearing on the average performance of hyperparameter inference in this experiment.

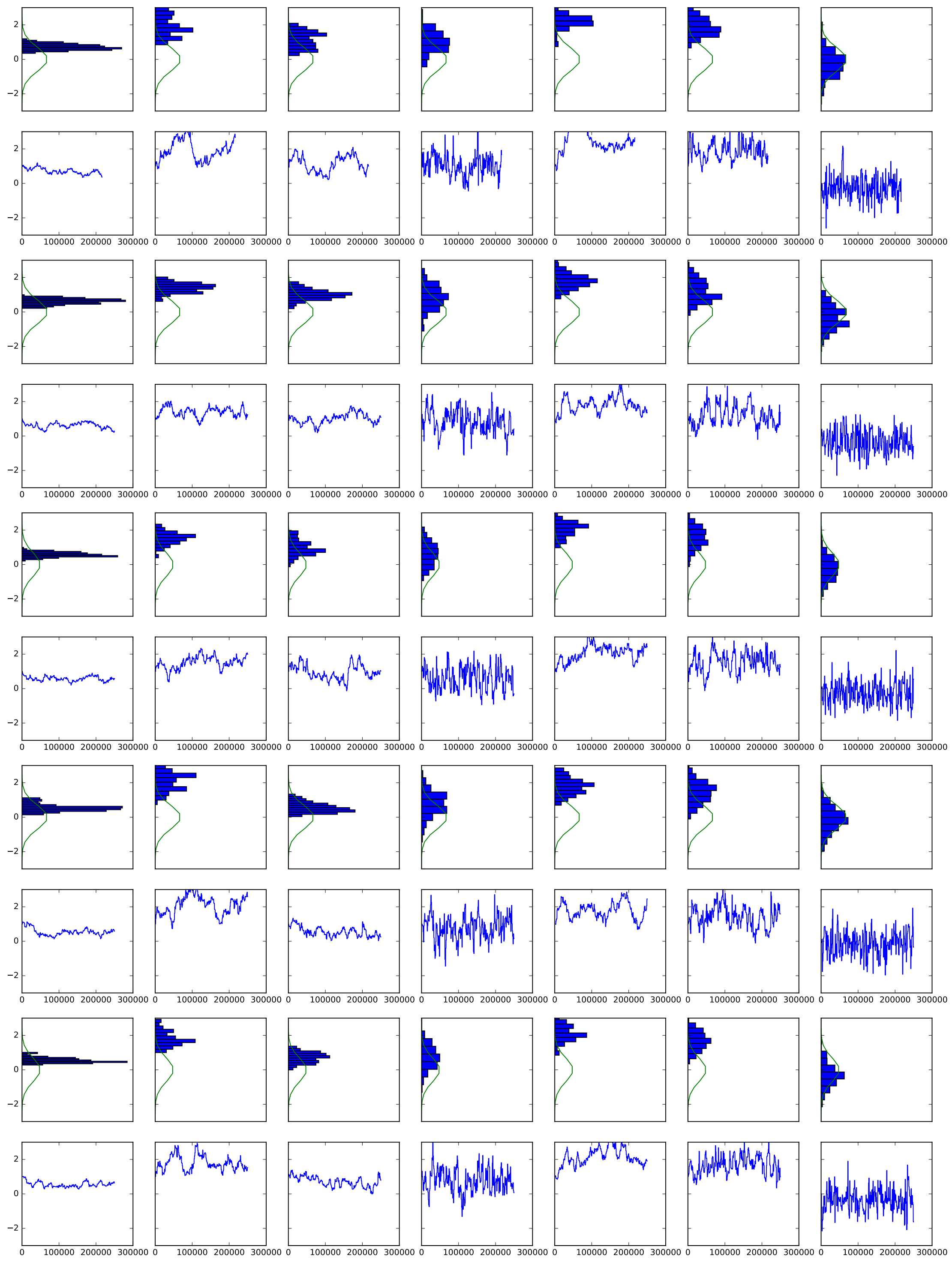


Figure 6.4.8: Hyperparameter samples paths over the course of an MCMC chain. Kernel exponential ARD. Each column shows one hyperparameter: $\psi_{1..6}$, last column: $\log h_b$. Groups of 2 rows are each for one fold of the data (five folds in total). Odd rows: histogram of relative frequency hyperparameter values, with plot of hyperprior for comparison, in green. Even rows: Hyperparameter sample path. Experimental configuration: Task: segmentation. Sampling hyperparameters every 100 latent variable updates. Initial values: $\log h_b = 0$, $\psi_{1..6} = \frac{1}{2}\log(2 * 7) = 1.32$. Sampling method: SS. Hyperparameter set: $\log h_b$, $\psi_{1..6}$, hyperprior broad.

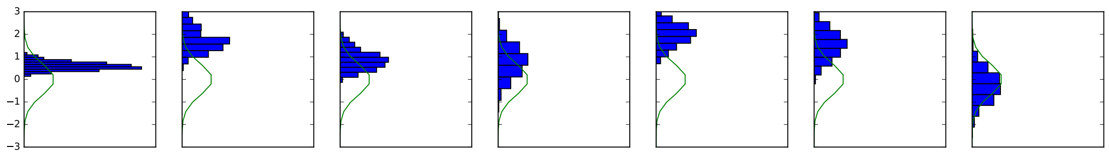


Figure 6.4.9: Hyperparameter sample paths, aggregated over the MCMC chains for all 5 folds, kernel exponential ARD, experiment as in figure 6.4.8. Each column shows one hyperparameter: $\psi_{1..6}$, last column: $\log h_b$. Task: segmentation. Sampling hyperparameters every 1 000 latent variable updates. Sampling method: SS. Initial values $\log h_b = 0$, $\psi_{1..6} = 1$. Hyperparameter set $\log h_b$, $\psi_{1..6}$. Hyperprior broad.

6.5 Conclusion

In this chapter, we have detailed several sampling methods suited to hyperparameter learning in GPstruct. Since the potential for a faulty MCMC implementation to go undetected is considerable, we put emphasis on synthetic data experiments and verification procedures. To assess how well the proposed methods work in practice, we conducted experiments on NLP datasets, with GPstruct using ARD kernels, and learning their variance hyperparameters. The scientific contributions of this chapter consist in:

- developing, implementing and demonstrating a correctness test for MCMC samplers which improves on Geweke's "Getting it right" test (Geweke, 2004)
- a synthetic task demonstrating hyperparameter learning[15]
- publicly available implementations[16] of the three hyperparameter learning schemes for GPstruct detailed section 6.2
- insights into hyperparameter inference with GPstruct using several different experimental settings (table 6.1)

As is often the case, the research was conducted in a different order from the order of exposition adopted in this chapter. After an initial implementation of the three reparameterisation schemes for hyperparameter inference, and preliminary experiments on the binary scaling hyperparameter, we moved on to experiments with ARD hyperparameters, to which we expected the task performance to be sensitive. Improvements did not turn out to be as strong as expected. These experiments run for several days each, and are thus computationally demanding, so we decided to focus only on slice sampling, despite being exposed to coupling between hyperparameters and latent variables. Fearing implementation issues, the MCMC samplers were rederived from their mathematical formulation, the pseudo-code was rewritten, the im-

[15]in a setting where GPstruct is restricted to Bayesian GP multi-class classification
[16]`http://github.com/sebastien-bratieres/pygpstruct`

plementation carefully tested with unit tests, and to remove any doubt, we carried out the Geweke tests detailed in section 6.3. We were able to demonstrate hyperparameter inference on a synthetic task, section 6.3.4. Several smaller bugs were uncovered, but the conclusions of the experiments conducted so far remained unchanged. To identify a favourable configuration, and to perform a solid analysis, we extended the experiments to cover the bulk of the cases listed in table 6.1, and carried out several analytic experiments, such as fixing the hyperparameters (figure 6.4.7), or inspecting hyperparameter sample path histograms, which demonstrated that hyperparameters were being learnt but affected task performance weakly (figure 6.4.9).

On the basis of these actions and insights, we believe we have conducted a thorough investigation, are convinced that our code is correct, and eliminated several causes of poor performance. Future work could include MCMC convergence diagnostics, which are computationally even more expensive than the current experiments, but with hindsight we believe we should have changed our dataset altogether earlier in our project. In fact, we hypothesise that the datasets and tasks we are working with do not have much leeway for performance improvement stemming from better learning procedures. This would explain issues found in section 4.12, namely the closeness of performance measurements for GPstruct against similar models, as well as, in section 6.4.3, the sometimes paradoxical behaviour of performance metrics (HE and ANLPM). Regarding this last point, however, we must point out that predictions are done on the basis of the Hamming loss and therefore optimise the Hamming error metric; nonetheless, HE has high cross-chain MCMC variance, and the visual analysis of MCMC trace plots shows that in-chain variance is relatively high too, which seems to show that it is hard to rely on at the levels of performance we are working with.

As described throughout this chapter, several alternative paradigms and methods for hyperparameter learning could have been applied. Noteworthy among these, HMC can jointly train both the latent parameters and the hyperparameters. When this research was undertaken (2014), HMC was thought to be difficult to tune; subsequent research (Filippone et al. (2013); Hensman et al. (2015)) has shown the issue to be a manageable one, and this method looks like a useful replacement. However, known methods require the likelihood function to factorise over $\mathbf{f}$, which calls for approximations or bounds. In fact, one solution is to resort to a pseudo-likelihood approximation of the model as in Srijith et al. (2016), discussed in section 5.6.1), where hyperparameters are estimated point-wise by a VB-EM algorithm (Beal and Ghahramani, 2003)[17].

Overall, the idea of using a hyperparameter learning scheme which closely integrates with latent variable inference paid off. We reaped the benefits of a probabilistic approach and avoided the curse of dimensionality brought about by grid search.

The surrogate data method was implemented and tested, but not found to be ef-

[17]an extension to VB in which at each iteration, the variational lower bound is subjected to a round of expectation maximisation to estimate the hyperparameters

ficient enough to compensate for its high computational cost (one inversion and two Cholesky factorisations of matrices of size $\sum_n T^{(n)}$). These findings are in line with the conclusions of Filippone et al. (2013). Nevertheless, we can offer improvements. In our implementation, we used a fixed auxiliary noise covariance matrix $S_\theta = \alpha I$, as suggested in Murray and Adams (2010). This could be improved, either by an MCMC pre-run to estimate the posterior covariance, or by working out a softmax version of the moment-matching approximation of the posterior variance, which for the experiments of Murray and Adams (2010) was obtained for logistic regression[18].

