Extreme multi-label learning with Gaussian processes

Aristeidis Panos

Department of Statistical Science

University College London

This dissertation is submitted for the degree of
Doctor of Philosophy

August 2019

Abstract

In modern probabilistic machine learning, Gaussian process models have provided both powerful and principled ways to approach a series of challenging problems. Nonetheless, their applicability can be significantly limited by cases where the number of training data points is large, something very typical in many modern machine learning applications. An additional restriction can be imposed when the posterior distribution is intractable due to non-Gaussian likelihoods used. Despite the fact that these two limitations have been efficiently addressed over the last decade, applications of Gaussian process models under extreme regimes where the number of the training data points and the dimensionality of both input and output space is extremely large have not appeared in literature so far. This thesis is focused on this kind of applications of Gaussian processes where supervised tasks such as multi-class and multi-label classification are considered.

We start by discussing the main mathematical tools required in order to successfully cope with the large scale of the datasets. Those include a variational inference framework, suitably tailored for Gaussian processes. Furthermore, in our attempt to alleviate the computational burden, we introduce a new parametrization for the variational distribution while a representation trick for reducing storage requirements for large input dimensions is also discussed.

A methodology is then presented which is based on this variational inference framework and a computationally efficient bound on the softmax function that allows the use of Gaussian processes for multi-class classification problems that involve arbitrarily large number of classes. A series of experiments test and compare the performance of this methodology with other methods.

Finally, we move to the more general multi-label classification task and we develop a method, also relied on the same variational inference framework, which can deal with datasets involving hundreds of thousands data points, input dimensions and labels. The effectiveness of our method is supported by experiments on several real-world multi-label datasets.

Impact Statement

The work presented in this thesis has a potential impact on both academic and industrial communities. In academia for example, our work revises the popular belief that Gaussian process models cannot be used for inference over large-scale datasets in the Big Data era. We show that the opposite is not just achievable but we can attain very promising performance by harnessing the power and flexibility of non-parametric models such as Gaussian processes. The domain of applications of our work involves supervised learning tasks, such as multi-class and multi-label classification which play a dominant role in many real-world scenarios.

Both of those supervised tasks find a large number of applications that are not restricted to traditional machine learning problems, such as text or image classification, speech recognition, and they can be useful to other scientific fields such as Biology (DNA expression microarray, DNA Sequence Classification) and in general any other scientific discipline that tries to classify any kind of objects into a finite, potentially large, set of classes. More importantly, the fact that our developed methodologies are able to scale well over the number of classes makes it appealing in a time that large amounts of data need to be analysed. To make the notion of the size of the datasets more clear, we note that one of the datasets used in this thesis to test both the performance and scalability of our methods consists of more than 190,000 data points, associated with a set of more than 200,000 classes while each of those data points are described by almost 782,000 different features. This clearly indicates the suitability of our proposed methodologies for a regime where extreme dimensions dominate. It should be also emphasised that all the proposed methodologies are accompanied with contemporary software which is employed by a large number of people in the machine learning community, and therefore rendering their deployment much easier.

Regarding applications of our work in industry, a large number of multi-national companies use multi-label classification algorithms as part of their recommendation systems that involve a huge number of labels. The most known examples of those companies are Netflix, Amazon, Youtube, and Spotify for product recommendation. Other large companies that focus on social media platforms such as Facebook or Twitter also make extensive use of these kind of algorithms. Hence, our propounded methodology could be applied to all those real-world scenarios since it is suitably designed for applications of that nature.

Acknowledgements

First and foremost, I would like to thank my supervisor Petros Dellaportas that most importantly gave me the opportunity to be part of an amazing academic environment as it is in UCL. The optimism of Petros and the fact he is always there to inspire and support your research, played a crucial role to fully enjoy the PhD experience over these four years. It was a pleasure working with him through all these years and I hope this collaboration will continue in the future.

I am also deeply grateful to Michalis Titsias who, acting both as collaborator and unofficial second supervisor, contributed greatly with his ideas to the fulfilment of this thesis. Collaborating with Michalis, who I consider a world expert in his field, has profoundly influenced my academic development all these years.

I wish also to thank all the people during the one year I spent at the Alan Turing Institute as an intern and an enrichment student. It was a valuable experience being part of such an intellectual environment and a very enjoyable one at the same time, due to all the good friends that I made through that time there. The most representative example of such friends is Seb, Abhinav, and Andreas. Cheers guys.

I would like to thank the Department of Statistical Science at UCL for its wonderful academic environment and particularly, my PhD mates Andrea, Long, and Alkeos for all the nice memories during the time I spent there.

I am also grateful to Prof Stephen Roberts and Dr Jinghao Xue for examining and improving with their insightful comments this thesis. Any remaining inaccuracies are of course because of me.

On a more personal note, I would like to thank all of my good friends back in Greece for their friendship over the years. Many thanks also to Lina, not only for her amazing typo-spotting skills that improved this thesis but mainly for making this PhD journey so special for me.

Last but not least, I want to express my greatest gratitude to the limitless love and support of my family. Not a single word would exist in this thesis without them.

Probability is the most important concept in modern science, especially as nobody has the slightest notion of what it means.
— Bertrand Russell, 1929

Contents

Notation **i**

List of Figures **v**

List of Tables **x**

1 Introduction **2**
- 1.1 Gaussian processes as an inference model 2
  - 1.1.1 An overview 2
  - 1.1.2 Gaussian processes for regression and multi-class classification 8
  - 1.1.3 The main challenges of using GP priors and how to address them 12
- 1.2 Extreme multi-label classification 14
  - 1.2.1 Performance evaluation metric 16
  - 1.2.2 The chosen XML methods 17
- 1.3 Thesis overview 18

2 Variational inference for sparse Gaussian processes **20**
- 2.1 Introduction 20
- 2.2 The evidence lower bound and tractable variational families $\mathcal{Q}$ . . . 22
  - 2.2.1 The evidence lower bound 22
  - 2.2.2 Tractable variational families $\mathcal{Q}$ 24
- 2.3 Sparse Gaussian processes in a variational inference framework . . . 25
  - 2.3.1 General sparse Gaussian process methods 25
  - 2.3.2 The variational inference framework for sparse Gaussian processes 27
- 2.4 Two potential parametrizations for $\hat{q}(\mathbf{u})$ 29
- 2.5 Subspace inducing inputs 32

3 One-versus-Each Multi-class Classification using Gaussian Processes **35**
- 3.1 Introduction 35
- 3.2 The One-vs-Each lower bound on the softmax 35

3.3 From Parametric to non-parametric score functions 37
3.4 Scalable variational inference over class labels 38
3.4.1 Partial class updates in a point estimation setting 41
3.5 Scalable variational inference over data points 42
3.6 Another surrogate for softmax 45
3.7 A collection of multi-class classification methods 47
3.8 Experiments 49
3.8.1 Datasets 49
3.8.2 Synthetic data 49
3.8.3 Multi-class Classification using MNIST 51
3.9 Conclusion 56

4 Extreme Multi-label Inference using Gaussian Processes 66
4.1 Introduction 66
4.2 From multi-output GP regression to multi-label classification 66
4.3 The Multi-Label GP Factor Model 67
4.4 Scalable Variational Inference 69
4.4.1 The augmented model 70
4.4.2 Derivation of the lower bound 72
4.4.3 Computing the lower bound 74
4.4.4 Scalable Training using Stochastic Optimization 76
4.4.5 Prediction 79
4.4.6 Optimality of the PP 80
4.5 Experiments 82
4.5.1 Datasets 83
4.5.2 Methods 85
4.5.3 Experimental set-ups & Results 86
4.6 Conclusion 130

5 Conclusion 133
5.1 Contributions 133
5.2 Future work 134

Bibliography 136

A Useful formulas for the multivariate Gaussian distribution 151
A.1 Marginal and conditional Gaussian distribution 151
A.2 KL-Divergence between Normal distributions 152

B Matrix-related identities 153
B.1 Matrix calculus identities 153
B.2 The Woodbury matrix identity 153

C Code Snippets **154**
C.1 Autograd-based Python Code Snippet 154
C.2 Tensorflow-based Python Code Snippet 157

Notation and acronyms

We present here the main notation and we describe a number of acronyms used over the course of this thesis.

Symbol	*Explanation*
$\triangleq$	an equality that acts as a definition
$\sim$	distributed according to; e.g. $x \sim \mathcal{N}(\mu, \sigma^2)$
$\nabla_{\mathbf{f}}$	partial derivatives (w.r.t. $\mathbf{f}$)
$\mathbf{0}_k$	a k-dimensional vector of all 0's
$\mathbf{1}_k$	a k-dimensional vector of all 1's
$\mathbf{I}(\cdot)$	the indicator function that gives 1 if the condition in the parentheses is satisfied otherwise 0
$\mathbb{E}$ or $\mathbb{E}_q[g(x)]$	expectation; expectation of $g(x)$ when the probability density function ofthe distribution of x is given by $q(x)$
f_V	a vector of GP function values evaluated at all points of the finite set V
$\mathcal{X}$	the index set of a function and the domain of a Gaussian process
$\mathbf{f}$	vector of GP latent function values, $\mathbf{f} = (f(\mathbf{x_1}), ..., f(\mathbf{x}_N))^\top$
$\mathcal{GP}$	Gaussian Process: $f \sim \mathcal{GP}\left(m(\mathbf{x}), \mathbf{k}(\mathbf{x}, \mathbf{x}')\right)$, the function f is distributed as a Gaussian Process with mean function $m(\mathbf{x})$ and covariance function $\mathbf{k}(\mathbf{x}, \mathbf{x}')$
K	number of classes (or labels) in a multi-class (or multi-label) classification problem
N	number of training instances
N and N^*	number of training and test cases respectively
D	dimension of input space $\mathcal{X}$
$\mathcal{R}$	dimensionality of the subspace used for the inducing inputs
M	number of inducing points

$\mathbf{k}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$	the kernel (or covariance) function evaluated at $\mathbf{x}^{(i)}$ and $\mathbf{x}^{(j)}$
K_X	$N \times N$ matrix $\mathbf{k}(X, X)$, the covariance between training points
K_{XZ}	$N \times M$ matrix $\mathbf{k}(X, Z)$, the cross-covariance matrix between training points and inducing inputs
K_{ZX}	the transpose of K_{XZ}
K_Z	$M \times M$ matrix $\mathbf{k}(Z, Z)$, the covariance between inducing inputs
$\log(t)$	natural logarithm (base e)
$\sigma(t)$	logistic function, $\sigma(t) = 1/(1 + e^{-t})$
$m(\mathbf{x})$	the mean function of a Gaussian process
$\mathbf{0}$	a zero mean function; $m(\mathbf{x}) = 0$, $\forall \mathbf{x} \in \mathcal{X}$
$\mathcal{N}(\boldsymbol{\mu}, \Sigma)$ or $\mathcal{N}(\mathbf{x}\|\boldsymbol{\mu}, \Sigma)$	(the variable $\mathbf{x}$ has a) Gaussian (Normal) distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix Σ
$p(\mathbf{x}) = \mathcal{N}(\mathbf{x}\|\boldsymbol{\mu}, \Sigma)$	$p(x)$ is the probability density function of a Gaussian distribution with mean vector $\boldsymbol{\mu}$ and covariance matrix Σ
$\mathbb{N}$	the set of natural numbers
$\mathcal{O}$	big Oh; for functions f and g on $\mathbb{N}$, we write $f(n) = O(g(n))$ if the ratio $\frac{f(n)}{g(n)}$ remains bounded as $n \to \infty$
$y\|x$ and $p(y\|x)$	conditional random variable y given x and its probability(density)
$\phi(\mathbf{x})$	feature map of input $\mathbf{x}$ (or input set X)
$\Phi(z)$	cumulative unit Gaussian: $\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{z} e^{\frac{-t^2}{2}} dt$
$\mathcal{S}_k(f_1, \cdots, f_K)$	the Softmax function: $\mathcal{S}_k(f_1, \cdots, f_K) = \frac{e^{f_k}}{\sum_{\ell=1}^{K} e^{f_\ell}}$
$\mathbb{R}$	the set of real numbers
σ_f^2	variance of the signal
σ_n^2	noise variance
Φ	factor loadings matrix
$\boldsymbol{\theta}$	vector of hyperparameters (parameters of the covariance function)
$\mathcal{X}$	input space and also the index set for the stochastic process
X	$N \times D$ matrix of the training inputs $\{\mathbf{x}^{(i)}\}_{i=1}^{N}$: the design matrix
Y	$N \times K$ matrix of the training binary label vectors $\{\mathbf{y}^{(i)}\}_{i=1}^{N}$: the label matrix

$\mathbf{x}^{(i)}$	the i^{th} training input
$x_d^{(i)}$	the d^{th} coordinate of the i^{th} training input $\mathbf{x}^{(i)}$
Z	$M \times D$ matrix of the inducing inputs
$\mathbf{u}$	the GP function values evaluated at the inducing inputs Z; the inducing variables
$\mathbf{u}_p$	the inducing variables for the p^{th} Gaussian process
$\text{tr}(A)$	trace of a (square) matrix A
$\lvert A \rvert$	determinant of a matrix A
$\lVert \mathbf{x} \rVert$	Euclidean norm of vector $\mathbf{x}$, i.e. $\sqrt{\sum_i x_i^2}$
$\mathbf{x}^\top$	the transpose of vector $\mathbf{x}$
$\mathcal{U}$	a universe of stocks
FP	full parametrization of the the variational distribution using $\mathcal{O}(M^2)$ variational parameters (see also Section 2.4)
PP	parsimonious parametrization of the the variational distribution using $\mathcal{O}(M)$ variational parameters (see also Section 2.4)
OvsE-p	the One-versus-Each method described in Section 3.5 where each variational densities q_k is parametrized by the parsimonious scheme we introduced in Section 2.4 (see also Section 3.7)
OvsE-f	same method as above, however, each variational density q_k is parametrized by $\mathcal{O}(M^2)$ parameters (see also Section 3.7)
OvsE-p-sh	the One-versus-Each method with PP of the variational distributions, however the set of inducing inputs Z is shared across all classes while the hyperparameters set is different for each of them (see also Section 3.7)
OvsE-f-sh	the One-versus-Each method is combined with the FP of the variational distributions now. Both the set of inducing inputs Z and hyperparameters are common for all classes (see also Section 3.7)
RM-p	a multi-class classification that uses the robustmax function accompanied with the PP. (see also Section 3.7)
RM-f	a multi-class classification that uses the robustmax function accompanied with the FP (see also Section 3.7)
ard	a multi-label classification method based on MLGP model that uses a ARD kernel where the full inducing inputs matrix Z is optimized with each row unit-normalized, a FP is

	employed and everything is implemented by Tensorflow (see also Section 4.5.2)
ARD-UN	a multi-label classification method based on MLGP model that uses a ARD kernel where the full inducing inputs matrix Z is optimized while each row is not unit-normalized, a FP is employed and everything is implemented by Tensorflow (see also Section 4.5.2)
ARD-S	a multi-label classification method based on MLGP model that uses a ARD kernel where the subspace inducing inputs matrix A is optimized but the basis matrix $\widetilde{X}$ is precomputed and kept fixed, the rows of the matrix product $A\widetilde{X}$ are unit-normalized,a FP is employed and everything is implemented by Tensorflow (see also Section 4.5.2)
ARD-S-B	a multi-label classification method based on MLGP model that uses a ARD kernel where both the subspace inducing inputs matrix A and the basis matrix $\widetilde{X}$ are optimized, the rows of the matrix product $A\widetilde{X}$ are unit-normalized,a FP is employed and everything is implemented by Tensorflow (see also Section 4.5.2)
ARD-S-AG	a multi-label classification method based on MLGP model that uses a ARD kernel where the subspace inducing inputs matrix A is optimized but the basis matrix $\widetilde{X}$ is precomputed and kept fixed, the rows of the matrix product $A\widetilde{X}$ are unit-normalized,a FP is employed and everything is implemented by Autograd (see also Section 4.5.2)
ARD-S-PP-AG	a multi-label classification method based on MLGP model that uses a ARD kernel where both the subspace inducing inputs matrix A and the basis matrix $\widetilde{X}$ are kept fixed, a PP is employed and everything is implemented by Autograd (see also Section 4.5.2)
ARD-S-FX-PP-AG	a multi-label classification method based on MLGP model that uses a ARD kernel where both the subspace inducing inputs matrix A and the basis matrix $\widetilde{X}$ are kept fixed, a PP is employed and everything is implemented by Autograd (see also Section 4.5.2)

List of Figures

1.1 Four sample functions generated by (a) an exponential kernel and (b) a isotropic SE kernel, with $w = \sigma_f = 1$ for both cases. The sample functions are generated by using 3000 equally-distanced points in the interval $[-5, 5]$. 7

1.2 An example of GP regression using an SE kernel as presented in Section 1.1.2. Panel (a) shows sample functions from the prior GP using initial hyperparameters $w = 1, \sigma_f^2 = 0.05$, $\sigma_n^2 = 10^{-8}$ while the log marginal likelihood value is $\log(p(\mathbf{y})) = -61.53$. Panel (b) shows the mean posterior function (black line) functions from the posterior GP using the optimized hyperparameters $w = 2.7, \sigma_f^2 = 0.26$ and $\sigma_n = 4.1 \times 10^{-10}$ from type-II maximum likelihood while the optimized value of $\log p(\mathbf{y})$ is -7.44. The blue line corresponds to the true latent function $f(x) = 0.1x^2 \sin(6x) + 0.1\,(x\cos(5x) + \exp(\cos(x)))$. The blue coloured areas indicate the 95% confidence intervals of each of the posterior mean values while the black dots are the ten data points used for training. Panel (c) shows sample functions (colourful lines) from the posterior GP using the optimized hyperparameters as in (b). The sample functions are generated by using 1000 equally-distanced points in the interval $[-3, 3]$. 11

3.1 The Synthetic dataset from Section 3.8.2. 50

3.2 Evolution of the error rate throughout epochs for the synthetic dataset in Section 3.8.2. The blue line corresponds to non-parametric score functions while the red one to linear score functions. 51

3.3 Error rates for the OvsE-p as a function of the number of inducing inputs M. The red vertical line indicates the error rate of using linear score functions, as described in Section 3.4.1 and Titsias (2016). 58

3.4 (a) Error rates per epoch comparison between OvsE-p and OvsE-f and (b) Evolution of the lower bound per iteration between OvsE-p and OvsE-f. Red color indicates use of OvsE-p, while blue color use of OvsE-f. 59

3.5 (a) Error rates per epoch comparison between OvsE-p-sh and OvsE-f-sh and (b) Evolution of the lower bound per iteration between OvsE-p and OvsE-f. Red color indicates use of OvsE-p-sh, while blue color use of OvsE-f-sh. 60
3.6 (a) Error rates per epoch comparison between RM-p and RM-f and (b) Evolution of the lower bound per iteration between RM-p and RM-f. Red color indicates use of RM-p, while blue color use of RM-f. 61
3.7 Visualization of the inducing inputs. Each row corresponds to different inducing point. Left: Initialized inducing input with a k-means center. Center: Optimized inducing point by using OvsE-p. Right: Optimized inducing point by using OvsE-f. 62
3.8 Visualization of the inducing inputs. Each row corresponds to different inducing point. Left: Initialized inducing input with a k-means center. Center: Optimized inducing point by using OvsE-p-sh. Right: Optimized inducing point by using OvsE-f-sh. 63
3.9 Visualization of the inducing inputs. Each row corresponds to different inducing point. Left: Initialized inducing input with a k-means center. Center: Optimized inducing point by using RM-p. Right: Optimized inducing point by using RM-f. 64

4.1 Performance comparison between various choices of kernels tested on Bibtex dataset. (a) shows the evolution of the lower bound per iteration, and (b) demonstrates how the P@1 metric varies across epochs for each kernel. Numerical results of (b) are demonstrated in Table 4.5. 88
4.2 Performance comparison between three different cases: (i) full inducing inputs optimization (red color line), (ii) subspace inducing inputs optimization and fixed basis (blue color line), (iii) optimization of both subspace inducing inputs and basis (black color line). All the cases are tested on Bibtex dataset using an ARD kernel. (a) shows the evolution of the lower bound per iteration, and (b) demonstrates how the P@1 metric varies across epochs for each kernel. 89
4.3 In those two plots, the performance of MLGPF model is compared with the **FP** (red color line) and **PP** (blue color line) using an ARD kernel. (a) shows the evolution of the lower bound, and (b) gives P@1 evolution across epochs for each parametrization. 91
4.4 Performance comparison of the MLGPF model between the **Tensorflow** (red color line) and **Autograd** implementation (blue color line). An ARD kernel combined by the PP are used while $M = 100$. (a) shows the evolution of the lower bound, and (b) gives the P@1 evolution across epochs for each Python package. 93

4.5 Performance comparison of the MLGPF model between **optimized** Z (red color line) and **fixed** one (blue color line) using an ARD kernel. (a) shows the evolution of the lower bound, and (b) gives the P@1 evolution across epochs for each parametrization. 94
4.6 Performance comparison of the MLGPF model between **normalized** Z (red color line) and **unnormalized** one (blue color line) using an ARD kernel. (a) shows the evolution of the lower bound, and (b) gives the P@1 evolution across epochs for each parametrization. 96
4.7 Performance of MLGPF model in terms of P@1 (Eq. (1.21)) as a function of epochs where an ARD method trained on the Bibtex dataset. (a) Assumes $P = 80$ and (b) $P = 159$ where in both cases $M = 10$, 50, 100, 200, 400, 800. Error bars of one standard deviation are also provided. 97
4.8 Lower bound values of MLGPF model as a function of iterations where an ARD method trained on the Bibtex dataset. (a) Assumes $P = 80$ and (b) $P = 159$ where in both cases $M = 10$, 50, 100, 200, 400, 800. The shaded area around curves represents 68% confidence intervals. 98
4.9 Performance of MLGPF model in terms of P@1 as a function of epochs where an ARD method trained on the Bibtex dataset. (a) Assumes $M = 50$ and (b) $M = 100$ where in both cases $P = 10$, 20, 40, 80, 159, 200, 500. Error bars of one standard deviation are also provided. 99
4.10 Lower bound values of MLGPF model as a function of iterations where an ARD method trained on the Bibtex dataset. (a) Assumes $M = 50$ and (b) $M = 100$ where in both cases $P = 10$, 20, 40, 80, 159, 200, 500. The shaded area around curves represents 68% confidence intervals. 100
4.11 Bar plots comparing P@1 values using several pair of values (M, P) where an ARD method trained on the Bibtex dataset and run for 50 epochs. (a) shared $P = 80$, 159, and (b) shared $M = 50$, 100. Error bars of one standard deviation are also provided. 101
4.12 Bar plots comparing lower bound values using several pair of values (M, P) where an ARD method trained on the Bibtex dataset and run for 50 epochs. (a) shared $P = 80$, 159, and (b) shared $M = 50$, 100. The lower the bar plots are the better performance is achieved by ARD. Error bars of one standard deviation are also provided. . . 102

4.13 Lower bounds of (a) Bibtex and (b) Delicious. The solid lines correspond to the methods that optimized the inducing points while the dashed ones correspond to those that kept them fixed. Blue color suggests the use of linear kernel while the red one the use of SE kernel. Performance results of those methods can be found in Table 4.13. 107
4.14 Lower bounds of (a) Mediamill, (b) EUR-Lex, and (c) RCV1. The solid lines correspond to the methods that optimized the inducing points while the dashed ones correspond to those that kept them fixed. Blue color suggests the use of linear kernel while the red one the use of SE kernel. Performance results of those methods can be found in Table 4.13. 108
4.15 Lower bounds of (a) Bibtex, (b) Delicious, and (c) Mediamill, (d) EUR-Lex, (e) RCV1, and (f) AmazonCat. The solid lines correspond to the methods that optimized the inducing points while the dashed ones correspond to those that optimized the subspace inducing points. Blue colour represents the use of linear kernel while the red one the use of squared exponential kernel. 115
4.16 Lower bounds of (a) EUR-Lex, (b) RCV1, and (c) AmazonCat. The solid lines correspond to the methods that optimized the inducing points while the dashed ones correspond to those that optimized the subspace inducing points. Blue color suggests the use of linear kernel while the red one the use of squared exponential kernel. . . . 116
4.17 Lower bounds of (a) Bibtex, (b) Delicious, and (c) Mediamill. The solid lines correspond to the methods that kept fixed the inducing points while the dashed ones correspond to those that kept fixed the subspace inducing points. Blue color suggests the use of linear kernel while the red one the use of squared exponential kernel. . . . 117
4.18 Lower bounds of (a) EUR-Lex and (b) AmazonCat. The solid lines correspond to the methods that kept fixed the inducing points while the dashed ones correspond to those that kept fixed the subspace inducing points. Blue color suggests the use of linear kernel while the red one the use of squared exponential kernel. 118
4.19 Evolution of the lower bound for (a) Bibtex, (b) Delicious, and (c) Mediamill. The blue line corresponds to maximization of the lower bound using fixed inducing inputs Z, while the red one to optimized Z. The ARD kernel is used for each dataset while all the other parameters are set as described in Table 4.20. 123
4.20 Evolution of the lower bound for (a) EUR-Lex and (b) Wiki10. The blue line corresponds to maximization of the lower bound using fixed inducing inputs Z, while the red one to optimized Z. The ARD kernel is used for each dataset while all the other parameters are set as described in Table 4.20. 124

4.21 Values of the the optimized inverse lengthscales w_d , $d = 1, \cdots , D$ across all input dimensions, for (a) Bibtex and (b) Delicious, (c) Eurlex and (d) Wiki10. The optimized values obtained by the ARD method where its performance is presented in Tables 4.23. 125
4.22 Evolution of the (mean) lower bound for Mediamill (first row) and EUR-Lex (second row). Right column corresponds to LINEAR-ℓ method while the left one to ARD. All the other parameters are set as described in Table 4.20. Shaded area depicts 68% confidence intervals. 127
4.23 Evolution of the (mean) lower bound for Bibtex (first row) and Delicious (second row). Right column corresponds to LINEAR-ℓ method while the left one to ARD. All the other parameters are set as described in Table 4.20. Shaded area depicts 68% confidence intervals. 128
4.24 (a) Error rates per epoch comparison between the methods SE (blue line) and SE-PP (red line) of the MLGPF model, using $M = 100$, and RM-F (black line). On the other hand, (b) depicts the evolution of the lower bound per iteration between the two parametrizations of the MLGPF model and RM-F. Colour line description is the same as (a). $N = 60000$ and $|B| = 1000$, thus each epoch corresponds to 60 iterations. 129
4.25 Visualization of the inducing inputs based on the results of Fig. 4.24. Each row corresponds to a different inducing input. Left: Initialized inducing input with a k-means center. Center: Optimized inducing point by using the MLGPF model with PP. Right: Optimized inducing point by using the MLGPF model with FP. . . 131

List of Tables

3.1 Time complexity and storage requirements for each of the described methods in Section 3.7. M is the number of inducing inputs, K the number of classes, and $|\mathcal{B}|$ the mini-batch size. 48

3.2 Error rate (%) (given by (3.42)) and average negative log predictive density (nlpd) (given by (3.43)) for six different methods using $M = 100$ inducing inputs. Training time (seconds/epoch) and memory footprint in gigabytes (GB) are also reported for the same six methods, however $M = 1000$ used for those cases. The first four methods (left to right) methods are all relied on the methodology of Section 3.5 and their differences lie in the parametrization choice, as discussed in 2.4, and the shared or separate set of inducing inputs used. The last two methods, RM-P and RM-F, are based on the robustmax likelihood of Section 3.6 and the two different parametrizations where for both cases all classes share a common set of inducing inputs Z. Therefore we have method (i) OvsE-P where each class has its own set of inducing inputs Z_k and the parsimonious parametrization (PP) is used, (ii) OvsE-F where each class has its own set of inducing inputs Z_k and the full parametrization (FP) is used, (iii) OvsE-P-SH where all classes share a common set of inducing inputs Z and the PP is used, (iv) OvsE-F-SH where all classes share a common set of inducing inputs Z and the FP is used, (v) RM-P where the PP is used, and (vi) RM-F where the FP used. For all those methods a SE kernel is employed (with single lengthscale). 52

3.3 Values of the initial and optimized hyperparameters of the SE kernel for each method in Table 3.2. Those values correspond to the methods used $M = 100$ in Table 3.2. 54

3.4 References for the methods used in Table 3.5. SVI stands for Stochastic Variational Inference and EP for Expectation-Propagation. 55

3.5 A summary of several methods, both GP-based and others, with their corresponding error rates for the MNIST dataset. Some of the results can be found at Lecun's website (LeCun, 2019). 57

3.6 Performance comparison using different algorithms for optimization of RM-F and their corresponding optimal learning rate. 57

4.1 Time complexity and storage requirements for the two different parametrizations needed for optimizing (4.28). M is the number of inducing inputs, K the number of classes, and $|\mathcal{B}|$ the minibatch size. Negative subsampling and the effect of the input dimensionality are not considered here. 76
4.2 Time complexity and storage requirements for the case of optimizing (i) Full inducing inputs Z (Full), (ii) Subspace inducing inputs with fixed $\widetilde{X}$ (SII), and (iii) both Subspace inducing inputs and basis $\widetilde{X}$ (SII+Basis). M is the number of inducing inputs, K the number of classes, and $|\mathcal{B}|$ the minibatch size. Negative subsampling is not considered here. 77
4.3 Summary of the parameters to be optimized where different cases are considered, such as parametrization of the variational distributions or subspace inducing inputs use. Blanks indicate that the corresponding parameters are independent from the choice of the specific method while ✓and ✗ indicate presence and absence, respectively, of the given scheme. 79
4.4 Data sets statistics: N and N^* are the number of the training and test points respectively, D and K are the number of features and labels respectively, and $\overline{K}$ is the average number of positive labels in an instance. 82
4.5 Results of using the MLGPF model in conjunction with different kernels for Bibtex dataset. All the P@k metrics (Eq. (1.21)) are computed after training our model with $M = 100$, $P = K = 159$, and epochs=50. 87
4.6 Numerical results of Fig. 4.2 (b). The performance metric Precision@k is used (Eq. (1.21)). 90
4.7 Numerical results of Fig. 4.3 (b). The performance metric Precision@k is used (Eq. (1.21)). 90
4.8 Numerical results of Fig. 4.4 (b) and running times (seconds/epoch) for the two implementations where the number of inducing inputs this time is set to 500. The performance metric Precision@k is used (Eq. (1.21)). 92
4.9 Training times (in minutes) of the MLGPF model for the Bibtex dataset using several values for P and M, running for 50 epochs and deploying the ARD kernel. 95
4.10 Elapsed training time comparison between different MLGPF methods, applied to Bibtex dataset using $M = |\mathcal{B}| = 500$ and $P = K = 159$. 103
4.11 Number of parameters (in millions) to be optimized for each dataset using different parametrizations. 105

4.12 Parameter settings of the MLGPF model for each dataset based on the methods preseneted in Table 4.13. 106
4.13 Predictive Performance of the MLGPF model for the seven multi-label datasets. Those methods that have not reported results for a dataset are indicated with the "-" sign. All the experimental settings of each dataset can be found in Table 4.12. 109
4.14 Training times (in minutes per epoch) of the MLGPF model for all used datasets based on the settings of Table 4.12. 110
4.15 Performance scores when predicting the partially observed label vectors based on the MAP estimate of $q(\mathbf{u}^{(*)})$ and $q(\mathbf{u}^{(*)}|\mathbf{y}_o^{(*)})$. In all datasets, except Mediamill the LINEAR-PP-AG was used. For Mediamill, the SE-PP-AG was chosen. 110
4.16 Parameter settings of the MLGPF model for each dataset used in this chapter. The methods presented in Table 4.17 use those settings for their parameters. 111
4.17 Predictive Performance of the MLGPF model for the seven multi-label datasets. Those methods that have not reported results for a dataset are indicated with the "-" sign. All the experimental settings of each dataset can be found in Table 4.16 111
4.18 Predictive Performance of the MLGPF model for five multi-label datasets. Those methods that have not reported results for a dataset are indicated with the "-" sign. 113
4.19 Computational time (in minutes per epoch) of the MLGPF model for six multi-label datasets. 114
4.20 Parameter settings of the MLGPF model for each dataset based on the methods preseneted in Table 4.23. 114
4.21 Sensitivity analysis of the jitter term used for the FP for each small-scale dataset. We run the ARD method using the same seed for our number generator in order to remove randomness from the algorithm and allow comparisons for a given dataset. Five different jitter values are tested, $\{10^i\}_{i=1}^5$, where the first 4 rows correspond to jitter=10^{-1}, the next 5th to 8th row corresponds to jitter=10^{-2} and so on. All the P@k metrics (Eq. (1.21)) and $\mathcal{F}$ are computed after training our model. For all datasets we use $M = 100$, $P = 80$, and epochs=50. 119

4.22 Sensitivity analysis of the jitter term used for the FP for each small-scale dataset. We run the ARD method using the same seed for our number generator in order to remove randomness from the algorithm and allow comparisons for a given dataset. Five different jitter values are tested, $\{10^{i+5}\}_{i=1}^{5}$, where the first 4 rows correspond to jitter=10^{-6}, the next 5th to 8th row corresponds to jitter=10^{-7} and so on. All the P@k metrics (Eq. (1.21)) and $\mathcal{F}$ are computed after training our model. For all datasets we use $M = 100$, $P = 80$, and epochs=50. 120
4.23 Predictive Performance of the MLGPF model for eight multi-label (test) datasets. Those methods that have not been applied on a dataset are indicated with the "-" sign. Error bars (one standard deviation) are provided for the small-scale datasets. The experimental settings of each method can be found in Table 4.20. 121
4.24 Computational time (in minutes per epoch) of the MLGPF model for six multi-label (training) datasets corresponding to the methods. Those methods that have not been applied on a dataset are indicated with the "-" sign. 121
4.25 Performance comparison between the MLGPF model using ARD and other state-of-the-art methods. The top-2 P@k ($k = 1, 3, 5$) for each dataset are in bold. "-" sign indicates absence of reported results for a dataset. 126
4.26 Error rates and computational times in seconds per epoch for the MLGPF model using different number of GPs P in tandem with RM-F method's error given by Table 3.2. For all cases an SE kernel and FP are used while both number of inducing inputs and epochs are set to 100. Nevertheless, computational times are calculated by using $M = 1000$. 130

Chapter 1

Introduction

This chapter aims to present a brief introduction to Gaussian processes as an inference model in conjunction with a description of the extreme multi-label classification problem. More specifically, in Section 1.1 we define the mathematical tool that constitutes the bedrock of all the theoretical results presented in this thesis, namely the Gaussian process, in tandem with a discussion of some of their interesting properties. Section 1.1.2 offers a brief overview of the methods used to solve multi-class classification tasks based on Gaussian process models, highlighting at the same time the most crucial obstacles. Continuing, Section 1.2.1 is devoted to the description of the state-of-the-art algorithms used, in order to deal with multi-label classification problems that are notoriously known for their extremely large scale which is also used as a baseline comparison for our proposed method in Chapter 4. Finally, we provide an overview of the chapters presented in this thesis.

1.1 Gaussian processes as an inference model

1.1.1 An overview

Gaussian processes (GPs) play a prominent role in the field of machine learning (Rasmussen and Williams, 2006) while their first appearance dates back almost a century ago in the seminal work of Kolmogoroff (1941) and Wiener (1949) for time series prediction. They have been extensively used in myriad of problems, spanning the full spectrum of the field, such as supervised learning (Rasmussen and Williams, 2006), unsupervised learning (Lawrence, 2004), and reinforcement learning (Engel et al., 2005) where details about the three fundamental classes of machine learning can be found in several textbooks, such as MacKay (2003) or Murphy (2012). This thesis is concerned with the use of GPs on supervised tasks, which are those that involve learning a mapping from inputs to outputs given a dataset of input-output pairs, and then use that mapping to predict outputs from new inputs. More specifically, our primary interest lies in classification problems where the outputs are categorical in contrast to regression, which are continuous.

In general, a GP is a special instance of the more general class of stochastic processes (Papoulis and Pillai, 2002) and as such they can be considered as a distribution over functions. This is a natural extension of the multivariate Gaussian distribution by considering infinite-dimensional vectors roughly speaking. Before we give the formal definition of the GP we establish some minimal notation. We start by considering an arbitrary function f that maps an index set $\mathcal{X}$ to the set of real numbers $\mathbb{R}$, i.e. $f : \mathcal{X} \to \mathbb{R}$; however, this is not restrictive and it can be extended to $\mathbb{R}^n$. For the index set, no limitation is imposed and it can be finite, countable or uncountable set. Moreover, $f \in \mathbb{R}^{\mathcal{X}}$ denotes the set of all values of f evaluated at all points of $\mathcal{X}$, meaning that f can be conceived as a vector where its dimensionality is defined by the cardinality of $\mathcal{X}$ which can be either finite or infinite. Nonetheless, the case of $\mathcal{X} \subseteq \mathbb{R}^D$ will be of interest for the rest of this thesis where D is the number of dimensions of the input space.

We say that f is distributed according to a Gaussian process with mean function $m : \mathcal{X} \to \mathbb{R}$ and covariance or kernel function $\mathbf{k} : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, if and only if for any finite set $X \subset \mathcal{X}$ of size $|X| = N$, we have

$$f_X \sim \mathcal{N}(\mathbf{m}_X, K_X), \tag{1.1}$$

where the N-dimensional random vector f_X comprised by the function values evaluated at all points X, is normally distributed. The mean vector of this Gaussian distribution is $\mathbf{m}_X \in \mathbb{R}^N$ which consists of the values of $m(\cdot)$ evaluated at all points in X. The covariance matrix $K_X \in \mathbb{R}^{N\times N}$ is similarly calculated via the kernel function $\mathbf{k}(\cdot, \cdot)$ where the element at the i^{th} row and j^{th} column of K_X is computed by $\mathbf{k}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \triangleq Cov(f(\mathbf{x}^{(i)}), f(\mathbf{x}^{(j)}))$ and written as $[K_X]_{ij}$. The above definition is succinctly denoted as

$$f \sim \mathcal{GP}(m(\mathbf{x}), \mathbf{k}(\mathbf{x}, \mathbf{x}')). \tag{1.2}$$

The above definition implies that a GP can be fully specified by its mean and kernel function. We should also note here that assuming another set $U \subset X \subset \mathcal{X}$, then the moments of the normally distributed random vector f^U, apart from referring to the mean and kernel function for their computation, can be also retrieved by just marginalizing out the random vector $f^{X\setminus U}$, where $X \setminus U$ is the usual difference of set U and X. This property is generally known as *consistency* of the marginal distributions of the stochastic process and it is linked with the Kolmogorov's extension theorem (Billingsley, 2008) for stochastic processes.

Additionally, the kernel function plays a crucial role in the nature of the "sampled"[1] functions by the GP since it controls the smoothness of the random functions f. Moreover, $\mathbf{k}(\cdot,\cdot)$ has to be such that it always produces symmetric and positive-definite matrices K_X which can be only achieved in the case of a symmetric positive-definite kernel $\mathbf{k}(\mathbf{x},\mathbf{x}')$, i.e.