Using a likelihood-based approach as we do offers several advantages in terms of making model selection modular in a machine learning system. More concretely, note we are using the (log) marginal likelihood as a score indicating how good the fit between hyperparameters and data is, as opposed to reporting a performance metric on the data, such as cross-validation performance. This way, we can compare model selection for a new model which was initially not considered; or we can train a set of hyperparameters jointly on several datasets (transfer learning), because log likelihoods over several datasets can be combined correctly (by adding them), so that hyperparameters can be trained using the sum of log likelihoods. This also opens the way to kernel learning as in e.g. Duvenaud et al. (2013).

Finally, by dealing only with covariance parameters, we have only scratched the surface of the possibilities offered by hyperparameter learning in the context of GPstruct. Many of the GPstruct *model design choices* introduced in chapter 4 could be learnt instead of being fixed. We think specifically of:

- the type of kernel, possibly a combination of kernels (with their respective scaling factors) when the feature vector is known to be a concatenation of very different features (e.g. a dense scalar part and a sparse binary part)
- the order of factors in the CRF; we have only considered binary factors in this work
- the size of the Markov blanket of a node in the CRF; we have only considered immediate neighbours
- parameter tying schemes (which would change the structure of $\mathbf{f}$)
- the kernel priors, and more specifically, their sparsity structure

A remark in terms of research agenda: implementing these options naïvely incurs a heavy computational cost; this cost has to be tackled by the development of acceleration techniques for GPstruct learning, such as GP sparsification, fast Cholesky matrix

[18]Cf. Murray and Adams (2010) after equation 12 for a discussion, and the code (file `logistic_aux.m`, line 27), which uses $(S_\theta)_{ii} = (\Sigma_\theta)_{ii} \cdot (1 - \frac{1}{\pi/2+4/(\Sigma_\theta)_{ii}})$. It is not clear how to obtain a similar approximation for the softmax case.

updates, and kernel approximations. Gordon and Desjardins (1995), in its section 2 “What is inductive bias?”, details two major sources of inductive bias: the representational and the procedural. This distinction is reflected throughout modern probabilistic machine learning, in treating models and algorithms separately. Applying this to our work on GPstruct, it is apparent that the exploration of GPstruct’s representational possibilities has to go hand in hand with progress on the procedural front, efficient inference.

Chapter 7

Conclusion

7.1 Summary of scientific contributions

We now point out the specific scientific contributions this thesis is making to the field of machine learning.

We contribute to the study of distributed MCMC algorithms with a map-reduce implementation of the "beam sampling" algorithm (Van Gael et al., 2008). Our implementation targeted a commodity platform (Amazon Elastic MapReduce, running Hadoop) for scientific reproducibility and re-use, was programmed in Java on top of Hadoop, and used the open-source Apache Mahout library.

This thesis offers good practices and lessons learnt for porting algorithms to a map-reduce framework, and details specific forms of mappers and reducers which are relevant for different algorithms. We explored alternative ways of adapting the original algorithm to map-reduce, and reported the corresponding experimental result. These are contributions to the field of distributed machine learning.

Our main experimental finding, communicated in our publication (Bratières et al., 2010a) obviates the need for new distributed platforms which support iteration as a base construct. This result has been obtained repeatedly in machine learning and systems research since, and the distributed computing research community has addressed this need with platforms supporting several algorithmic constructs beyond just the map-shuffle-reduce operations.

Our contribution to large-scale learning with the IHMM is still up-to-date. We are not aware of any new attempts to reproduce our experiments, which we would expect to take place on a distributed platform supporting highly iterative algorithms.

This thesis offers surveys of research on several connected issues: large scale Bayesian inference; distributed MCMC algorithms, and more specifically nonparametric Bayesian methods; and iterative map-reduce platforms.

Moving on to the part of this thesis dedicated to GPstruct, chapter 3 contains a selfcontained reasoned introductory presentation of discriminative probabilistic models in machine learning, leading up to the structured family, with several variants of the CRF.

We provide the “missing block” in this very presentation: we formulate GPstruct, a conceptually novel probabilistic model, endowed with a set of desirable properties which existing models exhibit only in isolation. It is a kernelised, non-parametric model; it supports fully Bayesian inference; it applies to structured output prediction tasks.

GPstruct is formulated in a modular way, which makes it flexible and adaptable. In particular, configurable modules are: the MRF structure, the parameter tying (i.e. sharing) scheme, the kernel design, in particular the form of the kernel for label-label and input-input similarity, and the likelihood formulation. All of these options are illustrated in this thesis and most are experimentally illustrated. Thanks to the probabilistic properties of GPstruct, all are amenable to automatic model selection; indeed, as a specific contribution to this issue, we research and experiment with hyperparameter learning in chapter 6.

We develop an MCMC training algorithm for GPstruct based on elliptical slice sampling (Murray et al., 2010). An open-source, easily accessible Python implementation of GPstruct is provided[1], which allows the reproduction of almost all of the experiments presented in chapters 4, 5 and 6.

Our experiments in chapters 4 and 5 provide several contributions at once:

- we demonstrate the flexibility of the model by adapting it to both sequence and grid models, and to different forms of the likelihood and predictive distribution
- we explore very different machine intelligence tasks: part-of-speech tagging, noun phrase tagging, named entity recognition, lexical segmentation, video classification, semantic image segmentation
- we benchmark GPstruct against several state-of-the-art methods, which shows its excellent predictive performance.

In analytic terms, we conducted complementary experiments to better understand the impact of MCMC configuration choices (section 4.7), cross-chain MCMC variance (section 4.8), how a MAP version of GPstruct performs (section 4.10), and how good the probabilistic calibration of GPstruct predictions is with respect to other models (section 5.5.2).

We address the challenge of taking GPstruct to large-scale tasks in chapter 5. Several complementary strategies are applied to achieve our goal; we detail them in this thesis and experimentally investigate the impact of approximations on predictive accuracy, in a small-scale experiment where exact inference is tractable (section

[1]`http://github.com/sebastien-bratieres/pygpstruct`

5.5.5). The approximations we exploit are not specific to the grid topology used in this chapter; pseudo-likelihood approximations for GPstruct are reused in Srijith et al. (2016), and the ensemble method we apply is very general indeed.

Our findings were shared with the research community in our publications Bratières et al. (2015) and Bratières et al. (2014).

Probabilistically calibrated predictions are one advantage of the fully Bayesian algorithm used in GPstruct; in chapter 6 we explore another, hyperparameter learning during MCMC training for latent variables. We compare reparameterisation schemes which can be used for this purpose and compare their usefulness on our problem, to conclude that the computationally cheapest, the forward parameterisation, seems the best suited. We obtain mitigated results in terms of predictive performance, which we attribute to the nature of the experimental task, a hypothesis which is supported by evidence from our other experiments (as discussed in sections 6.5, 4.10, 4.12). However, we demonstrate that hyperparameter posteriors are consistently different from their priors, which proves that we have achieved hyperparameter learning after all.

Finally, we make an independent, methodological contribution in the form of a method to test the correctness of MCMC implementations, based on an adaptation of Geweke (2004), in section 6.3.

7.2 Concluding thoughts

Individual chapters already discuss conclusions, open issues and opportunities for future work. Here, we bring up broader issues which seem worth pursuing, pointing to fields of research which are connected to this thesis.

Firstly, our research implied deploying algorithms to distributed settings in chapter 2 and 5 because of the scale of computation. Despite improved platforms, this process is still cumbersome. We believe that a successful approach relies on a user specifying a computational procedure in the form of a graph, and a platform taking care of the deployment. Such an approach inspired numerous distributed platforms[2], but it is in deep learning libraries that it has seen most successes. In Theano (Al-Rfou et al., 2016) or Tensorflow (Abadi et al., 2016), the developer constructs a computation graph which is optimised, compiled and deployed to an adapted target platform: initially GPUs (graphical processing units), but now also computing clusters and specialised microchips. This paradigm fully acknowledges that the person who writes the code, maybe a data scientist or a researcher, is not an expert, and is ultimately not interested, in how the code is optimised. By implementing this separation of concerns, libraries built according to this model can also seize the opportunity to optimise deployments dynamically during execution, or over several redesign and recompila-

[2]like Pig and Spark, but cf. also the user-side desiderata expressed by Lin (2012) for a distributed platform in an industry setting.

tion cycles. Indeed, it can instrument the resulting code with counters and timers, has access to measurements, and often also to costs which determine trade-offs, such as costs of different disk storage options, memory, computation, or network communication on a cloud computing platform, all while integrating with user-imposed performance metrics. Machine learning applications will benefit from work on efficient deployment tools which separate platform issues from algorithm issues.

Secondly, Bayesian methods offer many advantages in theory, but it seems that they don't fare well in structured settings. One fundamental reason is that the faith in a Bayesian inference result rests in part on the asymptotic consistency guarantees it offers that with more data, its posterior will converge as a point on the true posterior. This, however, only applies when the model space contains the data-generating model; in other words, when our model for the world is correct. However, on problems of practical relevance, e.g. in machine intelligence, we are usually quite sure that our model is mis-specified; for the image segmentation problems in chapter 5, we can be certain that the dependency structure of neighbouring labels we specify with a CRF is wrong, since it does not model traits of the spatial world which gave rise to the image, such as the continuity of material objects. On the contrary, machine learning methods based on ensembles, which do not perform Bayesian model averaging and do not converge to a single best model in the limit of infinite data, seem to have an advantage on empirical "data science" problems. Therefore, is it reasonable to be Bayesian when our model is known to be mis-specified? Are there variants, generalisations (such as Zhu et al. (2014), Grünwald (2016), or more generally divergence minimisation schemes) of the Bayesian paradigm which give better results, while still offering rational consistency guarantees? Because it is not necessarily the case that a learning strategy which appears correct under rational analysis is optimal in a real setting, should we work on hybrids between fully Bayesian inference and other inductive principles, such as search, ensemble methods, energy-based methods?

These interrogations bring us to our last point, which relates to both previous arguments. A major difference between the ideal setting for rational decision and decision-making in the real world is that the latter occurs under constrained computing resources. In practice, a variety of aspects might be constrained, bounded, or budgeted: computation, due to battery power limitation on small devices; memory or storage capacity; there might be training costs vs. prediction costs trade-offs; feature acquisition (from the real world) or feature extraction (through computation) costs; likelihood evaluation costs or simulation costs. There already is work on these issues, with approximate Bayesian computation, Bayesian optimisation, active learning, model compression, limited-precision machine learning, feature selection, approximate inference algorithms (such as in chapter 5), anytime prediction methods. Focusing on structured prediction, Bolukbasi et al. (2017) and Shi et al. (2015) are two recent examples of prediction under budget constraints. Section 5.5.5 illustrates how to obtain a trade-off between training cost and predictive performance. There is a need for computationally aware algorithms, even adaptive methods which can react to changing budget constraints, and ways to learn costs and constraints based on measurements. Such techniques are needed in many applied machine learning scenarios, from robotics to data science.