1. $\mathbf{k}(\mathbf{x},\mathbf{x}') = \mathbf{k}(\mathbf{x}',\mathbf{x})$, $\mathbf{x},\mathbf{x}' \in \mathcal{X}$
2. $\int f(\mathbf{x})\mathbf{k}(\mathbf{x},\mathbf{x}')f(\mathbf{x}')d\mu(\mathbf{x})d\mu(\mathbf{x}') > 0$, $\forall f \in L_2(\mathcal{X},\mu)$

where μ denotes the Lebesgue measure and $L_2(\mathcal{X},\mu)$ is the function-space of all square-integrable real functions under the measure μ where their domain is $\mathcal{X}$. The kernel function is usually parametrized by a vector of parameters $\boldsymbol{\theta}$, called hyperparameters, and they control the kernel's properties as we discuss later.

There is a large variety of kernel functions with different properties and they can be loosely classified into two main categories, *stationary* and *non-stationary* kernel functions. Stationary kernels can be written as a function of $\mathbf{x}-\mathbf{x}'$, which implies invariance to translations in the input space. Probably the most representative examples of such kernels are the *squared exponential* class of kernel functions. Most of the material presented here about kernels is based on Chapter 4 of Rasmussen and Williams (2006), thus the reader is referred there for a more detailed discussion about other stationary kernels that are closely related to squared exponential class, such as the Matérn class or the rational quadratic kernel function (Section 4.2.1 in Rasmussen and Williams (2006)).

Stationary kernels:

A common attribute of the stationary covariance functions is that they can be all written as functions of a (scaled) distance measure r between inputs $\mathbf{x}$ and $\mathbf{x}'$, where

$$r^2(\mathbf{x},\mathbf{x}') \triangleq r^2 = (\mathbf{x}-\mathbf{x}')^\top W(\mathbf{x}-\mathbf{x}'), \tag{1.3}$$

and W is a symmetric matrix. Maybe the most known class of stationary kernels is the squared exponential class that includes kernels of the following form

$$\mathbf{k}(\mathbf{x},\mathbf{x}') \triangleq \mathbf{k}(r) = \sigma_f^2 \exp\left(-\frac{r^2}{2}\right), \tag{1.4}$$

where $\sigma_f > 0$, and $\boldsymbol{\theta} = (\sigma_f, \{W\})^\top$ is a vector containing all the hyperparameters. In this thesis, we consider two choices for the matrix W,

[1]Apparently we cannot sample a vector of infinite dimensions but it is enough to sample from a finite subset of $\mathcal{X}$ for investigating the properties of the stochastic process as the definition suggests.

$$W_1 = wI, \quad W_2 = \mathrm{diag}(\mathbf{w}), \tag{1.5}$$

where $\mathbf{w}$ is D-dimensional vector with positive values, $w > 0$, and the operator $\mathrm{diag}(\mathbf{u})$ returns a $D \times D$ diagonal matrix with values $\mathbf{u}$. The distance measure W_1 gives rise to the common Euclidean distance and the isotropic[2] *squared exponential (SE)* kernel where the value of w, also known as inverse characteristic lengthscale, explains, loosely speaking, how far we need to move (along any axis) in input space such that the function values become uncorrelated, thus controlling the smoothness of the sampled functions. However, using W_2 as distance measure, then we deploy different (inverse) lengthscales $w_1, \cdots, w_D$ for each input dimension, leading to automatic relevance determination (ARD) (Neal, 1995), since a value of w_d close to zero indicates that the covariance is almost independent of the input for the d^{th} input dimension. Therefore, we call this (anisotropic) kernel ARD. Notice that both of those distance measures assume an independence between input dimensions. This can be remedied by introducing a full covariance matrix W which is able to capture correlation among input dimensions, however, in that case we would need to store and learn extra $\frac{D(D+1)}{2}$ hyperparameters which could be computationally prohibitive for large dimensions; this is the case for the input dimensionality of all datasets used in this thesis. The main characteristic of this class of kernels is that its kernel functions are all infinitely mean-square differentiable[3] rendering them very smooth. There is also a direct connection of the smoothness of a kernel with the rate of decay of its eigenvalues based on Mercer's kernel decomposition theorem (Mercer, 1909), where more details can be found in Section 4.3 in Rasmussen and Williams (2006). The main motivation of deploying the SE class of kernels in this thesis is mostly their wide-spread applicability within the Gaussian process community which provides us comparable results despite the fact that its strong smoothness assumptions could be considered unnatural in real-world problems.

Another instance of a well-known class of stationary kernels is the Matérn class which is parametrized by a positive scalar ν that controls the smoothness of the generated functions. These functions are k times differentiable (in the mean-square sense) if and only if $\nu > k$. A few examples of the Matérn family for half-integer values of ν (the kernels are reduced to simpler formulas in that cases) are given below

[2]Meaning that the kernel is a function of the euclidean norm of τ, and thus invariant to both translations and rotations.

[3]See Papoulis and Pillai (2002) for a formal definition of mean-square differentiability.

$$\mathbf{k}_{\nu=1/2}(r) = \sigma_f^2 \exp\left(-r\right) \tag{1.6}$$

$$\mathbf{k}_{\nu=3/2}(r) = \sigma_f^2(1+\sqrt{3}r)\exp\left(-\sqrt{3}r\right) \tag{1.7}$$

$$\mathbf{k}_{\nu=5/2}(r) = \sigma_f^2(1+\sqrt{5}r+\frac{1}{3}(\sqrt{5}r)^2)\exp\left(-\sqrt{5}r\right) \tag{1.8}$$

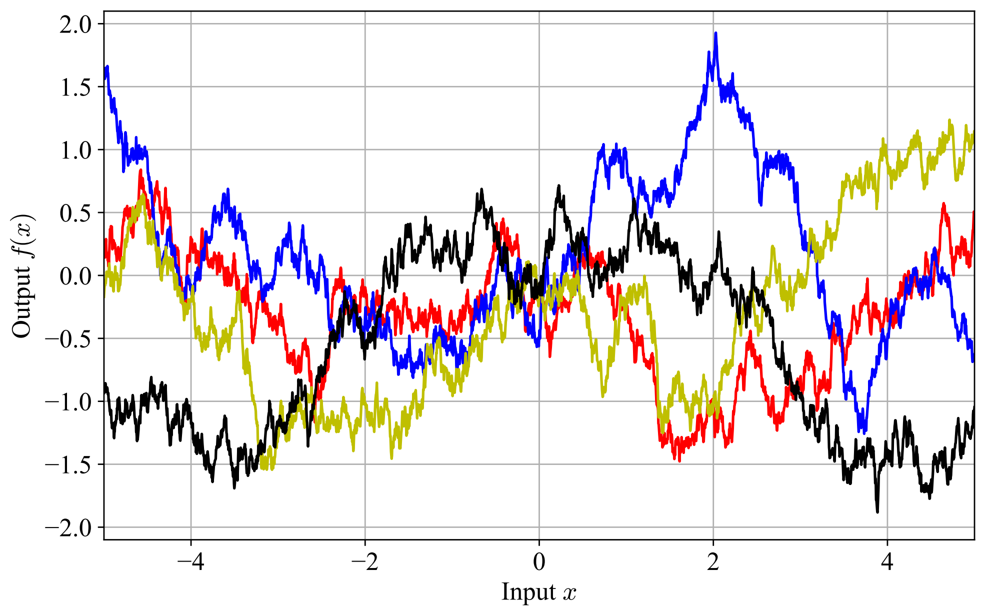
The kernel in Eq. (1.6) is also called the *exponential* covariance function and it is only employed in this chapter to demonstrate qualitative differences over other kernels as they are showed in Fig. 1.1; a GP prior on functions with the SE kernel (bottom panel) and with the exponential kernel which in that one dimensional case is an instance of the known Ornstein-Uhlenbeck process (Uhlenbeck and Ornstein, 1930) which is closely related with the Brownian motion (Brown, 1828; Einstein, 1905). In this figure we can also notice in practice how kernel's choice affects the smoothness of the sampled functions as the exponential kernel is only mean-square continuous (Papoulis and Pillai, 2002) but not mean-square differentiable leading to "rougher" generated functions as we also see in Fig. 1.1. We can see that by taking $\nu \to \inf$ the Matérn kernels converge to the squared exponential justifying in that way the infinitely-differentiability of the functions generated by the latter kernel.

Non-stationary kernels:

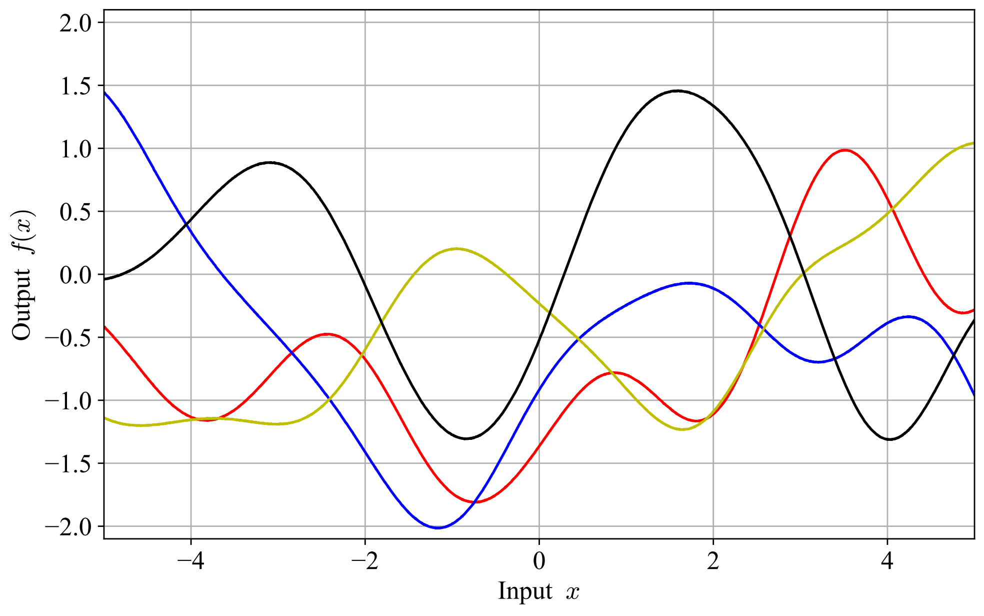
Regarding non-stationary kernels, one of the most known classes is the dot-product class where kernels are written as

$$\mathbf{k}(\mathbf{x},\mathbf{x}') = (\mathbf{x}^\top W \mathbf{x}')^p, \tag{1.9}$$

with p being a positive integer. In this thesis, we consider the case of the linear kernel ($p = 1$) with $W_1 = I$ and $W_2 = \mathrm{diag}(\mathbf{w})$. There are fundamental differences between the dot-product class and the SE class, which are not limited to stationarity. One of the most important ones is the degenerate kernels produced by the former class, i.e. kernels with finite non-zero eigenvalues. This also means that dot-product kernels can be written as finite expansion of basis functions in contrast to SE kernels that can be seen as infinite expansion of basis functions. For example, the isotropic SE kernel can be seen as a Bayesian linear regression model with an infinite number of basis functions (Rasmussen and Williams, 2006). Further, stationary kernels can be represented by the Fourier transform of a positive measure, which such a representation is guaranteed by the Bochner's theorem (Stein, 2012), providing useful information regarding the smoothness of the stochastic process based on its spectral density (Rasmussen and Williams, 2006).



(a)



(b)

Figure 1.1: Four sample functions generated by (a) an exponential kernel and (b) a isotropic SE kernel, with $w = \sigma_f = 1$ for both cases. The sample functions are generated by using 3000 equally-distanced points in the interval $[-5, 5]$.

1.1.2 Gaussian processes for regression and multi-class classification

Having defined what a GP is, we can now move to its description as an inference model for supervised tasks. Since stochastic processes define distributions over infinite objects and they obey the consistence rule of the marginals we discussed before, they can be used as priors for introducing a Bayesian framework for inference. This can be done by assuming the same finite set X with N elements from the previous section and a vector $\mathbf{y} \in \mathbb{R}^N$ where each entry is associated with an entry of X, i.e. the output data and then applying Bayes' theorem as follows,

$$p(f_X|\mathbf{y}) = \frac{p(\mathbf{y}|f_X)p(f_X)}{p(\mathbf{y})}. \tag{1.10}$$

Here, $p(f_X|\mathbf{y})$ is the posterior density, $p(\mathbf{y}|f_X)$ is the likelihood which is conditional to f_X, $p(f_X)$ is the prior which is given by (1.1) while the denominator is the marginal likelihood $p(\mathbf{y}) \triangleq \int p(\mathbf{y}|f_X)p(f_X)df_X$[4].

This framework is the core idea of Bayesian nonparametric models which extends the notion of traditional models that are parametrized by a finite parameter space, providing in that way additional flexibility since model complexity relies directly on the size of the dataset used. For a more detailed discussion about the subject the reader is referred to Hjort et al. (2010); Ghahramani (2013). Gaussian processes are an example of this kind of models, which can be used as discriminative models, aiming to find a mapping between input and output data. We can now introduce how inference is performed for the case of regression.

Regression

Consider a dataset with N D-dimensional input points $\{\mathbf{x}^{(i)}\}_{i=1}^N = X \in \mathbb{R}^{N \times D}$ and the corresponding continuous responses $\{y_i\}_{i=1}^N = \mathbf{y} \in \mathbb{R}^N$. In Gaussian process regression, we assume that the responses y_i have been produced by a latent function $f(\cdot)$ evaluated at inputs $\mathbf{x}^{(i)}$, i.e. $f(\mathbf{x}^{(i)}) \triangleq f_i$, and those latent function values have been corrupted by an additional independent Gaussian noise,

$$y_i \triangleq y(\mathbf{x}^{(i)}) = f_i + \epsilon \tag{1.11}$$

where $\epsilon \sim \mathcal{N}(0, \sigma_n^2)$. Using now a GP with a zero mean function[5] and a kernel

[4]The description we provide here for the stochastic processes as priors is not completely accurate, and one has to refer to measure-theoretic tools to define Bayes' theorem for infinite dimensional models. Schervish (2012) provides a good introduction to this matter.

[5]The choice of a zero mean function can be justified by the fact that the prior knowledge about f can be captured by using only the kernel function and its hyperparameters $\boldsymbol{\theta}$. It also leads to simpler linear algebra operations.

function $\mathbf{k}(\cdot,\cdot)$ as a prior over the function $f(\cdot)$, we can obtain the densities that comprise Eq. (1.10) as follows,

$$p(\mathbf{y}|f_X) = \prod_{i=1}^{N} \mathcal{N}(y_i|f_i, \sigma_n^2), \tag{1.12}$$

$$p(f_X) = \mathcal{N}(f_X|\mathbf{0}, K_X), \tag{1.13}$$

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{y}|\mathbf{0}, K_X + \sigma_n^2\mathbf{I}), \tag{1.14}$$

$$p(f_X|\mathbf{y}) = \mathcal{N}(f_X|K_X(K_X + \sigma_n^2\mathbf{I})^{-1}\mathbf{y}, K_X - K_X(K_X + \sigma_n^2\mathbf{I})^{-1}K_X). \tag{1.15}$$

where $\mathbf{I}$ is the identity matrix with the appropriate dimensionality. We see that all densities have a closed form solution which stems from the conjugacy relation between the prior (1.13) and the likelihood (1.12). The marginal likelihood (1.14) and the posterior (1.15) are derived by using marginalization properties of the normal distribution (see Appendix A.1). Moreover, by assuming an extra set of N^* points $U = \{\mathbf{x}_*^{(i)}\}_{i=1}^{N^*}$, we can impose a distribution over the latent values f_U by averaging out f_X using the posterior information of (1.15), i.e.

$$\begin{aligned} p(f_U|\mathbf{y}) &= \int p(f_U|f_X)p(f_X|\mathbf{y})df_X \\ &= \mathcal{N}(f_U|K_{UX}(K_X + \sigma_n^2\mathbf{I})^{-1}\mathbf{y}, K_{UX} - K_{UX}(K_X + \sigma_n^2\mathbf{I})^{-1}K_{XU}) \end{aligned} \tag{1.16}$$

where K_{UX} is the $N^* \times N$ cross-covariance matrix between U and X. In that way, we can generate samples or deploy the mean vector, as it is more common, of this predictive density in (1.16) and use it as predictions for the latent function values evaluated at U. Therefore, the whole procedure requires $\mathcal{O}(N^3)$ computational time due to the matrix inversion of $K_X + \sigma_n^2\mathbf{I}$.

It should be also mentioned that during the whole prediction process we indirectly imply that the kernel hyperparamters $\boldsymbol{\theta}$ are kept fixed. This means that we need a selection mechanism that allows us to choose their values in a principled way. The most common approach is based on a point estimation setting where the log marginal likelihood $p(\mathbf{y})$ is maximized as a function of $\boldsymbol{\theta}$ (Williams and Rasmussen, 1996) which is also known as "Emprical Bayes" or "Type-II maximum likelihood". There is also the more general approach that entails a Bayesian treatment over the hyperparameters where a prior has to be imposed over them (Williams and Rasmussen, 1996; Hensman et al., 2015a) leading to extra intractability and one has to refer to Markov Chain Monte Carlo (MCMC) based methods (Hastings, 1970; Gelfand and Smith, 1990).

Finally, we provide an example of Gaussian process regression where the unknown function form is $f(x) = 0.1x^2\sin(6x) + 0.1\,(x\cos(5x) + \exp(\cos(x)))$ and the observed values are contaminated by zero-mean Gaussian noise with variance equal to 0.001. We use $N = 10$ data points and $N^* = 1000$ test points while the used kernel here is an SE kernel. The kernel hyperparameters are optimized by employing type-II maximum likelihood. Fig. 1.2 shows how the smooth posterior samples, due to the use of the SE kernel, pass through the data points (black dots) after maximization of the log marginal likelihood while we can also notice that the posterior variances of the predictions close to the data points are considerable reduced comparing to more distant test points.

Multi-class classification

Multi-class classification in a GP framework is very similar with GP regression with one major difference, the output responses y_i are discrete numbers representing K distinct classes, which forces us to refer to non-Gaussian likelihoods to model the data while we also need to use K independent GPs instead of one as in regression. More formally, assume the design matrix X as in the regression case and the response vector $\mathbf{y}$ where each element $y_i \in \{1, \cdots, K\}$, $i = 1, \cdots, N$. The probability of y_i given the latent vector $\mathbf{f}^{(i)} = (f_1^{(i)}, \cdots, f_K^{(i)})$ is defined as

$$p(y_i|\mathbf{f}^{(i)}) = \frac{e^{f_{y_i}^{(i)}}}{\sum_{k=1}^{K} e^{f_k^{(i)}}} \triangleq \mathcal{S}_k(\mathbf{f}^{(i)}), \tag{1.17}$$

where $f_k^{(i)}$ is the function value of the k^{th} GP evaluated at $\mathbf{x}_i$ and $\mathcal{S}_k(\mathbf{f}^{(i)})$ is generally known as the softmax function. Hence, the likelihood is written as

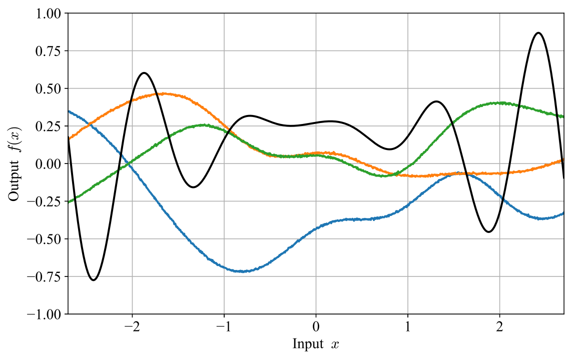
$$p(\mathbf{y}|\{\mathbf{f}_X^k\}_{k=1}^K) = \prod_{i=1}^{N} p(y_i|\mathbf{f}^{(i)}), \tag{1.18}$$

where $\mathbf{f}_X^k$ succinctly denotes the N-dimensional real vector consists of the k^{th} GP's function values evaluated at all training data points. We can notice the likelihood does not have a conjugacy relation with the NK-dimensional Gaussian prior $p(\{\mathbf{f}_X^k\}_{k=1}^K)$ used, leading therefore to an intractable posterior $p(\{\mathbf{f}_X^k\}_{k=1}^K|\mathbf{y})$. This intractability does not let us compute probabilistic predictions for a novel data point $\mathbf{x}^{(*)}$ that requires the evaluation of the following integral

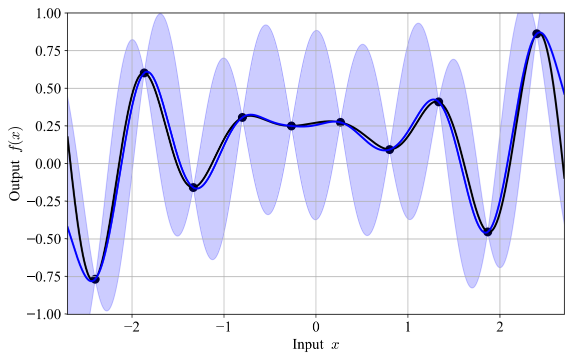
$$p(y_* = k|\mathbf{y}) = \int \mathcal{S}_k(\mathbf{f}^{(*)}) p(\mathbf{f}^{(*)}|\mathbf{y}) d\mathbf{f}^{(*)}, \tag{1.19}$$

since the computation of (1.19) involves the calculation of $p(\mathbf{f}^{(*)}|\mathbf{y})$ which is given by another integral,

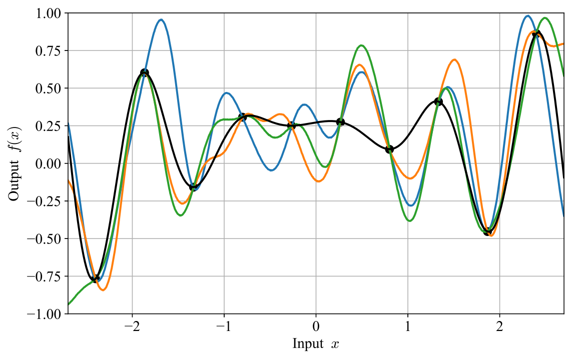
$$p(\mathbf{f}^{(*)}|\mathbf{y}) = \int p(\mathbf{f}^{(*)}|\{\mathbf{f}_X^k\}_{k=1}^K) p(\{\mathbf{f}_X^k\}_{k=1}^K|\mathbf{y}) d\mathbf{f}_X^1 \cdots d\mathbf{f}_X^K, \tag{1.20}$$



(a)



(b)



(c)

Figure 1.2: An example of GP regression using an SE kernel as presented in Section 1.1.2. Panel (a) shows sample functions from the prior GP using initial hyperparameters $w = 1, \sigma_f^2 = 0.05$, $\sigma_n^2 = 10^{-8}$ while the log marginal likelihood value is $\log(p(\mathbf{y})) = -61.53$. Panel (b) shows the mean posterior function (black line) functions from the posterior GP using the optimized hyperparameters $w = 2.7, \sigma_f^2 = 0.26$ and $\sigma_n = 4.1 \times 10^{-10}$ from type-II maximum likelihood while the optimized value of $\log p(\mathbf{y})$ is -7.44. The blue line corresponds to the true latent function $f(x) = 0.1x^2 \sin(6x) + 0.1\,(x\cos(5x) + \exp(\cos(x)))$. The blue coloured areas indicate the 95% confidence intervals of each of the posterior mean values while the black dots are the ten data points used for training. Panel (c) shows sample functions (colourful lines) from the posterior GP using the optimized hyperparameters as in (b). The sample functions are generated by using 1000 equally-distanced points in the interval $[-3, 3]$.

where it relies on the intractable posterior. For this reason, many methods have been proposed in the past to deal with this intractability by introducing some kind of approximation to the posterior. Such methods can be found in the work of Williams and Barber (1998) where Laplace's approximation (Azevedo-Filho and Shachter, 1994) is utilized to approximate the posterior of the latent values, Hernández-Lobato et al. (2011) where Expectation Propagation (Minka, 2001) is used this time and in the work of Hensman et al. (2015b,a) where a variational inference (see Chapter 2) approach of the problem is introduced. All those methods require $\mathcal{O}(KN^3)$ computational time when no sparsity assumptions are considered. Moreover, a more recent work (Milios et al., 2018) attempts to deal with the non-conjugacy relation between prior and likelihood by solving a GP regression where outputs are the logarithms of the discrete classes. Nonetheless, all of those methods have in common that they scale linearly with respect to the number of classes K. This can be prohibitive in cases where K is extremely large, rendering those methods impractical. In this thesis, we cope with this hindrance by presenting a methodology in Chapter 3 which uses a computationally efficient lower bound on the likelihood that allows us to reduce complexity and consider arbitrarily large K.

1.1.3 The main challenges of using GP priors and how to address them

Before we move to the next section, we would like to review and highlight the two main challenges that arise by employing Gaussian processes as priors. The first one that we also see in Section 1.1.2, has to do with the adverse $\mathcal{O}(N^3)$ scaling with the number of training data points that emerges from the need to compute densities like in (1.12), (1.13), (1.14), and (1.15) that require the calculation of the inverse and the determinant of the matrices K_X or $K_X + \sigma_n^2\mathbf{I}$. In general, we cannot avoid that cubic complexity $\mathcal{O}(N^3)$. For datasets that consist of up to 5000 data points this is not prohibitive, nevertheless, datasets in modern machine learning significantly surpass this number, and thus, we need to refer to approximation methods of the exact posterior. Nevertheless, there is a number of cases in the literature that circumvent the problem of solving large linear systems of the form $(K_X + \sigma_n^2\mathbf{I})\mathbf{x} = \mathbf{y}$ by making use of fast matrix-vector multiplication (Shen et al., 2006; Morariu et al., 2009), which solves the linear system by deploying conjugate gradients with $S \ll N$ iterations, giving as time complexity $\mathcal{O}(SN^2)$ while a more recent method (Cutajar et al., 2016) based on pre-conditioned conjugate gradients is used for approximating solutions of linear systems with applications to Gaussian process regression. Additionally, when the kernel matrix K_X has some structure, the matrix-vector multiplication methods can offer considerable speed-up and scalability. For example, methods exploiting the Kronecker structure of a tensor product kernel with form $\mathbf{k}(\mathbf{x}, \mathbf{x}') = \prod_{d=1}^{D} \mathbf{k}(x_d, x'_d)$, which allows us to write

$K_X = K_1 \otimes \cdots \otimes K_D$ and reduce the eigen-decomposition of K_X to $\mathcal{O}(DN^{1+1/D})$ for $D > 1$. The exploitation of the Toeplitz structure of a kernel matrix constructed by evenly spaced one dimensional points is also examined in Cunningham et al. (2008) which gives time complexity $\mathcal{O}N \log N$. A more general method, not limited to grid inputs as the previous Kronecker and Toeplitz methods, is the structured kernel interpolation in Wilson and Nickisch (2015) which considers M inducing inputs (see Chapter 2) and scales as $\mathcal{O}(N + DM^{1+1/D})$. We note here that kernels as the SE (or ARD) kernel can be written as a tensor product as the above kernel function, however we do not exploit this fact in our work here and we leave it for future work. Finally, Melkumyan and Ramos (2009) consider a sparse representation of K_X by imposing a distance threshold to $r(\mathbf{x}, \mathbf{x}')$, where the deduced kernel is known as compactly supported kernel, in order to build a sparse kernel matrix that can be used to train the GP model in time $\mathcal{O}(\alpha N^3)$ where $0 < \alpha < 1$ is a parameters that depends on the sparsity of the approximated kernel matrix.

Nevertheless, probably the most known way of dealing with the cubic complexity of the GP training in the last decade is via the so called *sparse approximations* which aim to build a generative probabilistic model by approximating either the prior (Quiñonero-Candela and Rasmussen, 2005a; Smola and Bartlett, 2001) or the posterior of the process (Titsias, 2009; Cao et al., 2013; Liu et al., 2018; Hensman et al., 2015b; Matthews, 2017; Hensman et al., 2015a) (see Chapter 2) for a more detailed discussion. We note that all these methods are heavily relied on the notion of the Nyström approximation (Gittens and Mahoney, 2016) which leads to $\mathcal{O}(NM^2)$ in general where an extra complexity reduction to $\mathcal{O}(M^3)$ is achieved by deploying data sub-sampling with variational inference techniques (Liu et al., 2018; Hensman et al., 2015b; Matthews, 2017; Hensman et al., 2015a). Alternatively, there are efficient approximations which are based on the sparsification of the spectrum of a stationary Gaussian process using M spectral frequencies. These methods (QuiÃ±onero-Candela et al., 2010; Tan et al., 2016; Gal and Turner, 2015; Hensman et al., 2017; Hoang et al., 2017) scale either as $\mathcal{O}(NM^2)$ or $\mathcal{O}(M^3)$ depending on the use of a sub-sampling technique of the training data points in the same manner as the sparse approximations methods that achieve same scalability.

The second challenge which is discussed in Section 1.1.2, concerns the absence of a closed form solution for the posterior density $p(f_X|\mathbf{y})$ when the likelihood is not Gaussian. Again, there is a significant amount of work on this matter such as in Flaxman et al. (2015); Sheth et al. (2015); Hensman et al. (2015a,b); Nickisch and Rasmussen (2008); Wenzel et al. (2019); Hernández-Lobato and Hernández-Lobato (2016a); Ruiz et al. (2018); Chai (2012); Fröhlich et al. (2013); Hensman et al. (2017) where in some cases the sparse approximations are utilized to achieve scalability. As we shall see in Chapter 2, those two difficulties can be addressed effectively by making use of the theory of variational inference.

1.2 Extreme multi-label classification

In this section we slightly divert from our main discussion about Gaussian process models and we proceed to the description of another subcategory of supervised learning, namely *Multi-label Learning* (MLL), which is the main topic of interest in Chapter 4.

MLL is concerned with the problem of finding a function g_{ML} that maps inputs $\mathbf{x} \in \mathcal{X} \subseteq \mathbb{R}^D$ to a subset of the label set $\{1, \cdots, K\}$ consisting of K classes, i.e. $g_{\text{ML}} : \mathcal{X} \rightarrow 2^{\{1,\cdots,K\}}$, where $2^{\{1,\cdots,K\}}$ is the usual power set of $\{1, \cdots, K\}$. Label associations for $\mathbf{x}$ can be also written as a K-dimensional binary vector $\mathbf{y} \in \{-1, 1\}^K$ where the k^{th} of $\mathbf{y}$, $y_k = 1$ if class k is annotated to $\mathbf{x}$. Clearly, MLL can be considered as a generalization of multi-class learning where $\mathbf{x}$ is labelled by only one class and the learning function is $g_{\text{MC}} : \mathcal{X} \rightarrow \{1, \cdots, K\}$. Many times the target function g_{ML}, given an input data $\mathbf{x}^{(*)}$, returns a K-dimensional probability vector $\mathbf{f}^{(*)}$ where its k^{th} entry $f_k^{(*)}$ indicates how probable the label class k is to be associated by this input data. A large number of methods have been developed over the last few years aiming to learn this function g_{ML} in a way that allows the exploitation of inter-label correlations (Tsoumakas and Zhang, 2009), given a dataset $\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^N = (X, Y)$ where $X \in \mathbb{R}^{N \times D}$ and $Y \in \{-1, 1\}^{N \times K}$, by using different approaches. Moreover, there is a consensus in the community (Gibaja and Ventura, 2015) that all the MML methods can be classified based on these approaches into two main families of methods, namely *problem transformation methods* and *algorithm adaptation methods* (Tsoumakas and Zhang, 2009). Methods in the former family transform the multi-label problem to a binary (Read et al., 2009; Boutell et al., 2004) or multi-class classification task (Tsoumakas and Vlahavas, 2007) while methods in the latter family extend common machine learning algorithms suitable for multi-class classification to be able to cope with multi-label data directly (Clare and King, 2001; Elisseeff and Weston, 2002; Ghamrawi and McCallum, 2005; Zaragoza et al., 2011; Zhang and Zhou, 2013). A good introduction on the subject of multi-label classification can be found in the work of Tsoumakas and Katakis (2007), Zhang and Zhou (2013), and Gibaja and Ventura (2015) where a review of the recent advances in the field is also provided.

Here, we focus our discussion to methods that attempt to deal with the MLL task under extreme settings where the number of training data points N, the feature dimensionality D and the label dimensionality K can be hundreds of thousands or even millions. This task is generally known in the machine learning community as the *Extreme Multi-label Learning* (XML) and there is a significant amount of work in the field (Hsu et al., 2009; Weston et al., 2011; Chen and Lin, 2012; Kapoor et al., 2012; Weston et al., 2013; Agrawal et al., 2013; Cisse et al., 2013; Lin et al., 2014; Prabhu and Varma, 2014a; Yu et al., 2014; Bhatia et al., 2015; Mineiro and Karampatziakis, 2015; Jain et al., 2016; Jasinska et al., 2016; Yen et al., 2016; Xu et al., 2016; Babbar and Schölkopf, 2017; Liu et al., 2017; Niculescu-Mizil and

Abbasnejad, 2017; Si et al., 2017; Tagami, 2017; Yen et al., 2017; Prabhu et al., 2018a,b; Siblini et al., 2018; Yen et al., 2018; Zhang et al., 2018). One of the main attributes of the datasets used for XML is that both input and label space present sparsity, otherwise it would be infeasible to store datasets such as the WikiLSHTC (see Section 4.5.1 of Chapter 4) which consists of $N = 1778351$ input data points with $D = 1617899$ features each. Moreover, most of those datasets are highly imbalanced, having just a few positive labels per data point (Section 4.5.1 while each label has just a few hundreds data points on average.

All the developed methods for XML can be broadly divided into three main categories depending on the approach they choose to solve the problem in a scalable manner. These are the *Embedding*-based methods , *Tree*-based methods, and *1-vs-All*-based methods. There is also an extra "pseudo-category" which contains "hybrid" methods that combines attributes of the main categories. A short description for each category follows.

Embedding-based

The main idea of those approaches is to reduce the label dimensionality by projecting the label vectors $\mathbf{y}^{(i)}$ onto a lower $\tilde{K}$-dimensional linear subspace where $\tilde{K} \ll K$, as $\mathbf{z}^{(i)} = U\mathbf{y}_i$ with $U \in \mathbb{R}^{\tilde{K}\times K}$ assuming that the label matrix Y is low-rank. After, a set of regressors $V \in \mathbb{R}^{\tilde{K}\times D}$ are learnt from the dataset such that $\mathbf{z}^{(i)} \approx V\mathbf{x}^{(i)}$. Finally, prediction of a novel point $\mathbf{x}^{(*)}$ can be either achieved by a decompression matrix $U' \in \mathbb{R}^{K\times\tilde{K}}$ that returns the projected label vectors to their original space(Hsu et al., 2009; Cisse et al., 2013; Bi and Kwok, 2013; Chen and Lin, 2012; Weston et al., 2011; Yu et al., 2014; Lin et al., 2014; Mineiro and Karampatziakis, 2015; Xu et al., 2016) or by means of the k-nearest neighbour algorithm in the embedded space (Bhatia et al., 2015; Tagami, 2017). These methods, except SLEEC (Bhatia et al., 2015), are considered obsolete in the field of XML the last 3 years, since they require large training/prediction times, high memory needs and they exhibit poor predictive performance.

Tree-based

Methods of this category based on the construction of decision trees (Agrawal et al., 2013) that partition the input space based on some criterion that is recursively optimized to finally give a tree-structure (Agrawal et al., 2013; Jain et al., 2016; Jasinska et al., 2016; Prabhu et al., 2018a; Prabhu and Varma, 2014a; Si et al., 2017). These methods are often characterized by low training and prediction times but large model sizes (in terms of memory footprint) and inferior predictive performance (Prabhu et al., 2018b).

1-vs-All-based

The last category can be considered as a subset of the problem transformation class. Here, as the name suggests, one binary classifier is learnt for each of the K labels. These methods (Babbar and Schölkopf, 2017; Liu et al., 2017; Niculescu-

Mizil and Abbasnejad, 2017; Weston et al., 2013; Yen et al., 2016, 2017) are described by their high prediction accuracies and low model sizes. Nonetheless, the fact that they have to train one classifier per label renders them slow in terms of training and prediction time.

Finally, we mention that the only hybrid method that combines the advantages of the tree and 1-vs-all based methods is Parabel (Prabhu et al., 2018b) which is one of the state-of-the-art methods in XML.

1.2.1 Performance evaluation metric

In contrast to traditional supervised tasks like multi-class classification, performance evaluation in multi-label learning cannot be achieved by conventional metrics such as accuracy, area under the ROC curve, etc. due to the more complicated nature of the problem that involves multiple labels simultaneously. Thus, one has to resort to different metrics suitably for multi-label problems. An overview of those metrics is offered in Zhang and Zhou (2013) where they are categorized into two groups, *example-based* and *label-based* metrics. The first group involves metrics that independently measure the performance on each test sample and then return the mean value of the performances of all test samples in the test set while the second group is concerned with the performance evaluation of each class label separately. In this thesis we focus our attention to a specific example-based metric.

As we mentioned previously, extreme multi-label classification datasets are described by positive label sparsity that associates just a few positive labels to each data point. This means that it is crucial for our learnt function g_{ML} to be able to accurately predict the few positive labels per data point instead of the colossal number of the negative labels. In other words, the indices of the larger values of the score vector $\mathbf{f}$ should correspond to the associated positive class labels. For this reason, assuming a test set $\{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{N^*}$, the prediction accuracy of a XML method is evaluated using the *precision* at $k \in \{1, \cdots, K\}$ metric (Agrawal et al., 2013; Hsu et al., 2009; Kapoor et al., 2012; Weston et al., 2011; Yu et al., 2014; Bhatia et al., 2015) which is defined as

$$\mathrm{P@k} = \frac{1}{N^*} \sum_{i=1}^{N^*} \frac{1}{k} \sum_{c \in \mathrm{rank}_k(\mathbf{f}^{(i)})} \frac{y_c^{(i)} + 1}{2}, \tag{1.21}$$

where $\mathbf{f}^{(i)} = g_{\mathrm{ML}}(\mathbf{x}^{(i)}) \in \mathbb{R}^K$ is the predicted score vector, $\mathrm{rank}_k(\cdot)$ is the rank function which returns a set of k indices that correspond to the largest values of an input vector, and $y_c^{(i)}$ is the c^{th} element of the i^{th} ground truth label vector $\mathbf{y}^{(i)}$. This metric counts the percentage of correct predictions in the top-k positive predictions. This metric is the most widely used one in the XML field which is motivated by real world applications such as tagging and recommendation systems where only the precision of the top recommendations matters (Bhatia et al., 2015). We extensively employ it in Section of 4.5.3 of Chapter 4 to evaluate performance of our proposed model.

1.2.2 The chosen XML methods

In this section we provide a short description of four XML algorithms which are used as comparison baselines for experiments conducted in Chapter 4. The reasons of their choice are discussed in 4.5.3 and can be summarized by the fact that those methods (i) attain high predictive performance, (ii) belong to at least one of the three aforementioned XML classes of methods, and (iii) provide predictive performance results in terms of P@k via the XML repository (Repository, 2010), for all the datasets used for experiments in Chapter 4. These methods are the SLEEC, FastXML, PFastreXML, and the PD-Sparse method.

SLEEC (Bhatia et al., 2015) : This is an embedding-based method which creates an ensemble of multi-label classifier in the following way. First the initial dataset X is partitioned into a specific number of clusters and then for each cluster embeddings $\mathbf{z}_i$ are learnt by regressors V, in a manner that allows capturing inter-label correlations by preserving the pairwise distances between only the closest label vectors. Prediction of a novel point is achieved by utilization of the k-nearest neighbour classifier, to find the cluster that this point is closer to and then use the appropriate embeddings to predict the label vector. The authors, to make their results robust against the high-dimensional input spaces, use an ensmble of these XML classifiers.

FastXML (Prabhu and Varma, 2014b): The main idea of this tree-based method is to learn a hierarchy over the input space motivated by the observation that just a small number of positive labels are found in regions of the input space. This hierarchy is learnt by optimizing a ranking loss function called normalized Discounted Cumulative Gain (nDCG) (Yu et al., 2014) that leads to leaf nodes that consist of constant classifiers. Therefore prediction can achieved by traversing this tree-like hierarchy and then taking into account solely the positive label in this region of the input space.

PFastreXML (Jain et al., 2016): The second tree-based method in our list. PFastreXML shares the same philosophy of FastXML and it was developed to improve tail label prediction, i.e. labels that do not appear frequently in training data set and thus their prediction is more challenging than frequently occurring ones. PFastreXML achieves that by optimizing a variant ranking loss function from FastXML's nDCG, suitable for this purpose.

PD-Sparse (Yen et al., 2016): The final method belongs to the 1-vs-All category and as such builds K binary classifiers. These classifiers are obtained by solving a primal-dual sparse problem where the well-known SVM margin-maximizing loss with l1-l2 penalties is maximized. The whole idea exploits the sparsity of both the primal and the dual while storing issues due to large variable arrays are solved by referring to hash-tables.

1.3 Thesis overview

As we discuss in the previous sections, applying Gaussian process models in practice can be proven challenging due to the adverse computational complexity $\mathcal{O}(N^3)$ and the absence of closed form solutions of the posterior and the marginal densities. This is a common scenario for multi-class tasks that involve non-Gaussian likelihoods. Nevertheless, none of the previous works on multi-class classification using Gaussian processes takes into account the cases where the number of classes is particularly large. Moreover, the use of Gaussian process models on the more general multi-label classification problems, has not been investigated so far. This thesis is concerned with addressing those two tasks by focusing on instances with extremely large dimensions. More specifically,

- Chapter 2 provides all the mathematical background needed for developing the methods in the coming chapters. It introduces the concepts of variational inference and how these are applied to Gaussian process models to attain scalable inference, mostly based on the work of Titsias (2009) and Hensman et al. (2015b). Furthermore, two new techniques are also presented in this chapter that the first technique involves a parsimonious representation of the variational distribution while the second one shows how we can reduce the burden of high dimensional input data.

- Chapter 3 is devoted to multi-class classification with Gaussian processes. It discusses a computationally effective way to deal with arbitrarily large number of classes based on a lower bound on the softmax likelihood as it is presented in Titsias (2016). This is combined by the variational framework suitable for Gaussian processes of Chapter 2 to give a new scalable method over the number of both data points and classes. Motivated by this method, a methodology that allows to approach regression tasks using multi-class classification algorithms is also introduced.

- Chapter 4 is concerned with the deployment of Gaussian process models on multi-label classification problems. A highly scalable method is developed that can successfully cope with extremely large number of training instances, input dimensions, and labels at the same time. The method is tested with a number of challenging datasets that involve such dimensions.

All the chapters have been written in a way that permits each of them to be read independently. Finally, we conclude and discuss some future research directions in 5.

Chapter 2

Variational inference for sparse Gaussian processes

2.1 Introduction

This chapter is focused on the presentation of the general ideas of the Variational Inference (VI) theory, mainly applied to sparse Gaussian Process models which are most relevant for the rest of this thesis. Most of the notions of VI can be found in several well-written introductions to VI such as Jordan et al. (1999) or related tutorials (Fox and Roberts, 2012). Finally, the reader is also referred to the excellent review of VI in Blei et al. (2017).

VI tries to tackle one of the most fundamental problems of modern statistics which is the approximation of probability densities that are computationally difficult (or even infeasible) to be calculated. These kind of probability densities are ubiquitous in Bayesian models where their posterior most of the times cannot be found analytically and thus, we have recourse to approximation or sampling strategies. VI is exactly this approximation strategy and it is employed as an alternative to Markov Chain Monte Carlo (MCMC) (Hastings, 1970; Gelfand and Smith, 1990) sampling to approximate posterior densities for Bayesian models while both of those methods emerged from the field of statistical physics. In general, VI constitutes a good surrogate of MCMC methods when our primary concern is to obtain fast results for datasets with large number of instances or more complex models (Blei et al., 2017), and we are willing to sacrifice some approximation precision at the same time.

The general problem can be mathematically formulated as follows. Assuming that we have a vector of N observations $\mathbf{y} \in \mathbb{R}^N$ and their corresponding latent variables $\mathbf{f}$, then their joint density is written as

$$p(\mathbf{y}, \mathbf{f}) = p(\mathbf{y}|\mathbf{f})p(\mathbf{f}), \tag{2.1}$$

where $p(\mathbf{f})$ and $p(\mathbf{y}|\mathbf{f})$ represent the prior density over the latent variables and the data likelihood, respectively. In the Bayesian paradigm, we would like to be able to compute the posterior density

$$p(\mathbf{f}|\mathbf{y}) = \frac{p(\mathbf{y}|\mathbf{f})p(\mathbf{f})}{p(\mathbf{y})}. \tag{2.2}$$

We denote the posterior distribution with density $p(\mathbf{f}|\mathbf{y})$ as $\widetilde{P}$. Generally in this chapter we denote distributions with capital Latin letters and densities with lower-case Latin letters. The computation intractability of the posterior density many times stems from the denominator term of 2.2. This denominator is the marginal likelihood of the observations or *evidence* and works as a normalization constant for the true posterior. Its calculation is derived by marginalizing out the latent variables $\mathbf{f}$ from the joint density, i.e.

$$p(\mathbf{y}) = \int p(\mathbf{y}, \mathbf{f}) d\mathbf{f}. \tag{2.3}$$

In contrast to an MCMC algorithm that uses sampling, where in the limit its random samples coincide with samples from the true posterior, VI refers to optimization for finding an approximation of the posterior. This is achieved by positing a family of tractable distributions over the latent variables $\mathcal{Q}$ and then looking for a distribution Q^* that is as similar as possible to the exact posterior $\widetilde{P}$. This similarity measure is given by the Kullback-Leibler (KL) divergence (Kullback and Leibler, 1951) and the optimal variational distribution Q^* is calculated by solving the following optimization problem,

$$Q^* = \operatorname*{argmin}_{Q \in \mathcal{Q}} \mathrm{KL}[Q \mid\mid \widetilde{P}]. \tag{2.4}$$

Therefore, the variational distribution Q^* can be employed as a proxy of the posterior. For notational convenience, we omit the latent variables of the distributions in order to emphasize that the minimization is over the distributions. Later in the chapter, we change the notation slightly regarding the approximation densities to increase readability. We should also note that the minimization in Eq. 2.4 is generally over infinite-dimensional objects, such as continuous densities and thus, we need to use[1] tools of the Calculus of Variations (Gelfand et al., 2000). Hence the term "variational inference".

Regarding the rest of the chapter, Section 2.2.1 presents how the optimization of 2.4 is achieved in practice and what factorization assumptions are usually taken. Moreover, Section 2.3 introduces the concept of sparse Gaussian processes and how

[1]Although, in practice, the choice of a specific parametric variational family $\mathcal{Q}$ makes the use of Calculus of Variations unnecessary.

it is combined with the theory of VI. Finally, two methods are described in Sections 2.4 and 2.5, which constitute some of the novelties of this thesis and allow sparse Gaussian Process models under a VI framework to scale well in extreme conditions of large number of data instances and input dimensionality.

2.2 The evidence lower bound and tractable variational families $\mathcal{Q}$

2.2.1 The evidence lower bound

In essence, having defined the family of approximation distributions $\mathcal{Q}$ for the intractable true posterior $\widetilde{P}$, VI aims to find the "closest" distribution Q^* in that family to $\widetilde{P}$, where "closeness" is defined in terms of the KL-divergence to the exact posterior; hence, inference amounts to the optimization task in 2.4, as we saw previously. Moreover, the difficulty of that optimization is inextricably linked to the complexity of the chosen variational family $\mathcal{Q}$. On the other hand, the intractability of $\widetilde{P}$ we mentioned before, can stem from at least three different scenarios. The first one, which is maybe the most common, is the case where an analytic expression of $\widetilde{P}$ is unavailable due to the non-closed form solution of the marginal likelihood in 2.3. This in Bayesian Inference amounts to a non-conjugate relation between prior and data likelihood densities. Secondly, there are examples where intractability means that evaluating $\widetilde{P}$ is provably a NP-hard problem as is the case for undirected graphical models. Finally, a closed-form solution may be available for $\widetilde{P}$ and other relevant expectations of interest whereas their computation can be achieved in polynomial time. Nonetheless, the exponent of the polynomial may be high enough to render those computations infeasible in practice. This is exactly the case of the Gaussian Process regression with Gaussian likelihood which requires $\mathcal{O}(N^3)$ computational time and cannot scale for large datasets. For the rest of the chapter, we assume that the posterior $p(\mathbf{f}|\mathbf{y})$ and marginal likelihood $p(\mathbf{y})$ are intractable in contrast to the joint density $p(\mathbf{y}, \mathbf{f})$ which is computable. This setting is familiar from the previous chapter when we tried to use GP models for multi-class classification problems in Section 1.1.2.

We re-write now the optimization problem in 2.4, using the densities $q(\mathbf{f})$ for the variational distributions Q in $\mathcal{Q}$ since the optimization problem is equivalent. Therefore, we have

$$q^*(\mathbf{f}) = \underset{q(\mathbf{f})\in\mathcal{Q}}{\operatorname{argmin}} \, \mathrm{KL}[q(\mathbf{f}) \;||\; p(\mathbf{f}|\mathbf{y})], \tag{2.5}$$

where $\mathrm{KL}[q(\mathbf{f}) \;||\; p(\mathbf{f}|\mathbf{y})] \triangleq \mathrm{KL}[Q \;||\; \widetilde{P}]$, i.e. we denote the Kullback-Leibler divergence between two distributions using their corresponding densities instead of the distributions themselves. We follow this notation for the Kullback-Leibler divergence between two distributions for the rest of the thesis. This optimization problem cannot be solved in practice since it involves the computation of the log marginal likelihood $p(\mathbf{y})$ in Eq. 2.3 which is the very reason we refer to VI. This is because we can expand the KL term in 2.5 as

$$\begin{aligned}\mathrm{KL}[q(\mathbf{f}) \mid\mid p(\mathbf{f}|\mathbf{y})] &= \mathbb{E}[\log q(\mathbf{f})] - \mathbb{E}[\log p(\mathbf{f}|\mathbf{y})] &(2.6)\\ &= \mathbb{E}[\log q(\mathbf{f})] - \mathbb{E}[\log p(\mathbf{f},\mathbf{y})] + \log p(\mathbf{y}), &(2.7)\end{aligned}$$

where all expectations are taken with respect to the variational density $q(\mathbf{f})$. The presence of $\log p(\mathbf{y})$ on the right-hand side of (2.7) is exactly why we cannot directly minimize the KL-divergence; however, using the non-negativeness property of KL-divergence we can impose a lower bound on the log evidence written as

$$\begin{aligned}&\mathrm{KL}[q(\mathbf{f}) \mid\mid p(\mathbf{f}|\mathbf{y})] \geq 0\\ \iff\ &\log p(\mathbf{y}) \geq \mathbb{E}[\log p(\mathbf{f},\mathbf{y})] - \mathbb{E}[\log q(\mathbf{f})]\\ \iff\ &\log p(\mathbf{y}) \geq \mathbb{E}[\log p(\mathbf{y}|\mathbf{f})] - \mathrm{KL}[q(\mathbf{f}) \mid\mid p(\mathbf{f})] = \mathcal{F}. \qquad (2.8)\end{aligned}$$

Therefore, the minimization problem of 2.7 is equivalent to maximization of the quantity $\mathcal{F}$ which sometimes referred as the *evidence lower bound* or ELBO. Notice that in this formulation, all the terms of $\mathcal{F}$ are computable.

For completeness, we would also like to add that the above description is the simplest (vanilla) case of the variational inference paradigm. There are more sophisticated methods based on variational inference, such as *collapsed variational inference* (Teh et al., 2007, 2008) (CVB) and *stochastic variational inference* (SVI) (Hoffman et al., 2013). The former technique considers marginalizing out some of the hidden variables which leads to lower dimensional posterior while a tighter lower bound on log marginal likelihood is emerged. Typical examples of its successful deployment is over latent Dirichlet allocation models (Blei et al., 2003). Regarding SVI, the main idea is to use stochastic optimization (Robbins and Monro, 1951) to maximize $\mathcal{F}$, by following noisy estimates of the natural gradients (Amari, 1998) where the noise is due to data subsampling. This allows us to deal with large datasets effectively while natural gradients effectively lead to better (and usually faster) optima than the vanilla approach.

As we shall see later, SVI plays a crucial role to the development of a scalable framework for GP models. However, SVI heavily relies its success on the fact that natural gradients are taken into account since ordinary gradients of the variational parameters with respect to $\mathcal{F}$ do not point to the direction that achieves the maximum change in KL divergence as natural gradients do. This is due to the nature of the optimization which is over a Riemannian manifold (Amari, 1982) and not over the Euclidean space that the variational parameters lie in. In this thesis, we should make clear from the beginning that we do not take into account this fact to the optimization of the ELBO in chapters 3 and 4, although the optimization algorithms that we use, such as Adam (Kingma and Ba, 2014), provides an approximation of the natural gradients[2]. Nevertheless, we leave the incorporation of natural gradients for future work (see Chapter 5).

2.2.2 Tractable variational families $\mathcal{Q}$

The derivation of $\mathcal{F}$ allows us to approximate the marginal likelihood in a tractable way. However, the determination of the variational family $\mathcal{Q}$ is of paramount importance for that endeavour. For this reason, factorization assumptions are imposed on the densities $q(\mathbf{f})$ of the members of $\mathcal{Q}$. More specifically, the density of the variational distribution over an N-dimensional Euclidean space is factorized into a product of densities by partitioning that space with $|\mathcal{E}|$ sub-groups and expressing $q(\mathbf{f})$ as

$$q(\mathbf{f}) = \prod_{e \in \mathcal{E}} q(\mathbf{f}_e). \tag{2.9}$$

This kind of family of factorized variational distributions is generally known as the *mean-field variational family* (Wainwright et al., 2008) and it has been widely-used for a large number of VI problems. Despite its simplicity and computability, the factorized nature of 2.9 fails to capture any correlations between variables that belong to different sub-groups in $\mathcal{E}$, something that may not be true for the exact posterior and thus, imposing limitations in some cases. These limitations can be overcome by using structured variational inference (Saul and Jordan, 1996; Barber and Wiegerinck, 1999) or mixture of variational densities (Bishop et al., 1998); nevertheless, better approximation results come at a computational cost. Moreover, mean-field-based inference often leads to underestimated values for the true marginal variance which is a direct consequence of the KL divergence from the variational distribution to the posterior in Eq. (2.6) (Blei et al., 2017; Fox and Roberts, 2012).

Apart from the factorization assumptions, another way to achieve tractability in the context of VI is to posit a specific parametric family of variational distributions in $\mathcal{Q}$ which are easily computable. The most common choice of such a parametric family is the exponential one (Kupperman et al., 1958) which is the most famous one in the VI literature. For instance, let the latent variables $\mathbf{f}$ follow

[2]More accurately Adam approximates the Fisher information matrix, which gives curvature information in the Riemannian space, by a diagonal matrix.

a N-dimensional Gaussian distribution with zero mean vector and covariance K, i.e.

$$p(\mathbf{f}) = \mathcal{N}(\mathbf{0}_N, K), \tag{2.10}$$

where the likelihood term factorizes as $p(\mathbf{y}|\mathbf{f}) = \prod_{i=1}^{N} p(y_i|f_i)$ with y_i, f_i being the i^{th} element of vectors $\mathbf{y}$ and $\mathbf{f}$ respectively. The case of choosing $\mathcal{Q}$ to be the family of multivariate Gaussian distribution is of particular relevance in this thesis. Under these assumptions about the Gaussian latent model 2.10 and $\mathcal{Q}$, we can write the KL divergence using Eq. 2.8 as

$$\mathrm{KL}[q(\mathbf{f}) \mid\mid p(\mathbf{f}|\mathbf{y})] = \mathrm{KL}[q(\mathbf{f}) \mid\mid p(\mathbf{f})] - \left[\sum_{i=1}^{N} \int q(f_i) \log p(y_i|f_i) df_i\right] + \log p(\mathbf{y}). \tag{2.11}$$

The whole computation of the KL-divergence in (2.11) scales as $\mathcal{O}(N^3)$ due to the computation (see Appendix A.2) of the KL-divergence term between two multivariate normal distributions $\mathrm{KL}[q(\mathbf{f})||p(\mathbf{f})]$. Additionally, each one-dimensional integral on the right-hand side of (2.11) can be efficiently calculated using Gauss-Hermite quadrature (Liu and Pierce, 1994) under the marginal variational density $q(f_i)$. Note that the cubic complexity of (2.11) renders this family of distributions $\mathcal{Q}$ intractable in practice; however, Section 2.3 presents a principled way to deal with that problem in the context of Gaussian processes. Apart from the computational issues, another pertinent question is how the variational density $q(\mathbf{f})$ (which is a multivariate normal distribution) is parametrized. The naive way requires $\mathcal{O}(N^2)$ parameters as result of the total number of parameters we need to parametrize its covariance matrix. Nonetheless, Opper and Archambeau (2009) showed that we can achieve the same approximation performance as the naive parametrization by just using $\mathcal{O}(N)$ parameters in total. We make extensive use of this idea through this PhD thesis, beginning by Section 2.4. Finally, it is worth mentioning that this kind of variational family $\mathcal{Q}$ can be more flexible and accurate than the mean-field approach we described before since it is possible to capture correlations between the posterior latent values.

2.3 Sparse Gaussian processes in a variational inference framework

2.3.1 General sparse Gaussian process methods

Sparse Gaussian Process theory was developed to cope with the computationally expensive full GP Inference which in the general case scales as $\mathcal{O}(N^3)$. There is a voluminous literature over the sparse Gaussian process framework where a comprehending overview over a large number of methods based on that framework is given by Quiñonero-Candela and Rasmussen (2005b). The main idea is the consistency of the Gaussian process, as discussed in Section 1.1 of Chapter 1, in conjunction with conditional independence assumptions over an approximation on GP prior. More specifically, Quiñonero-Candela and Rasmussen (2005b) start their discussion by using the exact GP prior,

$$p(\mathbf{f}^{(*)}, \mathbf{f}) = \int p(\mathbf{f}^{(*)}, \mathbf{f}|\mathbf{u})p(\mathbf{u})d\mathbf{u}, \tag{2.12}$$

where the function values $\mathbf{f} \in \mathbb{R}^N$ and $\mathbf{f}^{(*)} \in \mathbb{R}^{N^*}$ are given by a zero-mean GP $f \sim \mathcal{GP}(\mathbf{0}, \mathbf{k}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}))$, evaluated at a training set $X \in \mathbb{R}^{N \times D}$ and test set $X^* \in \mathbb{R}^{N^* \times D}$ of D-dimensional data points, respectively. Then, the prior $p(\mathbf{f}^{(*)}, \mathbf{f})$ is approximated by density $g(\mathbf{f}^{(*)}, \mathbf{f}))$ as

$$p(\mathbf{f}^{(*)}, \mathbf{f}) \approx g(\mathbf{f}^{(*)}, \mathbf{f}) = \int g(\mathbf{f}^{(*)}|\mathbf{u})g(\mathbf{f}|\mathbf{u})p(\mathbf{u})d\mathbf{u}, \tag{2.13}$$

where the extra latent function values $\mathbf{u}$ come from the same GP evaluated at M (potentially different from the training points) data points $Z \in \mathbb{R}^{M \times D}$. By using different choices of $g(\mathbf{f}^{(*)}|\mathbf{u})$ and $g(\mathbf{f}|\mathbf{u})$ but having the same conditional independence assumption of $\mathbf{f}$ and $\mathbf{f}^{(*)}$ given $\mathbf{u}$, several sparse Gaussian methods have emerged (Smola and Bartlett, 2001; Seeger et al., 2003; Snelson and Ghahramani, 2006). All those methods attain to reduce time complexity from $\mathcal{O}(N^3)$ to $\mathcal{O}(NM^2)$ due to this conditional independence assumption, where $M \ll N$. The latent function values $\mathbf{u}$ are usually called *inducing variables* while the corresponding points Z, where those values are evaluated at, are called *inducing inputs/points*. The name derives from the property of $\mathbf{u}$ to connect the two disjoint sets of variables $\mathbf{f}$ and $\mathbf{f}^{(*)}$; thus, *inducing* the dependencies between training and test variables as it is pointed out in Quiñonero-Candela and Rasmussen (2005b).

An additional challenge for the sparse Gaussian process framework stems from the fact that we need somehow to choose the M values of the inducing inputs in order to approximate the true posterior in a computationally cheap way. A naive choice would be to randomly pick M points from the training dataset. More sophisticated methods, based on greedy optimization, have been developed that choose a subset of points from the training dataset using various selection criteria (Smola and Schölkopf, 2000; Lawrence et al., 2002; Smola and Bartlett, 2001; Seeger et al., 2003; Keerthi and Chu, 2006). Nevertheless, those methods involve the exact or approximate solution of a difficult combinatoric problem while optimizing the continuous values of kernel hyperparameters at the same time. This can impose additional problems as discussed by Snelson and Ghahramani (2006). Those problems were solved temporarily by the Fully Independent Training Conditional (FITC) method of Snelson and Ghahramani who introduced a model approximation method that optimizes a modified version of the true marginal likelihood of the model over both hyperparameters and inducing inputs. This modified marginal likelihood derived by a specific independence assumptions that lead to computational tractability but results showed that this modification was prone to overfitting behaviour when the number of inducing inputs increased, although this increase did not always improve the posterior approximation.

Nonetheless, as we shall see in next section, Titsias (2009) overcame these issues of overfitting and absence of a mathematically rigorous method for posterior approximation by introducing a coherent variational inference framework for sparse Gaussian processes.

2.3.2 The variational inference framework for sparse Gaussian processes

As we discussed in the previous section, most of the methods relied on Sparse Gaussian process approximation had a number of limitations due to several reasons like (1) computationally challenging combinatoric optimization, (2) overffiting issues, and (3) lack of a similarity measure between the exact and approximated model. Nevertheless, in 2009, Titsias in his pioneering work (Titsias, 2009) managed to address these obstacles by presenting a rigorous way of minimizing a similarity measure between the true and the approximated model. More specifically, he considered the augmented GP posterior

$$\begin{aligned}\hat{P} \triangleq p(\mathbf{f},\mathbf{u}|\mathbf{y}) &= \frac{p(\mathbf{y},\mathbf{f},\mathbf{u})}{p(\mathbf{y})} \\ &= \frac{p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{u})p(\mathbf{u})}{p(\mathbf{y})} \\ &= \frac{p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{u})p(\mathbf{u})}{\int p(\mathbf{y}|\mathbf{f})p(\mathbf{f}|\mathbf{u})p(\mathbf{u})d\mathbf{f}d\mathbf{u}},\end{aligned} \tag{2.14}$$

where the M-dimensional random vector $\mathbf{u}$ represents the inducing variables in a similar manner as explained in the previous section. Subsequently, having chosen a variational distribution with density $\hat{q}(\mathbf{f},\mathbf{u})$, he minimized the KL-divergence between the augmented posterior $\hat{P}$ and that distribution ,

$$\mathrm{KL}[\hat{q}(\mathbf{f},\mathbf{u}) \mid\mid p(\mathbf{f},\mathbf{u}|\mathbf{y})], \tag{2.15}$$

where the $\hat{}$ symbol over the variational densities indicates use of the augmented model. Further, the chosen variational family $\mathcal{Q}$ includes variational distributions with factorized densities of the form

$$\hat{q}(\mathbf{f}, \mathbf{u}) = p(\mathbf{f}|\mathbf{u})\hat{q}(\mathbf{u}), \tag{2.16}$$

where $\hat{q}(\mathbf{u})$ is a M-dimensional Gaussian distribution parametrized by a mean vector and a covariance matrix where its parametrization is thoroughly discussed in Section 2.4. The required computational time for each optimization step of (2.15) is $\mathcal{O}(NM^2)$, similar to all the previously introduced sparse GP methods. The KL term in (2.15) allows us to choose both the kernel hyperparameters and the inducing inputs Z by minimizing it over them similarly as Snelson and Ghahramani (2006) via continuous optimization. However, as Bauer et al. (2016) mentions, the most serious drawbacks of FITC include overfitting, failing to improve approximation after increasing the number of inducing inputs and inability to recover the true posterior. Titsias' objective function is able to circumvent overfitting since inducing inputs Z are now treated as variational parameters instead of kernel hyperparameters as in FITC. Moreover, minimizing the KL-divergence between the variational and the true posterior GP provides a rigorous approximation method where the increase of the number of inducing inputs M provably gives a lower KL-divergence (Titsias, 2009; Bauer et al., 2016; Matthews, 2017), bringing the variational GP "closer" to the exact posterior. In the special case that we match the positions of the inducing inputs with those of the training dataset, i.e. $Z = X$, (2.15) coincides with the KL-diverence term in (2.5). These attributes constituted the variational sparse Gaussian process framework highly important for dealing with the adverse computational scaling required by full GP inference, leading to new research routes (Titsias and Lawrence, 2010; Damianou et al., 2012; Hensman et al., 2013; Damianou and Lawrence, 2013; Hensman et al., 2015b,a).

Despite the previously discussed benefits of Titsias' method, it still scaled as $\mathcal{O}(NM^2)$ which can be prohibitive for large number of training instances N. This limitation was solved a few years later by Hensman et al. (2013) where they made use of the idea of SVI. SVI has been extensively used to achieve scalable Bayesian inference for large datasets where its main idea relies on the fact that the variational parameters can be efficiently updated using stochastic but unbiased estimates of the ELBO's gradient on a mini-batch/subset of training points. Hensman et al. re-introduced Titsias' approximation framework in an suitable manner that allows considering mini-batches; hence, paving the way for performing SVI and reducing the computational complexity from $\mathcal{O}(NM^2)$ to $\mathcal{O}(|X_b|M^2)$, where X_b is a mini-batch of training data points with size $|X_b| \ll N$.

Finally, we would like to add that the minimization of the augmented KL-divergence in (2.15) is not the same as the minimization of the KL$[q(\mathbf{f})\ ||p(\mathbf{f}|\mathbf{y})]$ (see Section 2.2) as Titsias originally mistakenly claimed in Titsias (2009). Actually, the augmented KL-divergence is an upper bound of the unaugmented one as Matthews et al. (2016) showed using measure-theoretic tools. Nonetheless, in practice, the augmented model gives sufficiently good results and has been proven successful in a large number of sparse GP-based applications. Finally, the derivation steps of this section with its methodology constitutes the foundations of the methods described in the coming Chapters 3 and 4.

2.4 Two potential parametrizations for $\hat{q}(\mathbf{u})$

In the last few years, after the introduction of the coherent variational framework for sparse Gaussian Processes from Titsias (2009), the dominant way to parametrize the densities of variational distribution $\hat{q}(\mathbf{u})$, assuming that $\hat{q}(\mathbf{u}) = \mathcal{N}(\mathbf{u}|\mathbf{m}, S)$, was via $\frac{M(M+3)}{2}$ free variational parameters, since $\mathbf{m} \in \mathbb{R}^M$ and $S = LL^\top$ where L is an $M \times M$ lower triangular matrix. This parametrization effectively ensures the positive definiteness of the variational covariance matrix S while it is closely related to the Cholesky decomposition of the positive definite matrix S. Nonetheless, the uniqueness of the Cholesky decomposition stems from the fact that its diagonal entries are strictly positive, something that is not true for this parametrization. In practice though, we can sacrifice uniqueness for unconstrained optimization without any loss of performance as it is pointed out in Matthews (2017).

In Section 2.2.2 with the Gaussian latent model, we saw that without any augmentation assumptions, the KL-divergence term can be written as in (2.11). The computation of the augmented KL-divergence in 2.15 gives a very similar form,

$$\begin{aligned}\mathrm{KL}[\hat{q}(\mathbf{f},\mathbf{u})\ ||p(\mathbf{f},\mathbf{u}|\mathbf{y})] &= \mathbb{E}_{\hat{q}(\mathbf{f},\mathbf{u})}[\log \hat{q}(\mathbf{f},\mathbf{u})] - \mathbb{E}_{\hat{q}(\mathbf{f},\mathbf{u})}[\log p(\mathbf{y},\mathbf{f},\mathbf{u})] + \log p(\mathbf{y}) \\ &= \mathrm{KL}[\hat{q}(\mathbf{u})\ ||p(\mathbf{u})] - \left[\sum_{i=1}^{N} \int \hat{q}(f_i) \log p(y_i|f_i) df_i\right] + \log p(\mathbf{y}),\end{aligned} \tag{2.17}$$

where $\hat{q}(f_i) = \int p(f_i|\mathbf{u})\hat{q}(\mathbf{u})d\mathbf{u}$ and the final equation (2.17) is obtained by combining (2.6) and (2.14). Therefore, the ELBO we wish to maximize is written as

$$\mathcal{F}(\mathbf{m}, S) = \sum_{i=1}^{N} \mathbb{E}_{\hat{q}(\mathbf{u})}\left[\log G(y^{(i)}, \mathbf{u})\right] - \mathrm{KL}[\hat{q}(\mathbf{u})||p(\mathbf{u})], \tag{2.18}$$

where the data term $\log G(y^{(i)}, \mathbf{u}) = \mathbb{E}_{\hat{q}(f^{(i)}|\mathbf{u})}[\log p(y^{(i)}, \mathbf{u})]$ while the kernel hyperparameters and the inducing inputs Z are considered to be fixed in this section for notational simplicity. Notice that the variational and prior distributions are multivariate normal over $\mathbf{u}$, rendering the computation of their KL divergence analytically feasible (see Appendix A.2), scaling as $\mathcal{O}(M^3)$. Such a parametrization has appeared several times in literature, such as in Titsias and Lázaro-Gredilla (2014); Damianou and Lawrence (2013); Hensman et al. (2015b); Salimbeni and Deisenroth (2017), and to the best of our knowledge, there is no other different form of parametrization in sparse GPs literature. Ii this parametrization, which we call "Full" and denote by "FP", the $\hat{q}(\mathbf{f})$, induced by the $\hat{q}(\mathbf{u}) = \mathcal{N}(\mathbf{u}|\mathbf{m}, S = LL^\top)$, is $\hat{q}(\mathbf{f}) = \mathcal{N}(\mathbf{f}|\mathbf{m}^f, \mathbf{S}^f)$ where

$$\mathbf{m}^f = Q_{XZ}\mathbf{m}, \tag{2.19}$$
$$S^f = K_X - Q_{XZ}K_{ZX} + R_{XZ}R_{XZ}^\top, \tag{2.20}$$

where K_Z is the $M \times M$ kernel matrix evaluated at the inducing inputs Z, $K_{XZ} \in \mathbb{R}^{N\times M}$ is the cross-covariance matrix between training and inducing inputs with $K_{ZX} = K_{XZ}^\top$ and K_X is the corresponding $N \times N$ kernel matrix of the training inputs. Further, we have the matrices $Q_{XZ} = K_{XZ}K_Z^{-1} \in \mathbb{R}^{N\times M}$ and $R_{XZ} = Q_{XZ}L \in \mathbb{R}^{N\times M}$. For the derivation of (2.19) and (2.20), we use known multivariate Gaussian formulas presented in Appendix A.1. The FP leads to the following result for the KL term in (2.18),

$$\begin{aligned}\mathrm{KL}[\hat{q}(\mathbf{u})||p(\mathbf{u})] &= \int \hat{q}(\mathbf{u}) \log \frac{\hat{q}(\mathbf{u})}{p(\mathbf{u})} d\mathbf{u} \\ &= \frac{1}{2}\left(\mathbf{m}^T K_Z^{-1}\mathbf{m} + \mathrm{tr}\left(K_Z^{-1}S\right) + \log|K_Z| - \log|S| - M\right).\end{aligned} \tag{2.21}$$

Moreover, the marginal $\hat{q}(f^{(i)})$ in the expectations of the first data term in (2.18) becomes $\hat{q}(f^{(i)}) = \mathcal{N}(f^{(i)}|m^{(i)}, s^{(i)})$ where $m^{(i)}$ is the i^{th} element of the N-dimensional vector $\mathbf{m}^f$ in(2.19) and $s^{(i)}$ is the i^{th} element of the the main diagonal of S^f in (2.20) as well.

Nonetheless, FP depends on the optimization of $\mathcal{O}(M^2)$ parameters which could be restrictive for larger numbers of inducing inputs. Moreover, the use of FP introduces a strong dependence between $(\mathbf{m}, \mathbf{S})$ and the kernel matrix K_Z from the prior $p(\mathbf{u})$ which could lead to slow convergence[3]. To expose this dependence, we follow the derivation steps of Opper and Archambeau (2009) to find the maximum of $\mathcal{F}$. This is achieved by using $\mathbf{m} = K_Z\boldsymbol{\mu}$ and $\mathbf{S} = (K_Z^{-1} + \Lambda^{-1})^{-1}$ for some M-dimensional vector $\boldsymbol{\mu}$ and some full $M \times M$ (non-diagonal) positive definite matrix Λ associated with the second derivatives of the first data term in the above bound. The form of $\mathbf{m}$ and $\mathbf{S}$ suggests a parametrization of $\hat{q}(\mathbf{u})$ in terms of $(\boldsymbol{\mu}, \Lambda)$ in order to take advantage of the preconditioning with the kernel matrix K_Z. Nonetheless, this can still lead to slow optimization because the full Λ matrix requires optimizing over $\mathcal{O}(M^2)$ parameters. Therefore, here we propose to

[3]However, as we shall see in Chapters 3 and 4, experimental results present the exact opposite behaviour.

simplify this parametrization by replacing Λ with a diagonal covariance matrix Σ leading to the parametrization

$$\mathbf{m} = K_Z \boldsymbol{\mu}, \quad \mathbf{S} = (K_Z^{-1} + \Sigma^{-1})^{-1} = K_Z - K_Z(K_Z + \Sigma)^{-1} K_Z, \tag{2.22}$$

where $\boldsymbol{\mu} \in \mathbb{R}^M$ is a real-valued vector of tunable variational parameters and Σ is a diagonal positive definite matrix (i.e. with each diagonal element restricted to be non-negative) parametrized by M additional variational parameters. Hence, overall $\hat{q}(\mathbf{u})$ is parametrized by $2M$ variational parameters while all the remaining structure comes from a careful preconditioning with the model kernel matrix K_Z. We call the above parametrization as parsimonious one or PP in short. This kind of parametrization has been used before for full (i.e. non-sparse) GPs in Opper and Archambeau (2009); Damianou et al. (2011) in order to parametrize a full $\hat{q}(\mathbf{f})$, and it was motivated by the stationary conditions satisfied by the optimal $q^*(\mathbf{f})$ in a full GP variational approximation Opper and Archambeau (2009) where at maximum the covariance is $(K_X^{-1} + \Sigma^{-1})^{-1}$ with Σ being a diagonal positive definite matrix and K_X the $N \times N$ kernel matrix evaluated at the training inputs X. In our sparse GP setting the the moments of the N dimensional multivariate Gaussian $\hat{q}(\mathbf{f})$ this time are given by

$$\mathbf{m}^f = K_{XZ} \boldsymbol{\mu}, \tag{2.23}$$

$$S^f = K_X - K_{XZ}(K_Z + \Sigma)^{-1} K_{ZX}. \tag{2.24}$$

This parametrization can recover the optimal $q^*(\mathbf{f})$ when we place the inducing inputs on the training inputs, i.e. when $Z = X$. This property is rigorously proved in Section 4.4.6 of Chapter 4 in a slightly different framework. In other cases the restricted covariance in $\hat{q}(\mathbf{f})$ is not be able to match exactly the optimal one of $q^*(\mathbf{f})$, but still in practice it tends to be very flexible especially when we optimize over the inducing inputs Z so that a posteriori $\mathbf{f}$ is well reconstructed by $\mathbf{u}$.

Furthermore, the above parametrization of $\hat{q}(\mathbf{u})$ in (2.22) leads to a numerically stable and simplified form of the lower bound. Specifically, the KL divergence term in (2.18) reduces to

$$\mathrm{KL}[\hat{q}(\mathbf{u}) || p(\mathbf{u})] = \frac{1}{2} \left(\boldsymbol{\mu}^\top K_Z \boldsymbol{\mu} - \mathrm{tr}\left((K_Z + \Sigma)^{-1} K_Z \right) + \log |K_Z + \Sigma| - \log |\Sigma| \right), \tag{2.25}$$

while for each marginal $\hat{q}(f^{(i)}) = \mathcal{N}(f^{(i)} | m^{(i)}, s^{(i)})$, this time its moments $m^{(i)}, s^{(i)}$ are given by $\mathbf{m}^f$ and S^f in (2.23) and (2.24), respectively, in a similar manner as FP. Therefore, the overall bound in (2.18) obtains a quite simplified and numerically stable form because of the cancellation of all inverses and determinants of K_Z. At each optimization the only matrix we need to decompose using Cholesky is $K_Z+\Sigma$, which is in an already numerically stable form due to the inflation of the diagonal of K_Z with Σ. In a practical implementation we can constrain the diagonal variational parameters of Σ to be larger than a small value (typically 10^{-6}) to ensure numerical stability throughout optimization. The two described parametrizations are studied in depth in Chapters 3 and 4 regarding the convergence rate and predictive power of the methods that use those parametrizations.

2.5 Subspace inducing inputs

Despite the voluminous literature on sparse GP methods, none of them consider the dimensionality of the input space D when expressing time complexities and somehow D is assumed to be small or of the order of M. Only Quiñonero-Candela and Rasmussen (2005b) mention the required complexity for computing the gradients of (2.18) with respect to the inducing input matrix Z is $\mathcal{O}(DNM^2)$ (in Section 9). Since D appears in the lower bound only through the computation of the covariance matrix K_Z and the cross covariance matrix $K_{X_b Z}$, where X_b is a mini-batch of size $|X_b| \sim \mathcal{O}(M)$, the time complexity with respect to D is clearly $\mathcal{O}(DM^2)$ since evaluating any standard kernel function on each pair of instances scales as $\mathcal{O}(D)$. Thus, each optimization step of the bound in (2.18) actually scales overall as $\mathcal{O}(DM^2 + M^3)$ and when D is larger than M the term $\mathcal{O}(DM^2)$ dominates. For instance, in a dataset as MNIST (see Chapters 3 and 4) where $D = 784$ and $M = 500$ the optimization of the bound will roughly be of order $\mathcal{O}(M^3)$, while in other datasets with even slightly larger D, such as the CIFAR-10 dataset where $D = 3072$, the term $\mathcal{O}(DM^2)$ dominates; hence, resulting in slower training. Extremely large input dimesionality is typical in many applications of multi-label learning (Prabhu and Varma, 2014a) and in many cases can be larger than one million features, such as the WikiLSHTC dataset (Partalas et al., 2015) used in Chapter 4 where $D = 1617899$.