All these questions are rhetorical, of course, and fortunately some have received consideration and partial answers; our argument here is that in our own research experience, reflected in this thesis, they hold potential for practical benefit and for our understanding of machine learning. As a field, machine learning has seen many successful applications in the last decade, and yet its most important problems seem to lie ahead of us.

Appendix A

IHMM Gibbs sampling steps

This appendix details each of the steps of the Gibbs sampling procedure illustrated as a whole in figure 2.2.1 and table 2.1.

It will prove convenient to introduce a central data structure, the `Sentence` object, which contains:

- an array of tokens, of length T;
- an array of currently assigned cluster IDs (also called states), of length T;
- an array of auxiliary variables (one between each pair of adjacent tokens), of length $T - 1$. Auxiliary variables are introduced below, in section A.

Thus, we can pack all tokens, states, and auxiliary variables into a set of `Sentence` objects. Let Σ be the set of sentences, $\sigma \in \Sigma$ a sentence, σ.length its length denoted by T above, σ.states $= (z_1 \dots z_T)$, σ.tokens $= (y_1 \dots y_T)$, and σ.auxiliaryVariables $= (\nu_1 \dots \nu_{T-1})$. Note that hyperparameters α, γ, and H are not considered inputs since they are fixed throughout the program, as is V. K, the current number of used states, is always deduced from the size of $\boldsymbol{\beta}$.

Count transition frequencies (MR1)

```
input: $\Sigma$
output: $\boldsymbol{p} \in \mathbb{N}^{K \times K}$

$p_{ij}$ is the frequency of the subsequence $(i, j)$ in the state sequences,
    for all sentences
```

Sample $\boldsymbol{\pi}$ (MR1a)

```
input: $\boldsymbol{p}$
```

```
output: a new draw for $\boldsymbol{\pi}$ from the posterior distribution

$\boldsymbol{p}'$:=extend $\boldsymbol{p}$ with a column of zeroes to have matching sizes with $\boldsymbol{\beta}$
Draw each $\pi_k$ from Dirichlet$(\alpha\boldsymbol{\beta} + \boldsymbol{p}'_{k,\cdot})$
```

Count emission frequencies (MR2)

```
input: $\Sigma$
output: $\boldsymbol{f} \in \mathbb{N}^{K \times V}$

$f_{kw}$ is the frequency of state $k$ generating token $w$, across all sentences
```

Sample $\boldsymbol{\phi}$ (MR2a)

```
input: $\boldsymbol{f}$
output: a new draw of $\boldsymbol{\phi}$ from the posterior distribution

Draw each $\phi_k$ from Dirichlet$(\underline{\mathtt{H}} + f_k)$
```

Sample auxiliary variables, and determine the minimum auxiliary variable (MR3)

The auxiliary variables are an addition to the IHMM's original model, and hence do not change any other variable's marginal distribution, so the MCMC procedure converges to the true posterior. They allow auxiliary variable Gibbs sampling, effectively cutting down on the computational cost of state re-estimation. Details can be found in Van Gael et al. (2008).

```
input:$\Sigma$, $\boldsymbol{\pi}$
output: $\forall\sigma \in \Sigma$, a new draw of $\sigma$.auxiliaryVariables;
        $\nu$, the smallest auxiliary variable drawn

foreach $\sigma$:
    foreach $t \in \{1 \ldots \sigma.\text{length} - 1\}$
        $\sigma$.auxiliaryVariables$[t]$:= draw from Uniform$(0, \pi_{z_t, z_{t+1}})$
```

Sample Chinese Restaurant Franchise table counts (MR4)

Details can be found in Van Gael and Ghahramani (2010). Each transition $i \to j$ is considered a customer who either sits at a new table or not, and the resulting table number is stored.

```
input: $\boldsymbol{f}$, $\boldsymbol{\beta}$
```

```
output: $\boldsymbol{t} \in \mathbb{N}^{K \times K}$

foreach $(i, j) \in \{1 \ldots K\}^2$
    $t_{ij}$ = tableCount($j$, $f_{ij}$, $\beta_j$)

tableCount:
input: $j$, $f_{ij}$, $\beta_j$
output: the number of tables generated by the transitions $i \to j$

initialise $t = 0$
foreach $m \in \{1 \ldots f_{ij}\}$
    draw $r$ from Bernoulli($\frac{\alpha\beta_j}{\alpha\beta_j + m - 1}$)
    $t$:=$t$+$r$
return $t$
```

Sample $\boldsymbol{\beta}$

Details can be found in Teh et al. (2006, equation 35).

```
input: $\boldsymbol{t}$, $\boldsymbol{\beta}$
output: a new draw of $\boldsymbol{\beta}$ from the posterior

$g$:= extend $\sum_{k=1}^{k=K} t_{k,\cdot}$ with element $\gamma$, yielding a vector of
    length $K + 1$
Draw $\boldsymbol{\beta}$ from Dirichlet($g$)
```

Note that this takes care of the remaining mass element, as it ensures that the resulting draw sums to one.

"Re-break" $\boldsymbol{\beta}$ and $\boldsymbol{\pi}$

```
input: $\nu$, $\boldsymbol{\pi}$, $\boldsymbol{\phi}$, $\boldsymbol{\beta}$
output: new, expanded versions of $\boldsymbol{\pi}$, $\boldsymbol{\phi}$, $\boldsymbol{\beta}$
```

We omit details of this step here. The idea is to iteratively expand ("rebreak") the three variables by extracting new states from the remaining mass, in order to ensure that after the process, none of the π_{ij} is larger than ν. This is done by drawing a breaking point according to a Beta(1, γ), and splitting the remaining mass according to this. Each rebreak increases K by 1. The exact procedure involves quite some book-keeping in order to enforce the following constraints on $\boldsymbol{\pi}$, $\boldsymbol{\phi}$, $\boldsymbol{\beta}$:

- Summation to 1

- Supervision for dummy start and end states, which must correspond to start and end tokens, and which have restrictions on
  - transitions (end cannot transition anywhere, no state can transition to start);
  - emissions (newly created state cannot emit the start or end tokens).

Sample states of entire sequence (MR5)

This is carried out using beam sampling (Van Gael et al., 2008), which is a dynamic program of complexity $O(\sigma.\text{length} \times K^2)$ in the worst case.

```
input: $\Sigma$, $\boldsymbol{\pi}$, $\boldsymbol{\phi}$
output: $\forall \sigma \in \Sigma$, a new draw of $\sigma$.states

for a given $\sigma$:
initialise $DP \in$
$\mathbb{R}^{K\times T}$ to hold temporary values of the dynamic program
for $t$=1 to $T$
    for $k$=0 to $K$
        for $l$=0 to $K$
            if $\pi_{lk} > \sigma.\texttt{auxiliaryValues}[t-1]$
                $DP_{kt} = DP_{kt} + DP_{l,t+1}$
        $DP_{kt} = DP_{kt} \cdot \phi_{k,\,\sigma.\texttt{tokens}[t]}$
now backtrack to sample the state sequence:
for $t$=$T$-1-1 to 0
    initialise $\tau \in \mathbb{R}^K$
    for $k$=0 to $K$
        $\tau_k = \begin{cases} 1 & \texttt{if } \pi_{k,\,\sigma.\texttt{states}[t+1]} > \sigma.\texttt{auxiliaryValues}[t] \\ 0 & \texttt{else} \end{cases}$
    draw $\sigma$.states$[t]$ from Categorical($\tau \circ DP_{\cdot,t}$),
```

Here $\circ$ denotes the Hadamard, or element-wise, product (of vectors of dimension K, in the present case).

List states effectively used in the new state sample (MRMarkUsedStates)

```
input: $\Sigma$
output: the list of unused states, obtained by going through all the $\sigma$.states
```

Collapse $\boldsymbol{\beta}$

```
input: $\boldsymbol{\beta}$, the list of unused states
```

```
output: a collapsed $\boldsymbol{\beta}$, where unused states no longer appear.
        Their probability mass is added to the remaining mass element $\beta_{K+1}$.
```

States are renumbered in this step and in the immediately following step. This cleaning-up step ensures that we always only consider the minimum number of states required.

Renumber states in state sample (MRCleanupUnusedStates)

```
input: $\Sigma$, the list of unused states
output: $\Sigma$, with the following constraint enforced:
        $\forall \sigma \in$
$\Sigma$, $\sigma$.states contains no element from the list of
            unused states
```

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., and Zheng, X. (2016). TensorFlow: A System for Large-scale Machine Learning. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, pages 265–283, Berkeley, CA, USA. USENIX Association. (cited p. 157)

Agarwal, A., Chapelle, O., Dudik, M., and Langford, J. (2012). A reliable effective terascale linear learning system. *arXiv preprint arXiv:1110.4198.* (cited p. 51)

Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., Bastien, F., Bayer, J., Belikov, A., Belopolsky, A., Bengio, Y., Bergeron, A., Bergstra, J., Bisson, V., Bleecher Snyder, J., Bouchard, N., Boulanger-Lewandowski, N., Bouthillier, X., de Brébisson, A., Breuleux, O., Carrier, P.-L., Cho, K., Chorowski, J., Christiano, P., Cooijmans, T., Côté, M.-A., Côté, M., Courville, A., Dauphin, Y. N., Delalleau, O., Demouth, J., Desjardins, G., Dieleman, S., Dinh, L., Ducoffe, M., Dumoulin, V., Ebrahimi Kahou, S., Erhan, D., Fan, Z., Firat, O., Germain, M., Glorot, X., Goodfellow, I., Graham, M., Gulcehre, C., Hamel, P., Harlouchet, I., Heng, J.-P., Hidasi, B., Honari, S., Jain, A., Jean, S., Jia, K., Korobov, M., Kulkarni, V., Lamb, A., Lamblin, P., Larsen, E., Laurent, C., Lee, S., Lefrancois, S., Lemieux, S., Léonard, N., Lin, Z., Livezey, J. A., Lorenz, C., Lowin, J., Ma, Q., Manzagol, P.-A., Mastropietro, O., McGibbon, R. T., Memisevic, R., van Merriënboer, B., Michalski, V., Mirza, M., Orlandi, A., Pal, C., Pascanu, R., Pezeshki, M., Raffel, C., Renshaw, D., Rocklin, M., Romero, A., Roth, M., Sadowski, P., Salvatier, J., Savard, F., Schlüter, J., Schulman, J., Schwartz, G., Serban, I. V., Serdyuk, D., Shabanian, S., Simon, É., Spieckermann, S., Subramanyam, S. R., Sygnowski, J., Tanguay, J., van Tulder, G., Turian, J., Urban, S., Vincent, P., Visin, F., de Vries, H., Warde-Farley, D., Webb, D. J., Willson, M., Xu, K., Xue, L., Yao, L., Zhang, S., and Zhang, Y. (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.0. (cited p. 157)