Learning the inducing inputs Z (as opposed to fixing them to a subset of training instances) is rather crucial in order to obtain good approximations, as was initially observed in GP regression (Titsias, 2009) but also more recently in non-Gaussian likelihoods such as in GP classification (Hensman et al., 2015b; Hernández-Lobato and Hernández-Lobato, 2016a). By optimizing over Z we are reducing the KL divergence between the approximate posterior and the exact posterior process (Matthews et al., 2016; Titsias, 2009) in a way that both the likelihood $p(\mathbf{y}|\mathbf{f})$ and the kernel function are taken into account. The benefit from optimizing Z can be even more profound in high dimensions where simple heuristics such as placing Z in a grid or setting it using clustering could be non-applicable or sub-optimal. Nevertheless, free-form gradient-based optimization over Z in high dimensions is challenging since at each step it requires computing gradients over DM parameters, and clearly when D is very large this becomes very expensive. To cope with this, we propose to restrict the gradient-based optimization over Z in a data-informed lower dimensional manifold or subspace.

Our key idea is to represent Z through the use of a precomputed basis so as the optimizable parameters in Z reduce from $\mathcal{O}(DM)$ to $O(\mathcal{R}M)$ where $\mathcal{R} \ll D$ (or $\mathcal{R} \sim \mathcal{O}(M)$) and the overall computations scale as $\mathcal{O}(M^2 R)$ (or $\mathcal{O}(M^3)$). We firstly consider the case of a linear kernel function $k(\mathbf{x}', \mathbf{x}) = \mathbf{x}'^\top \mathbf{x}$. Suppose we have precomputed a basis of R vectors stored as separate rows in matrix $\widetilde{X} \in \mathbb{R}^{R \times D}$. For instance, $\widetilde{X}$ can be obtained either by clustering the rows of X or by applying a matrix decomposition technique. In all our experiments in Section 4.5 of Chapter 4 where the proposed method is applied, we construct $\widetilde{X}$ using singular value decomposition, i.e. by computing the $\mathcal{R}$ right-singular vectors that correspond to the $\mathcal{R}$ largest singular values of X using the efficient subset singular-value decomposition (SVDs) algorithm Baglama and Reichel (2005). We then parametrize Z as

$$Z = A\widetilde{X}, \tag{2.26}$$

where $A \in \mathbb{R}^{M \times \mathcal{R}}$ is a real-valued matrix of tunable/variational parameters. This allows to construct Z so that each individual inducing input $\mathbf{z}_i \in \mathbb{R}^D$ is a linear combination of the basis vectors in $\widetilde{X}$ and where the weights in this combination are given by the i-th row of A. At each optimization step of the lower bound we need to compute the square kernel matrix K_Z and the cross kernel matrix $K_{X_b Z}$. We can compute K_Z as follows

$$K_Z^{\mathrm{L}} = ZZ^\top = A\widetilde{X}\widetilde{X}^\top A^\top = AK_{\widetilde{X}}A^\top, \tag{2.27}$$

where crucially the $\mathcal{R} \times \mathcal{R}$ matrix $K_{\widetilde{X}}$ can be precomputed and stored before the optimization starts, i.e. while such computation requires $\mathcal{O}(DM^2)$ time it needs to be performed only once. Then, any subsequent computation of K_Z and its gradient wrt A costs $\mathcal{O}(\mathcal{R}M^2)$. Similarly, the computation of the cross covariance matrix $K_{X_b Z}$ reduces to

$$K_{X_b Z}^{\mathrm{L}} = X_b Z^\top = (X_b \widetilde{X}^\top) A^\top = K_{X_b \widetilde{X}} A^\top, \tag{2.28}$$

where again the computation of $K_{X_b \widetilde{X}}$ can be done only once beforehand, i.e. by pre-computing the whole $N \times R$ matrix $K_{X\widetilde{X}}$ and then selecting for any mini-batch the corresponding block. Note also that for many datasets, as the majority of the multi-label classification datasets with extreme input dimensionality, X is a sparse matrix and therefore instead of keeping the full matrix $K_{X\widetilde{X}}$ in memory, we could alternatively perform the matrix multiplication $X_b \widetilde{X}^\top$ at each mini-batch optimization step with low computational cost by taking advantage of the sparsity of X_b. Thus, given that $|X_b| \sim \mathcal{O}(M)$ the computation of K_{X_bZ} and its gradients scales as $\mathcal{O}(\mathcal{R}M^2)$.

This representation trick can be also applied to any non-linear kernels, leaving the computational time exactly the same as in (2.27) and (2.28) since we only need to compute the euclidean distances between inputs. Therefore, by evaluating the $M \times M$ matrix D_Z as

$$D_Z = \mathrm{DG}_Z \mathbf{1}_M^\top + \mathbf{1}_M \mathrm{DG}_Z^\top - 2K_Z^{\mathrm{L}}, \tag{2.29}$$

where $\mathrm{DG}_Z \in \mathbb{R}^M$ includes the elements of the main diagonal of K_Z^{L} , and $\mathbf{1}_M$ is an M-dimensional column vector containing ones, we can easily build the kernel matrix K_Z. Similarly, the cross covariance matrix between a mini-batch of inputs X_b and Z can be computed using the $|X_b| \times M$ matrix D_{XZ}, defined as,

$$D_{XZ} = \mathrm{DG}_{X_b} \mathbf{1}_M^\top + \mathbf{1}_{|X_b|} \mathrm{DG}_Z^\top - 2K_{X_bZ}^{\mathrm{L}} \tag{2.30}$$

with $\mathrm{DG}_{X_b} \in \mathbb{R}^{|X_b|}$ being the main diagonal of the matrix $X_b X_b^\top$.

The matrix A can be initialized by the M centroids given by k-means with M clusters over the matrix $US \in \mathbb{R}^{N \times \mathcal{R}}$ where $U \in \mathbb{R}^{N \times \mathcal{R}}$ contains as columns the left-singular vectors of X and $S \in \mathbb{R}^{\mathcal{R} \times \mathcal{R}}$ is a diagonal matrix with the $\mathcal{R}$ largest singular values of X. Notice that both U and S are obtained by the singular-value decomposition of X, such that $US \approx X$, when we construct the basis $\widetilde{X}$. Moreover, in the case of using SVD, we can extra speed-up computations by the fact that $K_{\widetilde{X}}$ is equal to the $\mathcal{R}$-dimensional identity matrix due to the orthonormalization property of the eigenvectors collected in V.

Chapter 3

One-versus-Each Multi-class Classification using Gaussian Processes

3.1 Introduction

Having already discussed in Section 1.1.2 the challenges of applying Gaussian process models to multi-class classification tasks, such as non-conjugate likelihood intractability and scalability issues, and then discussing in Chapter 2 how those challenges can be addressed by a variational inference framework suitably tailored for sparse GPs (Section 2.3.2), in this chapter we introduce a scalable way to perform multi-class classification with Gaussian Processes using Variational Inference for arbitrarily large number of both classes and data points. The idea is based on an approximation of the softmax representation as it was presented in Titsias (2016). After that, there is an introduction to various methods that combine the two parametrizations discussed in Section 2.4 of Chapter 2 with different likelihoods. Lastly, a number of experiments on both real-world and synthetic datasets demonstrate the different attributes of each method and provide an insightful comparison with other baselines.

3.2 The One-vs-Each lower bound on the softmax

We start our discussion by introducing the derived lower bound and some of its properties as presented in Titsias (2016), since it involves the bedrock of the derivation of our multi-class method. More specifically, Titsias (2016) based on the observation that for any $\alpha_k \in \mathbb{R}_{\geq 0}$ with $k = 1, \cdots, K$ the following inequality holds,

$$1 + \sum_{k=1}^{K} \alpha_k \leq \prod_{k=1}^{K} (1 + \alpha_k), \tag{3.1}$$

derived a lower bound on the softmax function.

Let us first define the softmax function $\mathcal{S} : \{1, \cdots, K\} \times \mathbb{R}^K \to [0, 1]$ as

$$\mathcal{S}_k(f_1, \cdots, f_K) = \frac{e^{f_k}}{\sum_{\ell=1}^{K} e^{f_\ell}}, \tag{3.2}$$

where $f_1, \cdots, f_K \in \mathbb{R}$ are the score functions values and $k \in \{1, \cdots, K\}$. This is the multi-class generalization of the logit likelihood and it can be also seen as a mapping from $\mathbb{R}^K$ to the $(K-1)$-simplex, i.e. it defines how probable is the class k to happen, denoted by $p(k) = \mathcal{S}_k(f_1, \cdots, f_K)$ (see also Section 1.1.2 for an introduction to multi-class classification using the softmax function as likelihood). Hence, combining 3.1 and 3.2, we have the following lower bound

$$p(k) \geq \prod_{\ell \neq k} \sigma(f_k - f_\ell), \tag{3.3}$$

where $\sigma(t) = 1/(1 + e^{-t})$, the sigmoid function. Clearly, the above bound provides a computationally efficient way to approximate (3.2), since the presence of the product factorization circumvents the need of computing the cumbersome normalizing constant of softmax. Assume now a training dataset $\mathcal{D} = \{\mathbf{x}^{(i)}, y^{(i)}\}_{i=1}^N = (X, \mathbf{y})$ where $X \in \mathbb{R}^{N \times D}$ is the design matrix containing the D-dimensional feature vector $\mathbf{x}^{(i)}$ at its i^{th} and $\mathbf{y}$ is a N-dimensional discrete response vector where its i^{th} entry $y^{(i)} \in \{1, \cdots K\}$ labels the feature vector $\mathbf{x}^{(i)}$. The log likelihood of the data takes the following form,

$$\log \prod_{i=1}^{N} p(y^{(i)}) = \sum_{k=1}^{K} N_k \log p(y^{(i)}), \tag{3.4}$$

where N_k denotes the number of observations belonging in class k. If we now replace the score values $f_1, \cdots, f_K$ with parametrized[1] score functions $f_k(\mathbf{x}^{(i)}; \mathbf{w}_k)$ indexed by $\mathbf{x}^{(i)}$ and parametrized by a D-dimensional real vector of weights $\mathbf{w}_k$ (e.g. $f_k(\mathbf{x}) = \mathbf{w}_k^\top \mathbf{x}$), having as goal to obtain the optimal parameters by optimizing the likelihood, (3.4) can be written as

$$\log \prod_{i=1}^{N} p(y^{(i)}) = \sum_{i=1}^{N} \left[f_{y_i}(\mathbf{x}_i; \mathbf{w}_{y_i}) - \log \left(\sum_{k=1}^{K} e^{f_k(\mathbf{x}_i; \mathbf{w}_k)} \right) \right]. \tag{3.5}$$

[1]Later we generalize the notion of parametrized score functions using the Gaussian process framework.

We can notice that (3.5) can cope with large number of training data points via a stochastic optimization scheme. Nevertheless, large number of classes K can render the optimization procedure infeasible since we need to compute the normalization constant at each optimization step. Fortunately, the inequality in (3.1) lets us obtain a lower bound on the log likelihood which can effectively deal with both large number of data points and classes. Thus, we have

$$\begin{aligned}\log \prod_{i=1}^{N} p(y^{(i)}) &\geq \log \prod_{i=1}^{N} \prod_{\ell \neq y_i} \frac{1}{1 + e^{f_m(\mathbf{x}_i;\mathbf{w}_\ell) - f_{y_i}(\mathbf{x}_i;\mathbf{w}_{y_i})}} \\ &= -\sum_{i=1}^{N} \sum_{\ell \neq y_i} \log\left(1 + e^{f_\ell(\mathbf{x}_i;\mathbf{w}_\ell) - f_{y_i}(\mathbf{x}_i;\mathbf{w}_{y_i})}\right), \qquad (3.6)\end{aligned}$$

where the new bound is now a double summation over both data points and labels which paves the way for a doubly stochastic approximation scheme. Notice also that each term in the sum depends on the pairwise score difference $f_\ell(\mathbf{x}_i;\mathbf{w}_\ell) - f_{y_i}(\mathbf{x}_i;\mathbf{w}_{y_i})$. Last but not least, Titsias (2016) demonstrated the superiority of that lower bound when compared to the Bouchard's bound Bouchard (2007), another bound on softmax in (3.2), which was employed for multi-class classification by applying variational inference. Additionally, he provided compelling evidence that the former bound achieves better performance than the latter one in practice while the latter cannot achieve scalability of the magnitude of the former.

Having derived the lower bound of Titsias and justified its usefulness, we are now ready to move from the parametrized score functions $f_k(\mathbf{x}_i;\mathbf{w}_k)$ to general functions $f_k(\cdot)$, where the use of the Gaussian process framework emerges naturally. This generalization of Titsias' work is one of our main contributions in this thesis.

3.3 From Parametric to non-parametric score functions

In order to deal with distributions on infinite-dimensional objects, like arbitrary functions f_k, we introduce K (zero-mean) GP priors, one for each class, $f_k \sim \mathcal{GP}\left(0, \mathbf{k}\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)\right)$ where $\mathbf{k}\left(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}\right)$ is a common[2] kernel function parametrized by a vector of hyperparameters $\boldsymbol{\theta}$ although for notational simplicity, we omit including it in the notation. We also denote by $\mathbf{f}_k \in \mathbb{R}^N$, the N-dimensional vector including all the function values of $f_k(\mathbf{x})$ evaluated at all training inputs X, while $\mathbf{f}^{(i)} \in \mathbb{R}^K$ denotes all the function values of each $f_k(\cdot)$ evaluated at data point $\mathbf{x}^{(i)}$. The full vector $\mathbf{f}_k$ follows the Gaussian prior

[2]We can easily define different kernels for each class or a common kernel with different hyperparameters for each class. We avoid it in our definitions just to keep the notation uncluttered.

$$p(\mathbf{f}_k) = \mathcal{N}(\mathbf{f}_k | \mathbf{0}, K_X), \tag{3.7}$$

where K_X is the $N \times N$ covariance matrix, evaluated at the training inputs X defined in previous section. In GP multi-class, we parametrize each likelihood term $p(y^{(i)}|\mathbf{f}^{(i)})$ following the definition we gave in 3.2, i.e.

$$p(y^{(i)}|\mathbf{f}^{(i)}) = \frac{e^{f_k^{(i)}}}{\sum_{m=1}^{K} e^{f_m^{(i)}}}. \tag{3.8}$$

The joint density of observed class labels (given their inputs) and the latent function values is given by

$$p(\mathbf{y}, \mathbf{f}_1, \ldots, \mathbf{f}_K) = \left(\prod_{i=1}^{n} p(y^{(i)}|\mathbf{f}^{(i)}) \right) \prod_{k=1}^{K} p(\mathbf{f}_k). \tag{3.9}$$

An equivalent way to write this joint density is to re-arrange the product over the likelihood factors based on the class labels, so that

$$p(\mathbf{y}, \mathbf{f}_1, \ldots, \mathbf{f}_K) = \prod_{k=1}^{K} \left[\left(\prod_{i \in \mathcal{N}_k} p(k|\mathbf{f}^{(i)}) \right) p(\mathbf{f}_k) \right], \tag{3.10}$$

where $\mathcal{N}_k = \{i | y_i = k\}$ indicates the subset of data points that belong to k^{th} class. Eq. (3.10) implies that we could train the model by stochastic mini-batch training over classes (since the log of the joint would be written as a sum over labels) in the same manner as it has been proposed in Hoffman et al. (2013). Unfortunately, as we pointed out in Section 3.2, this cannot be realized in practice due to the softmax individual likelihood factors $p(k|\mathbf{f}^{(i)})$ that couple the latent values $\mathbf{f}^{(n)}$ across all K classes. This is where the One-versus-each bound in (3.1) comes into play and help us to overcome softmax's computational impediment. In the next two sections, we develop a stochastic variational inference framework that can deal with both very large number of classes (see section 3.4) and very large number of data points (see section 3.5).

3.4 Scalable variational inference over class labels

The first ingredient of our method is the one-vs-each bound on the softmax given by (3.1)

$$p(y^{(i)}|\mathbf{f}^{(i)}) \geq \prod_{\ell \neq y^{(i)}} \sigma\left(f^{(i)}_{y^{(i)}} - f^{(i)}_{\ell}\right) \triangleq \widetilde{p}(y^{(i)}|\mathbf{f}^{(i)}). \tag{3.11}$$

As we shall see shortly, this lower bound allows us to introduce what we call *stochastic partial class updates* where the variational parameters of just two (or very few) classes can be updated in a single stochastic optimization step. If we plug in the above lower bound in (3.10) we obtain

$$\begin{aligned}&\prod_{k=1}^{K}\left[\left(\prod_{i\in\mathcal{N}_k}\prod_{\ell\neq k}\sigma(f^{(i)}_k - f^{(i)}_\ell)\right)p(\mathbf{f}_k)\right] = \\ &\prod_{k=1}^{K}\left[\prod_{\ell\neq k}\left(\prod_{i\in\mathcal{N}_k}\sigma(f^{(i)}_k - f^{(i)}_\ell)\right)p(\mathbf{f}_k)\right],\end{aligned} \tag{3.12}$$

where crucially we have re-arranged the order of the products $\prod_{n\in\mathcal{N}_k}$ and $\prod_{\ell\neq k}$. This bound has a product form that could lead to scalable training over classes. At this point, we introduce a variational distribution, based on the mean-field variational family (see Section 2.2.2 of Chapter 2) with density given by

$$Q = \prod_{k=1}^{K} q(\mathbf{f}_k), \tag{3.13}$$

where each factor $q(\mathbf{f}_k)$ is a N-dimensional multivariate Gaussian. We omit to specify the parametrization of each $q(\mathbf{f}_k)$ since it would not affect the derivation process of the the lower bound that we present. Similar to Section 2.2.1 of Chapter 2, we would like to approximate the true posterior

$$\widetilde{P} = p(\{\mathbf{f}_k\}_{k=1}^{K}|\mathbf{y}), \tag{3.14}$$

by the variational density Q in (3.13), where $\{\mathbf{f}_k\}_{k=1}^{K}$ concisely denotes NK random variables. This can be achieved by minimizing the KL-divergence $\mathrm{KL}[Q \;||\; \widetilde{P}]$, which leads to a lower bound on the log marginal likelihood $p(\mathbf{y})$, as we showed in Chapter 2, Section 2.2.1, and combined with the re-arrangement trick in (3.12) takes the following form

$$\begin{aligned}&\mathbb{E}_Q\left[\log\frac{\prod_{k=1}^{K}\left[\prod_{\ell\neq k}\left(\prod_{i\in N_k}\sigma(f^{(i)}_k - f^{(i)}_\ell)\right)p(\mathbf{f}_k)\right]}{\prod_{k=1}^{K} q(\mathbf{f}_k)}\right] \\ &= \sum_{k=1}^{K}\sum_{\ell\neq k}\sum_{i\in N_k}\mathbb{E}_Q\left[\log\sigma(f^{(i)}_k - f^{(i)}_\ell)\right] - \sum_{k=1}^{K}\mathbb{E}_Q\left[\log\frac{q(\mathbf{f}_k)}{p(\mathbf{f}_k)}\right].\end{aligned} \tag{3.15}$$

Notice this bound is a looser bound than the usual ELBO on $p(\mathbf{y})$ since it involves the One-vs-Each approximation on the data likelihood $p(y^{(i)}|\mathbf{f}^{(i)})$. Since each Q is given by (3.13), each term in the single sum simplifies to become an expectation over $\mathbf{f}_k$,

$$\mathbb{E}_Q\left[\log\frac{q(\mathbf{f}_k)}{p(\mathbf{f}_k)}\right]=\mathbb{E}_{q(\mathbf{f}_k)}\left[\log\frac{q(\mathbf{f}_k)}{p(\mathbf{f}_k)}\right], \tag{3.16}$$

which is precisely the KL divergence $\mathrm{KL}[q(\mathbf{f}_k)||p(\mathbf{f}_k)]$. Regarding each i^{th} term in the first nested sum, we first observe that

$$\log\sigma(f_k^{(i)}-f_\ell^{(i)})=-\log(1+e^{-(f_k^{(i)}-f_\ell^{(i)})}), \tag{3.17}$$

where each $f_k^{(i)}$ $\forall k$, is a scalar random variable that under Q follows the univariate Gaussian distribution

$$q(f_k^{(i)})=\mathcal{N}(f_k^{(i)}\mid m_k^{(i)},s_k^{(i)}). \tag{3.18}$$

For computing all the moments of this normal distribution we can refer to Appendix A.1. Since we only care for the difference $f_{k-\ell}^{(i)}=f_k^{(i)}-f_\ell^{(i)}$, the variational density takes the following form $q(f_{k-\ell}^{(i)})=\mathcal{N}(f_{k-\ell}^{(i)}\mid m_k^{(i)}-m_\ell^{(i)},s_k^{(i)}+s_\ell^{(i)})$, where only four scalars are needed to define it. The whole data reconstruction term in (3.15) simplifies to

$$-\sum_{k=1}^{K}\sum_{\ell\neq k}\sum_{i\in\mathcal{N}_k}\mathbb{E}_{q(f_{k-\ell}^{(i)})}\left[\log(1+e^{-f_{k-\ell}^{(i)}})\right]. \tag{3.19}$$

Plugging now (3.16) and (3.19) in (3.15) gives us the final lower bound

$$-\sum_{k=1}^{K}\left[\sum_{\ell\neq k}\sum_{i\in\mathcal{N}_k}\mathbb{E}_{q(f_{k-\ell}^{(i)})}[\log(1+e^{-f_{k-\ell}^{(i)}})]+\mathrm{KL}[q(\mathbf{f}_k)||p(\mathbf{f}_k)]\right]. \tag{3.20}$$

A stochastic unbiased estimate of the lower bound (3.20) can be obtained by randomly sub-sampling one class $k\in\{1,\dots,K\}$ and a remaining class $\ell\in\{1,\dots,K\}\backslash\{k\}$ so that

$$-K\left[(K-1)\sum_{i\in\mathcal{N}_k}\mathbb{E}_{q(f_{k-\ell}^{(i)})}[\log(1+e^{-f_{k-\ell}^{(i)}})]+\mathrm{KL}[q(\mathbf{f}_k)||p(\mathbf{f}_k)]\right], \tag{3.21}$$

is an unbiased stochastic estimate of (3.20). This has the interesting property that depends solely on the variational factors $q(\mathbf{f}_k)$ and $q(\mathbf{f}_\ell)$ associated with two different classes. Thus, a stochastic gradient ascent step only requires computing and updating these two factors. Furthermore, it is worth mentioning that the expected value $\mathbb{E}_{q(f_{k-\ell}^{(i)})}\left[\log\left(1+e^{-f_{k-\ell}^{(i)}}\right)\right]$ in the summation of (3.21) can be accurately approximated using Gauss-Hermite quadrature (Liu and Pierce, 1994) since the expectation is a univariate integral which is much easier than the high dimensional integrals encountered in non-conjugate marginal likelihoods. Another way to approximate such integrals and their gradients would be by using unbiased stochastic estimates, e.g. a naive Monte Carlo estimate that samples from $q(f_{k-\ell}^{(i)})$ as suggested in Titsias and Lázaro-Gredilla (2011). In practice though, we found that Gauss-Hermite quadrature performs well, so we adhered to that approximation method.

It is straightforward to extend the above stochastic estimate to include more samples, e.g. from the set of remaining classes, or use importance sampling to reduce variance in class imbalanced problems; however, we did not conduct any research on that direction in this thesis. Finally, every time that a computation of (3.21) and its gradients is required, we have to encounter the adverse $\mathcal{O}(N^3)$ scaling. Given that burden, in the next section, we present a scalable way to deal with it effectively.

3.4.1 Partial class updates in a point estimation setting

Partial class updates can be also applied in a simple point estimate setting, such as regularized/penalized maximum likelihood. Specifically, considering the parametric score functions introduced in section 3.3, let the functions $f_k(\mathbf{x})$ be not random but parametrized according to $f_k(\mathbf{x}) = \mathbf{w}_k^T\boldsymbol{\phi}(\mathbf{x})$ where $\boldsymbol{\phi}(\mathbf{x})$ is a fixed deterministic feature vector (e.g. in the simplest case $\boldsymbol{\phi}(\mathbf{x}) = \mathbf{x}$), and $\mathbf{w}_k$ are class-specific parameters. Each $\mathbf{w}_k$ is assigned a prior, e.g. $\log p(\mathbf{w}_k) = -\frac{\lambda}{2}||\mathbf{w}_k||^2 + const$. By using the one-vs-each lower bound on the softmax likelihood we obtain the following lower bound on the regularized log likelihood

$$-\sum_{k=1}^{K}\left[\sum_{\ell\neq k}\sum_{i\in\mathcal{N}_k}\log(1+e^{-(\mathbf{w}_k-\mathbf{w}_\ell)^T\boldsymbol{\phi}(\mathbf{x}_i)}) + \frac{\lambda}{2}||\mathbf{w}_k||^2\right]. \tag{3.22}$$

An unbiased stochastic estimate can be obtained by sub-sampling one class k and a remaining class ℓ exactly as done previously. Further, it is also possible to subsample mini-batches of training instances, i.e. to pick $\mathcal{B}_k \subseteq \mathcal{N}_k$ so that $\mathcal{B}_k$ is a subset of data all belonging to the subsampled class k, and then construct the unbiased estimate

$$- K \left[\frac{(K-1)|\mathcal{N}_k|}{|\mathcal{B}_k|} \sum_{i \in \mathcal{B}_k} \log \left(1 + e^{-(\mathbf{w}_k - \mathbf{w}_\ell)^T \boldsymbol{\phi}(\mathbf{x}_i)} \right) + \frac{\lambda}{2} ||\mathbf{w}_k||^2 \right]. \tag{3.23}$$

This leads to a partial class stochastic step, where only two parameter vectors $(\mathbf{w}_k, \mathbf{w}_\ell)$ are updated in a single gradient ascent iteration. 3.23 is exactly the method used for the experiments in Titsias (2016) and we refer to this as OvsE-lin method in the next of this chapter. The implementation of the above scheme is straightforward.

3.5 Scalable variational inference over data points

To deal with extreme number of training data we consider the variational inference method based on inducing variables as described in Chapter 2, section 2.3.2, we use similar derivation steps as Titsias (2009), Hoffman et al. (2013), and Hensman et al. (2015b) to achieve that. More specifically, for each latent function, we introduce a vector $\mathbf{u}_k \in \mathbb{R}^M$ of inducing variables. We assume a separate set of inducing inputs $Z_k \in \mathbb{R}^{M \times D}$ for each class k where M is the number of inducing inputs per class so that

$$p(\mathbf{u}_k) = \mathcal{N}(\mathbf{u}_k | \mathbf{0}, K_{Z_k}), \tag{3.24}$$

and K_{Z_k} is the $M \times M$ kernel matrix obtained by evaluating the kernel function at the inducing inputs Z_k. By expanding now the joint density with the inducing variables we have

$$\prod_{k=1}^{K} \left[\left(\prod_{i \in \mathcal{N}_k} p(k|\mathbf{f}^{(i)}) \right) p(\mathbf{f}_k|\mathbf{u}_k) p(\mathbf{u}_k) \right], \tag{3.25}$$

where $p(\mathbf{f}_k|\mathbf{u}_k)$ is the standard conditional GP prior and can be derived by using the Gaussian identity described in A.1. Thus, we have

$$p(\mathbf{f}_k|\mathbf{u}_k) = \mathcal{N}\left(\mathbf{f}_k | K_{XZ_k} K_{Z_k}^{-1} \mathbf{u}_k, K_X - K_{XZ_k} K_{Z_k}^{-1} K_{Z_k X}\right), \tag{3.26}$$

where K_{XZ_k} is the cross-covariance matrix between the training inputs X and the inducing inputs Z_k and $K_{Z_k X} = K_{XZ_k}^T$. For notational convenience we write $\mathbf{f} = \{\mathbf{f}_k\}_{k=1}^K$ and $\mathbf{u} = \{\mathbf{u}_k\}_{k=1}^K$. By employing now as density of the variational distribution the following

$$\hat{Q}(\mathbf{f}, \mathbf{u}) = \prod_{k=1}^{K} p(\mathbf{f}_k|\mathbf{u}_k)\hat{q}(\mathbf{u}_k), \tag{3.27}$$

where $p(\mathbf{f}_k|\mathbf{u}_k)$ is the conditional GP prior and $\hat{q}(\mathbf{u}_k)$ is the density of M-dimensional Gaussian variational distribution over the inducing variables for class k, a bound on the $p(\mathbf{y})$ can be also derived using the same arguments as in previous section. Notice that we added a ˆ symbol over each augmented distribution to differentiate them with original distributions of the previous section. We keep this notation through the rest of this chapter. The augmented posterior is now written as

$$\hat{P}(\mathbf{f}, \mathbf{u}) = p(\{\mathbf{f}_k, \mathbf{u}_k\}_{k=1}^{K}|\mathbf{y}). \tag{3.28}$$

The minimization of KL$[\hat{Q} \mid\mid \hat{P}]$ gives a similar lower bound as in (3.20), but the KL-divergence terms are this time between $\hat{q}(\mathbf{u}_k)$ and $p(\mathbf{u}_k)$ and the marginal means and variances are calculated given the information of the inducing variables. More specifically, the corresponding lower bound now is

$$\begin{aligned}
&\mathbb{E}_{\hat{Q}}\left[\log \frac{\prod_{k=1}^{K}\left[\prod_{\ell\neq k}\left(\prod_{i\in N_k}\sigma(f_k^{(i)} - f_\ell^{(i)})\right)p(\mathbf{f}_k|\mathbf{u}_k)p(\mathbf{u}_k)\right]}{\prod_{k=1}^{K} p(\mathbf{f}_k|\mathbf{u}_k)\hat{q}(\mathbf{u}_k)}\right] \\
&= \sum_{k=1}^{K}\sum_{\ell\neq k}\sum_{i\in N_k}\mathbb{E}_{\hat{Q}}\left[\log\sigma(f_k^{(i)} - f_\ell^{(i)})\right] - \sum_{k=1}^{K}\mathbb{E}_{\hat{Q}}\left[\log\frac{\hat{q}(\mathbf{u}_k)}{p(\mathbf{u}_k)}\right],
\end{aligned} \tag{3.29}$$

which can be further written, using the same derivation steps of the previous section, as

$$-\sum_{k=1}^{K}\left[\sum_{\ell\neq k}\sum_{i\in N_k}\mathbb{E}_{\hat{q}(f_{k-\ell}^{(i)})}[\log(1 + e^{-f_{k-\ell}^{(i)}})] + \text{KL}[\hat{q}(\mathbf{u}_k)||p(\mathbf{u}_k)]\right]. \tag{3.30}$$

Despite its resemblance with (3.20), a different process must be followed in order to calculate the marginal means and variances of the one-dimensional Gaussian distributions $\hat{q}(f_{k-\ell}^{(i)})$ in (3.30). We firstly need to marginalize out the inducing variables for class k, i.e.

$$\hat{q}(\mathbf{f}_k) = \int p(\mathbf{f}_k|\mathbf{u}_k)\hat{q}(\mathbf{u}_k)d\mathbf{u}_k, \tag{3.31}$$

where this integral is analytically tractable due to a common-used Gaussian identity (A.1). Thus, the density of marginal variational distribution is $\hat{q}(f_k^{(i)})$ is Gaussian so we have to calculate its scalar mean and variance. As usual, in order to compute its moments we need to predefine the form of the variational parametrization. For instance, given a FP, the density of the variational distribution of class k is written as,

$$\hat{q}(\mathbf{u}_k) = \mathcal{N}(\mathbf{u}_k|\mathbf{m}_k, S_k = L_k L_k^\top). \tag{3.32}$$

where L_k is a $M \times M$ lower triangular matrix. Therefore, each $m_k^{(i)}$ and $s_k^{(i)}$ are given by

$$\begin{aligned} m_k^{(i)} &= Q_{\mathbf{x}^{(i)}Z_k}\mathbf{m}_k, & (3.33)\\ s_k^{(i)} &= k(\mathbf{x}^{(i)}, \mathbf{x}^{(i)}) - Q_{\mathbf{x}^{(i)}Z_k}\mathbf{k}(Z_k, \mathbf{x}^{(i)}) + R_{\mathbf{x}^{(i)}Z_k}R_{\mathbf{x}^{(i)}Z_k}^\top, & (3.34) \end{aligned}$$

where $Q_{\mathbf{x}^{(i)}Z_k} = \mathbf{k}(\mathbf{x}^{(i)}, Z_k)K_{Z_k}^{-1}$ and $R_{\mathbf{x}^{(i)}Z_k} = Q_{\mathbf{x}^{(i)}Z_k}L_k$ while equations (3.33) and (3.34) are derived using a known Gaussian formula that can be found in A.1. The defined parametrization allows us also to calculate the KL terms in (3.30), which can be obtained by,

$$\begin{aligned} \mathrm{KL}[\hat{q}(\mathbf{u}_k) \mid\mid p(\mathbf{u}_k)] &= \int \hat{q}(\mathbf{u}_k) \log \frac{\hat{q}(\mathbf{u}_k)}{p(\mathbf{u}_k)} d\mathbf{u}_k \\ &= \frac{1}{2}\left(\mathbf{m}_k^\top K_{Z_k}^{-1}\mathbf{m}_k + \mathrm{tr}\left(K_{Z_k}^{-1}S_k\right) + \log|K_{Z_k}| - \log|S_k| - M\right). \end{aligned} \tag{3.35}$$

Using the same arguments, we can derive the corresponding quantities of interest for the PP as we similarly did in equations (2.23) and (2.24) (see Section 2.4 of Chapter 2 for a more detailed discussion). We would like to mention that the PP combined with the factorized variational density $\hat{Q}$ can recover the covariance matrix of the optimal variational distribution as described in Opper and Archambeau (2009) and discussed in Chapter 2, Section 2.4. We do not provide the proof here as it is given in the next Chapter in Section 4.4.6 in the more general multi-label framework.

Having derived all the required quantities for the computation of the bound, we can now demonstrate the corresponding unbiased stochastic estimate of (3.30) in order to render computations feasible in cases of large-scale datasets. That is

$$-K\left[\frac{(K-1)|\mathcal{N}_k|}{|\mathcal{B}_k|}\sum_{i\in\mathcal{B}_k}\mathbb{E}_{\hat{q}(f_{k-\ell}^{(i)})}[\log(1+e^{-f_{k-\ell}^{(i)}})] + \mathrm{KL}[\hat{q}(\mathbf{u}_k)||p(\mathbf{u}_k)]\right]. \tag{3.36}$$

Hence, the whole derivation steps led us to a quantity in (3.36) that can be fully scalable in terms of both number of class and data points since it is up to us to predefine the (i) number of inducing inputs, (ii) the size of the mini-batch, and (iii) the number of the sub-sampled classes. The computational complexity of this method is discussed thoroughly in Section 3.7 later.

Having maximized the bound 3.36, we can use both the optimized hyperparameters and variational parameters to make predictions on hold out data. For instance, if we consider a single novel data point $\mathbf{x}^{(*)}$, the predictive distribution of $y^{(*)} = k$, for some $k \in \{1, \cdots K\}$ can be approximated by

$$\widetilde{p}(y^{(*)} = k|\mathbf{y}) \approx \int \widetilde{p}(y^{(*)} = k|\mathbf{f}^{(*)})\hat{q}(\mathbf{f}^{(*)})d\mathbf{f}^{(*)}, \tag{3.37}$$

where, as before, the K-dimensional vector $\mathbf{f}^{(*)}$ corresponds to the latent function values of each class evaluated at $\mathbf{x}^{(*)}$ and the variational density $\hat{q}(\mathbf{f}^{(*)})$ is calculated by marginalizing out the inducing variables $\mathbf{u}$ of $\hat{Q}$ in (3.27), i.e.

$$\hat{q}(\mathbf{f}^{(*)}) = \int \hat{Q}(\mathbf{f}, \mathbf{u})d\mathbf{u}. \tag{3.38}$$

Despite the fact that the above integral has a closed form solution (see A.1), the integral in (3.37) is not analytically tractable so we approximate it by Monte Carlo integration where we draw samples[3] from the K-dimensional isotropic Normal distribution where its mean vector and covariance matrix can be computed by using equations (3.33) and (3.34) for the FP while the case of PP is similarly derived as in Section 2.4. As we shall see later, the described One-vs-Each approximation with GPs can achieve very promising results given its approximation nature.

3.6 Another surrogate for softmax

Another choice of modelling $p(y^{(i)}|\mathbf{f}^{(i)})$, instead of the softmax function in 3.8, is through the robust max function (Girolami and Rogers (2006); Hernández-Lobato et al. (2011); Kim and Ghahramani (2003)) which gives the following likelihood,

$$p(y^{(i)}|\mathbf{f}^{(i)}) = \begin{cases} 1-\epsilon, & \text{if } y^{(i)} = \operatorname{argmax}_k \mathbf{f}^{(i)} \\ \frac{\epsilon}{K-1}, & \text{otherwise.} \end{cases} \tag{3.39}$$

This likelihood translates as choosing class $y^{(i)}$ with probability $1-\epsilon$ if the maximum value of the K-dimensional score vector $\mathbf{f}^{(i)}$ is achieved at its $y^{(i)}$-th entry. If not then with probability ϵ, we choose randomly one of the remaining $K-1$ classes. The parameter ϵ can be seen as a robustness threshold to outliers.

[3]In all experiments later, we draw no more than 1000 samples.

Firstly we need to adjust this likelihood in the variational sparse GP framework we presented before. Following the same derivation steps as in Sections 3.4 and 3.5, we find that the ELBO can be expressed as

$$\sum_{i=1}^{N} \int \hat{q}(f_i) \log p(y_i|f_i) df_i - \sum_{k=1}^{K} \mathrm{KL}[\hat{q}(\mathbf{u}_k)\ ||p(\mathbf{f}_k)], \tag{3.40}$$

where the variational density $\hat{Q}$ is identical with (3.27). The main reason that makes the robust max function appealing is the fact that all the one-dimensional variational expectations with respect to $\hat{q}(f_i)$ that appear in (3.40) can be easily computed by one dimensional quadrature and the standard normal cumulative distribution function. Finally, as Girolami and Rogers (2006) argue, the multinomial probit behaviour can be guaranteed by adding extra Gaussian noise to the latent functions.

Once again, when it comes to predicting a new test point we need to approximate the same integral as in (3.37). Nevertheless, we can omit employing Monte Carlo integration since there is a more algorithmically elegant way to compute this quantity as it is presented in Hernández-Lobato et al. (2011), where the K-dimensional integral can be reduced to an one-dimensional integral. Thus, we have

$$\begin{aligned} p(y^{(*)} = k|\mathbf{y}) &= \int p(y^{(*)} = k|\mathbf{f}^{(*)}) q(\mathbf{f}^{(*)}) d\mathbf{f}^{(*)} \\ &= (1-\epsilon)\, \mathbb{E}_{\mathcal{N}(f_k^{(*)})|m_k^{(*)}, s_k^{(*)})} \left[\prod_{\ell \neq k} \Phi\left(\frac{f_k^{(*)} - m_\ell^{(*)}}{\sqrt{s_\ell^{(*)}}} \right) \right], \end{aligned} \tag{3.41}$$

where $\Phi(\cdot)$ is the standard normal cumulative distribution function and $m_l^{(*)}, s_l^{(*)}$ with $l \in \{1, \cdots K\}$ are the variational means and variances, respectively, of each class l. Similarly, as discussed in 3.4, all the moments can be computed exactly, given the chosen variational distribution parametrization (see also Chapter 2 Section 2.4). The final one-dimensional integral is now able to be effectively approximated by Gauss-Hermite quadrature.

This likelihood has been successfully applied to multi-class classification on a Variational Sparse Gaussian Processes framework (Matthews, 2017; Hensman et al., 2015a) and thus, as we shall see in the Experiments Section later, it is our main baseline comparison with our proposed One-versus-Each method of Section 3.5.

3.7 A collection of multi-class classification methods

Having derived previously (Section 3.5) an approximation method for multi-class classification and introducing an alternative way to parametrize the variational density $\hat{q}(\mathbf{u})$ (Section 2.4), we are able to "generate" six different multi-class classification methods that they are probed meticulously in Section 3.8. All those six methods have emerged by combining two different likelihoods, the robustmax and One-vs-Each one, with the two variational density's parametrizations while two of methods have also considered the case of a shared set of inducing inputs across all K different classes. We present now the following methods:

OvsE-p: This is the One-versus-Each method described in Section 3.5 where each variational density q_k is parametrized by the parsimonious scheme we introduced in Section 2.4.

OvsE-f: Same method as above; however, each variational density q_k is parametrized by $\mathcal{O}(M^2)$ parameters.

OvsE-p-sh: Here we use the One-versus-Each method with PP of the variational distributions. Nevertheless the set of inducing inputs Z is shared across all classes while the hyperparameters set is different for each of them.

OvsE-f-sh: The One-versus-Each method is combined with the FP of the variational distributions now. Both the set of inducing inputs Z and hyperparameters are common for all classes.

RM-p: This is where we use the robustmax function accompanied with the PP. In the robustmax case, we always use shared inducing inputs and hyperparameters as well. This is a combination of our work and Matthews (2017); Hensman et al. (2015a).

RM-f: Lastly, the robustmax function is used in tandem with the FP as it was introduced in Matthews (2017); Hensman et al. (2015a).

As we mentioned before, all of the above methods are employed on a Variational Inference framework suitably designed for Sparse Gaussian Process models. Thus, we can apply the theory presented in Chapter 2, Section 2.4 to each of the methods for evaluating their time complexity needed for each optimization step of their corresponding lower bounds (and their derivatives). More specifically, for the One-vs-Each based methods (i.e. OvsE-p, OvsE-f, OvsE-p-sh, and OvsE-f-sh) we need to perform two Cholesky decompositions, one for the randomly chosen class k and one for the other class ℓ on the corresponding kernel matrices K_{Z_k} and K_{Z_ℓ}, consequently, which scales as $\mathcal{O}(M^3)$. This is enough for computing the KL term $\mathrm{KL}[q(\mathbf{u}_k)||p(\mathbf{u}_k)]$. For calculating the one-dimensional integrals due to the expectation terms in (3.30), we only need to compute the marginal mean and

Method	Time Complexity	Storage
OvsE-p	$\mathcal{O}(M^3 + \lvert\mathcal{B}\rvert M^2)$	$\mathcal{O}(KMD)$
OvsE-f	$\mathcal{O}(M^3 + \lvert\mathcal{B}\rvert M^2)$	$\mathcal{O}(KM^2 + KMD)$
OvsE-p-sh	$\mathcal{O}(M^3 + \lvert\mathcal{B}\rvert M^2)$	$\mathcal{O}(KM + MD)$
OvsE-f-sh	$\mathcal{O}(M^3 + \lvert\mathcal{B}\rvert M^2)$	$\mathcal{O}(KM^2 + MD)$
RM-p	$\mathcal{O}(KM^3 + K\lvert\mathcal{B}\rvert M^2)$	$\mathcal{O}(KM + MD)$
RM-f	$\mathcal{O}(M^3 + K\lvert\mathcal{B}\rvert M^2)$	$\mathcal{O}(KM^2 + MD)$

Table 3.1: Time complexity and storage requirements for each of the described methods in Section 3.7. M is the number of inducing inputs, K the number of classes, and $|\mathcal{B}|$ the mini-batch size.

variance evaluated at each data point $\mathbf{x}^{(i)}$, and then just apply Gauss-Hermite quadrature using a few[4] Gauss Hermite quadrature points. This can be done in $\mathcal{O}(M^3)$, therefore, for equally sized mini-batches across all classes, i.e. $|\mathcal{B}| = |\mathcal{B}_k|$, $\forall\, k$, we need $\mathcal{O}(M^3+|\mathcal{B}|M^2)$. By choosing a mini-batch size such that $|\mathcal{B}| \sim \mathcal{O}(M)$, the overall time complexity scales as $\mathcal{O}(M^3)$.

With a similar argument, but using different mathematical expressions, we can show that the time complexity appears in RM-f is $\mathcal{O}(M^3+K|\mathcal{B}|M^2)$. Notice now, that we need to compute the marginal means and variances for all K Gaussian process instead of just two as before, for a specific data point $\mathbf{x}^{(i)}$. This is a direct consequence of abandoning the One-vs-Each approximation in favour of the exact[5] likelihood. Essentially, this means that in case of large number of classes, the term $K|\mathcal{B}|M^2$ dominates and renders the method computationally prohibitive, unless we dramatically reduce the mini-batch size which inevitably increases the variance of the stochastic estimates of ELBO. Nevertheless, the RM-p method is the one that scales the worst since it requires $\mathcal{O}(KM^3 + K|\mathcal{B}|M^2)$. The main culprit of this adverse computational time is the sparse parametrization of the variational distribution since K Cholesky decompositions are calculated this time for computing the ELBO. All the computational time complexities of the methods accompanied by their corresponding space complexities are summarized in Table 3.1. Lastly, we would like to mention here that an extra bias term $b_k \in \mathbb{R}$ is added to each of the K latent functions to increase model's flexibility for all the above methods and they are treated as variational parameters during the optimization of the lower bound.

[4]Practically, in all experiments we found that a number of ten Gauss Hermite quadrature points suffices to approximate those integrals accurately

[5]Exact in the sense that we do not need to perform any approximation for computing it. However it does not coincide with the exact likelihood of One-vs-Each which of course is softmax.

3.8 Experiments

3.8.1 Datasets

We carry out experiments on a world-dataset which has been widely used in the machine learning field, the MNIST dataset LeCun et al. (1998). The MNIST dataset has been employed in several classification models, LeCun et al. (1998); Keysers et al. (2007); Romanuke (2016), providing us useful comparisons for a high-dimensional[6] dataset. This is the main reason we deploy it here for our experiments. Finally, a 2-dimensional synthetic dataset was also employed for emphasizing the superiority of the non-parametric (Section 3.3) over the parametric (Section 3.4.1) One-vs-Each method.

Regarding the datasets, MNIST consists of 28×28 grey scale images depicting a digit from zero to nine. It is split in the standard training and test datasets, with 60000 and 10000 data points respectively. Further, to make classification feasible, we deskewed the images of MNIST as it is typical for this dataset (LeCun et al., 1998). Finally, the synthetic data is highly non-linear dataset in 2 dimensions. More specifically, to generate the dataset, we considered 5 different classes where the data points of each class lies in the circumference of the corresponding circle of a given radius. We also add independent noise for each of the data points. Each class has in total 1500 data points where 1000 of them are used for training and the rest of them for testing performance.

Finally, we would like to mention that all the experiments are conducted by employing our own implementation for any Gaussian-Process-based model. This was coded up from scratch in Python, combined with the open-source software library, Tensorflow (Abadi et al., 2016), due to its highly efficient automatic differentiation tools and optimizers as well. Moreover, many parts of the code have been based on the excellent work of the highly-efficient Tensorflow-based python package, GPflow (Matthews et al., 2017). A code snippet based on Tensorflow that implements the lower bound in (3.36) using the PP can be found in C.2 of Appendix.

3.8.2 Synthetic data

A depiction of the described dataset can be found in Figure 3.1. For this dataset, we test the OvsE-lin and OvsE-p methods, and then we evaluate their performance.

Unsurprisingly, the point estimation method utterly fails on this dataset, giving an extremely high error rate of 67.76%; even greater than just predicting the class labels randomly. On the other hand, the non-parametric case manages to achieve

[6] We shall see that the perspective of a high dimensional dataset we espouse in this chapter is way different than the one in Chapter 4.

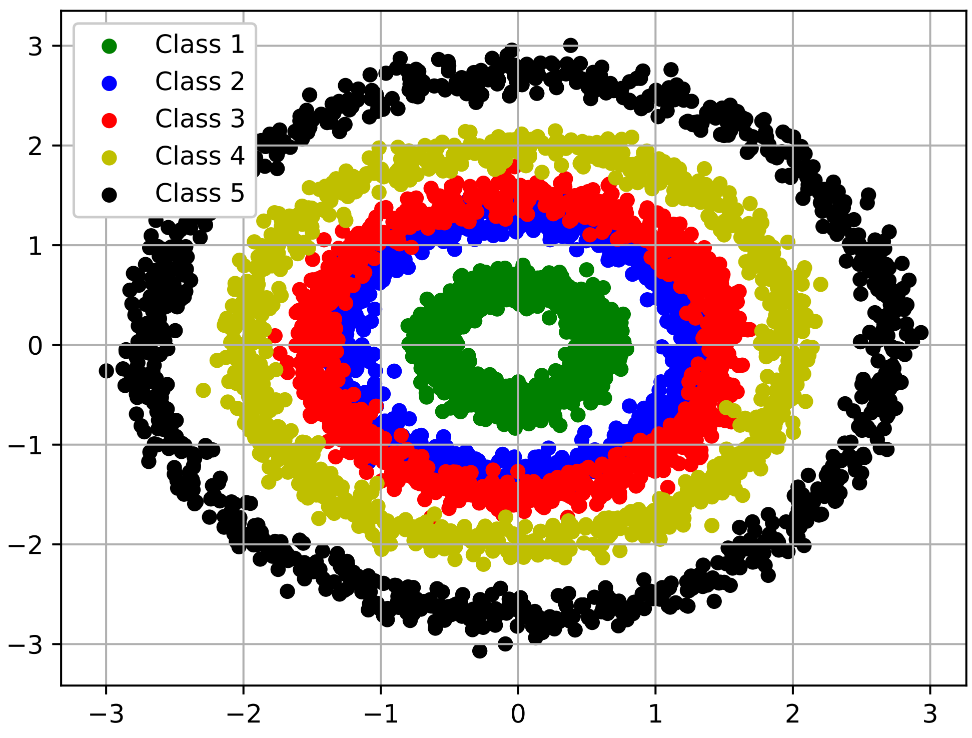


Figure 3.1: The Synthetic dataset from Section 3.8.2.

.

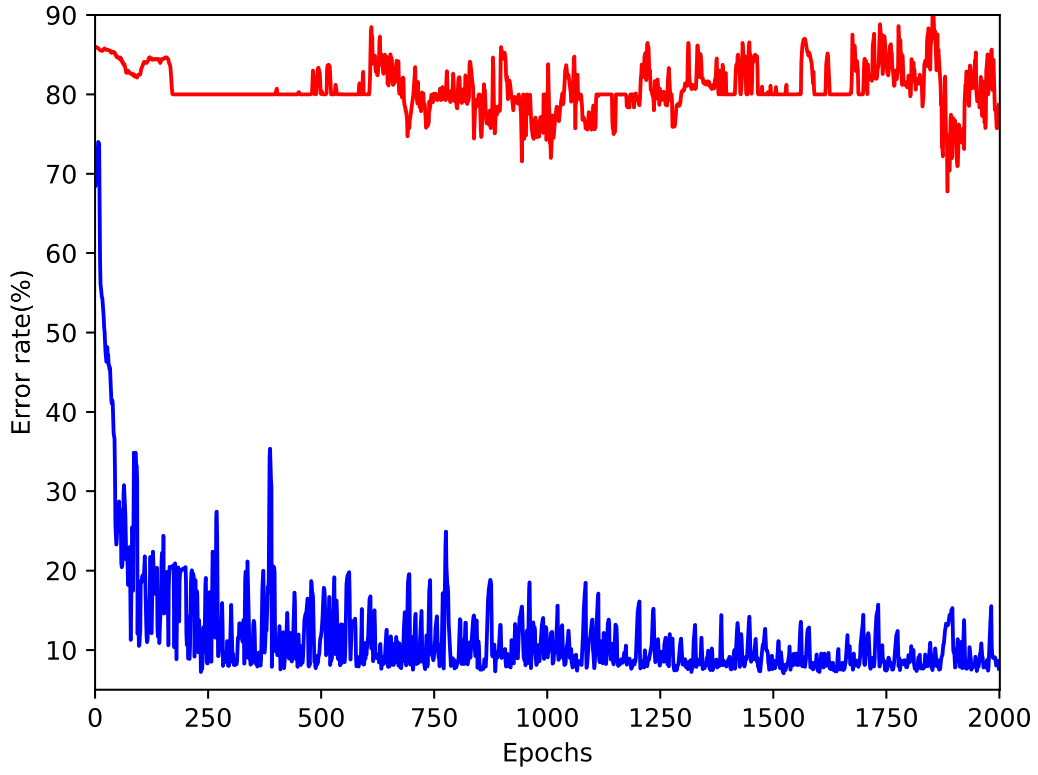


Figure 3.2: Evolution of the error rate throughout epochs for the synthetic dataset in Section 3.8.2. The blue line corresponds to non-parametric score functions while the red one to linear score functions.

.

7.08% error rate by using a Squared Exponential kernel. We run both of the methods for 2000 epochs and we plot the evolution of the error rate as a function of the number of epochs in Fig. 3.2. Apparently, this demonstrates the importance of our proposed generalization into the space of the non-parametric functions which allows to adjust the flexibility of the model via the kernel function's choice, and as we shall see in next sections, its usefulness when applied on real-world datasets.

3.8.3 Multi-class Classification using MNIST

Due to its high popularity, MNIST provides us several results from different methods to compare it. Nonetheless, we mostly focus on comparing the One-versus-Each method (OvsE) with the robustmax function described in Section 3.6 which has been proven highly effective in the past few years.

For all the experiments here, we employ the SE kernel (Eq. (1.4) of Chapter 1)

Method	OvsE-p	OvsE-f	OvsE-p-sh	OvsE-f-sh	RM-p	RM-f
Error Rate	3.68	3.28	6.8	4.04	3.9	2.6
nlpd	0.1377	0.1128	0.2295	0.3578	0.1192	0.081
Time	63.72	97.65	53.60	86.42	220.5	141.1
Memory	1.834	2.378	1.365	1.546	1.902	2.162

Table 3.2: Error rate (%) (given by (3.42)) and average negative log predictive density (nlpd) (given by (3.43)) for six different methods using $M = 100$ inducing inputs. Training time (seconds/epoch) and memory footprint in gigabytes (GB) are also reported for the same six methods, however $M = 1000$ used for those cases. The first four methods (left to right) methods are all relied on the methodology of Section 3.5 and their differences lie in the parametrization choice, as discussed in 2.4, and the shared or separate set of inducing inputs used. The last two methods, RM-p and RM-f, are based on the robustmax likelihood of Section 3.6 and the two different parametrizations where for both cases all classes share a common set of inducing inputs Z. Therefore we have method (i) OvsE-p where each class has its own set of inducing inputs Z_k and the parsimonious parametrization (PP) is used, (ii) OvsE-f where each class has its own set of inducing inputs Z_k and the full parametrization (FP) is used, (iii) OvsE-p-sh where all classes share a common set of inducing inputs Z and the PP is used, (iv) OvsE-f-sh where all classes share a common set of inducing inputs Z and the FP is used, (v) RM-p where the PP is used, and (vi) RM-f where the FP used. For all those methods a SE kernel is employed (with single lengthscale).

with a single lengthscale for all input dimensions. The reason we prefer this kernel instead of some other, more flexible kernel (e.g. ARD as defined in Eq. (1.4) of Chapter 1 using Λ_2 as distance measure), is to allow similar comparisons with other GP-based methods that have reported results for this Dataset. Consequently, we dot not expect to achieve the highest possible performance on this non-linear high dimensional dataset, since the curse of dimensionality (Marimont and Shapiro, 1979) renders the euclidean distances in those spaces extremely similar, making the supervised learning task more challenging. For robustmax-based methods RM-p and RM-f, we set the outlier threshold ϵ equals to 10^{-3}, as it is suggested in Hensman et al. (2015a).

Regarding experimental set-ups for all the previously described methods, we chose $M = 100$ inducing inputs and mini-batches of size 1000 (i.e. $|\mathcal{B}| = |\mathcal{B}_k| = 1000$, $\forall\, k$). Moreover, the number of epochs for the OvsE-p, OvsE-p-sh, and OvsE-f-sh was set equal to 10000 while for RM-p and RM-f, 100. This difference of the number of epochs is justified by the fact that One-vs-Each methods are based on the lower bound in (3.36), where the first sum is over classes, while in the robust max methods' lower bound, the first sum runs over data points (see (2.18)) which are much larger in number than K. Further, in order to reduce the stochastic effect of the methods on the reported results and facilitate comparisons, we choose to initialize the variational parameters and hyperparameters as well, with identical values for all the reported methods, while at the same time, we seed the random generator with the same value. We employ the Adam optimizer (Kingma and Ba, 2014) for all the optimization problems in this Section since we found to be successful in practice. Finally, for running all the aforementioned methods, a local machine with 16 GB RAM and an Intel i7-4720HQ CPU @ 2.6GHz was used.

Table 3.2 shows the corresponding error rates and average negative log predictive density (nlpd) for each of the methods, where their values are computed by

$$\text{error rate} = \frac{1}{N^*} \sum_{t=1}^{N^*} \mathbf{I}(y^{(t)} \neq \hat{y}^{(t)}), \tag{3.42}$$

$$\text{nlpd} = \frac{1}{N^*} \sum_{t=1}^{N^*} p(y^{(t)}|\mathbf{y}), \tag{3.43}$$

where N^* denotes the number of the test data points and $\mathbf{I}(\cdot)$ is the indicator function while $y^{(t)}$ and $\hat{y}^{(t)}$ are the ground truth and predicted class label, respectively. RM-F is able to achieve the lowest error while the OvsE-P-SH the highest. Although, unsurprisingly, it is RM-F that has the lowest nlpd again, the largest value comes from the OvsE-F-SH indicating that we could have methods that have low error rates but relatively higher nlpd due to the predictive variance. We can also notice that the PP always achieves inferior performance than its full counterpart, regardless the choice of the likelihood. Nevertheless, our propounded One-vs-Each method is proved to be very competitive comparing to the RM-based methods, leading to similar results, in terms of both error rate and nlpd. This is a promising result since it provides us a Bayesian principled way to successfully address classification problems where the number of both classes and data points is extremely large. The only potential drawback of the One-vs-Each methods is that their performance is highly connected with different sets of inducing inputs per class. Admittedly, we see a performance deterioration for the OvsE-P-SH where there is a shared set of inducing inputs across classes. This could be restricting in cases where the number of classes is large since we would need $\mathcal{O}(KMD)$ parameters for the inducing inputs storage[7]. Nevertheless, if we opt for a FP combined with shared Z, then the error is much closer to OvsE-P and OvsE-F, although we might need to be more conservative with increasing the number of the inducing inputs M, since $\mathcal{O}(KM^2)$ storage is required to parametrize all the variational distributions.

[7]Here we ignore the computational effect of the input dimensionality. Nonetheless, it can be proven burdensome in cases where $D \gg 1000$ and it is in detail in Chapter 4.

Method	Inital values	OvsE-P	OvsE-F	RM-P	RM-F
ℓ	10.0	22.422	52.977	43.103	54.585
σ_f^2	1.0	1.3992	2.4214	11.079	0.1359

Table 3.3: Values of the initial and optimized hyperparameters of the SE kernel for each method in Table 3.2. Those values correspond to the methods used $M = 100$ in Table 3.2.

We also consider both computational time and memory footprint for all the aforementioned methods where the results are also demonstrated in Table 3.2. In this case, we increase the number of the inducing inputs to $M = 1000$ in order to have a better understanding of the results. We run each of the methods for 60 iterations which corresponds to one complete scan (one epoch) of the training dataset in the case of the non-One-vs-Each methods and batches of 1000 data points. As it is expected from the complexity analysis in Table 3.1, the most computationally demanding method is RM-P, since for each optimization step the stochastic computation of ELBO with its corresponding unbiased estimates of its gradients, requires $\mathcal{O}(KM^3 + |\mathcal{B}|M^2)$. We can also notice how the memory footprint varies across the methods, providing us an insightful way to discern the scalability/performance trade-off that emerges with the use for each of those methods. By observing the memory footprints, it is worth mentioning at this point, the high performance we obtain by using Tensorflow in our implementation. This can be observed by investigating the memory requirements of OvsE-F. Admittedly, it does not require more than 2.378 GB memory space in RAM for computing the value of the objective function and then temporarily storing its derivatives for updating all the parameters, despite the fact that the number of all the variational parameters we need to optimize here is 12855000. This is clearly an optimization task that is not encountered frequently in Bayesian frameworks and it is more reminiscent of training very large neural networks.

In spite of initializing all the methods with same values for their corresponding hyperparameters[8], the optimized hyperparameters are much different for each method. For instance, the RM-P's optimized kernel variance is found to be more than 100 times larger than RM-F's, while, on the other hand, its corresponding length-scale is smaller than RM-F's one. A similar behaviour is also observed between OvsE-P and OvsE-F. However, their optimized variances does not exhibit so large differences as before.

Next, we present how the inducing inputs Z move from their initialized positions to the optimized ones. The very nature of the dataset allows us to visualize inducing inputs as pictures and observe their connection with the optimized ones.

[8]And variational parameters as well, when that was feasible.

Method	Reference
Linear classifier	LeCun et al. (1998)
OvsE-lin	Titsias (2016)
GP Latent Variable Model	Gal et al. (2014)
1-Nearest Neighbour	Wilder (1998)
OvsE-f	this work
GP Classifier RBF kernel 1 (SVI)	Dezfouli and Bonilla (2015)
GP Classifier RBF kernel 2 (SVI)	Hensman et al. (2015a)
GP Classifier ARD kernel (EP)	Hernández-Lobato et al. (2016b)
RM-f	this work
SVM RBF kernel	LeCun et al. (1998)
Convolutional GP	Van der Wilk et al. (2017)
Deep convolutional network	Ciresan et al. (2011)

Table 3.4: References for the methods used in Table 3.5. SVI stands for Stochastic Variational Inference and EP for Expectation-Propagation.

This has been investigated in Hensman et al. (2015a) and Hensman et al. (2015b), where they have showed that the positions of the optimized inducing inputs tend to approach the classification decision boundary. We also confirm that behaviour for all the used methods here. Figures 3.7 , 3.8 , and 3.9 show 2 different inducing inputs, before and after optimization, for 3 different pairs of methods, namely OvsE-p vs OvsE-f, OvsE-p-sh vs OvsE-f-sh, and RM-p vs RM-f, respectively. The inducing inputs of each pair are initialized by k-means clustering. In the case of the OvsE-p-OvsE-f pair, we run k-means 10 different times, one for each class, using as data points for each clustering only those that belong in the corresponding class. As we can see, regardless the choice of the method, inducing inputs move away from their initialized values, heading towards the classification boundary. For example, in Fig. 3.7 both OvsE-p and OvsE-f are initialized with same k-means center that looks like a three. Nevertheless, after optimization, the inducing input optimized by OvsE-p bears similarities with a six, an eight and a five, while for the OvsE-f case it resembles both a nine and a five digit.

We continue by showing a diversified collection of methods are applied on MNIST, providing both references and error rates in tables 3.4 and 3.5, respectively. For both methods OvsE-f and RM-f, we use again the isotropic Squared Exponential kernel, but this time we dramatically increase the number of inducing inputs to $M = 3000$. To the best of our knowledge, this is the first time that a Sparse Gaussian Process model is trained with that number of inducing inputs. For instance, we note that OvsE-f, which is the most memory demanding method, needed no more than 11 GB to run, meaning that a standard desktop machine could be employed. Nevertheless, due to the large time budget we offered for those two experiments, they run on an Intel Xeon Processor E5-2667 v3 server. Regarding the error rates of table 3.5, the convolutional network, as expected, achieves the best possible performance[9] of all methods. It is encouraging that RM-F, with a simple kernel function as the SE, managed to be the second most competitive GP-based method. Even our proposed method OvsE-F, performs relatively well, given its approximating nature, having an error rate very close to other Sparse GP models like Dezfouli and Bonilla (2015).

Last but not least, we provide experimental evidence in favour of our choice of using Adam throughout all the experiments. More specifically, we test four of the most popular algorithms for stochastic optimization, namely, Adam (Kingma and Ba, 2014), RMSProp (Hinton et al., 2012), Adagrad (Duchi et al., 2011), and Adadelta (Zeiler, 2012). All of the four algorithms are employed to optimize the ELBO for the RM-F while we set the $M = 20$, $|\mathcal{B}| = 5000$, and epochs = 200. We also perform a grid-search for the learning rate parameter of each algorithm, using the following set of values, $\{0.1, 0.05, 0.01, 0.005, 0.001\}$. All the initial variational parameters and the kernel hyperparameters values are common when we run each of the algorithms. Adam and RMSProp appear to have almost similar performance while Adadelta does not have competitive results, despite its successful use in the past for similar models (Hensman et al., 2015b). Further, our experiments show that Adam is more robust than the other algorithms with respect to different learning rate values.

3.9 Conclusion

In this chapter we try to address the scalability issues that emerge in multi-class classification problems using Gaussian Process models, in terms of both number of training data points and classes. Driven by the One-versus-Each approximation of Titsias (2016), we extend this idea by generalizing the parametrized score functions to arbitrary random functions via Gaussian process priors and then inspired by Hoffman et al. (2013) and Hensman et al. (2015b) this gives birth to a new scalable method that can efficiently cope with both arbitrarily large number of classes and data. This extension incorporates the flexibility provided by Gaussian processes and it is verified by Section 3.8 which shows the superiority over linear score functions used in Titsias (2016). Moreover, we see that our proposed method is able to attain predictive performance similar to other GP-based methods in the literature despite its approximating nature (Section 3.8.3), even though slightly lower. Regarding the two parametrizations, our proposed PP and the conventional FP, we observe that in general, methods that utilize the FP are able to achieve better performance than those that use the PP. Therefore there is a trade off

[9]To be precise, the lowest ever achieved error rate on MNIST is 0.23 in Cireşan et al. (2012). However, there is an extra preprocessing step for width normalization of the pictures, something that is not present in the last reported method of table 3.4.

Method	Error rate (%)
Linear classifier	12.0
OvsE-lin	7.4
GP Latent Variable Model	5.95
1-Nearest Neighbour	3.09
OvsE-f	2.9
GP Classifier RBF kernel 1 (SVI)	2.51
GP Classifier RBF kernel 2 (SVI)	1.96
GP Classifier ARD kernel (EP)	1.80
RM-f	1.63
SVM RBF kernel	1.4
Convolutional GP	1.17
Deep convolutional network	0.35

Table 3.5: A summary of several methods, both GP-based and others, with their corresponding error rates for the MNIST dataset. Some of the results can be found at Lecun's website (LeCun, 2019).

Algorithm	Optimal learning rate	Error rate
AdaDelta	0.05	14.0
AdaGrad	0.05	5.41
RMSProp	0.005	4.93
Adam	0.001	4.89

Table 3.6: Performance comparison using different algorithms for optimization of RM-f and their corresponding optimal learning rate.

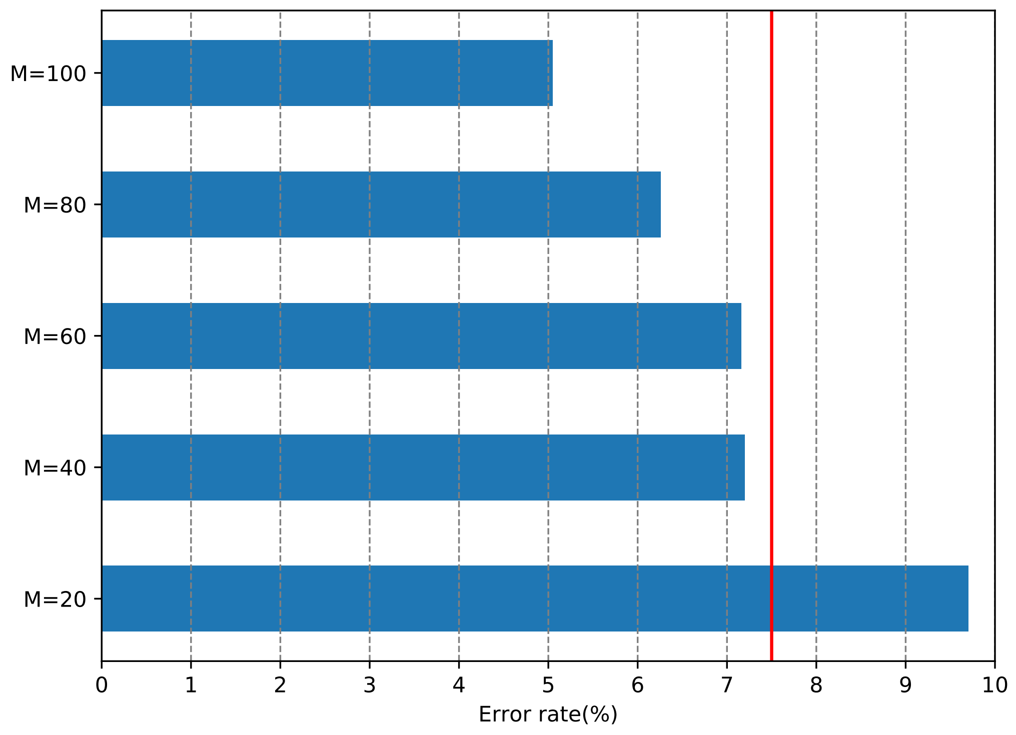
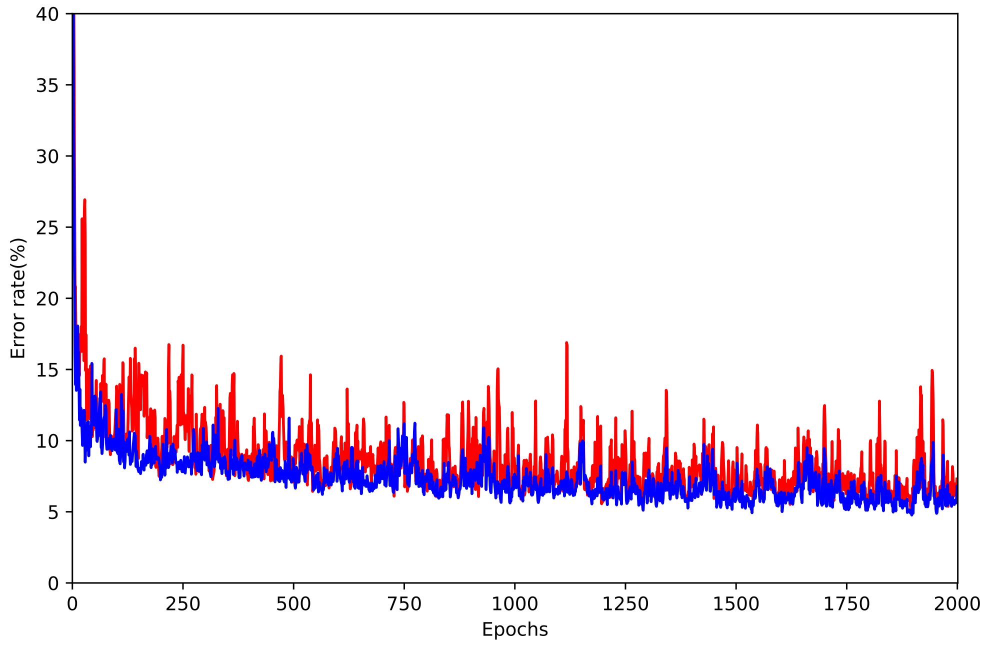
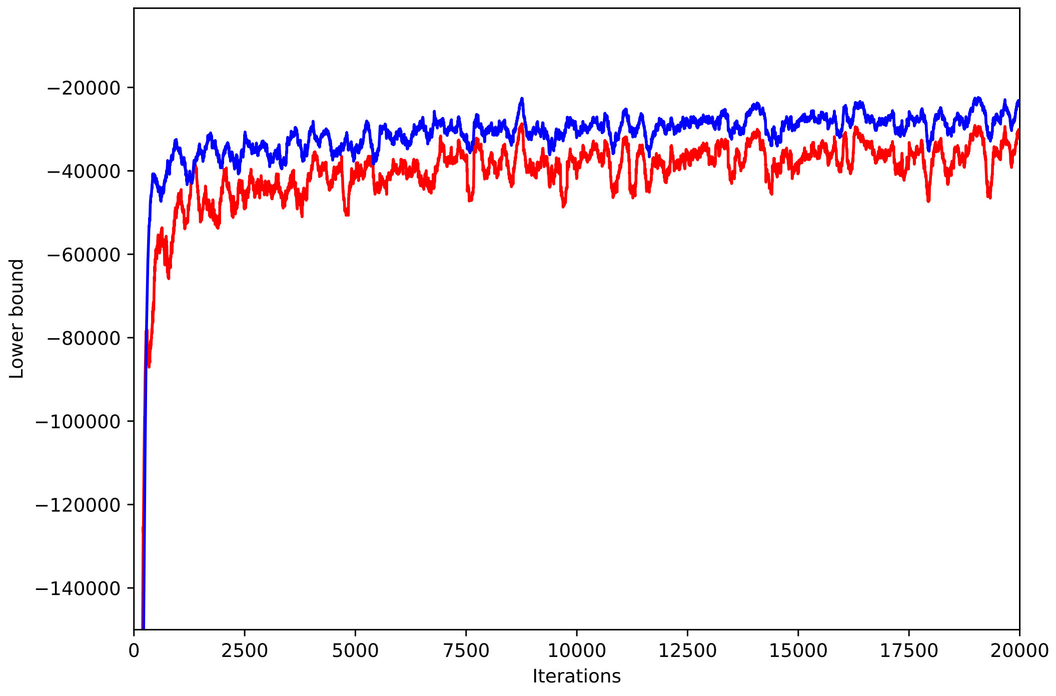


Figure 3.3: Error rates for the OvsE-p as a function of the number of inducing inputs M. The red vertical line indicates the error rate of using linear score functions, as described in Section 3.4.1 and Titsias (2016).

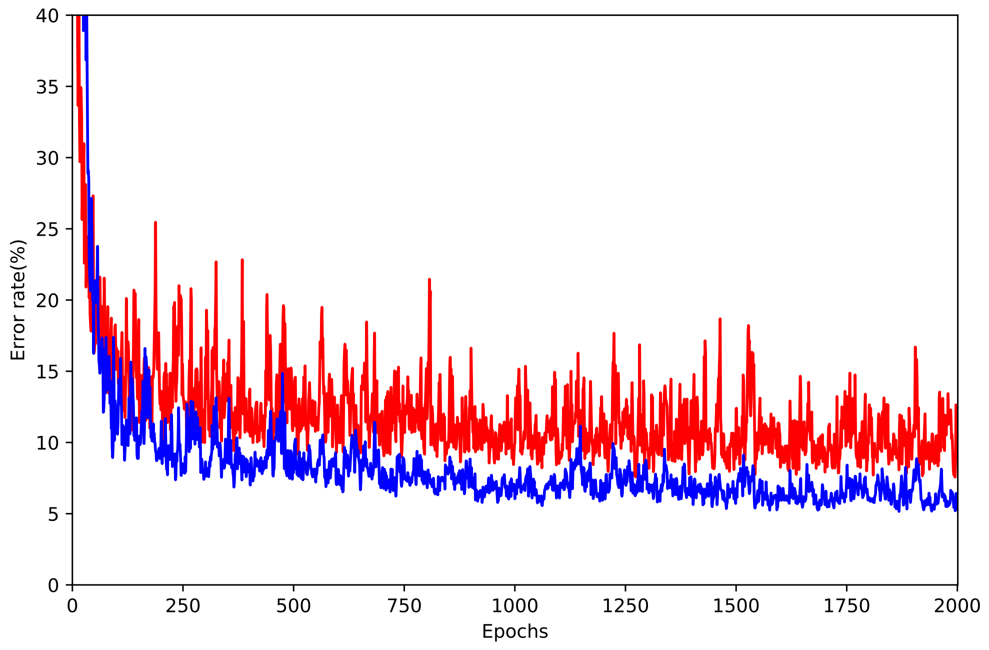


(a)

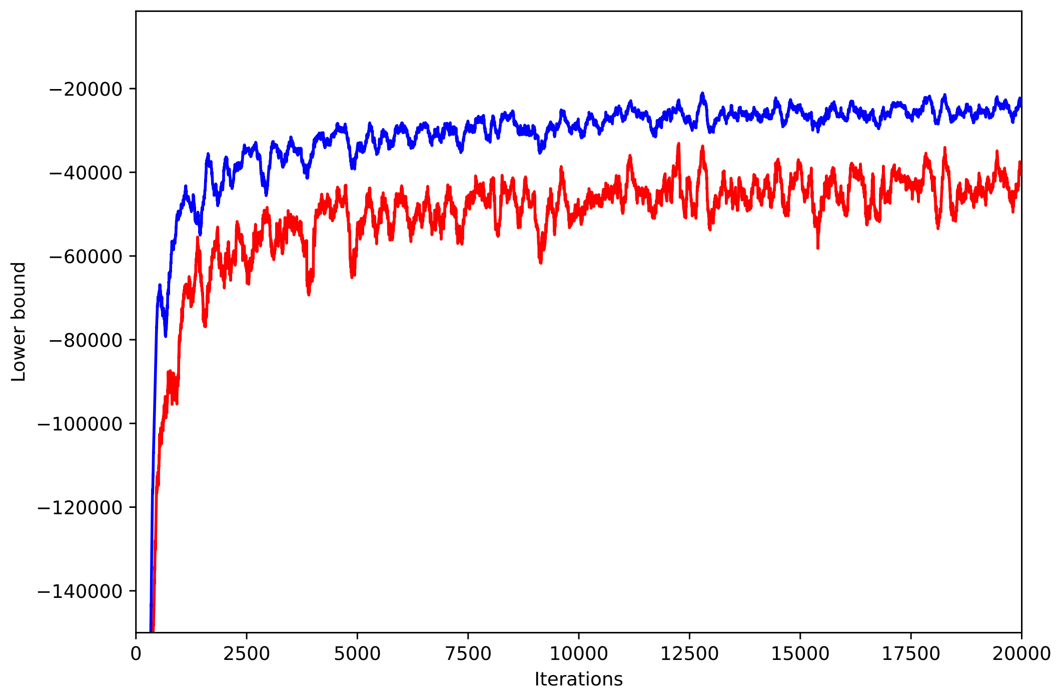


(b)

Figure 3.4: (a) Error rates per epoch comparison between OvsE-p and OvsE-f and (b) Evolution of the lower bound per iteration between OvsE-p and OvsE-f. Red color indicates use of OvsE-p, while blue color use of OvsE-f.