Altun, Y. (2005). *Discriminative Methods for Label Sequence Learning*. PhD thesis, Brown University. (cited p. 94)

Altun, Y. and Hofmann, T. (2003). Gaussian process classification for segmenting and annotating sequences (Technical Report CS-03-23). Technical report, Department of Computer Science, Brown University. (cited p. 74, 76)

Altun, Y., Hofmann, T., and Smola, A. J. (2004a). Gaussian process classification for segmenting and annotating sequences. In *Proceedings of the twenty-first International Conference on Machine Learning*, page 4. ACM. (cited p. 65, 74, 75, 76, 77, 84, 94)

Altun, Y., Smola, A. J., and Hofmann, T. (2004b). Exponential families for conditional random fields. In *UAI 2004: Proceedings of the Twentieth Conference Conference on Uncertainty in Artificial Intelligence*, Banff, Canada. AUAI Press, ISBN 0-9749039-0-6. (cited p. 94)

Altun, Y., Tsochantaridis, I., and Hofmann, T. (2003). Hidden Markov support vector machines. In *Proceedings of the 21st International conference on Machine Learning*, volume 20, page 3. (cited p. 76)

Alvarez, M. A., Rosasco, L., and Lawrence, N. D. (2011). Kernels for vector-valued functions: A review. *arXiv preprint arXiv:1106.6251*, pages 1–37. (cited p. 100)

Andrés-Ferrer, J., Ortiz-Martínez, D., García-Varea, I., and Casacuberta, F. (2008). On the use of different loss functions in statistical pattern recognition applied to machine translation. *Pattern Recognition Letters*, 29(8):1072–1081. (cited p. 62)

Antoniak, C. E. (1974). Mixtures of Dirichlet Processes with Applications to Bayesian Nonparametric Problems. *The Annals of Statistics*, 2(6):1152–1174. (cited p. 20, 29)

Antoniano-Villalobos, I. and Walker, S. G. (2013). Bayesian Nonparametric Inference for the Power Likelihood. *Journal of Computational and Graphical Statistics*, 22(4):801–813. (cited p. 115)

Ardia, D. (2008). Bayesian Estimation of the GARCH(1, 1) Model with Normal Innovations. In *Financial Risk Management with Bayesian Estimation of GARCH Models: Theory and Applications*, pages 17–37. Springer Berlin Heidelberg. (cited p. 132)

Armbrust, M., Fox, A., Griffith, R., Joseph, A., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., and Zaharia, M. (2009). Above the Clouds: A Berkeley View of Cloud Computing. Technical report, EECS Department, University of California, Berkeley. (cited p. 33)

Asuncion, A., Smyth, P., and Welling, M. (2008). Asynchronous distributed learning of topic models. *Advances in Neural Information Processing Systems*, pages 81–88. (cited p. 55, 56)

Babu, G. J. and Feigelson, E. D. (2006). Astrostatistics: Goodness-of-Fit and All That! In Gabriel, C., Arviset, C., Ponz, D., and Solano, E., editors, *Astronomical Data Analysis Software and Systems XV ASP Proceedings of the Conference Held 2-5 October 2005 in San Lorenzo de El Escorial, Spain*, page 127, San Francisco. Astronomical Society of the Pacific. (cited p. 132)

Bakir, G. H., Hofmann, T., Schölkopf, B., Smola, A. J., Taskar, B., and Vishwanathan, S. V. N. (2007). *Predicting Structured Data*. MIT Press. (cited p. 92)

Balan, A. K., Rathod, V., Murphy, K., and Welling, M. (2015). Bayesian Dark Knowledge. *CoRR*, abs/1506.0. (cited p. 119)

Bartlett, P. L., Jordan, M. I., and McAuliffe, J. D. (2006). Convexity, classification, and risk bounds. *Journal of the American Statistical Association*, 101(473):138–156. (cited p. 75)

Beal, M. J. and Ghahramani, Z. (2003). The Variational Bayesian EM Algorithm for Incomplete Data: with Application to Scoring Graphical Model Structures. *Bayesian Statistics*, 7:453–454. (cited p. 152)

Beal, M. J., Ghahramani, Z., and Rasmussen, C. E. (2002). The infinite hidden Markov model. In *Advances in Neural Information Processing Systems*, pages 14:577–584. (cited p. 20, 25, 31, 37)

Bekkerman, R., Bilenko, M., and Langford, J. (2012). Scaling Up Machine Learning: Parallel and Distributed Approaches. In *Proceedings of the 17th ACM SIGKDD International Conference Tutorials*, page 475. (cited p. 51)

Belanger, D. (2017). *Deep Energy-Based Models for Structured Prediction*. PhD thesis, University of Massachusetts Amherst. (cited p. 95)

Belanger, D. and McCallum, A. (2016). Structured Prediction Energy Networks. In *ICML*. (cited p. 95)

Belanger, D., Yang, B., and McCallum, A. (2017). End-to-End Learning for Structured Prediction Energy Networks. In *ICML*. (cited p. 95)

Berger, J. O. (1985). *Statistical decision theory and Bayesian analysis*. Springer-Verlag. (cited p. 61)

Besag, J. (1975). Statistical analysis of non-lattice data. *Journal of the Royal Statistical Society. Series D (The Statistician)*, 24(3):179–195. (cited p. 106, 107)

Bishop, C. M. (2006). *Pattern recognition and machine learning*. Springer, New York. (cited p. 20, 67, 69, 101)

Blake, A., Kohli, P., and Rother, C. (2011). *Markov random fields for vision and image processing*. MIT Press. (cited p. 92)

Blundell, C., Cornebise, J., Kavukcuoglu, K., and Wierstra, D. (2015). Weight Uncertainty in Neural Networks. In Bach, F. R. and Blei, D. M., editors, *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pages 1613–1622. Journal of Machine Learning Research. (cited p. 103)

Bo, L. and Sminchisescu, C. (2010). Twin Gaussian Processes for Structured Prediction. *International Journal of Computer Vision*, 87(1-2):28–52. (cited p. 99)

Bolukbasi, T., Chang, K.-W., Wang, J., and Saligrama, V. (2017). Resource Constrained Structured Prediction. In Singh, S. P. and Markovitch, S., editors, *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA.*, pages 1756–1762. AAAI Press. (cited p. 158)

Bouchard, G. (2007). Bias-variance tradeoff in hybrid generative-discriminative models. In *Sixth International Conference on Machine Learning and Applications (ICMLA 2007)*, pages 124–129. IEEE. (cited p. 63)

Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge University Press. (cited p. 73)

Brants, T., Popat, A. C., Xu, P., Och, F. J., and Dean, J. (2007). Large Language Models in Machine Translation. In Eisner, J., editor, *EMNLP-CoNLL 2007, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, June 28-30, 2007, Prague, Czech Republic*, pages 858–867. ACL. (cited p. 44)

Bratières, S., Quadrianto, N., and Ghahramani, Z. (2015). GPstruct: Bayesian Structured Prediction Using Gaussian Processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(7). (cited p. 21, 65, 79, 119, 120, 157)

Bratières, S., Quadrianto, N., Nowozin, S., and Ghahramani, Z. (2014). Scalable Gaussian process structured prediction for grid factor graph applications. In *Proceedings of the 31st International Conference on Machine Learning, ICML 2014*. (cited p. 21, 157)

Bratières, S., Van Gael, J., Vlachos, A., and Ghahramani, Z. (2010a). Learning the iHMM through iterative map-reduce. In *Unpublished. Poster at the Thirteenth International Conference on Artificial Intelligence and Statistics (AISTATS 2010)*, Chia Laguna, Sardinia, Italy. (cited p. 51, 155)

Bratières, S., Van Gael, J., Vlachos, A., and Ghahramani, Z. (2010b). Scaling the iHMM: Parallelization versus Hadoop. In *10th IEEE International Conference on Computer and Information Technology, CIT 2010, Bradford, West Yorkshire, UK, June 29-July 1, 2010*, pages 1235–1240. (cited p. 21, 44, 51)

Breiman, L. (1996). Bagging Predictors. *Machine Learning*, 24(2):123–140. (cited p. 108)

Brill, E. (1995). Transformation-Based Error-Driven Learning and Natural Language Processing : A Case Study in Part-of-Speech Tagging. *Computational Linguistics*, 21(4):543–565. (cited p. 26)

Brockwell, A. E. (2006). Parallel Markov chain Monte Carlo Simulation by Pre-Fetching. *Journal of Computational and Graphical Statistics*, 15(1):246–261. (cited p. 56)

Brown, P. F., DeSouza, P. V., Mercer, R. L., Della Pietra, V. J., and Lai, J. C. (1992). Class-Based n-gram Models of Natural Language. *Computational Linguistics*, 18:467–479. (cited p. 59)

Bu, Y., Howe, B., Balazinska, M., and Ernst, M. D. (2010). HaLoop: efficient iterative data processing on large clusters. *Proceedings of the VLDB Endowment*, 3(1-2):285–296. (cited p. 52)

Calderhead, B. (2014). A general construction for parallelizing Metropolis-Hastings algorithms. *Proceedings of the National Academy of Sciences of the United States of America*, 111(49):17408–13. (cited p. 56)

Chang, F., Dean, J., Ghemawat, S., Hsieh, W. C., Wallach, D. A., Burrows, M., Chandra, T., Fikes, A., and Gruber, R. (2006). Bigtable: A Distributed Storage System for Structured Data. In Bershad, B. N. and Mogul, J. C., editors, *7th Symposium on Operating Systems Design and Implementation (OSDI '06), November 6-8, Seattle, WA, USA*, pages 205–218. USENIX Association. (cited p. 33)

Chen, L.-C., Schwing, A. G., Yuille, A. L., and Urtasun, R. (2015). Learning deep structured models. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, pages 1785–1794. (cited p. 95)