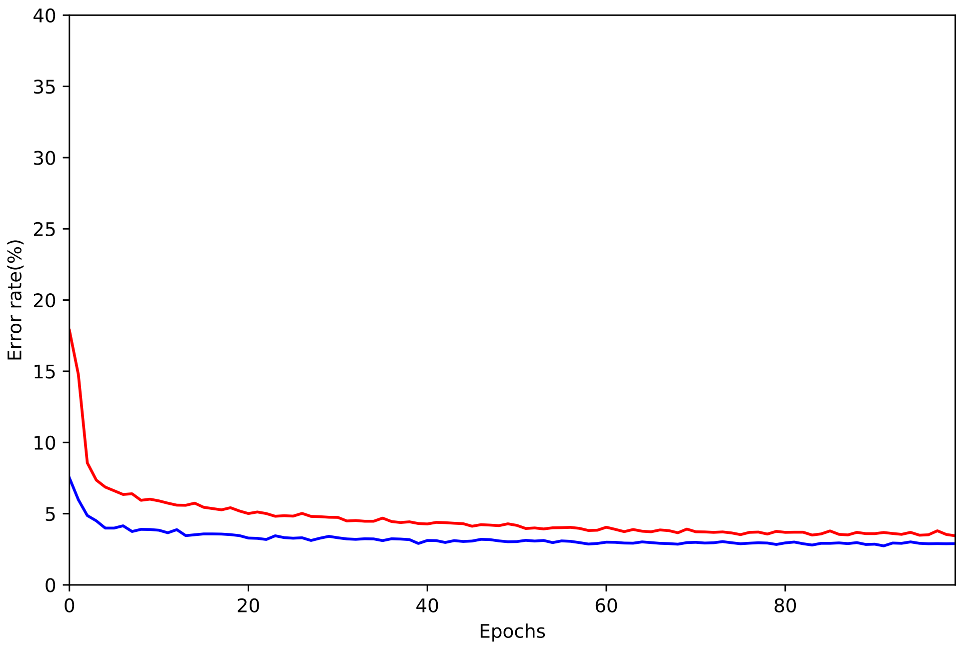


(a)

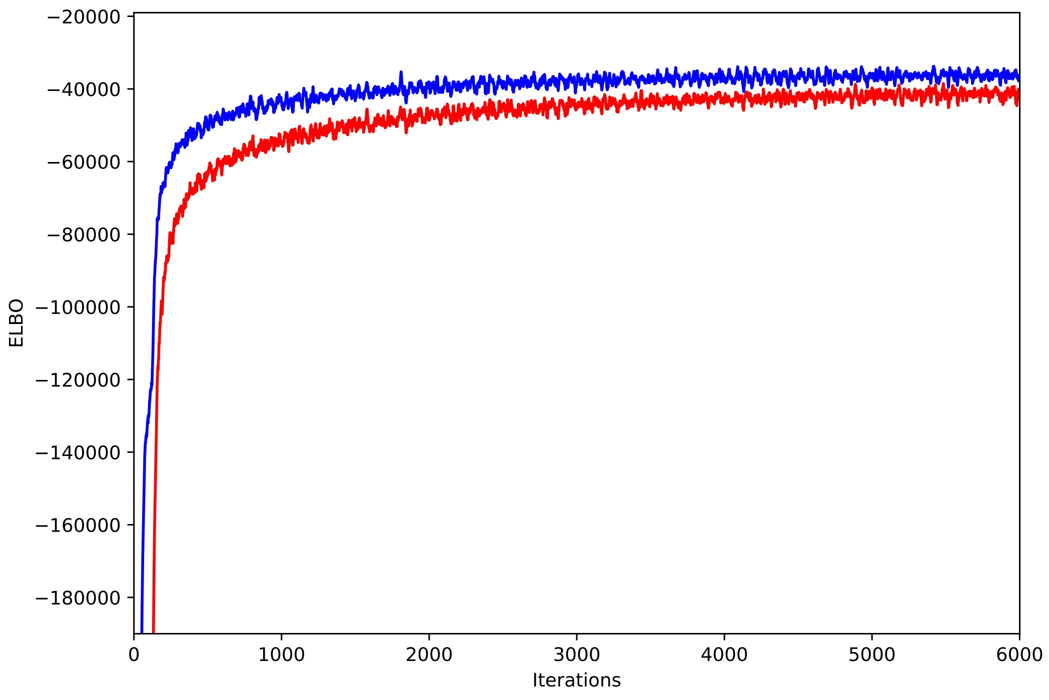


(b)

Figure 3.5: (a) Error rates per epoch comparison between OvsE-p-sh and OvsE-f-sh and (b) Evolution of the lower bound per iteration between OvsE-p and OvsE-f. Red color indicates use of OvsE-p-sh, while blue color use of OvsE-f-sh.



(a)



(b)

Figure 3.6: (a) Error rates per epoch comparison between RM-P and RM-F and (b) Evolution of the lower bound per iteration between RM-P and RM-F. Red color indicates use of RM-P, while blue color use of RM-F.

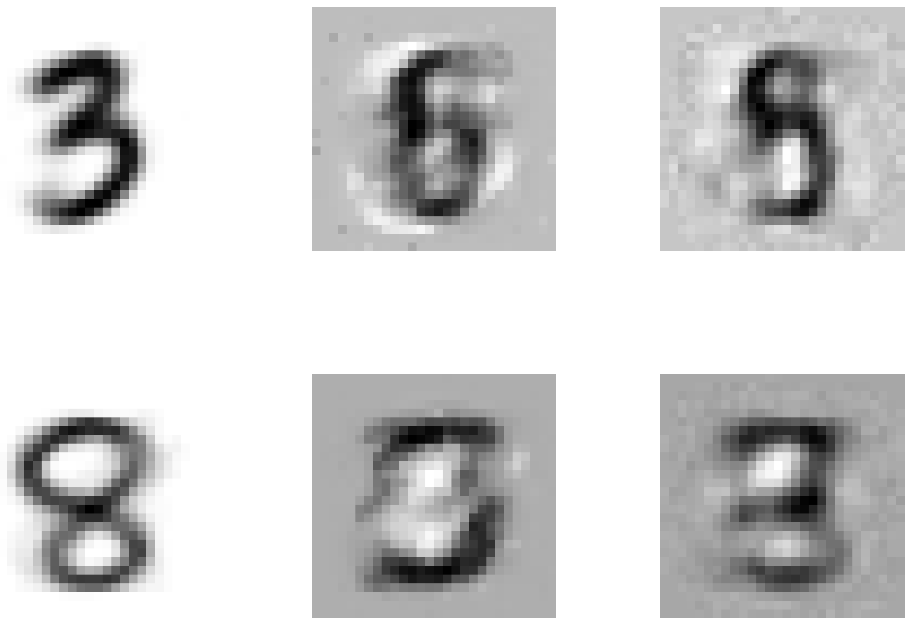


Figure 3.7: Visualization of the inducing inputs. Each row corresponds to different inducing point. Left: Initialized inducing input with a k-means center. Center: Optimized inducing point by using OvsE-p. Right: Optimized inducing point by using OvsE-f.

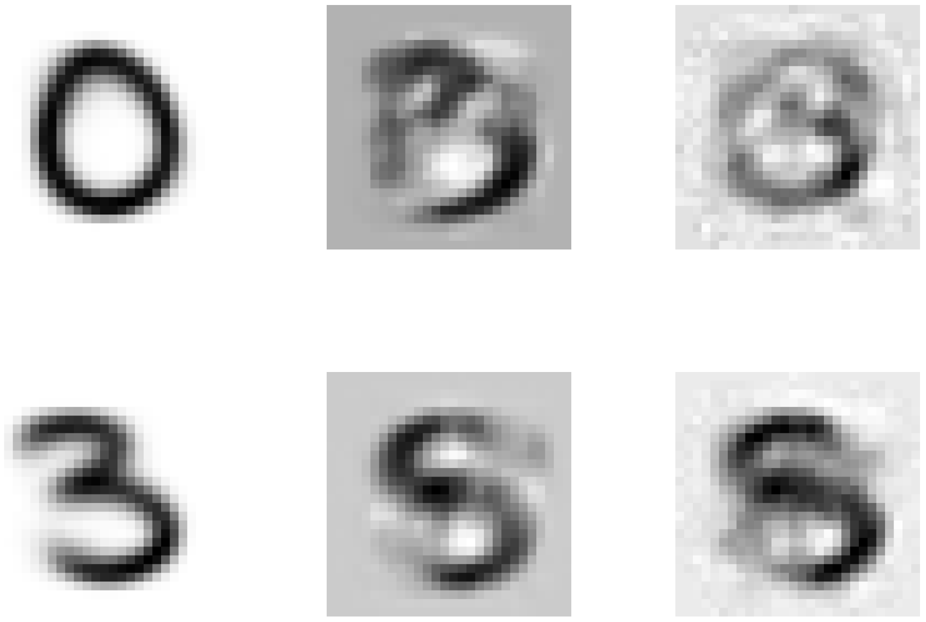


Figure 3.8: Visualization of the inducing inputs. Each row corresponds to different inducing point. Left: Initialized inducing input with a k-means center. Center: Optimized inducing point by using OvsE-p-sh. Right: Optimized inducing point by using OvsE-f-sh.

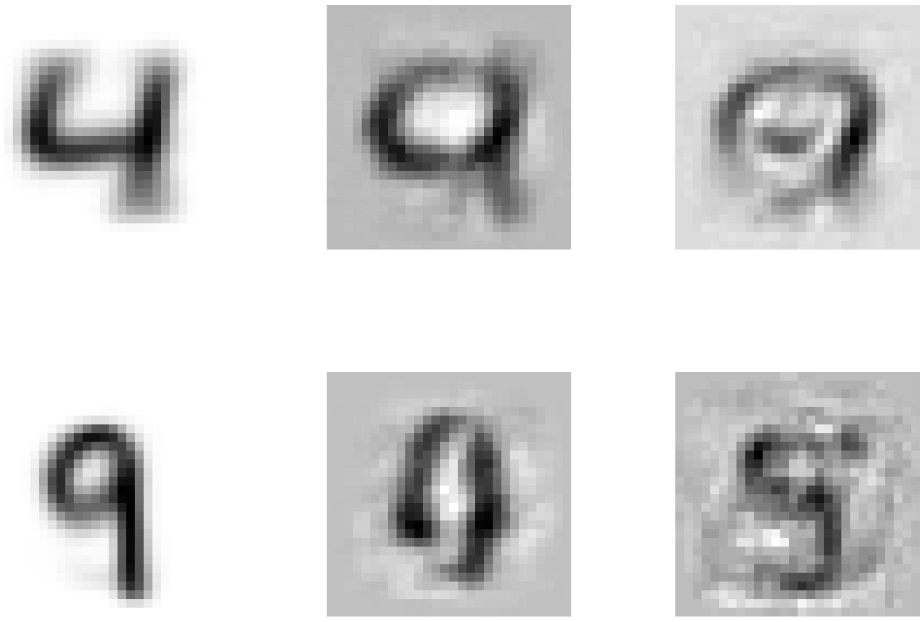


Figure 3.9: Visualization of the inducing inputs. Each row corresponds to different inducing point. Left: Initialized inducing input with a k-means center. Center: Optimized inducing point by using RM-P. Right: Optimized inducing point by using RM-F.

between number of variational parameters and model's flexibility. Finally, we show that Gaussian process approximation methods have inferior performance than neural networks; however, we run the method of Hensman et al. (2015b) with a large number of inducing inputs using our own implementation and we achieve the highest ever reported accuracy for a GP-based method using a simple SE kernel with a single lenghtscale for the MNIST dataset.

Chapter 4

Extreme Multi-label Inference using Gaussian Processes

4.1 Introduction

This chapter is mainly focused on introducing a scalable way of employing Gaussian Process models on multi-label classification problems. It is discussed how multi-output GP models can be transformed to a suitable multi-label classification scheme and different perspectives of scalability issues are addressed. The proposed methodology is thoroughly tested on a number of both small and large-scale datasets where some of them involve more than (i) 16×10^5 input dimensions, (ii) 17×10^5 number of data instances, and (iii) 3×10^5 number of labels.

4.2 From multi-output GP regression to multi-label classification

Multi-output regression using Gaussian processes has a long history. The idea of capturing output correlations between multiple single-output GP regression models has been described in Cressie (1992) (Section 3.2.3), where a general framework for this problem was introduced under the name *co-kriging* and focused on geostatistics applications. Since then, there is voluminous research on the subject where it can be found in the work of Williams and Rasmussen (1996); Goovaerts et al. (1997); Lawrence (2004); Teh et al. (2005); Boyle and Frean (2005); Alvarez and Lawrence (2009); Álvarez et al. (2010); Titsias and Lawrence (2010); Álvarez and Lawrence (2011); Wilson et al. (2011); Skolidis and Sanguinetti (2011); Alvarez et al. (2012); Nguyen et al. (2014); Dai et al. (2017); Moreno-Muñoz et al. (2018) among many others. In most of the cases, the GP-based multi-output models produce correlated outputs by mixing a number of independent latent Gaussian Processes. This problem is well known in geostatistics community as the *linear model of coregionalization* (Goovaerts et al., 1997). This kind of models has been employed on a Bayesian framework where a prior has been imposed over the mixing coefficients (Titsias and Lázaro-Gredilla, 2011). Further work on this direction, can be found in (Nguyen and Bonilla, 2013; Wilson et al., 2011; Alvarez and Lawrence, 2009; Álvarez and Lawrence, 2011) where most advanced techniques are developed which are based on input-dependent coefficients (Nguyen and Bonilla, 2013; Wilson et al., 2011) or convolving processes (Alvarez and Lawrence, 2009; Álvarez and Lawrence, 2011), and scalability addressing methods (Nguyen et al., 2014). Multi-output GP models have been used in an unsupervised manner for non-linear dimensionality reduction (Lawrence, 2004; Titsias and Lawrence, 2010) while their theory has also been applied on hierarchical models of multiple GPs (Damianou and Lawrence, 2013).

Most of the previous work mentioned on multi-output (or multi-task) GPs has in common that it was mainly focused on regression problems, except Teh et al. (2005) and Skolidis and Sanguinetti (2011), without taking into account discrete or more specifically binary outputs which demands reformulation of the likelihood. This consideration of binary outputs leads naturally to a GP-based method for multi-label learning where a Bernoulli or a binary regression type likelihood is required. That is exactly what we discuss in the next section where the main model is introduced with a factorized Bernoulli over the labels employed as likelihood to cope with the binary nature of the task. Nevertheless, the pertinent question that arises here is how the model would be able to capture correlations among labels? The answer comes from applying the Semiparametric latent factor model (SLFM) proposed by Teh et al. (2005). Note that this was the only method that was applied to a multi-label synthetic dataset. Despite its simplicity, SLFM is suitable for large-scale paradigms since it lets us adjusting the number of the latent Gaussian Processes accordingly, while at the same time simple linear algebra operations are needed for its computation which can be performed fast and efficiently.

At this point we would like to add that there is one more recently published work (Moreno-Muñoz et al., 2018) that became aware to us during the writing of this thesis, and it can be seen as a generalization of our proposed model in Section 4.3. The model introduced in Moreno-Muñoz et al. (2018) is able to consider outputs than can take both discrete and continuous values at the same time. However, this work is not concerned with large-scale datasets and its associated publicly available code[1] is not suitable to cope with datasets of the scale we use here.

4.3 The Multi-Label GP Factor Model

We start our discussion by introducing, as usual, a training dataset $\mathcal{D} = \{\mathbf{x}^{(i)}, \mathbf{y}^{(i)}\}_{i=1}^{N}$ where each $\mathbf{x}^{(i)} \in \mathbb{R}^D$ is the input vector while $\mathbf{y}^{(i)} \in \{-1, 1\}^K$ is now the binary

[1]https://github.com/pmorenoz/HetMOGP

vector that corresponds to the class labels assigned to $\mathbf{x}^{(i)}$, such that $y_k = 1$ indicates presence of the k-th label while $y_k = -1$ indicates absence. We collectively denote all input vectors by $X \in \mathbb{R}^{N\times D}$ and the binary labels by $Y \in \{-1, 1\}^{N\times K}$ so that rows of these matrices store respective data points. We wish to model these data using a flexible probabilistic model that captures the correlation between different labels. As it was presented previously, a multi-label extension of the SLFM is considered which combines a linear latent variable model with GPs. Specifically, SLFM is a general-purpose multi-output GP model Teh et al. (2005); Álvarez and Lawrence (2011); Alvarez et al. (2012) that uses a small number of P latent GPs (factors) to generate the K outputs through a linear mapping. The full hierarchical model for generating the training examples is,

$$h_p \sim \mathcal{GP}(0, \mathbf{k}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})),\ p = 1, \dots, P, \tag{4.1}$$

$$\mathbf{f}^{(i)} = \Phi\mathbf{h}^{(i)} + \mathbf{b},\ i = 1, \dots, N, \tag{4.2}$$

$$\mathbf{y}^{(i)} \sim p(\mathbf{y}^{(i)}|\mathbf{h}^{(i)}) = \prod_{k=1}^{K} \sigma(y_k^{(i)} f_k^{(i)}),\ i = 1, \dots, N, \tag{4.3}$$

where h_p denotes a latent function drawn from a GP with zero-mean and kernel function $k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ that depends on kernel hyperparameters $\boldsymbol{\theta}$ (although $\boldsymbol{\theta}$ is suppressed throughout to keep the notation uncluttered). We refer to that model as the Multi-Label GP Factor Model (MLGPF). Apparently, we could consider different kernels $\mathbf{k}_p(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$ with different sets of hyperparameters $\boldsymbol{\theta}_p$ for each of the latent Gaussian Processes, however, this would increase both computational time and storage requirements without providing any significant performance improvement in practice and thus, we omit its use. Further, $\mathbf{h}^{(i)} = (h_1^{(i)}, \dots, h_P^{(i)})^\top \in \mathbb{R}^P$ denotes the vector of all function values evaluated at input $\mathbf{x}^{(i)}$, i.e. $h_p^{(i)} \triangleq h_p(\mathbf{x}^{(i)})$, while the parameters $\Phi \in \mathbb{R}^{K\times P}$ and $\mathbf{b} \in \mathbb{R}^K$ correspond to the factor loadings matrix and the bias vector of the linear mapping, respectively. By using these parameters, the latent vector $\mathbf{h}^{(i)}$ is deterministically mapped into $\mathbf{f}^{(i)} = (f_1^{(i)}, \dots, f_K^{(i)})^\top \in \mathbb{R}^K$, such that each

$$f_k^{(i)} = \sum_{p=1}^{P} \phi_{kp} h_p^{(i)} + b_k, \tag{4.4}$$

defines the so-called *utility* score that finally generates the k-th binary label through a sigmoidal/Bernoulli likelihood. The *utility* score plays a crucial role in the prediction of new label vectors as we probe in Section 4.4.5. Notice that while the labels are conditionally independent given $\mathbf{h}^{(i)}$, they become fully coupled once these variables are integrated out, rendering in that way the model suitable for capturing inter-label correlations. The full joint distribution is given by

$$\prod_{i=1}^{N} p(\mathbf{y}^{(i)}|\mathbf{h}^{(i)}) \prod_{p=1}^{P} p(\mathbf{h}_p), \tag{4.5}$$

where $p(\mathbf{h}_p) = \mathcal{N}(\mathbf{h}_p|\mathbf{0}, K_X)$ is an N-dimensional Gaussian distribution induced by evaluating the GP prior at the training inputs X with K_X denoting the kernel or covariance matrix, $[K_X]_{ij} = \mathbf{k}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$. An equivalent way to write the above model is by using the concept of kernels for multi-task or vector-valued functions Bonilla et al. (2008); Álvarez and Lawrence (2011); Alvarez et al. (2012). More precisely, observe that the utility scores $f_k^{(i)}$ that directly interact with the data in (4.4) follow a GP prior with mean given by the bias b_k (that depends on the label but not on the input) and covariance function

$$\mathrm{Cov}(f_l^{(i)}, f_k^{(j)}) = \mathbf{k}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \sum_{p=1}^{P} \phi_{lp}\phi_{kp}. \tag{4.6}$$

For regression problems with Gaussian likelihoods the above multi-task GP, as we mentioned in the previous section, is known as the intrinsic correlation model Stoyan (1996); Bonilla et al. (2008), a specific case of co-kriging in geostatistics. For more details about multi-task GP inference and separable kernels, as the one in (4.6), we refer the reader to Alvarez et al. (2012) for a full review. Here, we use this model for multi-label learning where the tasks correspond to different class labels.

Inference in the above model is very challenging since real applications in multi-label classification involve both very large number of training instances N and very large number of class labels K (Zhang and Zhou, 2013; Gibaja and Ventura, 2014, 2015). This forces us to resort to a scheme that combines sparse GPs with stochastic variational inference (Hoffman et al., 2013) (see also Section 2.3.2 of Chapter 2) which, as we show later, scales well for both number of training instances and labels.

4.4 Scalable Variational Inference

The approximate inference procedures derived in this section are mainly based on the representation that uses the latent GP vectors $\mathbf{u}_p$ rather than the multi-task kernel representation in (4.6). More specifically, Section 4.4.1, following the theory of Chapter 2, presents a scalable way to deal with the large number of training data points, similar with the one described in Chapter 3, Section 3.5. The utility scores $f_k^{(i)}$ are only used to simplify the computations of some final Gaussian integrals. Section 4.4.4 shows how doubly stochastic optimization can allow to deal simultaneously with arbitrarily large N and K, giving rise to a doubly stochastic optimization scheme and Section 4.4.5, finally, provides details of how prediction of new label vectors can be achieved.

4.4.1 The augmented model

To deal with large number of training data we consider the variational sparse GP inference framework based on inducing variables introduced by Titsias (2009) and described in Section 2.3.2 of Chapter 2. The reader can be also referred to Matthews et al. (2016) for a measure-theoretic derivation of this method and Bauer et al. (2016) for a useful discussion about its properties and how it compares with other GP sparse approximations. For each latent function h_p we introduce a vector $\mathbf{u}_p \in \mathbb{R}^M$ of function values of h_p evaluated at inputs Z, where for simplicity we take the inputs Z to be shared by all latent GPs. This is not restrictive at all but the extremely dataset sizes accompanied by poor performance improvement using different sets of inducing inputs renders the choice of shared Z practically sound. The vector $\mathbf{u}_p$ is often referred to as inducing variables and Z as the inducing or pseudo inputs (Quiñonero-Candela and Rasmussen, 2005a; Snelson and Ghahramani, 2006). In the variational sparse GP method Z plays the role of a variational parameter that can be optimized to improve the approximation. By following Titsias (2009) we augment the joint distribution in (4.5) with the inducing variables to obtain

$$\prod_{i=1}^{n} p(\mathbf{y}^{(i)}|\mathbf{h}^{(i)}) \prod_{p=1}^{P} p(\mathbf{h}_p|\mathbf{u}_p)p(\mathbf{u}_p). \tag{4.7}$$

Here, $p(\mathbf{u}_p) = \mathcal{N}(\mathbf{u}_p|\mathbf{0}, K_Z)$ is the marginal GP prior over $\mathbf{u}_p$ and K_Z is the $M \times M$ kernel matrix obtained by evaluating the kernel function at Z while $p(\mathbf{h}_p|\mathbf{u}_p)$ is the conditional GP prior which, by making use of (A.4), can be written as

$$p(\mathbf{h}_p|\mathbf{u}_p) = \mathcal{N}\left(\mathbf{h}_p|K_{XZ}K_Z^{-1}\mathbf{u}_p, K_X - K_{XZ}K_Z^{-1}K_{ZX}\right), \tag{4.8}$$

where K_{XZ} is the cross-covariance matrix between the training inputs X and the inducing inputs Z while $K_{ZX} = K_{XZ}^\top$. For any value of Z this augmentation does not change the model (e.g. the exact marginal likelihood is invariant to the value of Z), however by applying a certain variational approximation in the space of $(\mathbf{h}_p, \mathbf{u}_p)$ we can both reduce the time complexity and also treat Z as a variational parameter. We would like to shortly note here that the augmentation argument of Titsias (2009) has been thoroughly studied by Matthews et al. (2016), who showed that, in general even if the original model does not necessarily have the same optimal variational parameters as the augmented one, it performs well in practice (see Section 2.3.2 of Chapter 2 for a similar discussion).

We are now ready to introduce the density of the variational distribution for the augmented model,

$$Q = \prod_{p=1}^{P} p(\mathbf{h}_p|\mathbf{u}_p)q(\mathbf{u}_p), \tag{4.9}$$

where $p(\mathbf{h}_p|\mathbf{u}_p)$ is the conditional GP prior that appears also in the joint (4.7), while $q(\mathbf{u}_p)$ is a the density of the M-dimensional Gaussian variational distribution over the inducing variables for the p-th latent GP. The given choice of the density of the variational distribution is such that allows the cancellation of the $p(\mathbf{h}_p|\mathbf{u}_p)$ of each factor, facilitating in that way crucial computations in a straightforward manner.

At this point, for being able to continue the derivation of the lower bound in next section, we need to choose a specific parametrization for $q(\mathbf{u}_p)$, as described in Chapter 2, Section 2.4. This provides us with the appropriate equations for computing the marginal means and variances needed for the evaluation of the expectation terms in (4.28). We start with the full parametrization (FP) and then we introduce the parsimonious parametrization (PP).

For the FP, we get

$$q(\mathbf{u}_p) = \mathcal{N}(\mathbf{u}_p|\mathbf{m}_p, S_p), \tag{4.10}$$

where $\mathbf{u}_p$ is a vector of M entries and $S_p = L_p L_p^\top$ is an $M \times M$ covariance matrix parametrized by the $\frac{M(M+1)}{2}$ tunable parameters of the lower triangular matrix L_p. Therefore, by combining 4.10 and a well-known Gaussian identity (A.1), we are able to marginalize out the inducing variables of the p-th latent function from the approximate distribution (4.9). Thus, we obtain

$$q(\mathbf{h}_p) = \int p(\mathbf{h}_p|\mathbf{u}_p)q(\mathbf{u}_p)d\mathbf{u}_p = \mathcal{N}(\mathbf{h}_p|\mathbf{m}_p^h, S_p^h), \tag{4.11}$$

where

$$\mathbf{m}_p^h = Q_{XZ}\mathbf{m}_p, \tag{4.12}$$

$$S_p^h = K_X - Q_{XZ}K_{ZX} + R_{XZ}^p R_{XZ}^{p^\top}, \tag{4.13}$$

and $Q_{XZ} = K_{XZ}K_Z^{-1}$ and $R_{XZ}^p = Q_{XZ}L_p$.

On the other hand, in the case of choosing to parametrize the variational distribution using the parsimonious scheme, then the pair $(\mathbf{m}_p, S_p)$ is parametrized as follows,

$$\mathbf{m}_p = K_Z \boldsymbol{\mu}_p, \tag{4.14}$$

$$S_p = (K_Z^{-1} + \Sigma_p^{-1})^{-1} = K_Z - K_Z(K_Z + \Sigma_p)^{-1} K_Z, \tag{4.15}$$

where the second equality of Eq. (4.15) is due to application of the Woodbury matrix identity (Eq. (B.3) in Appendix) and $\Sigma_p \in \mathbb{R}^{M \times M}$ is a diagonal matrix with positive entries in its main diagonal. Hence, the moments of 4.10 are now given by

$$\mathbf{m}_p^h = K_{XZ} \boldsymbol{\mu}_p, \tag{4.16}$$

$$S_p^h = K_X - K_{XZ}(K_Z + \Sigma_p)^{-1} K_{ZX}. \tag{4.17}$$

Having all the appropriate mathematical tools, we are now able to introduce the complete derivation of the lower bound in the next section.

4.4.2 Derivation of the lower bound

To express the lower bound on the log marginal likelihood $\log p(Y)$ under the variational distribution in (4.9) we follow similar derivation steps as in Chapter 3, Section 3.5, which is mainly based on Titsias (2009) and Hensman et al. (2015b) work.

We start our derivation by expressing the joint density as

$$\prod_{i=1}^{N} p(\mathbf{y}^{(i)} | \mathbf{h}^{(i)}) \prod_{p=1}^{P} p(\mathbf{h}_p | \mathbf{u}_p) p(\mathbf{u}_p). \tag{4.18}$$

Our main goal again is to approximate the true augmented posterior density $\hat{P} \equiv p(\{\mathbf{h}_p, \mathbf{u}_p\}_{p=1}^{P} | Y)$ with some computationally "convenient" distribution. In other words, we want a distribution that factorizes across the latent factors. That is exactly the reason we defined our density of the variational distribution in (4.9) as it is. In order to find the best possible variational parameters such that the approximation to the true posterior is tight enough, we have to minimize the KL divergence $\mathrm{KL}[Q||\hat{P}]$. As it is already presented in the Chapters 2 and 3, this task is equivalently expressed as the maximization of the following lower bound on the log marginal likelihood $\log p(Y)$,

$$\begin{aligned}
&\mathbb{E}_Q\left[\log\frac{\prod_{i=1}^N p(\mathbf{y}^{(i)}|\mathbf{h}^{(i)})\prod_{p=1}^P p(\mathbf{h}_p|\mathbf{u}_p)p(\mathbf{u}_p)}{\prod_{p=1}^P p(\mathbf{h}_p|\mathbf{u}_p)q(\mathbf{u}_p)}\right]\\
=&\mathbb{E}_Q\left[\log\frac{\prod_{i=1}^N p(\mathbf{y}^{(i)}|\mathbf{h}^{(i)})\prod_{p=1}^P p(\mathbf{u}_p)}{\prod_{p=1}^P q(\mathbf{u}_p)}\right]\\
=&\sum_{i=1}^N \mathbb{E}_Q\left[\log p(\mathbf{y}^{(i)}|\mathbf{h}^{(i)})\right]-\sum_{p=1}^P \mathbb{E}_Q\left[\log\frac{q(\mathbf{u}_p)}{p(\mathbf{u}_p)}\right]. \qquad (4.19)
\end{aligned}$$

The factorized nature of Q, given by (4.9), requires only the presence of $\mathbf{u}_p$ to compute each of the P expectations in the second sum. Therefore we can re-write each of those terms as

$$\mathbb{E}_Q\left[\log\frac{q(\mathbf{u}_p)}{p(\mathbf{u}_p)}\right]=\mathbb{E}_{q(\mathbf{u}_p)}\left[\log\frac{q(\mathbf{u}_p)}{p(\mathbf{u}_p)}\right], \qquad (4.20)$$

where the right-hand side of Eq. (4.20) can be recognised as the KL divergence $\mathrm{KL}[q(\mathbf{u}_p)||p(\mathbf{u}_p)]$. Shifting our attention now towards the i^{th} likelihood term in the first sum, we can expand $\log p(\mathbf{y}^{(i)}|\mathbf{h}^{(i)})$ using the conditional independence of the labels given the latent function values, which decomposes as

$$\log p(\mathbf{y}^{(i)}|\mathbf{h}^{(i)})=\sum_{k=1}^K \log\sigma(y_k^{(i)}f_k^{(i)})=-\sum_{k=1}^K \log(1+e^{-y_k^{(i)}f_k^{(i)}}), \qquad (4.21)$$

where $f_k^{(i)}=\sum_{p=1}^P \phi_{kp}h_p^{(i)}+b_k$ is a scalar random variable. However, we have already showed in (4.10) that the density of the marginal variational distribution is a multivariate normal one; thus,

$$q(f_k^{(i)})=\mathcal{N}(f_k^{(i)}|\sum_{p=1}^P \phi_{kp}m_p^{(i)}+b_k,\sum_{p=1}^P \phi_{kp}^2 s_p^{(i)}). \qquad (4.22)$$

In the case of the PP the scalar mean $m_p^{(i)}$ can be computed by the i^{th} entry of $\mathbf{m}_p^h$ given by (4.16) and similarly, the variance $s_p^{(i)}$ can be computed by the i^{th} entry of the main diagonal of S_p^h in (4.17), i.e.

$$m_p^{(i)}=\mathbf{k}(\mathbf{x}^{(i)},Z)\boldsymbol{\mu}_p, \qquad (4.23)$$

$$s_p^{(i)}=\mathbf{k}(\mathbf{x}^{(i)},\mathbf{x}^{(i)})-\mathbf{k}(\mathbf{x}^{(i)},Z)(K_Z+\Sigma_p)^{-1}\mathbf{k}(Z,\mathbf{x}^{(i)}), \qquad (4.24)$$

while in the case of the FP, we make use of the suitable equations, (4.12) and (4.13), for the mean and variance respectively, which gives us

$$m_p^{(i)} = Q_{iZ}\mathbf{m}_p, \tag{4.25}$$
$$s_p^{(i)} = \mathbf{k}(\mathbf{x}^{(i)}, \mathbf{x}^{(i)}) - Q_{iZ}\mathbf{k}(Z, \mathbf{x}^{(i)}) + R_{iZ}^p R_{iZ}^{p\top}. \tag{4.26}$$

where Q_{iZ} and R_{iZ}^p are the i^{th} row of Q_{XZ} and R_{XZ}^p respectively. Therefore, the whole data reconstruction term $\sum_{i=1}^N \mathbb{E}_Q \left[\log p(\mathbf{y}^{(i)}|\mathbf{h}^{(i)})\right]$ simplifies to

$$-\sum_{i=1}^N \sum_{k=1}^K \mathbb{E}_{q(f_k^{(i)})} \left[\log(1 + e^{-y_k^{(i)} f_k^{(i)}})\right]. \tag{4.27}$$

Finally, combining (4.20) and (4.27), we can obtain the final form of the lower bound, which is

$$\mathcal{F} = -\sum_{i=1}^N \sum_{k=1}^K \mathbb{E}_{q(f_k^{(i)})} \left[\log(1 + e^{-y_k^{(i)} f_k^{(i)}})\right] - \sum_{p=1}^P \mathrm{KL}[q(\mathbf{u}_p) \mid\mid p(\mathbf{u}_p)]. \tag{4.28}$$

Having derived now the lower bound, we continue our discussion in next section on how we can efficiently calculate it in a computationally scalable way, suitable for dealing with large-scale datasets.

4.4.3 Computing the lower bound

The lower bound in (4.28) can be divided into two main quantities of interest. These are the double-nested sum of expectations and the sum of the KL-divergence terms. Regarding the first one, we have already seen that all the expectations are with respect to the univariate normal distributions $q(f_k^{(i)})$ in (4.22). Therefore, in a similar manner as 3.4, all expectations over the likelihood terms reduce to performing NK one-dimensional integrals under Gaussian distributions and each such integral can be accurately approximated by Gauss-Hermite quadrature using a few quadrature points. In practice, we make use of ten quadrature points for all of our experiments.

Regarding the KL divergence term of the lower bound in the second line of Eq. (4.28), its computation depends on the choice of the parametrization. Choosing a FP, it leads to the following formula

$$\mathrm{KL}^{\mathrm{full}}[q(\mathbf{u}_p) \;||\; p(\mathbf{u}_p)] = \frac{1}{2}\mathbf{m}_p^{\top} K_Z^{-1} \mathbf{m}_p + \frac{1}{2}\operatorname{tr}\left(K_Z^{-1} S_p\right) + \frac{1}{2}\log|K_Z| - \frac{1}{2}\log|S_p| - \frac{M}{2}. \tag{4.29}$$

In a same manner, for the parsimonious parametrization we obtain

$$\mathrm{KL}^{\mathrm{pars}}[q(\mathbf{u}_p) \;||\; p(\mathbf{u}_p)] = \frac{1}{2}\boldsymbol{\mu}_p^{T} K_Z \boldsymbol{\mu}_p - \frac{1}{2}\operatorname{tr}\left((\Sigma_p + K_Z)^{-1} K_Z\right) + \frac{1}{2}\log|\Sigma_p + K_Z| - \frac{1}{2}\log|\Sigma_p|, \tag{4.30}$$

where in both cases equations (2.21) and (2.25) from Chapter 2 are utilized.

At this point, we try to elaborate the strengths and weaknesses of each of the parametrization, when applied to the computation of the lower bound in (4.28). Apparently, the parsimonious parametrization, as the name indicates, has the storage advantage of the full one. This means that we only need $\mathcal{O}(PM)$ variational parameters to define all the P densities $q(\mathbf{u}_p)$ of the variational distributions, in contrast to the FP, where space complexity is one order higher, i.e. $\mathcal{O}(PM^2)$. This can be very useful in cases when we wish to increase the number of the latent factors in order to boost up the flexibility of our model. Moreover, as we already stressed in Chapter 2, Section 2.4, the parsimonious parametrization cancels all the inverses and determinants of K_Z; thus, there is no need of computing its Cholesky decomposition which can be unstable sometimes, contrary to the diagonal inflated matrices $K_Z + \Sigma_p$. That is exactly the reason why we have to add a tiny jitter (e.g. 10^{-6}) to the main diagonal of K_Z when we use the FP, otherwise the eigenvalues of K_Z can be very close to zero, rendering the optimization then infeasible. In the case of the FP, instead of adding a fixed jitter term, we can equivalently consider to corrupt the latent functions with extra noise where its variance can be learnt throughout the optimization of the lower bound as a kernel hyperparameter[2].

Another interesting aspect of both the parametrizations, is the fact that they can efficiently compute the P KL-divergences, having calculated the necessary Cholesky decompositions in advance. More accurately, using the PP and having pre-computed the P Cholesky decompositions $\hat{L_p}$ of each of the $K_Z + \Sigma_p$, then the trace term in 4.30 can be computed by using backward substitution to invert the $\hat{L_p}$ and then the computation is straightforward. Moreover, the other two log determinants can be easily calculated by adding the logarithm of the values in the main diagonal for each of the matrices $\hat{L_p}$ and Σ_p. In the same fashion, the pre-computation of the Cholesky factor L of K_Z combined with the fact that we

[2]This is exactly the same as adding an extra White kernel to the already used one $\mathbf{k}(\cdot,\cdot)$.

Method	Time Complexity	Storage
PF	$\mathcal{O}(\lvert\mathcal{B}\rvert KP + \lvert\mathcal{B}\rvert PM^2 + M^3)$	$\mathcal{O}(PM^2 + MD)$
PP	$\mathcal{O}(\lvert\mathcal{B}\rvert KP + \lvert\mathcal{B}\rvert PM^2 + PM^3)$	$\mathcal{O}(PM + MD)$

Table 4.1: Time complexity and storage requirements for the two different parametrizations needed for optimizing (4.28). M is the number of inducing inputs, K the number of classes, and $|\mathcal{B}|$ the minibatch size. Negative subsampling and the effect of the input dimensionality are not considered here.

only keep the lower triangular matrix L_p of each variational covariance matrix, allows us to efficiently calculate all the remaining terms of each KL divergence.

Finally, we are ready to compute the whole bound. Assuming that we parametrize our P variational distributions according to the FP, we need firstly to perform one Cholesky decomposition of the matrix K_Z that overall scales as $\mathcal{O}(M^3)$ and allows us to fully calculate the sum of the KL divergence terms in the second line in (4.28) as we showed before. Then, with this Cholesky decomposition precomputed, for each i-th data point we need to compute $(\mu_p^{(i)}, s_p^{(i)})_{p=1}^P$, an operation that scales as $\mathcal{O}(PM^2)$, and subsequently calculate the K variational distributions (i.e. their means and variances) over the utility scores in (4.22) which requires additional $\mathcal{O}(KP)$ time. Therefore, in order to compute the whole data reconstruction term of the bound (first line in Eq. (4.28)) we need $\mathcal{O}(NKP + NPM^2)$ time and for the full bound we need $\mathcal{O}(NKP + NPM^2 + M^3)$ time. Given that $N \gg M$ and $K \gg P$, the terms that can dominate are either $\mathcal{O}(NKP)$ or $\mathcal{O}(NPM^2)$ which can make the computations very expensive when the number of data instances and/or labels is very large. Using the same arguments, we can find similar computational complexity of the PP as FP; however, we need here to compute P Cholesky decompositions, rendering the computation of the bound to scale as $\mathcal{O}(NKP + NPM^2 + PM^3)$. As we see in the next section, those extra $P - 1$ Cholesky decompositions impose a practical hindrance of the PP on the lower bound computation which is proved of vital importance in the coming experiments section. All the time and space complexities of the two parametrizations are summarized in Table 4.1.