Chou, W. and Juang, B. (2003). *Pattern recognition in speech and language processing*. CRC Press. (cited p. 62)

Christodoulopoulos, C., Goldwater, S., and Steedman, M. (2010). Two decades of unsupervised POS induction: How far have we come? *EMNLP 2010 - Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, pages 575–584. (cited p. 59)

Chu, C.-T., Kim, S. K., Lin, Y.-A., Yu, Y., Bradski, G. R., Ng, A. Y., and Olukotun, K. (2007). Map-Reduce for Machine Learning on Multicore. In Schölkopf, B., Platt, J. C., and Hoffman, T., editors, *Proceedings of the Twentieth Annual Conference on Neural Information Processing Systems*, pages 281–288, Vancouver. MIT Press. (cited p. 35)

Collins, M. (2002). Discriminative Training Methods for Hidden Markov Models: Theory and Experiments with Perceptron Algorithms. In *Proceedings of the ACL-02 Conference on Empirical Methods in Natural Language Processing - Volume 10*, EMNLP '02, pages 1–8, Stroudsburg, PA, USA. Association for Computational Linguistics. (cited p. 73)

Crammer, K. and Singer, Y. (2001). On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2:265–292. (cited p. 65, 76)

Crick, T., Hall, B. A., and Ishtiaq, S. (2015). Reproducibility as a Technical Specification. *CoRR*, abs/1504.0. (cited p. 34)

de Andrade e Silva, R. B. and Ghahramani, Z. (2006). Bayesian Inference for Gaussian Mixed Graph Models. In *UAI '06, Proceedings of the 22nd Conference in Uncertainty in Artificial Intelligence, Cambridge, MA, USA, July 13-16, 2006*. AUAI Press. (cited p. 100)

Dean, J. and Ghemawat, S. (2004). MapReduce: Simplified data processing on large clusters. In *Operating Systems Design and Implementation*, pages 137–149. (cited p. 20, 35, 36)

Della Pietra, S., Della Pietra, V. J., and Lafferty, J. (1997). Inducing features of random fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(4):380–393. (cited p. 73)

Dezfouli, A. and Bonilla, E. V. (2015). Scalable Inference for Gaussian Process Models with Black-Box Likelihoods. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28*, pages 1414–1422. Curran Associates, Inc. (cited p. 120)

Dillon, J. V. and Lebanon, G. (2010). Stochastic Composite Likelihood. *J. Mach. Learn. Res.*, 11:2597–2633. (cited p. 108)

Do, T.-M.-T. and Artières, T. (2010). Neural conditional random fields. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9. JMLR: W&CP. (cited p. 95)

Domke, J. (2013). Learning Graphical Model Parameters with Approximate Marginal Inference. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(10):2454–2467. (cited p. 108, 110, 111, 113)

Doshi-Velez, F., Miller, K., Van Gael, J., and Teh, Y. W. (2009). Variational inference for the Indian buffet process. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, volume 12, pages 12:137–144. (cited p. 35)

Dubey, A., Williamson, S., and Xing, E. P. (2014). Parallel Markov chain Monte Carlo for Pitman-Yor mixture models. In *Proceedings of the Conference on Uncertainty and Artificial Intelligence.* (cited p. 57)

Duvenaud, D. K., Lloyd, J., Grosse, R., Tenenbaum, J., and Ghahramani, Z. (2013). Structure Discovery in Nonparametric Regression through Compositional Kernel Search. *JMLR*, 28(3):1166–1174. (cited p. 153)

Ekanayake, J., Li, H., Zhang, B., Gunarathne, T., Bae, S.-H., Qiu, J., and Fox, G. (2010). Twister. In *Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing - HPDC '10*, page 810, New York, New York, USA. ACM Press. (cited p. 52)

Elnikety, E., Elsayed, T., and Ramadan, H. E. (2011). IHadoop: Asynchronous iterations for MapReduce. In *Proceedings - 2011 3rd IEEE International Conference on Cloud Computing Technology and Science, CloudCom 2011*, pages 81–90. (cited p. 52)

Ewens, W. J. (1988). Population genetics theory – the past and the future. In Lessard, S., editor, *Mathematical and Statistical Problems in Evolution.* University of Montréal Press, Montréal. (cited p. 30)

Filippone, M., Zhong, M., and Girolami, M. (2013). A comparative evaluation of stochastic-based inference methods for Gaussian process models. *Machine Learning*, 93(1):93–114. (cited p. 125, 128, 152, 153)

Finley, T. and Joachims, T. (2008). Training Structural SVMs when Exact Inference is Intractable. In *Proceedings of the Twenty-Fifth International Conference on Machine Learning (ICML 2008), Helsinki, Finland, June 5-9, 2008.* (cited p. 94)

Franc, V., Zien, A., and Schölkopf, B. (2011). Support Vector Machines as Probabilistic Models. In Getoor, L. and Scheffer, T., editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 665–672. Omnipress. (cited p. 102)

Fröhlich, B., Rodner, E., and Denzler, J. (2010). A Fast Approach for Pixelwise Labeling of Facade Images. In *Proceedings of the International Conference on Pattern Recognition (ICPR 2010).* (cited p. 110)

Fujimaki, R. and Hayashi, K. (2012). Factorized Asymptotic Bayesian Hidden Markov Models. In *Proceedings of the 29th International Conference on Machine Learning, ICML 2012, Edinburgh, Scotland, UK, June 26 - July 1, 2012.* (cited p. 25)

Fushiki, T., Komaki, F., and Aihara, K. (2005). Nonparametric bootstrap prediction. *Bernoulli*, 11(2):293–307. (cited p. 108)

Gal, Y. and Ghahramani, Z. (2014). Pitfalls in the use of Parallel Inference for the Dirichlet Process. In *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*, pages 208–216. (cited p. 57)

Gales, M. and Young, S. (2008). The Application of Hidden Markov Models in Speech Recognition. *Foundations and Trends in Signal Processing*, 1(3):195–304. (cited p. 115)

Galliani, P., Dezfouli, A., Bonilla, E. V., and Quadrianto, N. (2017). Gray-box inference for structured Gaussian process models. In *AISTATS.* (cited p. 119, 120)

Gao, J. and Johnson, M. (2008). A comparison of Bayesian estimators for unsupervised Hidden Markov Model POS taggers. *EMNLP '08: Proceedings of the Conference on Empirical Methods in Natural Language Processing*, (October):344–352. (cited p. 26)

Gelman, A., Carlin, J. B., Stern, H. S., and Rubin, D. B. (2009). *Bayesian Data Analysis.* CRC Press. (cited p. 129)

Geweke, J. (2004). Getting It Right: Joint Distribution Tests of Posterior Simulators. *Journal of the American Statistical Association*, 99(1):799–804. (cited p. 21, 129, 130, 131, 151, 157)

Gonzalez, J., Low, Y., and Guestrin, C. (2009). Residual splash for optimally parallelizing belief propagation. *Artificial Intelligence and Statistics (AISTATS).* (cited p. 55)

Gonzalez, J. E. (2014). Emerging systems for large-scale machine learning. *ICML tutorial.* (cited p. 43)

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative Adversarial Nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. Curran Associates, Inc. (cited p. 63)

Gordon, D. F. and Desjardins, M. (1995). Evaluation and selection of biases in machine learning. *Machine Learning*, 20(1-2):5–22. (cited p. 154)

Gould, S., Fulton, R., and Koller, D. (2009). Decomposing a Scene into Geometric and Semantically Consistent Regions. In *IEEE 12th International Conference on Computer Vision, ICCV 2009, Kyoto, Japan, September 27 - October 4, 2009.* (cited p. 110)

Green, P. J., Łatuszyński, K., Pereyra, M., and Robert, C. P. (2015). Bayesian computation: a summary of the current state, and samples backwards and forwards. *Statistics and Computing*, 25(4):835–862. (cited p. 55, 56)

Griffiths, T. L. and Goldwater, S. (2007). A fully Bayesian approach to unsupervised part-of-speech tagging. *ACL'07 - 45th Annual Meeting of the Association of Computational Linguistics*, pages 744–751. (cited p. 26)

Grosse, R. B. and Duvenaud, D. K. (2014). Testing MCMC code. *2014 NIPS workshop on Software Engineering for Machine Learning.* (cited p. 131)

Grünwald, P. (2016). Safe Probability. *CoRR*, abs/1604.0. (cited p. 158)

Gygli, M., Norouzi, M., and Angelova, A. (2017). Deep Value Networks Learn to Evaluate and Iteratively Refine Structured Outputs. In *ICML.* (cited p. 95)

Hammersley, J. M. and Clifford, P. (1971). Markov fields on finite graphs and lattices. *Unpublished manuscript.* (cited p. 72)

Hannun, A., Case, C., Casper, J., Catanzaro, B., Diamos, G., Elsen, E., Prenger, R., Satheesh, S., Sengupta, S., Coates, A., and Ng, A. Y. (2014). Deep Speech: Scaling up end-to-end speech recognition. *arXiv preprint.* (cited p. 25)

Heaukulani, C., Knowles, D. A., and Ghahramani, Z. (2014). Beta Diffusion Trees. *Proceedings of the 31st International Conference on Machine Learning (ICML 2014)*, 32(2011):1809–1817. (cited p. 132)

Hensman, J., Fusi, N., and Lawrence, N. D. (2013). Gaussian Processes for Big Data. In Nicholson, A. and Smyth, P., editors, *Proceedings of the Twenty-Ninth Conference on Uncertainty in Artificial Intelligence, UAI 2013, Bellevue, WA, USA, August 11-15, 2013.* AUAI Press. (cited p. 112, 117)

Hensman, J., Matthews, A. G. d. G., Filippone, M., and Ghahramani, Z. (2015). MCMC for variationally sparse Gaussian processes. *Advances in Neural Information Processing Systems*, pages 1648–1656. (cited p. 152)

Huang, S. and Renals, S. (2007). Hierarchical Pitman-Yor language models for ASR in meetings. In *Proceedings of the IEEE Workshop on Automatic Speech Recognition and Understanding*, volume 10. (cited p. 56)

Isard, M., Budiu, M., Yu, Y., Birrell, A., and Fetterly, D. (2007). Dryad: Distributed Data-parallel Programs from Sequential Building Blocks. In *Proceedings of the 2007 Eurosys Conference.* Association for Computing Machinery. (cited p. 33, 35, 54)

Jancsary, J., Nowozin, S., Sharp, T., and Rother, C. (2012). Regression Tree Fields - An Efficient, Non-parametric Approach to Image Labeling Problems. In *IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society. (cited p. 95)