4.4.4 Scalable Training using Stochastic Optimization

To ensure that the time complexity $\mathcal{O}(NKP + NPM^2 + M^3)$ and $\mathcal{O}(NKP + NPM^2 + PM^3)$ for the FP and PP respectively, is reduced to $\mathcal{O}(PM^3)$ when it comes to very large datasets, we optimize the bound using stochastic gradient ascent by following a similar procedure used in stochastic variational inference for GPs Hensman et al. (2013). Given that the sum of KL divergences in (4.28) is already within the desired complexity $\mathcal{O}(PM^3)$, we only need to speed up the

Method	Time Complexity
Full	$\mathcal{O}(\lvert\mathcal{B}\rvert KP + \lvert\mathcal{B}\rvert PM^2 + M^3 + DM^2 + \lvert\mathcal{B}\rvert DM)$
SII	$\mathcal{O}(\lvert\mathcal{B}\rvert KP + \lvert\mathcal{B}\rvert PM^2 + M^3 + \mathcal{R}M^2 + \lvert\mathcal{B}\rvert \mathcal{R}M)$
SII+Basis	$\mathcal{O}(\lvert\mathcal{B}\rvert KP + \lvert\mathcal{B}\rvert PM^2 + M^3 + M\mathcal{R}^2 + DM^2 + D\mathcal{R}^2 + \lvert\mathcal{B}\rvert D\mathcal{R} + \lvert\mathcal{B}\rvert \mathcal{R}M)$

Method	Storage
Full	$\mathcal{O}(PM^2 + MD)$
SII	$\mathcal{O}(PM^2 + M\mathcal{R} + \mathcal{R}D)$
SII+Basis	$\mathcal{O}(PM^2 + M\mathcal{R} + \mathcal{R}D)$

Table 4.2: Time complexity and storage requirements for the case of optimizing (i) Full inducing inputs Z (Full), (ii) Subspace inducing inputs with fixed $\widetilde{X}$ (SII), and (iii) both Subspace inducing inputs and basis $\widetilde{X}$ (SII+Basis). M is the number of inducing inputs, K the number of classes, and $|\mathcal{B}|$ the minibatch size. Negative subsampling is not considered here.

remaining data reconstruction term. This term involves a double sum over data instances and class labels, a setting suitable for stochastic approximation. Thus, a straightforward procedure is to uniformly sub-sample terms in the double sum in (4.28) which leads to an unbiased estimate of the bound and its gradients. It turns out that we can further reduce the variance of this basic strategy by applying a more stratified sub-sampling over class labels as discussed next.

Suppose $\mathcal{B} \subset \{1, \ldots, N\}$ denotes the current minibatch at the t^{th} iteration of stochastic gradient ascent. For each $i \in \mathcal{B}$ the internal sum over class labels can be written as

$$-\sum_{k \in \mathcal{P}_i} \mathbb{E}_{q(f_k^{(i)})} \log(1 + e^{-f_k^{(i)}}) - \sum_{\ell \in \mathcal{N}_i} \mathbb{E}_{q(f_\ell^{(i)})} \log(1 + e^{f_\ell^{(i)}}), \tag{4.31}$$

where $\mathcal{P}_i = \{k | y_k^{(i)} = 1\}$ is the set of present or positive labels of $\mathbf{x}^{(i)}$ while $\mathcal{N}_i = \{k | y_k^{(i)} = -1\}$ is the set of absent or negative labels such that $\mathcal{P}_i \cup \mathcal{N}_i = \{1, \cdots, K\}$. Multi-label classification problems most of the times are characterized by a very small size of positive labels $\mathcal{P}_i$ compared to the negative set which can be extremely large (Zhang and Zhou, 2013; Gibaja and Ventura, 2014, 2015). Thus, we can enumerate exactly the first sum and use (if needed) sub-sampling to approximate the second sum over the negative labels. The whole process becomes somehow similar to negative sampling used in large scale classification and for learning word embeddings Mikolov et al. (2013). Overall, we get the following unbiased stochastic estimate of the lower bound,

$$- \frac{N}{|\mathcal{B}|} \sum_{i \in \mathcal{B}} \left[\sum_{k \in \mathcal{P}_i} \mathbb{E}_{q(f_k^{(i)})} \log(1 + e^{-f_k^{(i)}}) + \frac{|\mathcal{N}_i|}{|\mathcal{L}_i|} \sum_{\ell \in \mathcal{L}_i} \mathbb{E}_{q(f_\ell^{(i)})} \log(1 + e^{f_\ell^{(i)}}) \right]$$
$$- \sum_{p=1}^{P} \mathrm{KL}[q(\mathbf{u}_p)||p(\mathbf{u}_p)], \tag{4.32}$$

where $\mathcal{L}_i$ is the set of negative classes for the i-th data point. In general, the computation of this stochastic bound scales as $\mathcal{O}(|\mathcal{B}|(|\mathcal{P}_i|+|\mathcal{L}_i|)P+|\mathcal{B}|PM^2+M^3)$ for the FP and $\mathcal{O}(|\mathcal{B}|(|\mathcal{P}_i|+|\mathcal{L}_i|)P+|\mathcal{B}|PM^2+PM^3)$ for the PP. Therefore, by choosing $|\mathcal{B}| \sim \mathcal{O}(M)$ and $|\mathcal{P}_i|+|\mathcal{L}_i| \sim \mathcal{O}(M^2)$ we can ensure that the overall time is $\mathcal{O}(PM^3)$. Notice that the second condition is not that restrictive at all and in many cases might not be needed. In practice, all the experiments we carry out, i.e. use as negative sets $\mathcal{L}_i$ the very full negative set $\mathcal{N}_i$, reducing the variance coming from the sub-sampling of the negative labels.

Although we showed that the computation of the lower bound asymptotically scales as $\mathcal{O}(PM^3)$ for both PP and FP, in practice, there is a considerable computational speed up of FP over PP. This can be justified by the extra $P-1$ expensive Cholesky decompositions of PP, that make the computation of the bound and its gradients slower, which seriously limits us to use a larger number of latent factors P to improve performance. This argument is also supported later where experiments reveal the computational burden of PP and its inferior performance comparing to FP.

We implemented the above stochastic bound in Python in order to jointly optimize using stochastic gradient ascent and automatic differentiation tools. Regarding the automatic differentiation tools, we employed both Tensorflow and Autograd (Maclaurin et al., 2015). Autograd was used at the beginning of this PhD, since Tensorflow was still in its infancy at that time. However, after a while, when Tensorflow applied successfully in many machine learning models including GPs, we switched our python implementation to Tensorflow, enabling to speed up all the computations required for the optimization of the bound while the efficiency of the package based on Tensors and computational graphs, allowed us to run experiments on datasets that would be an insurmountable task in the case of Autograd. Therefore we provide experimental results for both the packages. For a more detailed look in the code used for the experiments for both of the packages, we refer the reader to Appendix C. Lastly, Table 4.3 provides all the parameters that we need to optimize the bound accompanied by their corresponding dimensionality.

Parameters	Dimensionality(in $\mathbb{R}$)	FP	PP	Full	Sub	Sub & Basis
Φ	$K \times P$					
M_q	$P \times M$					
Σ_q	$P \times M$	✗	✓			
L_q	$P \times \frac{M^2+M}{2}$	✓	✗			
Z	$M \times D$			✓	✗	✗
A	$M \times \mathcal{R}$			✗	✓	✓
$\widetilde{X}$	$\mathcal{R} \times D$			✗	✗	✓
$\boldsymbol{\theta}$	kernel depending					
$\mathbf{b}$	K					

Table 4.3: Summary of the parameters to be optimized where different cases are considered, such as parametrization of the variational distributions or subspace inducing inputs use. Blanks indicate that the corresponding parameters are independent from the choice of the specific method while ✓and ✗ indicate presence and absence, respectively, of the given scheme.

4.4.5 Prediction

Given a novel data point $\mathbf{x}^{(*)}$ we would like to make prediction over its unknown label vector $\mathbf{y}^{(*)}$. This requires approximating the predictive distribution $p(\mathbf{y}^{(*)}|Y)$,

$$p(\mathbf{y}^{(*)}|Y) \approx \int p(\mathbf{y}^{(*)}|\mathbf{u}^{(*)})q(\mathbf{u}^{(*)})d\mathbf{u}^{(*)}. \tag{4.33}$$

Here, $q(\mathbf{u}^{(*)})$ is the variational predictive posterior over the latent function values $\mathbf{u}^{(*)}$ evaluated at $\mathbf{x}^{(*)}$. An interesting aspect of the variational sparse GP method of Titsias (2009) is that to obtain $q(\mathbf{u}^{(*)})$ we need to make no further approximations since everything follows from the GP consistency property, i.e.

$$\begin{aligned} q(\mathbf{u}^{(*)}) &= \prod_{p=1}^{P} \int p(u_p^{(*)}|\mathbf{h}_p, \mathbf{u}_p)p(\mathbf{h}_p|\mathbf{u}_p)q(\mathbf{u}_p)d\mathbf{h}_p d\mathbf{u}_p \\ &= \prod_{p=1}^{P} \int p(u_p^{(*)}|\mathbf{u}_p)q(\mathbf{u}_p)d\mathbf{u}_p = \prod_{p=1}^{P} q(u_p^{(*)}). \end{aligned} \tag{4.34}$$

Here, GP consistency given by Equations (A.3) and (A.4), tractably simplifies each integral

$$\int p(u_p^{(*)}|\mathbf{h}_p, \mathbf{u}_p)p(\mathbf{h}_p|\mathbf{u}_p)d\mathbf{h}_p = p(u_p^{(*)}|\mathbf{u}_p) \tag{4.35}$$

so that the obtained $p(u_p^{(*)}|\mathbf{u}_p)$ is the conditional GP prior of $u_p^{(*)}$ given the inducing variables. The final form of each univariate Gaussian $q(u_p^{(*)})$ has a mean and variance given precisely by equations (4.12) and (4.13) for FP or (4.16) and (4.17), with X replaced by $\mathbf{x}^{(*)}$. In practice, when we compute several accuracy ranking-based scores that are often used in the literature to report multi-label classification performance (Zhang and Zhou, 2013; Gibaja and Ventura, 2014, 2015) it suffices to further approximate $q(\mathbf{u}^{(*)})$ by a delta mass centred at the MAP. This reduces the whole computation of such scores to only requiring the evaluation of the mean utility vector $\bar{\mathbf{f}}^{(*)} = [\bar{f}_1^{(*)} \dots \bar{f}_K^{(*)}]^\top$ such that

$$\bar{f}_k^{(*)} = \sum_{p=1}^{P} \phi_{kp} m_p^{(*)} + b_k, \tag{4.36}$$

where $\mathbf{k}(\mathbf{x}^{(*)}, Z)$ is the cross covariance row vector between $\mathbf{x}^{(*)}$ and the inducing points Z. Depending on the parametrization, we have $m_p^{(*)} = \mathbf{k}(\mathbf{x}^{(*)}, Z)K_Z^{-1}\mathbf{m}_p$ for the FP and $m_p^{(*)} = \mathbf{k}(\mathbf{x}^{(*)}, Z)\boldsymbol{\mu}_p$ for PP.

Having computed the utility vector $\bar{\mathbf{f}}^{(*)}$, we can use it to predict the k most probable labels of the input vector $\mathbf{x}^{(*)}$ in tandem with its ranking score $P@k$ introduced in Section 1.2.1 of Chapter 1.

Furthermore, our probabilistic framework can capture correlations between the different labels, a property that can be useful when we wish to predict partially observed label vectors. More precisely, for the novel data point $\mathbf{x}^{(*)}$ assume that we partially observe its label vector $\mathbf{y}^{(*)} = (\mathbf{y}_o^{(*)}, \mathbf{y}_m^{(*)})$ so that $\mathbf{y}_o^{(*)}$ corresponds to the observed labels and $\mathbf{y}_m^{(*)}$ to the missing labels. In this setting the approximate predictive distribution over $\mathbf{y}_m^{(*)}$ we wish to compute is

$$p(\mathbf{y}_m^{(*)}|\mathbf{y}_o^{(*)}, Y) \approx \int p(\mathbf{y}_m^{(*)}|\mathbf{u}^{(*)})q(\mathbf{u}^{(*)}|\mathbf{y}_o^{(*)})d\mathbf{u}^{(*)}, \tag{4.37}$$

where $q(\mathbf{u}^{(*)}|\mathbf{y}_o^{(*)}) \propto p(\mathbf{y}_o^{(*)}|\mathbf{u}^{(*)})q(\mathbf{u}^{(*)})$ and $q(\mathbf{u}^{(*)})$ (given by (4.34)) acts now as a prior in this new posterior that needs to accommodate also for the information coming from the observed $\mathbf{y}_o^{(*)}$. Again to get fast estimates of ranking scores we can rely on the MAP estimate of $q(\mathbf{u}^{(*)}|\mathbf{y}_o^{(*)})$ that requires few gradient-based optimization steps to maximize over $\mathbf{u}^{(*)}$. Other more elaborate posterior estimates, e.g. a variational approximation to $q(\mathbf{u}^{(*)}|\mathbf{y}_o^{(*)})$, are also possible but we did not pursue any further research on that direction.

4.4.6 Optimality of the PP

Before start discussing about the conducted experiments, we provide the proof of the optimality of PP. Opper and Archambeau have already proved that the PP in the case of a single latent function is able to recover the optimal variational distribution if we use as inducing inputs the whole design matrix X. We generalize this result by considering the density $\prod_{p=1}^{P} q(\mathbf{h}_p)$ of the factorized variational distribution. Thus, we have the following proposition,

Proposition 1. *The density $\prod_{p=1}^{P} q(\mathbf{h}_p)$ of the factorized variational distribution, parametrized by* (4.15) *can recover the covariance matrix of the optimal factorized distribution with density $\prod_{p=1}^{P} q^*(\mathbf{h}_p)$.*

Proof. This proof relies on the derivation steps provided by Opper and Archambeau (2009). Assume that we have the density of the factorized variational distribution

$$\prod_{p=1}^{P} q(\mathbf{h}_p), \tag{4.38}$$

where $q(\mathbf{h}_p) = \mathcal{N}(\mathbf{h}_p|\mathbf{m}_p^h, S_p^h)$. The variational lower bound is

$$-\sum_{i=1}^{N}\sum_{k=1}^{K} \mathbb{E}_{q(f_k^{(i)})}[\log(1 + e^{-y_k^{(i)} f_k^{(i)}})] - \sum_{p=1}^{P} \mathrm{KL}[q(\mathbf{h}_p)||p(\mathbf{h}_p)],$$

where each KL divergence term is given by

$$\begin{aligned}\mathrm{KL}[q(\mathbf{h}_p)||p(\mathbf{h}_p)] = \frac{1}{2}[&\mathrm{tr}\left(K_X^{-1}(S_p^h + \mathbf{m}_p^h \mathbf{m}_p^{h\top})\right) \\ &+ \log|K_X| - \log|S_p^h| - N].\end{aligned} \tag{4.39}$$

Rewriting now the bound by defining the term

$$V_i = \sum_{k=1}^{K} \mathbb{E}_{q(f_k^{(i)})}\left[\log(1 + e^{-y_k^{(i)} f_k^{(i)}})\right], \quad i = 1, \cdots, N, \tag{4.40}$$

we obtain

$$\mathcal{F} = -\sum_{i=1}^{N} V_i - \sum_{p=1}^{P} \mathrm{KL}[q(\mathbf{h}_p)||p(\mathbf{h}_p)]. \tag{4.41}$$

Notice that each term V_i is a sum of K univariate Gaussian expectations with respect to the marginal $q(f_k^{(i)}) = \mathcal{N}(f_k^{(i)}| \sum_{p=1}^{P} \phi_{kp} m_p^{(i)} + b_k, \sum_{p=1}^{P} \phi_{kp}^2 s_p^{(i)})$ which

Data set	D	K	N	N^*	$\overline{K}$
Bibtex	1836	159	4880	2515	2.40
Delicious	500	983	12920	3185	19.03
Mediamill	120	101	30993	12914	4.38
EUR-Lex	5000	3993	15539	3809	5.31
RCV1	47236	2456	623847	155962	4.79
Wiki10	101938	30938	14146	6616	18.64
AmazonCat	203882	13330	1186239	306782	5.04
Delicious-Large	782585	205443	196606	100095	75.54
WikiLSHTC	1617899	325056	1778351	587084	3.19

Table 4.4: Data sets statistics: N and N^* are the number of the training and test points respectively, D and K are the number of features and labels respectively, and $\overline{K}$ is the average number of positive labels in an instance.

means that these expectations depend only on the linear combination of the P means $m_p^{(i)}$ and the P variances $s_p^{(i)}$, i.e. the i^{th} diagonal elements of each covariance matrix $\mathbf{S}_p^h$.

Therefore, by differentiating the variational lower bound with respect to each S_p^h and setting it equal to zero we have for the covariance of the density of the optimal variational distribution $q^*(\mathbf{h}_p)$ that

$$\begin{aligned} \nabla_{S_p^h}\mathcal{F} &= -\sum_{i=1}^{N} \nabla_{S_p^h} V_i - \frac{1}{2}(K_X^{-1} - \mathbf{S}_p^h) = 0 \\ \Rightarrow \mathbf{S}_p^h &= (K_X^{-1} + \Lambda_p)^{-1}, \end{aligned} \tag{4.42}$$

where $\Lambda_p \in \mathbb{R}^{N \times N}$ is a diagonal matrix with positive entries $\lambda_p^{(i)} = 2\frac{\partial V_i}{\partial s_p^{(i)}}$ and for the right-hand side of the first line of the previous equation we made use of known matrix calculus identities (see identities B.1). □

We would like to also note that the above result can be similarly proven in the case where the PP is applied to a multi-class task, as we briefly mentioned in the previous chapter.

4.5 Experiments

Undoubtedly, the realization of this section possesses a considerable portion of time and endeavour throughout this 4-year PhD journey. For that reason, a larger amount of material is provided, compared to the previous chapter, accompanied by the corresponding discussion over the results presented. The section starts with a brief introduction and description of each of the datasets used. Then it continues with a significant number of results, produced by the MLGPF model, and various comparisons with state-of-the-art methods in the field of multi-label classification. Finally, we apply the proposed model to classification tasks using the MNIST dataset and compare its predictive performance to that of the RobustMax likelihood from the previous section.

4.5.1 Datasets

To test the performance of our model, we consider several datasets of different size in terms of (a) input dimensionality D, (b) label space dimensionality K, and (c) number of training instances N, where the vast majority of them include text data. More specifically, we chose to use both four small-scale and large-scale datasets where each of them is publicly available and can be found in Repository (2010) website.

Small-scale Datasets

- Bibtex (Katakis et al., 2008): The smallest dataset in terms of training data points. It contains metadata information for several bibtex items like paper's title, author's name, etc. The final dataset includes only binary vectors.

- Delicious (Tsoumakas et al., 2008): The data has been extracted by del.icio.us [3] social bookmarking web service. Each data point of this dataset corresponds to textual data coming form various bookmarks' web pages where the the label vectors indicate different tags for a specific bookmark. The input data are binary as before.

- Mediamill (Snoek et al., 2006): This is the only dataset in our collection that does not include text data. More precisely, it is formed by video data which are labelled by 101 different semantic concepts. Both feature and label dimensionality are the smallest ones across all datasets.

- EUR-Lex (Mencia and Fürnkranz, 2008): This time the data represents European Union Law documents where each of the labels classifies every text, based on a multilingual thesaurus. Despite its medium size in terms of training instances, this dataset would be considered as large-scale in many other applications.

- RCV1 (Lewis et al., 2004): That is the last of the small-scale datasets. Although its size resembles a large-scale dataset, as we observe in Table 4.4, it is justifiably considered a small one in the XML framework. The dataset consists of newswire stories in English language obtained by the news agency Reuters where each label can represent a specific topic, industry or region of a given story.

[3]`https://en.wikipedia.org/wiki/Delicious_(website)`

Large-scale Datasets

- Wiki10 (Zubiaga, 2012): The smallest of the large-scale datasets. It includes documents generated by various articles of the english version of Wikipedia in conjunction with several tags for each document. The difference between this dataset and the small-scale datasets, in terms of input and label space dimensionality, is conspicuous.

- AmazonCat (McAuley et al., 2015): This is another textual dataset, where at this time, documents represent reviews of Amazon products and they are mapped to a wide range of different Amazon product categories. More than a million documents are used for training.

- Delicious-Large (Wetzker et al., 2008): As the small-scale Delicious dataset, this is another example of bookmarking text data retrieved from the del.icio.us web service, however, in a much larger scale, and the second most challenging dataset after WikiLSHTC.

- WikiLSHTC (Partalas et al., 2015): Finally, we have the largest dataset that used in our experiments. The titanic size of the dataset stems from around two million Wikipedia documents which are labelled based on a set of more than 310^5 labels.

For summary statistics and size information of each dataset, the reader is referred to Table 4.4. As we can observe, even the small-scale datasets could be considered very challenging to be dealt with, let alone the large-scale ones. However, as someone would expect for textual data, all the described datasets above, except Mediamill, are consist of both sparse X and Y, where the most dense of them has no more than 7% non-zero values on average. This is just a consequence of using text data since each feature represents a word of a given dictionary that it is rare to appear in all documents of the corpus. Therefore, data sparsity allows us to store and handle those immerse datasets. Moreover, due to text nature of most of the datasets, they are all normalized in order each of the training instances has unit norm. This is very typical to text-based applications, since we are usually interested in the similarity of two documents and not their vector length, and there are examples in literature that normalizing the dataset does improve performance (Baroni et al., 2014).

Having now introduced and described all the datasets, it is more easily conceived the difficulty of our endeavour to cope with those extremely high-dimensional spaces of both features and labels. Taking also into account that the goal is to train a Bayesian model under these circumstances, then someone could justifiably argue that the whole attempt is insuperable. Nevertheless, as we see in the next section, the MLGPF model has not only achieved to be applied to all of the aforementioned datasets, but most importantly, it has managed to surpass in performance the majority of the state-of-the-art algorithms, rendering it, at the best of our knowledge, the first ever, not only GP-based, but more generally, Bayesian method that has been successfully used to solve problems of that scale.

4.5.2 Methods

Before we start discussing about the experimental results, we need to introduce the name abbreviation for each of the tested methods in order to make the next section more readable and less burdensome to the reader. All of the methods which is described here, are different combinations of kernel functions (e.g. linear, SE), variational distributions parametrization (PP or FP), implementations (Tensorflow or Autograd) etc, however they are all based on the MLGPF model. Therefore, we firstly present the logic of how its method's name is derived. More precisely we follow the name convention: *"kernel name"-"optimized subspace inducing inputs"-"optimized basis"-"fixed Z"-"unnormalized Z"-"Python implementation"* where **Kernel name** can be:

LINEAR: A linear kernel with no hyperparameters as defined in (1.9).

LINEAR-ℓ: A linear kernel with D lengthscales as hyperparameters, each for every input dimension.

SE: The well-known SE kernel with a single lengthscale as hyperparameter[4] as defined in (1.4).

SE-p-ℓ: The SE kernel where in that case P different lengthscales are used, each one for the P latent functions h_p.

ARD: The Automatic Relevance Determination kernel as defined in (1.4) using as distance measure Λ_2.

ARD/LINEAR: This is a method that uses as kernel function the sum of an ARD kernel with a Linear one.

For the second field the only possible value is "S" which denotes that the method optimizes the subspace inducing inputs while keeping the basis matrix $\widetilde{X}$ fixed. In case that $\widetilde{X}$ is optimized too, then an extra B is added to the end of the name. Furthermore, we consider the cases where the inducing inputs are kept fixed during the optimization process, denoted by "FX", and the inducing inputs are normalized at each optimization step such that each of them lies in the unit hyper-sphere, denoted by "UN". This implies that all the methods which their names do not include the "UN" string, normalize the inducing inputs each time they are updated. Finally, when the P variational distributions are parsimoniously parametrized and the method is implemented by the Autograd Python package then the letters "PP" and "AG" need to used as suffix to the method's name. To

[4]The reason that we do not consider the usual output variance σ_f^2 is the fact that it is redundant since its role is replaced by the entries of the factor loadings matrix Φ.

eliminate any remained confusion, we provide a few examples of methods' names, accompanied with a short description.

ARD: this method uses an ARD kernel where the full inducing inputs matrix Z is optimized with each row unit-normalized, a FP is employed and everything is implemented by Tensorflow.

ARD-S-AG: this method uses an ARD kernel where the subspace inducing inputs matrix A is optimized but the basis matrix $\widetilde{X}$ is precomputed and kept fixed, the rows of the matrix product $A\widetilde{X}$ are unit-normalized, a FP is deployed and everything is implemented by Autograd.

ARD-S-FX-PP-AG: this method uses an ARD kernel where both the subspace inducing inputs matrix A and the basis matrix $\widetilde{X}$ are kept fixed, a PP is employed and everything is implemented by Autograd.

Having described all the above methods, we are ready to move to the next section where all of them are tested on the datasets of Section 4.5.1 in order to gain a deeper insight of their advantages and disadvantages and their performance is compared with other baselines from the XML literature.

4.5.3 Experimental set-ups & Results

To begin with, all of our experiments run on an Intel Xeon Processor E5-2667 v3 server with 128G RAM, due to the scale of the datasets and the long time required to train the MLGPF model on them. Moreover, to test and compare predictive performance of different methods on a specific dataset, the precision metrics P@1, P@3, and P@5 (see Section 1.2.1 of Chapter 1 for a detailed description and the reason we choose this specific metric) while it is also demonstrated how well the lower bound is optimized during the training process. Further, it should be mentioned that choosing large size for the negative label set $\mathcal{L}_i$ can dramatically reduce variance in the stochastic optimization of the lower bound. Fortunately, the computational complexity analysis of Section 4.4.4 allows us to choose $\mathcal{L}_i$ such that $|\mathcal{L}_i| \sim \mathcal{O}(M^2)$ which is not restrictive at all, and in practice, for all datasets used in the experiments, $\mathcal{L}_i$ is selected to be equal to the size of the full negative set $\mathcal{N}_i$. Further, we would like to make clear that we avoid to use error bars for our P@k, $k = 1, 3, 5$, in many of the experiments of for two reasons. The first one is due to the training time limitations that the scale of the datasets impose where some of them require a few weeks training. Secondly, all the reported performance results from other state-of-the-art methods obtained by the XML repository (Repository, 2010) do not provide that information. Nevertheless, we include error bars in all of the small scale datasets by running each method for ten times using different random initializations. Finally, we define one epoch as a complete scan of the dataset based on the minibatch used, therefore one epoch corresponds to $\frac{N}{|\mathcal{B}|}$ iterations.

Method	LINEAR	LINEAR-ℓ	SE	SE-p-ℓ	ARD	ARD/LINEAR
P@1	57.85	61.27	59.88	58.41	62.98	62.70
P@3	34.90	37.51	36.34	34.84	38.36	38.46
P@5	25.83	27.49	26.97	26.08	28.22	28.35

Table 4.5: Results of using the MLGPF model in conjunction with different kernels for Bibtex dataset. All the P@k metrics (Eq. (1.21)) are computed after training our model with $M = 100$, $P = K = 159$, and epochs=50.

Various results based on Bibtex dataset

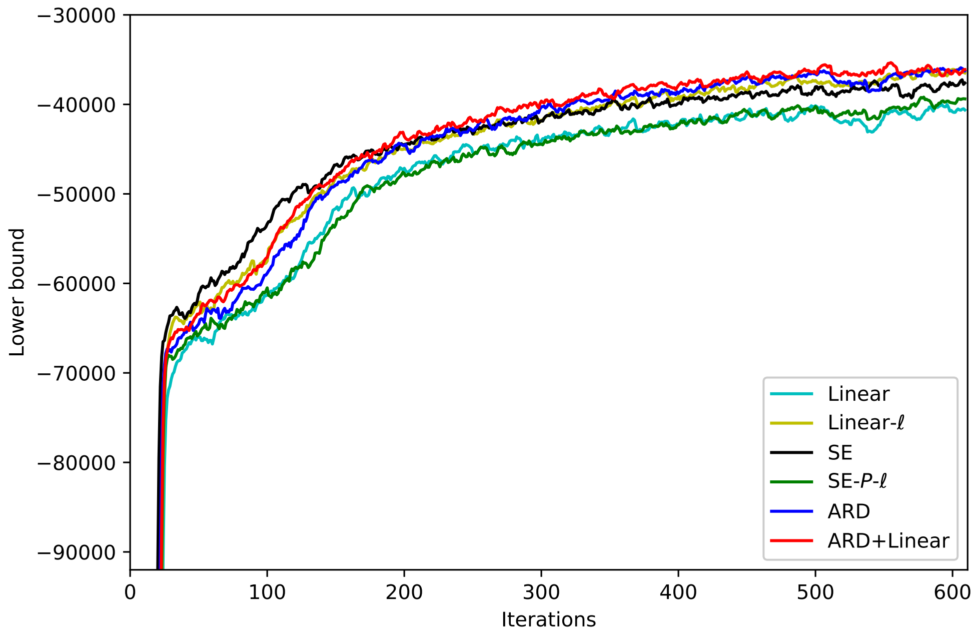
In this section we run various a number experiments on Bibtex using different methods in order to gain a better insight of how performance of the MLGPF varies under different used schemes. The specific dataset was chosen due to its relatively small size (Table 4.4) and its sparse high dimensional input space which is ubiquitous in all other datasets and thus, rendering its choice suitable for generalizing our conclusions. Regarding experimental set-ups, we use $M = 100$ inducing inputs and let the experiments run for 50 epochs. The reason we do not increase further the number of inducing inputs or epochs is that we primarily focus on the comparison of those methods and not in the achievement of the best possible performance.

Kernels comparison

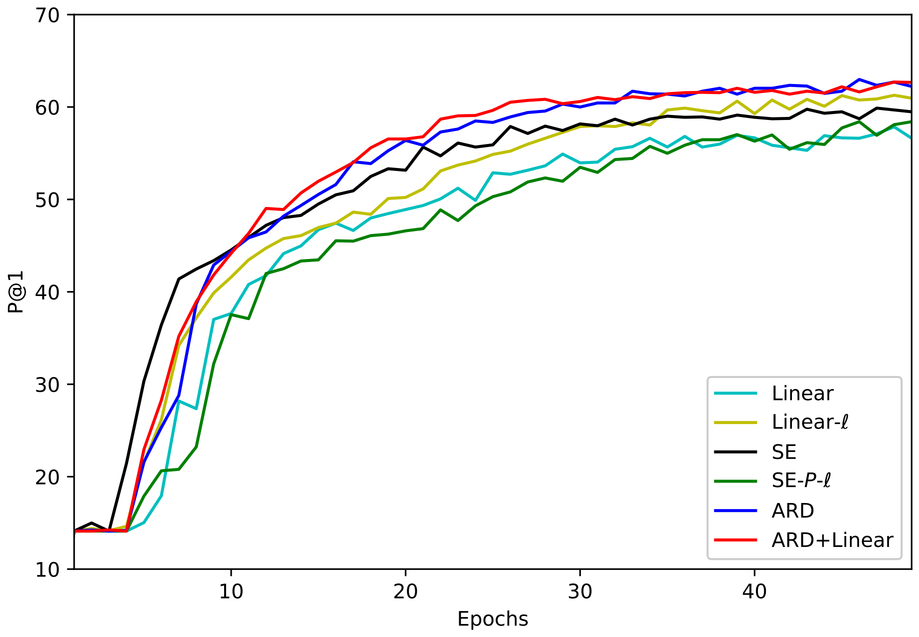
The experiments start with a thorough investigation of the effect that different kernels have on the performance of the MLGPF model. We run the LINEAR, LINEAR-ℓ, SE, SE-p-ℓ, and ARD, ARD/LINEAR method. By examining Table 4.5, it can be seen the superiority of ARD/LINEAR over the rest of the methods. It is also conspicuous how crucial is the addition of the extra D lengthscales as kernel hyperparamters to the overall performance of the MLGPF model regardless the choice of the kernel. For instance, a simple linear kernel gives P@1 = 57.85% accuracy while the same kernel, enhanced with D lengthscales, improves P@1 more than 3.4%. A similar behaviour is also observed in SE kernel. On top of that, Fig. 4.1 (a) shows how a well-maximized lower bound leads to a better predictive performance, revealing this positive correlation between lower bound maximization and predictive performance. Another interesting aspect that emerges by comparing the methods' performance is that despite the high dimensional input spaces where decision boundaries tend to become linearly separable, the SE kernel gives better results than the linear one.

Subspace inducing inputs comparison

The next experiments are conducted by using ARD, ARD-S, and the ARD-S-B methods. In this case, we would like to check how the performance varies de-

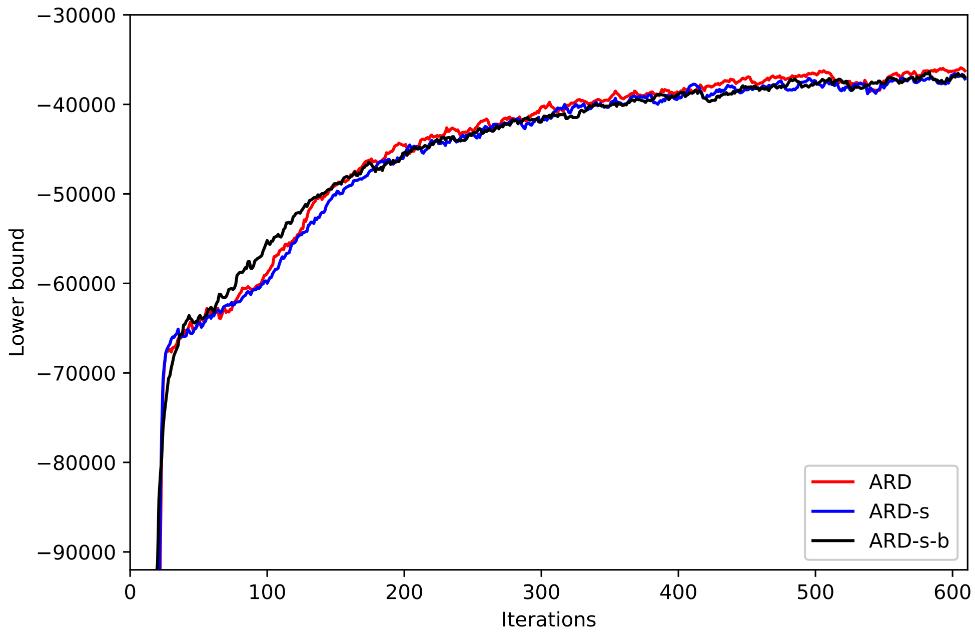


(a)

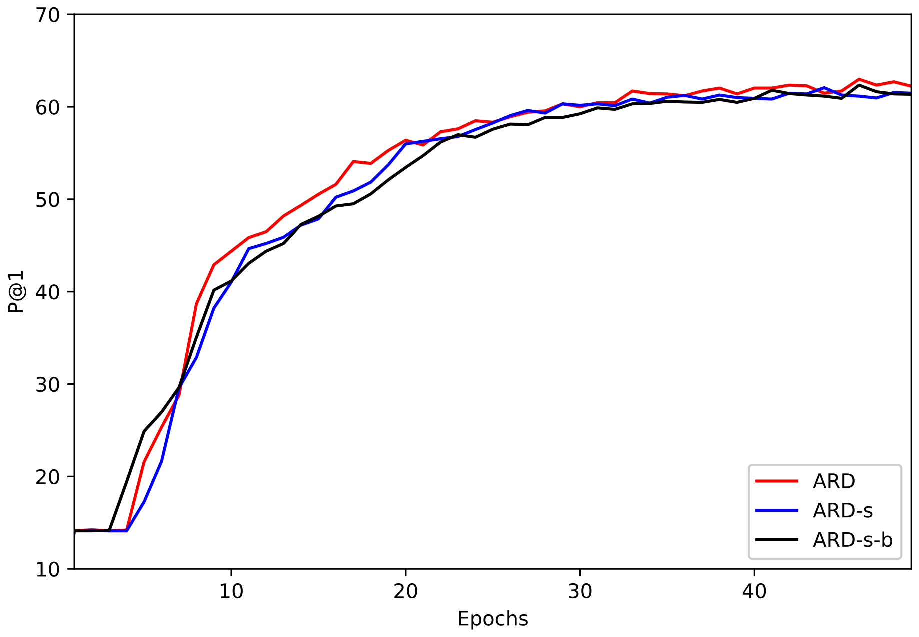


(b)

Figure 4.1: Performance comparison between various choices of kernels tested on Bibtex dataset. (a) shows the evolution of the lower bound per iteration, and (b) demonstrates how the P@1 metric varies across epochs for each kernel. Numerical results of (b) are demonstrated in Table 4.5.



(a)



(b)

Figure 4.2: Performance comparison between three different cases: (i) full inducing inputs optimization (red color line), (ii) subspace inducing inputs optimization and fixed basis (blue color line), (iii) optimization of both subspace inducing inputs and basis (black color line). All the cases are tested on Bibtex dataset using an ARD kernel. (a) shows the evolution of the lower bound per iteration, and (b) demonstrates how the P@1 metric varies across epochs for each kernel.

Method	ARD	ARD-S	ARD-S-B
P@1	62.98	62.07	62.35
P@3	38.36	38.09	37.68
P@5	28.22	27.98	27.79

Table 4.6: Numerical results of Fig. 4.2 (b). The performance metric Precision@k is used (Eq. (1.21)).

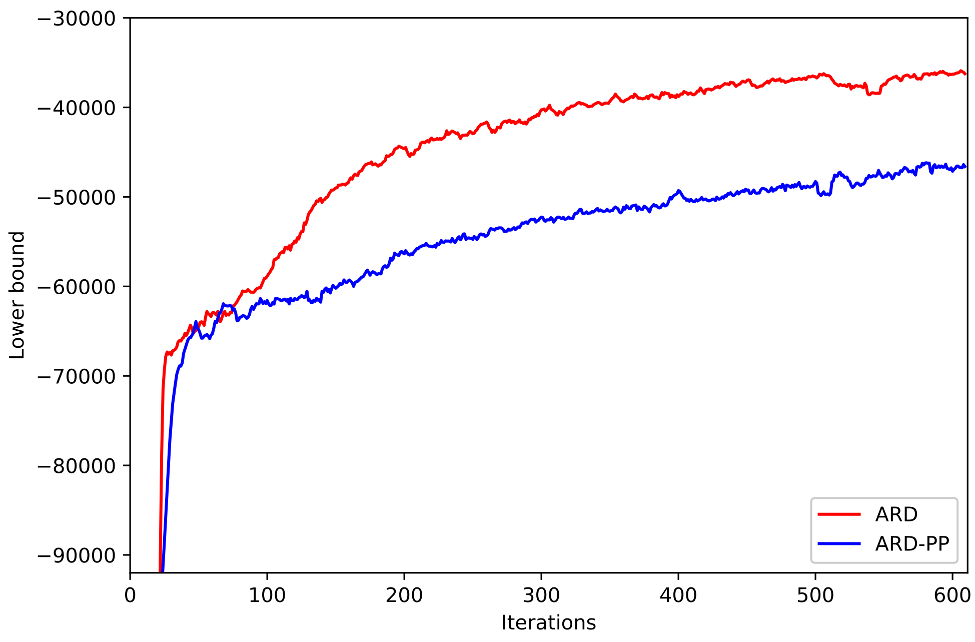
Method	ARD	ARD-PP
P@1	62.98	41.39
P@3	38.36	23.21
P@5	28.22	18.02

Table 4.7: Numerical results of Fig. 4.3 (b). The performance metric Precision@k is used (Eq. (1.21)).