Joachims, T., Finley, T., and Yu, C.-N. J. (2009). Cutting-plane training of structural SVMs. *Machine Learning*, 77(1):27–59. (cited p. 64)

Johnson, M. (2007). Why doesn't EM find good HMM POS-taggers. *EMNLP-CoNLL 2007, Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning, June 28-30, 2007, Prague, Czech Republic*, (June):296–305. (cited p. 26)

Keshet, J. (2014). Optimizing the Measure of Performance in Structured Prediction. In Nowozin, S., Gehler, P. V., Jancsary, J., and Lampert, C. H., editors, *Advanced Structured Prediction*, chapter 11. MIT Press. (cited p. 77)

Knorr-Held, L. and Rue, H. (2002). On block updating in Markov random field models for disease mapping. *Scandinavian Journal of Statistics*, 29(4):597–614. (cited p. 128)

Koller, D. and Friedman, N. (2009). *Probabilistic Graphical Models*. MIT Press. (cited p. 20, 72, 77)

Kowalczyk, W. and Vlassis, N. A. (2005). Newscast EM. *Advances in Neural Information Processing Systems*, 17. (cited p. 55)

Kuleshov, V. and Liang, P. (2015). Calibrated Structured Prediction. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R., editors, *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 3474–3482. (cited p. 102)

Kuss, M. and Rasmussen, C. E. (2005). Assessing Approximate Inference for Binary Gaussian Process Classification. *JMLR*, 6:1679–1704. (cited p. 71)

Kuss, M. and Rasmussen, C. E. (2006). Assessing Approximations for Gaussian Process Classification. In *Advances in Neural Information Processing Systems 2005*. (cited p. 71)

Lafferty, J., McCallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning 18*, pages 282–289. Morgan Kaufmann. (cited p. 18, 65, 71, 73)

Lafferty, J., Zhu, X., and Liu, Y. (2004). Kernel conditional random fields: representation and clique selection. In *ICML '04: Proceedings of the twenty-first international conference on Machine learning*, page 64, Banff. ACM. (cited p. 65, 74, 75, 76, 77, 82)

Lakshman, A. and Malik, P. (2010). Cassandra: A Decentralized Structured Storage System. *SIGOPS Oper. Syst. Rev.*, 44(2):35–40. (cited p. 33)

Lampert, C. H. (2009). *Kernel Methods in Computer Vision*, volume 4. (cited p. 140)

Laptev, I., Marszalek, M., Schmid, C., and Rozenfeld, B. (2008). Learning realistic human actions from movies. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE. (cited p. 88)

Lasserre, J. (2008). *Hybrid of Generative and Discriminative Methods for Machine Learning*. PhD thesis, University of Cambridge. (cited p. 63)

Lasserre, J. and Bishop, C. M. (2007). Generative or Discriminative? Getting the Best of Both Worlds. In Bernardo, J. M., Bayarri, M. J., Berger, J. O., Dawid, A. P., Heckerman, D., Smith, A. F. M., and West, M., editors, *Bayesian Statistics*, volume 8, pages 3–24. Oxford University Press. (cited p. 63)

Lawrence, N. D. (2005). Probabilistic Non-linear Principal Component Analysis with Gaussian Process Latent Variable Models. *The Journal of Machine Learning Research*, 6:1783–1816. (cited p. 99)

Lehmann, E. L. and Casella, G. (1998). *Theory of point estimation*. Springer. (cited p. 61)

Liang, P. and Jordan, M. I. (2008). An asymptotic analysis of generative, discriminative, and pseudolikelihood estimators. In *Proceedings of the 25th international conference on Machine learning - ICML '08*, pages 584–591, New York, New York, USA. ACM Press. (cited p. 63)

Lin, J. (2012). MapReduce is Good Enough? If All You Have is a Hammer, Throw Away Everything That's Not a Nail! *arXiv preprint*. (cited p. 50, 51, 157)

Lovell, D., Adams, R. P., and Mansinghka, V. K. (2012). Parallel Markov chain Monte Carlo for Dirichlet process mixtures. In *NIPS Workshop on Big Learning*. (cited p. 57)

Ma, X. and Hovy, E. (2016). End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 1064–1074, Stroudsburg, PA, USA. Association for Computational Linguistics. (cited p. 95)

MacKay, D. J. C. (1996). Bayesian Methods for Backpropagation Networks. In E. Domany, J. L. v. H. and Schulten, &. K., editors, *Models of neural networks III*, chapter 6, pages 211–254. Springer, Berlin. (cited p. 139)

Malewicz, G., Austern, M. H., Bik, A. J., Dehnert, J. C., Horn, I., Leiser, N., and Czajkowski, G. (2010). Pregel. In *Proceedings of the 2010 international conference on Management of data - SIGMOD '10*, page 135, New York, New York, USA. ACM Press. (cited p. 53)

Marroquin, J., Mitter, S., and Poggio, T. (1987). Probabilistic solution of ill-posed problems in computational vision. *Journal of the American Statistical Association*, 82(397):76–89. (cited p. 106)

Meila, M. (2007). Comparing clusterings – an information based distance. *Journal of Multivariate Analysis*, 98(5):873–895. (cited p. 27)

Minka, T. P. (2005). Discriminative models, not discriminative training. Technical report, Microsoft Research Ltd. (cited p. 62)

Mooij, J. M. (2010). libDAI: A Free and Open Source C++ Library for Discrete Approximate Inference in Graphical Models. *Journal of Machine Learning Research*, 11:2169–2173. (cited p. 115)

Moore, D. A. and Russell, S. J. (2015). Gaussian process random fields. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, pages 3357–3365. MIT Press. (cited p. 99)

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT Press. (cited p. 20, 67, 69, 77, 101)

Murray, I. and Adams, R. P. (2010). Slice sampling covariance hyperparameters of latent Gaussian models. In *NIPS*, pages 1732–1740. (cited p. 125, 126, 127, 128, 129, 134, 153)

Murray, I., Adams, R. P., and Mackay, D. J. C. (2010). Elliptical slice sampling. *Artificial Intelligence and Statistics (AISTATS)*, (2):541–548. (cited p. 21, 85, 156)

Murray, I. and Ghahramani, Z. (2004). Bayesian learning in undirected graphical models: approximate MCMC algorithms. In *Uncertainty in Artificial Intelligence*. (cited p. 107)

Murray, I., Ghahramani, Z., and MacKay, D. J. C. (2006). MCMC for doubly-intractable distributions. In *UAI 2006, Proceedings of the 22nd Conference in Uncertainty in Artificial Intelligence, Cambridge, MA, USA, July 13-16, 2006*. AUAI Press. (cited p. 107)

Nallapati, R., Cohen, W., and Lafferty, J. (2007). Parallelized Variational EM for Latent Dirichlet Allocation: An Experimental Evaluation of Speed and Scalability. In *ICDMW'07: Proceedings of the Seventh IEEE International Conference on Data Mining Workshops*, pages 349–354. IEEE Computer Society. (cited p. 35, 55)

Neal, R. M. (2003). Slice sampling. *Annals of Statistics*, 31(3):705–767. (cited p. 125)

Neal, R. M. (2011). MCMC using Hamiltonian dynamics. *Handbook of Markov Chain Monte Carlo*, pages 113–162. (cited p. 125)

Newman, D., Asuncion, A., Smyth, P., and Welling, M. (2007). Distributed inference for latent Dirichlet allocation. *Advances in Neural Information Processing Systems*, 20(1081-1088):17–24. (cited p. 55, 56)

Ng, A. Y. and Jordan, M. I. (2001). On Discriminative vs. Generative Classifiers: A comparison of logistic regression and naive Bayes. In *Advances in Neural Information Processing Systems*, pages 841–848. (cited p. 63)

Nguyen, T. V. and Bonilla, E. V. (2014). Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 1404–1412. (cited p. 99)

Nickisch, H. (2008). Approximations for Binary Gaussian Process Classification. *JMLR*, 9:2035–2078. (cited p. 71, 84)

Nowozin, S., Gehler, P. V., Jancsary, J., and Lampert, C. H. (2014). *Advanced Structured Prediction.* MIT Press. (cited p. 92)

Nowozin, S. and Lampert, C. H. (2010). Structured Learning and Prediction in Computer Vision. *Foundations and Trends in Computer Graphics and Vision*, 6(3-4):185–365. (cited p. 19, 92, 108)

Nowozin, S., Rother, C., Bagon, S., Sharp, T., Yao, B., and Kohli, P. (2011). Decision tree fields. In Metaxas, D. N., Quan, L., Sanfeliu, A., and Gool, L. J. V., editors, *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, pages 1668–1675. IEEE Computer Society. (cited p. 110, 112, 121)

Olea, R. A. and Pawlowsky-Glahn, V. (2009). Kolmogorov–Smirnov test for spatially correlated data. *Stochastic Environmental Research and Risk Assessment*, 23(6):749–757. (cited p. 132)

Owen, S., Anil, R., Dunning, T., and Friedman, E. (2011). *Mahout in Action.* Manning Publications Co., Greenwich, CT, USA. (cited p. 35)

Panda, B., Herbach, J. S., Basu, S., and Bayardo, R. J. (2009). PLANET: massively parallel learning of tree ensembles with MapReduce. *VLDB*, 2(2):1426–1437. (cited p. 51)

Parise, S. and Welling, M. (2005). Learning in Markov Random Fields: An Empirical Study. In *Joint Statistical Meeting*. (cited p. 107)

Pedersen, M. S., Baxter, B., Templeton, B., Rishøj, C., Theobald, D. L., Hoegh-rasmussen, E., Casteel, G., Gao, J. B., Dedecius, K., Strim, K., Christiansen, L., Hansen, L. K., Wilkinson, L., He, L., Bar, M., Winther, O., Sakov, P., and Hattinger, S. (2008). The Matrix Cookbook. Technical report, Technical University of Denmark. (cited p. 67, 68)

Pérez, F. and Granger, B. E. (2007). IPython: A system for interactive scientific computing. *Computing in Science and Engineering*, 9(3):21–29. (cited p. 34)

Perez-Cruz, F., Pontil, M., and Ghahramani, Z. (2007). Conditional graphical models. In *Predicting Structured Data*, pages 265–282. MIT Press, Cambridge, MA (USA). (cited p. 74, 75)

Platt, J. (1999). Probabilistic outputs for support vector machines and comparisons to regularized likelihood methods. *Advances in large margin classifiers*, 10(3):61–74. (cited p. 102)

Plummer, M., Best, N., Cowles, K., and Vines, K. (2006). CODA: Convergence Diagnosis and Output Analysis for MCMC. *R News*, 6(1):7–11. (cited p. 130)

Qi, Y., Szummer, M., and Minka, T. P. (2005). Bayesian Conditional Random Fields. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*. (cited p. 65, 74)

Qiu, Z., Wu, B., Wang, B., Shi, C., and Yu, L. (2014). Collapsed Gibbs Sampling for Latent Dirichlet Allocation on Spark. *JMLR: Workshop and Conference Proceedings*, (2004):17–28. (cited p. 57)

Quiñonero-Candela, J. and Rasmussen, C. E. (2005). A Unifying View of Sparse Approximate Gaussian Process Regression. *Journal of Machine Learning Research*, 6:1939–1959. (cited p. 117)

Rabiner, L. (1989). A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286. (cited p. 24)

Rainforth, T., Naesseth, C. A., Lindsten, F., Paige, B., van de Meent, J.-W., Doucet, A., and Wood, F. (2016). Interacting Particle Markov Chain Monte Carlo. In *International Conference on Machine Learning*, volume 48. (cited p. 57)

Rakitsch, B. and Lippert, C. (2013). It is all in the noise: Efficient multi-task Gaussian process inference with structured residuals. *Advances in Neural . . .*, pages 1–9. (cited p. 100)

Rasmussen, C. E. and Williams, C. K. I. (2006). *Gaussian Processes for Machine Learning*. MIT Press. (cited p. 20, 65, 69, 70, 71, 136, 139)

Rosen, J., Polyzotis, N., Borkar, V., and Bu, Y. (2013). Iterative MapReduce for Large Scale Machine Learning. *arXiv preprint*. (cited p. 51, 52)

Rue, H., Martino, S., and Chopin, N. (2009). Approximate Bayesian inference for latent Gaussian models by using integrated nested Laplace approximations. *Journal of the Royal Statistical Society. Series B: Statistical Methodology*, 71(2):319–392. (cited p. 125)

Salvatier, J., Wiecki, T. V., and Fonnesbeck, C. (2016). Probabilistic programming in Python using PyMC3. *PeerJ Computer Science*, 2:e55. (cited p. 130)

Sandve, G. K., Nekrutenko, A., Taylor, J., Hovig, E., Crocker, J., Cooper, M., Jasny, B., Chin, G., Chong, L., Vignieri, S., Peng, R., Mesirov, J., Nekrutenko, A., Taylor, J., Ioannidis, J., Allison, D., Ball, C., Coulibaly, I., Cui, X., Steen, R., Prinz, F., Schlange, T., Asadullah, K., Begley, C., Ellis, L., Reich, M., Liefeld, T., Gould, J., Lerner, J., Tamayo, P., Giardine, B., Riemer, C., Hardison, R., Burhans, R., Elnitski, L., Rex, D., Ma, J., Toga, A., Oinn, T., Addis, M., Ferris, J., Marvin, D., Senger, M., Piwowar, H., Day, R., Fridsma, D., Schwab, M., Karrenbach, M., Claerbout, J., Goble, C., Bhagat, J., Aleksejevs, S., Cruickshank, D., Michaelides, D., Deelman, E., Singh, G., Su, M.-H., Blythe, J., Gil, Y., Goecks, J., Nekrutenko, A., Taylor, J., Brazma, A., Hingamp, P., Quackenbush, J., Sherlock, G., Spellman, P., Brazma, A., Parkinson, H., Sarkans, U., Shojatalab, M., Vilo, J., Edgar, R., Domrachev, M., Lash, A., Sneddon, T., Li, P., Edmunds, S., Prlić, A., and Procter, J. (2013). Ten Simple Rules for Reproducible Computational Research. *PLoS Computational Biology*, 9(10):e1003285. (cited p. 34)

Schaul, T., Antonoglou, I., and Silver, D. (2013). Unit Tests for Stochastic Optimization. *CoRR*, abs/1312.6. (cited p. 130)

Schmidhuber, J. (2015). Deep learning in neural networks: An overview. *Neural Networks*, 61:85–117. (cited p. 22)

Schmidt, M. (2009). A Note on Structural Extensions of SVMs. (cited p. 76)

Schölkopf, B. and Smola, A. J. (2002). *Learning with Kernels*. MIT Press. (cited p. 20, 65, 69, 75)

Schwarz, G. (1978). Estimating the Dimension of a Model. *Annals of Statistics*, 6(2):461–464. (cited p. 25)

Scott, S. L., Blocker, A. W., Bonassi, F. V., Chipman, H. A., George, E. I., and McCulloch, R. E. (2013). Bayes and big data: the consensus Monte Carlo algorithm. In *EFaBBayes 250 Conference*. (cited p. 56)

Scott, S. L., Blocker, A. W., Bonassi, F. V., Chipman, H. A., George, E. I., and McCulloch, R. E. (2016). Bayes and big data: the consensus Monte Carlo algorithm. *International Journal of Management Science and Engineering Management*, 11(2):78–88. (cited p. 56)

Sculley, D., Holt, G., Golovin, D., Davydov, E., Phillips, T., Ebner, D., Chaudhary, V., Young, M., Crespo, J.-F., and Dennison, D. (2015). Hidden technical debt in Machine learning systems. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, pages 2503–2511. MIT Press. (cited p. 51)

Seeger, M. (2004). Gaussian processes for machine learning. *International Journal of Neural Systems*, 14(2):69–106. (cited p. 69)

Sethuraman, J. (1994). A constructive definition of Dirichlet priors. *Statistica Sinica*, 4(2):639–650. (cited p. 32)

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., and de Freitas, N. (2016). Taking the Human Out of the Loop: A Review of Bayesian Optimization. *Proceedings of the IEEE*, 104(1):148–175. (cited p. 123)

Shi, T., Steinhardt, J., and Liang, P. (2015). Learning Where to Sample in Structured Prediction. In Lebanon, G. and Vishwanathan, S. V. N., editors, *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2015, San Diego, California, USA, May 9-12, 2015*, volume 38 of *JMLR Workshop and Conference Proceedings*. JMLR.org. (cited p. 158)

Shotton, J., Fitzgibbon, A. W., Cook, M., Sharp, T., Finocchio, M., Moore, R., Kipman, A., and Blake, A. (2011). Real-time human pose recognition in parts from single depth images. In *24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2011, Colorado Springs, CO, USA, 20-25 June 2011*. (cited p. 112)

Snelson, E. and Ghahramani, Z. (2005a). Compact approximations to Bayesian predictive distributions. In Raedt, L. D. and Wrobel, S., editors, *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, volume 119 of *ACM International Conference Proceeding Series*, pages 840–847. ACM. (cited p. 119)

Snelson, E. and Ghahramani, Z. (2005b). Sparse Gaussian Processes using Pseudo-inputs. In *Advances in Neural Information Processing Systems*. (cited p. 117)

Sollich, P. (2002). Bayesian Methods for Support Vector Machines: Evidence and Predictive Class Probabilities. *Machine Learning*, 46(1/3):21–52. (cited p. 76)

Srijith, P. K., Balamurugan, P., and Shevade, S. (2014a). Efficient Variational Inference for Gaussian Process Structured Prediction. In *Advances in Variational Inference NIPS 2014 Workshop*. (cited p. 119)

Srijith, P. K., Balamurugan, P., and Shevade, S. (2014b). Gaussian Process Pseudo-Likelihood Models for Sequence Labeling. (cited p. 119)

Srijith, P. K., Balamurugan, P., and Shevade, S. K. (2016). Gaussian Process Pseudo-Likelihood Models for Sequence Labeling. In Frasconi, P., Landwehr, N., Manco, G., and Vreeken, J., editors, *Machine Learning and Knowledge Discovery in Databases - European Conference, ECML PKDD 2016, Riva del Garda, Italy, September 19-23, 2016, Proceedings, Part I*, volume 9851 of *Lecture Notes in Computer Science*, pages 215–231. Springer. (cited p. 119, 120, 152, 157)

Stern, D., Herbrich, R., and Graepel, T. (2009). Matchbox: Large Scale Online Bayesian Recommendations. In *Proceedings of the 18th international conference on World wide web*, pages 111–120. ACM, New York, USA. (cited p. 55)

Stoyanov, V. and Eisner, J. (2012). Minimum-risk training of approximate CRF-based NLP systems. In *Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, June 3-8, 2012, Montréal, Canada*, pages 120–130. (cited p. 94)

Stoyanov, V., Ropson, A., and Eisner, J. (2011). Empirical risk minimization of graphical model parameters given approximate inference, decoding, and model structure. In *Artificial Intelligence and Statistics (AISTATS).* (cited p. 94)

Strid, I. (2008). Metropolis-Hastings prefetching algorithms. In *Working Paper Series in Economics and Finance*, Working Paper Series in Economics and Finance. Stockholm School of Economics, Stockholm School of Economics. (cited p. 56)

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *NIPS*, pages 3104–3112. MIT Press. (cited p. 95)

Sutton, C. (2012). An Introduction to Conditional Random Fields. *Foundations and Trends in Machine Learning*, 4(4):267–373. (cited p. 65, 87, 92)

Tanner, M. and Wong, W. H. (2010). From EM to Data Augmentation: The Emergence of MCMC Bayesian Computation in the 1980s. *Statistical Science*, 25(4):506–516. (cited p. 22)

Taskar, B. (2004). *Learning Structured Prediction Models: A Large Margin Approach.* PhD thesis, Stanford University. (cited p. 101)

Taskar, B., Guestrin, C., and Koller, D. (2004). Max-margin Markov networks. In *Advances in Neural Information Processing Systems.* (cited p. 65, 76, 77)

Teh, Y. W., Jordan, M. I., Beal, M. J., and Blei, D. M. (2006). Hierarchical Dirichlet Processes. *Journal of the American Statistical Association*, 101(476):1566–1581. (cited p. 30, 31, 37, 162)

Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Anthony, S., Liu, H., Wyckoff, P., and Murthy, R. (2009). Hive: A Warehousing Solution over a Map-reduce Framework. *Proc. VLDB Endow.*, 2(2):1626–1629. (cited p. 33)

Titsias, M. K. (2009). Variational Learning of Inducing Variables in Sparse Gaussian Processes. In Dyk, D. V. and Welling, M., editors, *Proceedings of the Twelfth International Conference on Artificial Intelligence and Statistics (AISTATS-09)*, volume 5, pages 567–574. Journal of Machine Learning Research - Proceedings Track. (cited p. 120)

Tresp, V. (2000). A Bayesian committee machine. *Neural Computation*, 12:2719–2741. (cited p. 108)

Tripuraneni, N., Gu, S., Ge, H., and Ghahramani, Z. (2015). Particle Gibbs for Infinite Hidden Markov Models. In *Proceedings of the 28th International Conference on Neural Information Processing Systems*, pages 2395–2403. MIT Press. (cited p. 57, 58)

Tsochantaridis, I., Hofmann, T., Joachims, T., and Altun, Y. (2004). Support vector machine learning for interdependent and structured output spaces. *Proceedings of the Twenty-first International Conference on Machine Learning (ICML 2004), Banff, Alberta, Canada, July 4-8, 2004*, page 104. (cited p. 76)

Tsochantaridis, I., Joachims, T., Hofmann, T., Altun, Y., and Singer, Y. (2005). Large margin methods for structured and interdependent output variables. *Journal of Machine Learning Research*, 6(2):1453–1484. (cited p. 65, 76, 84)

Tu, L. and Gimpel, K. (2018). Learning approximate inference networks for structured prediction. In *ICLR*. (cited p. 95)

Uřičář, M., Franc, V., and Hlaváč, V. (2013). Bundle Methods for Structured Output Learning—Back to the Roots. *Image Analysis*, (2). (cited p. 94)

Urtasun, R. and Darrell, T. (2007). Discriminative Gaussian process latent variable model for classification. In *Proceedings of the 24th international conference on Machine learning - ICML '07*, pages 927–934, New York, New York, USA. ACM Press. (cited p. 99)

Valiant, L. G. (1990). A bridging model for parallel computation. *Communications of the ACM*, 33(8):103–111. (cited p. 53)

Van Gael, J. and Ghahramani, Z. (2010). Nonparametric Hidden Markov Models. In *Bayesian Time-Series Models*. Cambridge University Press. (cited p. 161)

Van Gael, J., Saatci, Y., Teh, Y. W., and Ghahramani, Z. (2008). Beam sampling for the infinite hidden Markov model. In *ICML '08: Proceedings of the 25th international* *conference on Machine learning*, pages 1088–1095, Helsinki, Finland. ACM. (cited p. 43, 57, 155, 161, 163)

Van Gael, J., Vlachos, A., and Ghahramani, Z. (2009). The infinite HMM for unsupervised PoS tagging. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, volume 2, pages 678–687. Association for Computational Linguistics. (cited p. 24, 27, 39, 44, 45, 49)

VanDerwerken, D. N. and Schmidler, S. C. (2013). Parallel Markov Chain Monte Carlo. *arXiv preprint*. (cited p. 56)

Vlachos, A. (2011). Evaluating unsupervised learning for natural language processing tasks. In *Proceedings of the First workshop on Unsupervised Learning in NLP*, pages 35–42, Edinburgh, Scotland. Association for Computational Linguistics. (cited p. 27)

Wainwright, M. J. and Jordan, M. I. (2008). Graphical Models, Exponential Families, and Variational Inference. *Foundations and Trends in Machine Learning*, 1(1–2):1–305. (cited p. 108)

Walker, S. and Hjort, N. L. (2001). On Bayesian Consistency. *Journal of the Royal Statistical Society. Series B (Statistical Methodology)*, 63(4):811–821. (cited p. 115)

Wang, L., Liu, J., and Li, S. Z. (2000). MRF parameter estimation by MCMC method. *Pattern Recognition*, 33(11):1919–1925. (cited p. 107)

Wang, L.-H., Liu, J., Li, Y.-F., and Zhou, H.-B. (2004). Predicting protein secondary structure by a support vector machine based on a new coding scheme. *Genome informatics. International Conference on Genome Informatics*, 15(2):181–90. (cited p. 18)

Wang, Y., Bai, H., Stanton, M., Chen, W.-Y., and Chang, E. Y. (2009). PLDA: Parallel Latent Dirichlet Allocation for Large-Scale Applications. In *Algorithmic Aspects in Information and Management*, pages 301–314. Springer-Verlag. (cited p. 35, 56)

Weiss, M. S. (1978). Modification of the Kolmogorov-Smirnov Statistic for Use with Correlated Data. *Journal of the American Statistical Association*, 73(364):872. (cited p. 132)

Welling, M., Teh, Y. W., and Kappen, B. (2008). Hybrid variational-MCMC inference in Bayesian networks. In *Uncertainty in Artificial Intelligence*. (cited p. 117)

Weston, J., Chapelle, O., Vapnik, V., Elisseeff, A., and Schölkopf, B. (2002). Kernel Dependency Estimation. In Thrun, S. and Obermayer, K., editors, *Advances in Neural Information Processing Systems 15*, pages 873–880. MIT Press, Cambridge, MA. (cited p. 99)

Weston, J. and Watkins, C. (1998). Multi-class support vector machines (Technical Report CSD-TR-98-04). Technical report, Department of Computer Science, Royal Holloway, University of London. (cited p. 76)

Weston, J. and Watkins, C. (1999). Support vector machines for multi-class pattern recognition. In *Proceedings of the Seventh European Symposium On Artificial Neural Networks*, volume 4. (cited p. 65)

White, T. (2009). *Hadoop: The Definitive Guide.* O'Reilly & Associates Inc. (cited p. 20, 33, 36)

Williams, C. K. I. and Barber, D. (1998). Bayesian Classification With Gaussian Processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351. (cited p. 65, 70)

Williamson, S., Dubey, A., and Xing, E. P. (2013). Parallel Markov chain Monte Carlo for nonparametric mixture models. In *Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16-21 June 2013.* (cited p. 57)

Williamson, S., Wang, C., Heller, K. A., and Blei, D. M. (2015). Nonparametric mixed membership models using the IBP compound Dirichlet process. In *NIPS Workshop on Bayesian Nonparametrics: The Next Generation.* (cited p. 57)

Wilson, A. G., Knowles, D. A., and Ghahramani, Z. (2012). Gaussian Process Regression Networks. In *Proceedings of the 29th International Conference on on Machine Learning*, pages 1139–1146. (cited p. 17, 97)

Wolfe, J., Haghighi, A., and Klein, D. (2008). Fully distributed EM for very large datasets. In *Proceedings of the 25th international conference on Machine learning*, pages 1184–1191, Helsinki, Finland. ACM. (cited p. 35, 55)

Wood, F. and Griffiths, T. L. (2007). Particle Filtering for Nonparametric Bayesian Matrix Factorization. In *Advances in Neural Information Processing Systems*, volume 19, pages 1513–1520. (cited p. 56)

Xue, J.-H. and Titterington, D. M. (2008). Comment on "On Discriminative vs. Generative Classifiers: A Comparison of Logistic Regression and Naive Bayes". *Neural Processing Letters*, 28(3):169–187. (cited p. 63)

Xue, J.-H. and Titterington, D. M. (2009). Interpretation of hybrid generative/discriminative algorithms. *Neurocomputing*, 72(7-9):1648–1655. (cited p. 63)

Yang, J., van Dalen, R. C., Zhang, S.-X., and Gales, M. J. F. (2014). Infinite structured support vector machines for speech recognition. In *IEEE International Conference* *on Acoustics, Speech and Signal Processing, ICASSP 2014, Florence, Italy, May 4-9, 2014*, pages 3320–3324. IEEE. (cited p. 99)

Yu, Y., Isard, M., Fetterly, D., Budiu, M., Erlingsson, Ú., Gunda, P. K., and Currey, J. (2008). DryadLINQ: A System for General-Purpose Distributed Data-Parallel Computing Using a High-Level Language. In USENIX, editor, *OSDI'08: Eighth Symposium on Operating System Design and Implementation.* (cited p. 35, 54)

Yu, Y. and Meng, X.-L. (2011). To Center or Not to Center: That Is Not the Question—An Ancillarity–Sufficiency Interweaving Strategy (ASIS) for Boosting MCMC Efficiency. *Journal of Computational and Graphical Statistics*, 20(3):531–570. (cited p. 125, 126, 128)

Yuille, A. and He, X. (2012). Probabilistic models of vision and max-margin methods. *Frontiers of Electrical and Electronic Engineering in China*, 7(1):94–106. (cited p. 94)

Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S., and Stoica, I. (2010). Spark: Cluster computing with working sets. In *2nd USENIX Workshop on Hot Topics in Cloud Computing, HotCloud'10, Boston, MA, USA, June 22, 2010.* (cited p. 54)

Zhai, K., Boyd-Graber, J., Asadi, N., and Alkhouja, M. L. (2012). Mr. LDA: a Flexible Large Scale Topic Modeling Package using Variational Inference in MapReduce. *Proceedings of the 21st international conference on World Wide Web - WWW '12*, page 879. (cited p. 57)

Zhang, A., Gultekin, S., and Paisley, J. (2016a). Stochastic Variational Inference for the HDP-HMM. In *Proceedings of the 19th International Conference on Artificial Intelligence and Statistics*, pages 800–808, Cadiz, Spain. (cited p. 58)

Zhang, A., Zhu, J., and Zhang, B. (2014). Max-Margin Infinite Hidden Markov Models. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 315–323. (cited p. 99)

Zhang, Y., Chen, S., Wang, Q., and Yu, G. (2016b). i2MapReduce: Incremental mapreduce for mining evolving big data. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 1482–1483. IEEE. (cited p. 52)

Zhang, Y., Gao, Q., Gao, L., and Wang, C. (2012). iMapReduce: A Distributed Computing Framework for Iterative Computation. *Journal of Grid Computing*, 10(1):47–68. (cited p. 52)

Zhong, G., Li, W.-J., Yeung, D.-Y., Hou, X., and Liu, C.-L. (2010). Gaussian process latent random field. In *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence*, pages 679–684. AAAI Press. (cited p. 99)

Zhu, J., Chen, N., and Xing, E. P. (2011). Infinite SVM: a Dirichlet Process Mixture of Large-margin Kernel Machines. In Getoor, L. and Scheffer, T., editors, *Proceedings of the 28th International Conference on Machine Learning, ICML 2011, Bellevue, Washington, USA, June 28 - July 2, 2011*, pages 617–624. Omnipress. (cited p. 99)

Zhu, J., Chen, N., and Xing, E. P. (2014). Bayesian inference with posterior regularization and applications to infinite latent SVMs. *Journal of Machine Learning Research*, 15(1):1799–1847. (cited p. 99, 158)

Zhu, J. and Hastie, T. (2005). Kernel Logistic Regression and the Import Vector Machine. *Journal of Computational and Graphical Statistics*, 14(1):185–205. (cited p. 75)