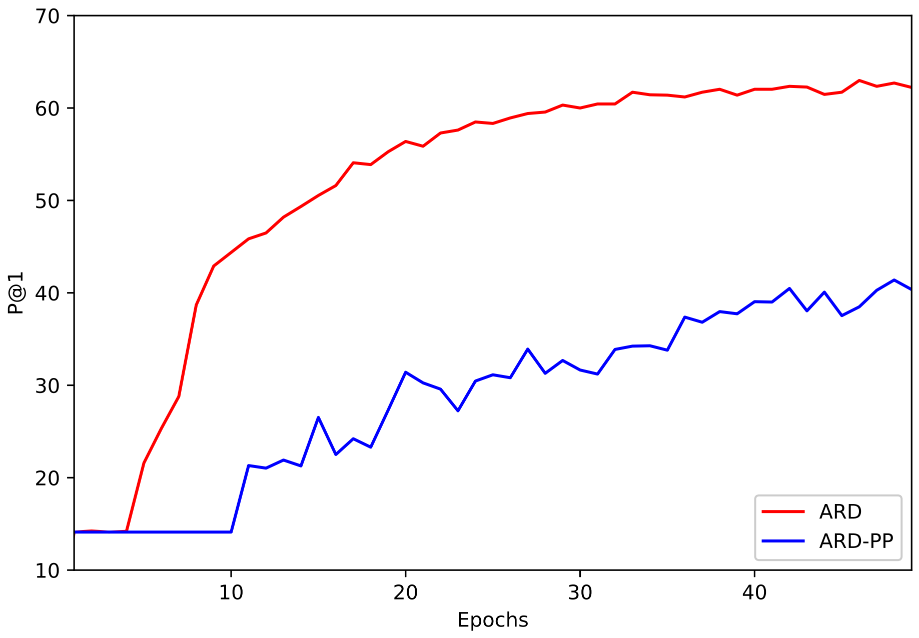
pending on the choice of optimizing the inducing inputs either in their original high-dimensional space or in lower dimensional subspace. The subspace inducing inputs are chosen to lie in the $\mathbb{R}^{500}$ and an ARD kernel is employed for all cases. We chose this number because more than 75% of dataset's variance is explained by the 500 largest singular values. The results in Table 4.6 reveal that optimizing the subspace inducing inputs gives marginally worse performance than optimizing Z. Moreover, the fact that we additionally optimized the basis $\widetilde{X}$, did not lead to better performance than keeping it fixed, on the contrary, in some cases the predictions were less accurately than ARD-S . Nevertheless, we see later that trying to choose the subspace inducing inputs dimensionality $\mathcal{R}$ such that $\mathcal{R} \sim \mathcal{O}(M)$ in order the overall complexity of the bound to scale as $\mathcal{O}(PM^3)$ (see Table 4.2), leads to poor predictive performance. The corresponding lower bounds and P@k , $k = 1, 2, 3$ as a function of iterations or epochs can be found in Fig. (a) and (b) of 4.2.

Parametrization comparison

The question of how each of the two parametrization schemes, FP and PP behaves for the MLGPF model naturally arises, and Table 4.7 in conjunction with Fig. 4.3 attempt to answer it. As usual, the experimental set-ups, the dataset tested, and the kernel function are all same as the previous paragraph. Fig. (b) 4.3 shows a clear superiority of the FP over the PP. This can be also verified by the maximization of the lower bound in Fig. (a) 4.3 where there is a wide gap between the plots of the two methods. Overall, the only benefit of the PP over the FP is the storage required for the variational parameters for each of the $q(\mathbf{u}_p), p =$



(a)



(b)

Figure 4.3: In those two plots, the performance of MLGPF model is compared with the **FP** (red color line) and **PP** (blue color line) using an ARD kernel. (a) shows the evolution of the lower bound, and (b) gives P@1 evolution across epochs for each parametrization.

Method	ARD-PP	ARD-PP-AG
P@1	41.39	1.31
P@3	23.21	1.06
P@5	18.02	1.03
time	110	648

Table 4.8: Numerical results of Fig. 4.4 (b) and running times (seconds/epoch) for the two implementations where the number of inducing inputs this time is set to 500. The performance metric Precision@k is used (Eq. (1.21)).

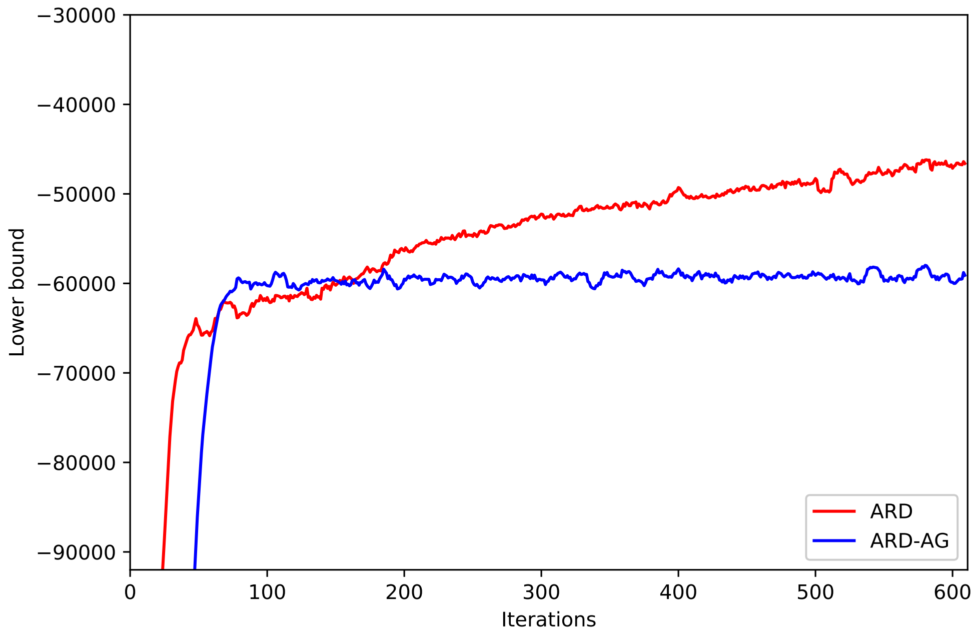
$1, \cdots, P$. Nonetheless, we never use more than 600 inducing inputs in practice, meaning that FP does not impose any practical restriction when it comes to storage requirements.

Implementation comparison

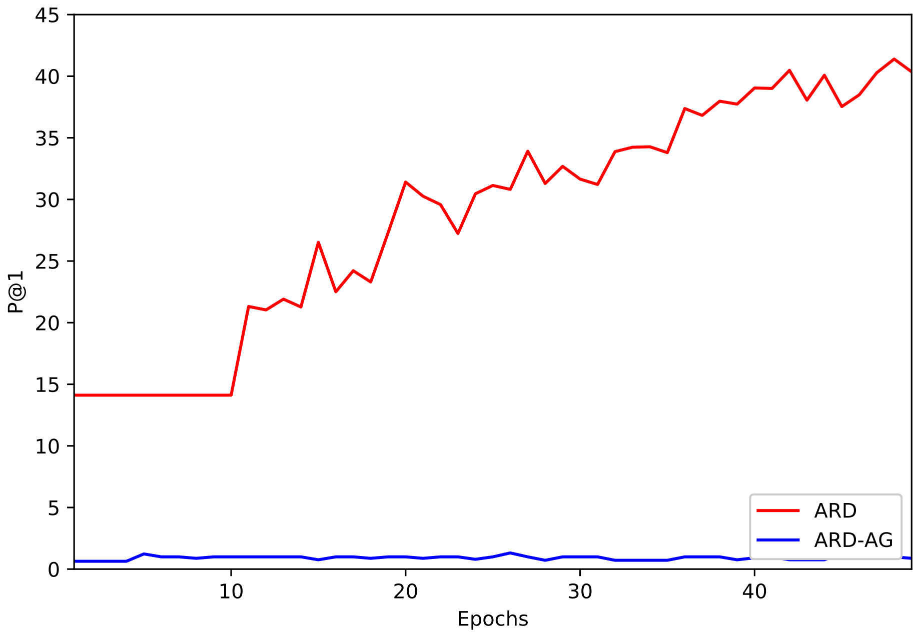
As we mentioned in the previous section, the optimization of the lower bound is achieved by a Python implementation which is either based on Autograd or Tensorflow package. Here, we address the efficiency of each method by considering two different implementations of the ARD-PP method. As we can see in Table 4.8 and Fig. 4.4, the Autograd implementation fails dramatically in terms of predictive performance while the corresponding lower bound is much lower than the Tensorflow's implementation. This is a general issue we encountered throughout all experiments based on SE kernel and Autograd. However, the Linear kernel works well for Autograd as we shall see in a few later in other experiments. We speculate that this result is due to the fact that we had to code up our own optimizer implementation (it is an RMSProp optimizer), since Autograd does not provide any, which is not as efficient as the one provided by Tensorflow itself. This result might also be the case of problematic initialization of the learning rate value that we use for our experiments and better calibration might have been required. We do not encounter this issue using the Adam optimizer of Tensorflow.

Fixed Z comparison

The fact that we treat the inducing inputs as variational parameters and we optimize the lower bound over them is one of the reasons we are able to achieve much better performance than by just keep them fixed throughout the optimization procedure. This is demonstrated in Fig. 4.5 (a) and (b), where both plots of P@1 precision and lower bound values are significantly larger in the case of optimized Z than fixed Z. The compared methods here are ARD and ARD-FX where fixed matrix Z consists of $M = 100$ k-means centres obtained by running k-means algorithm on the original dataset.

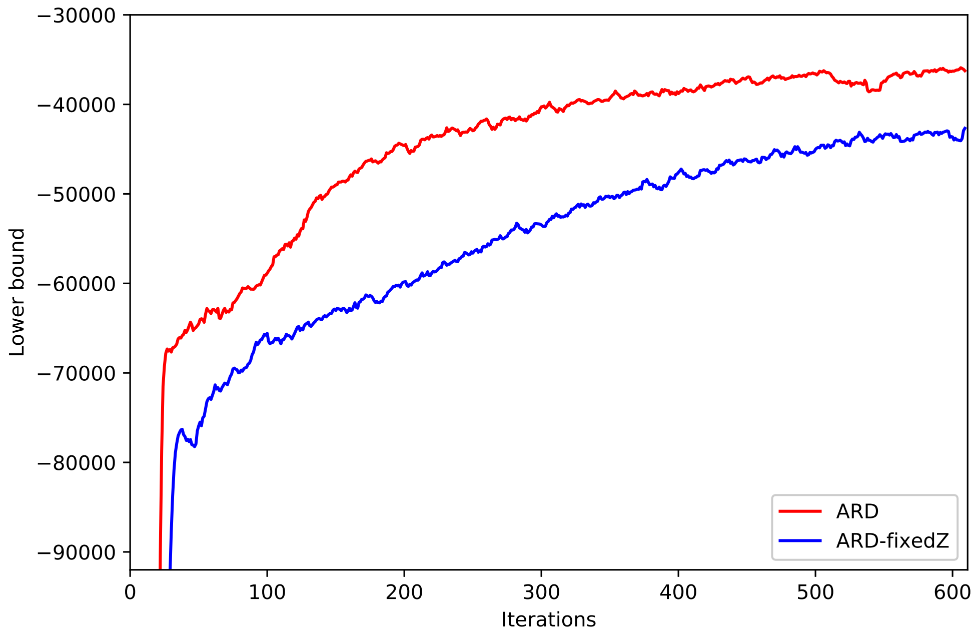


(a)

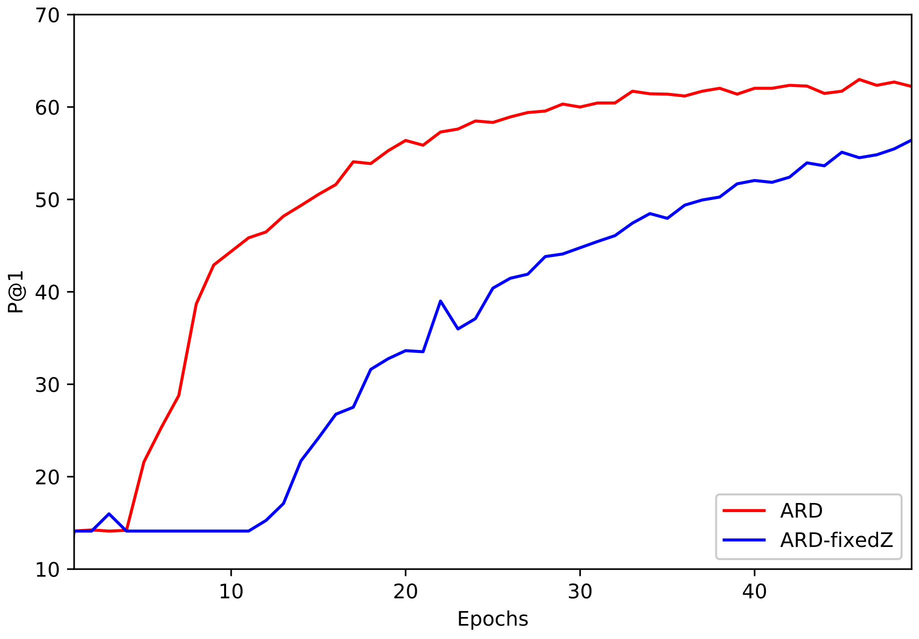


(b)

Figure 4.4: Performance comparison of the MLGPF model between the **Tensorflow** (red color line) and **Autograd** implementation (blue color line). An ARD kernel combined by the PP are used while $M = 100$. (a) shows the evolution of the lower bound, and (b) gives the P@1 evolution across epochs for each Python package.



(a)



(b)

Figure 4.5: Performance comparison of the MLGPF model between **optimized** Z (red color line) and **fixed** one (blue color line) using an ARD kernel. (a) shows the evolution of the lower bound, and (b) gives the P@1 evolution across epochs for each parametrization.

P	10	20	40	80	159	200	500
$M = 50$	0.55	1.02	1.15	1.26	2.01	2.17	5.21
$M = 100$	0.75	1.12	1.92	2.62	5.14	6.42	9.43
M	10	50	100	200	400	600	800
$P = 80$	1.11	1.26	2.62	3.62	15.46	70.23	125.21
$P = 159$	1.52	2.01	5.14	3.62	20.46	79.58	363.03

Table 4.9: Training times (in minutes) of the MLGPF model for the Bibtex dataset using several values for P and M, running for 50 epochs and deploying the ARD kernel.

Normalized Z comparison

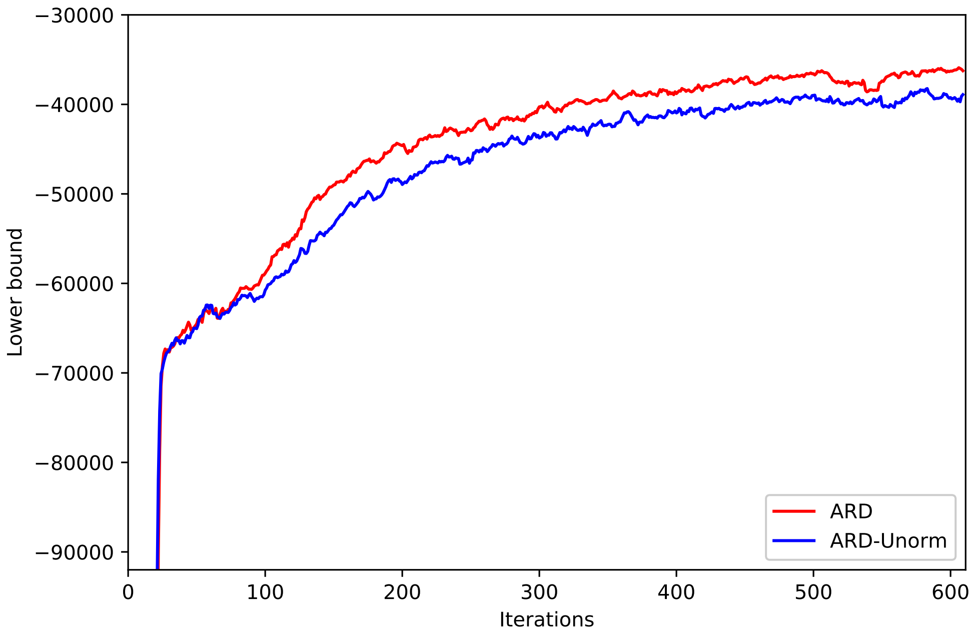
As we noted earlier in this section, all the text-based datasets are unit-normalized to improve model's performance. For that reason we would like to see how different results we would get in case we constrained the inducing inputs to lie in the unit-hypersphere compared to those which are optimized unconstrainedly. Fig. 4.6 reveals that constraining the optimization of Z leads to considerably better performance. The comparison is between ARD and ARD-UN.

Effect of the number of inducing inputs M used

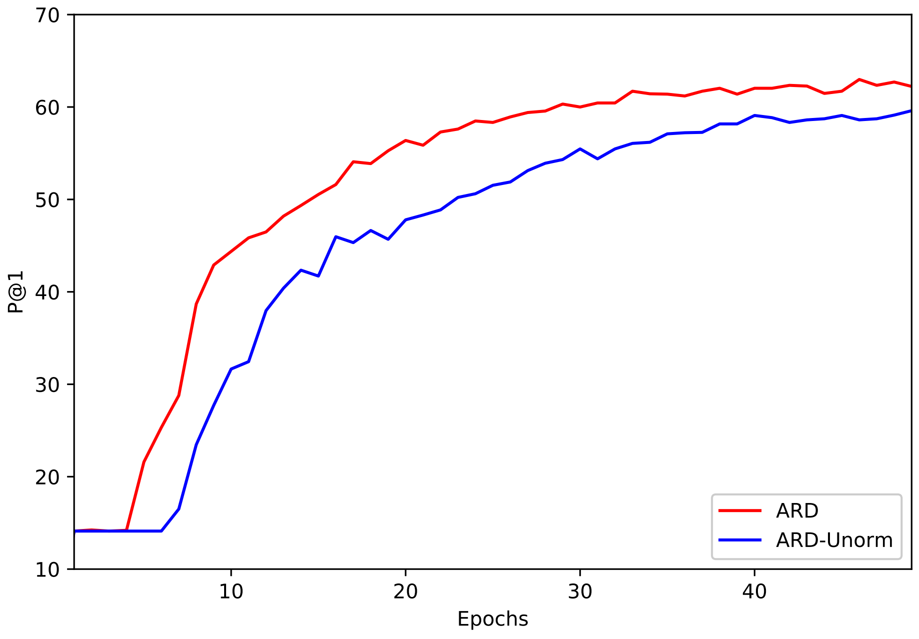
It is generally known (Titsias, 2009; Matthews et al., 2016) that increasing the number of inducing inputs leads to a tighter lower bound on the log marginal likelihood $p(Y)$, consequently leading to a better performance most of the times. Fig. 4.7 verifies this relation as a function of the number of inducing points for the case of the P@1 metric where in the top plot (a) we kept the number of latent factors equal to 80 while in (b) $P = 159$. Similarly, Fig. 4.8 demonstrates the corresponding lower bounds. For all cases the ARD method was employed. An extra illustration of those results using bar plots can be found in Fig. (a) 4.11 and (a) 4.12.

Effect of the number of GPs P used

Another aspect that plays a significant role on the overall performance of the MLGPF model is the number of latent factors or GPs P used to train the model. Since P has to be defined in advance as like M, it would be important to examine how much our model can be benefited with an increase of P. By looking carefully Fig. 4.9, it can be concluded that using a large number of GPs leads to a better performance and consequently better optimized lower bound. Moreover, in some cases, it can be noticed that by using a large number of GPs, an increase of the number of the inducing inputs gives a marginal, if any, performance improvement, meaning that the model is flexible enough using this number of GPs. For instance,

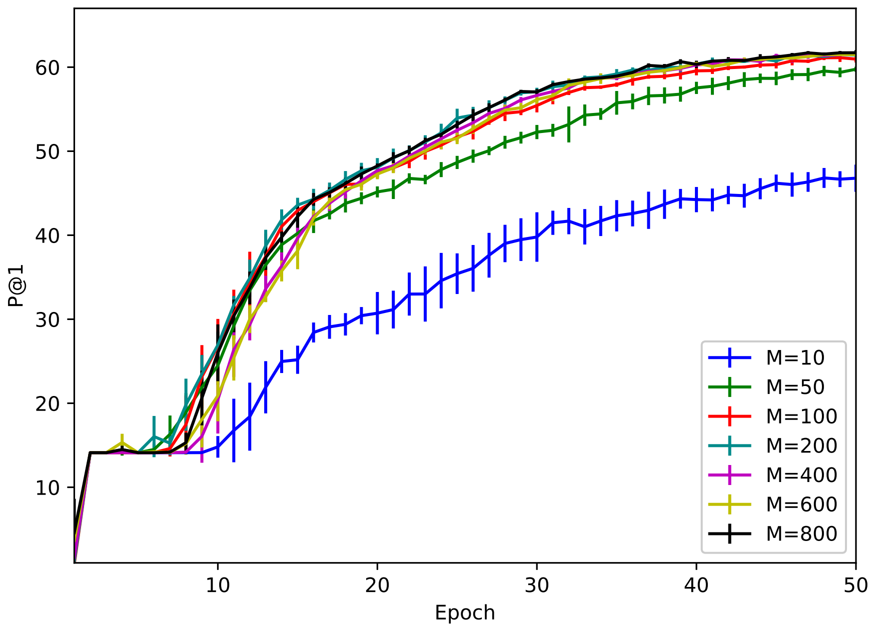


(a)

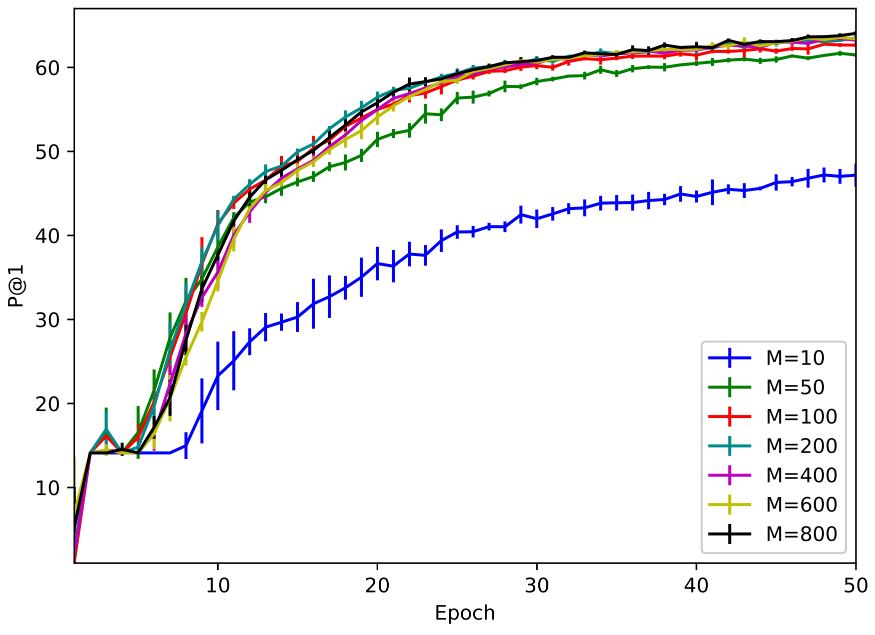


(b)

Figure 4.6: Performance comparison of the MLGPF model between **normalized** Z (red color line) and **unnormalized** one (blue color line) using an ARD kernel. (a) shows the evolution of the lower bound, and (b) gives the P@1 evolution across epochs for each parametrization.

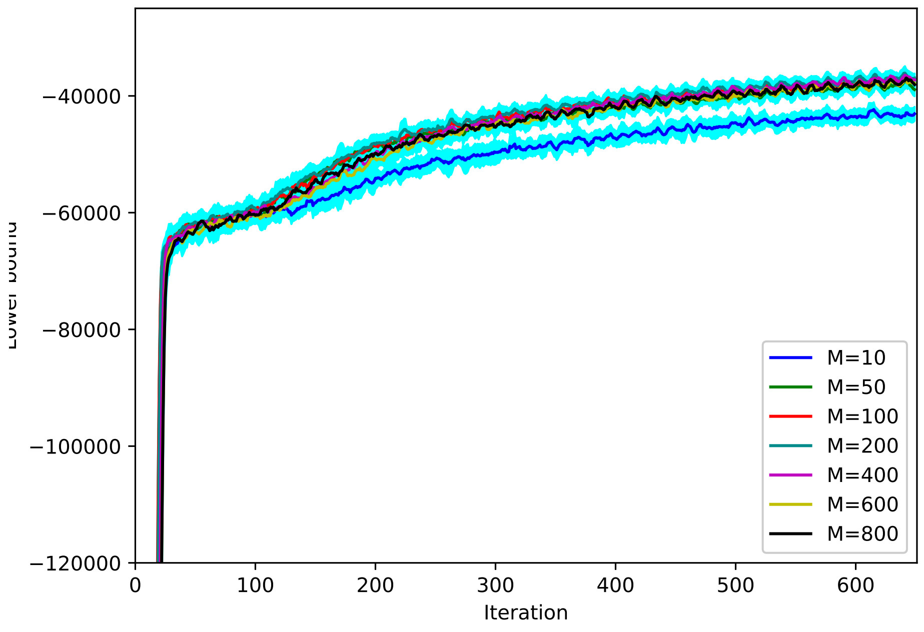


(a)

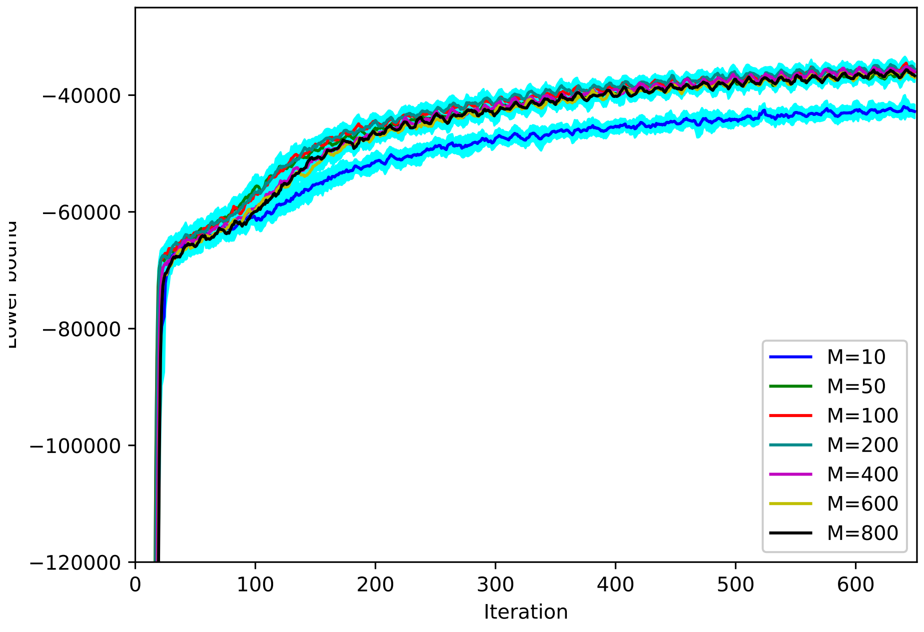


(b)

Figure 4.7: Performance of MLGPF model in terms of P@1 (Eq. (1.21)) as a function of epochs where an ARD method trained on the Bibtex dataset. (a) Assumes $P = 80$ and (b) $P = 159$ where in both cases $M = 10, 50, 100, 200, 400, 800$. Error bars of one standard deviation are also provided.

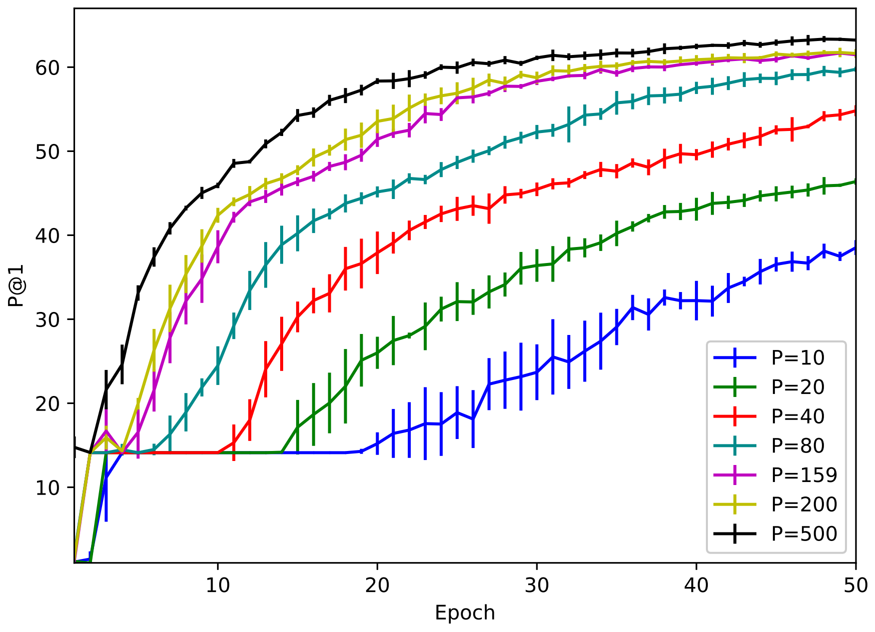


(a)

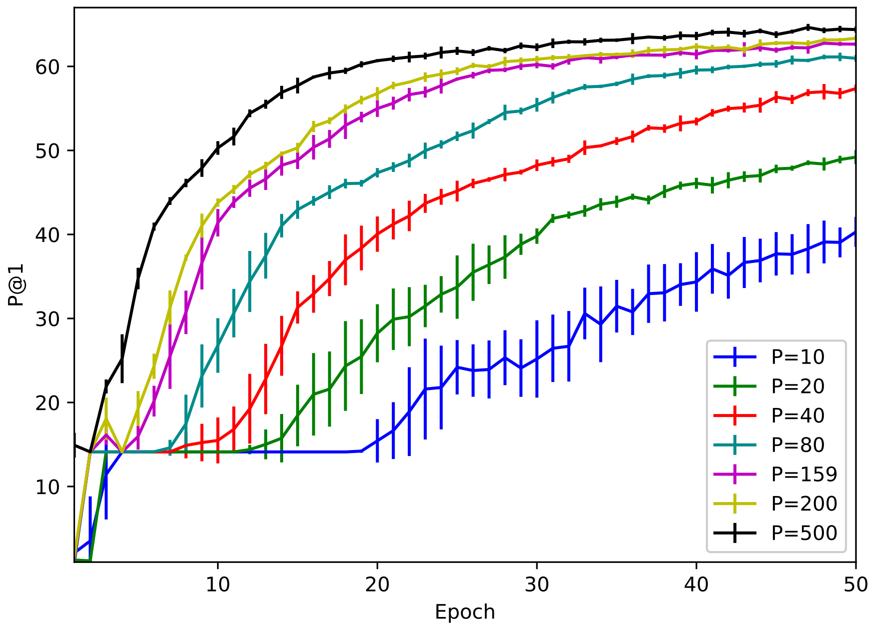


(b)

Figure 4.8: Lower bound values of MLGPF model as a function of iterations where an ARD method trained on the Bibtex dataset. (a) Assumes $P = 80$ and (b) $P = 159$ where in both cases $M = 10, 50, 100, 200, 400, 800$. The shaded area around curves represents 68% confidence intervals.

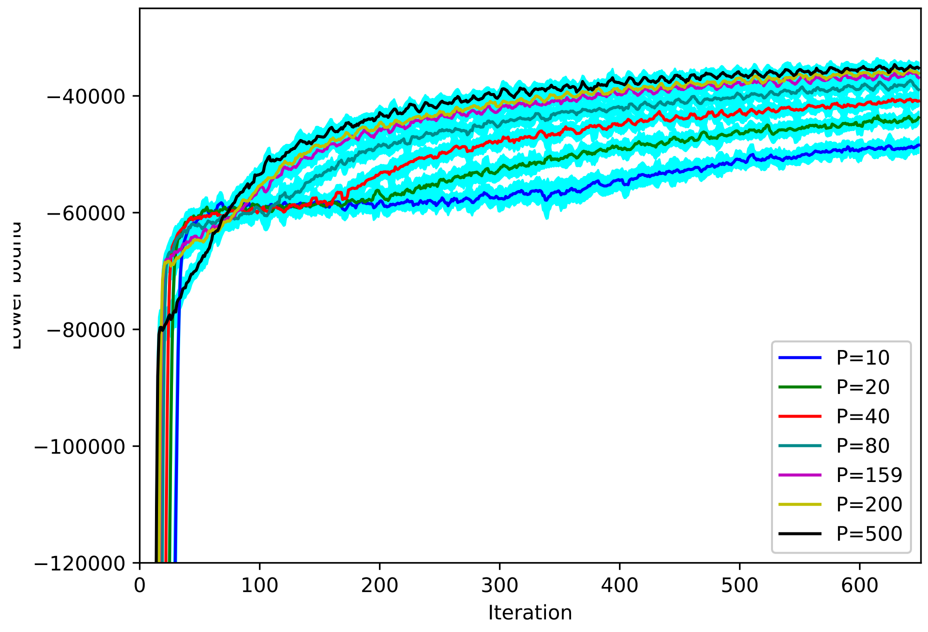


(a)

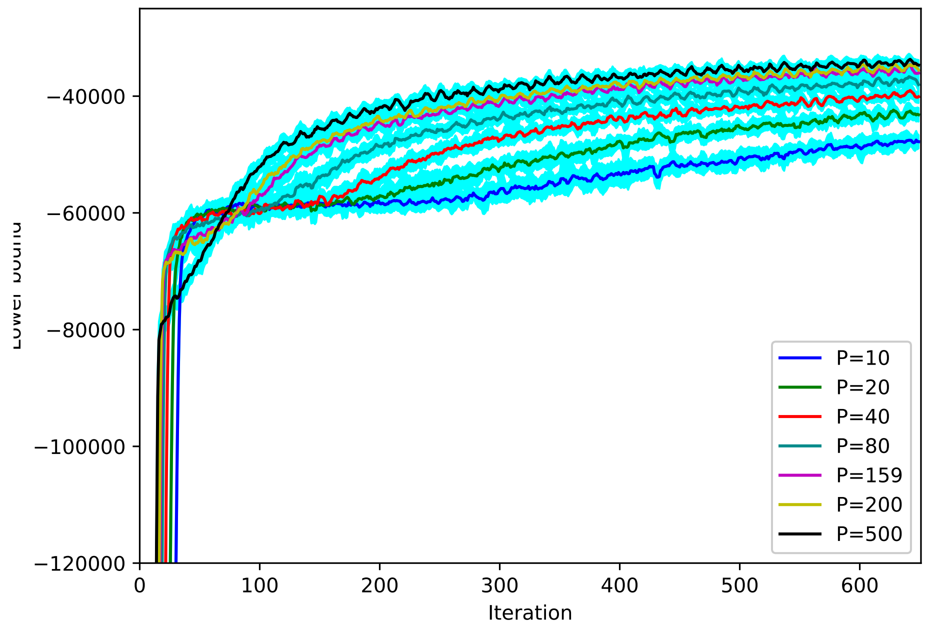


(b)

Figure 4.9: Performance of MLGPF model in terms of P@1 as a function of epochs where an ARD method trained on the Bibtex dataset. (a) Assumes $M = 50$ and (b) $M = 100$ where in both cases $P = 10, 20, 40, 80, 159, 200, 500$. Error bars of one standard deviation are also provided.

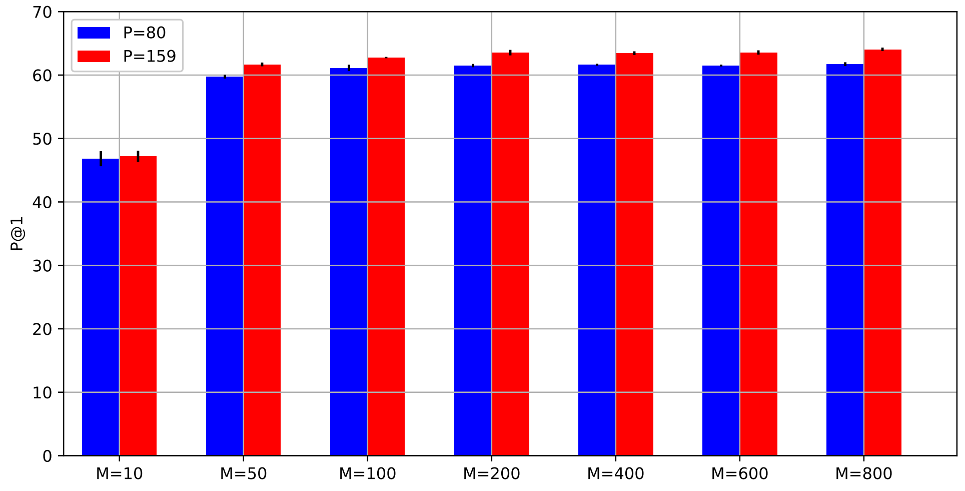


(a)

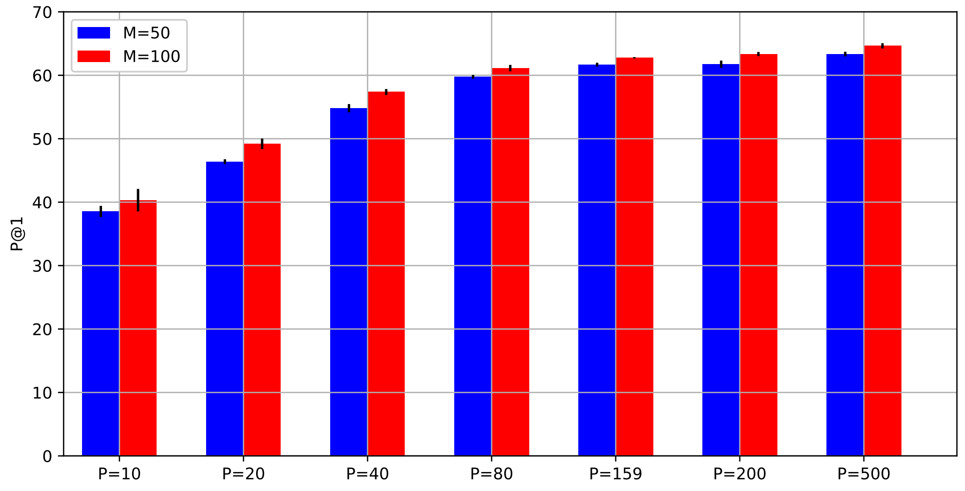


(b)

Figure 4.10: Lower bound values of MLGPF model as a function of iterations where an ARD method trained on the Bibtex dataset. (a) Assumes $M = 50$ and (b) $M = 100$ where in both cases $P = 10, 20, 40, 80, 159, 200, 500$. The shaded area around curves represents 68% confidence intervals.

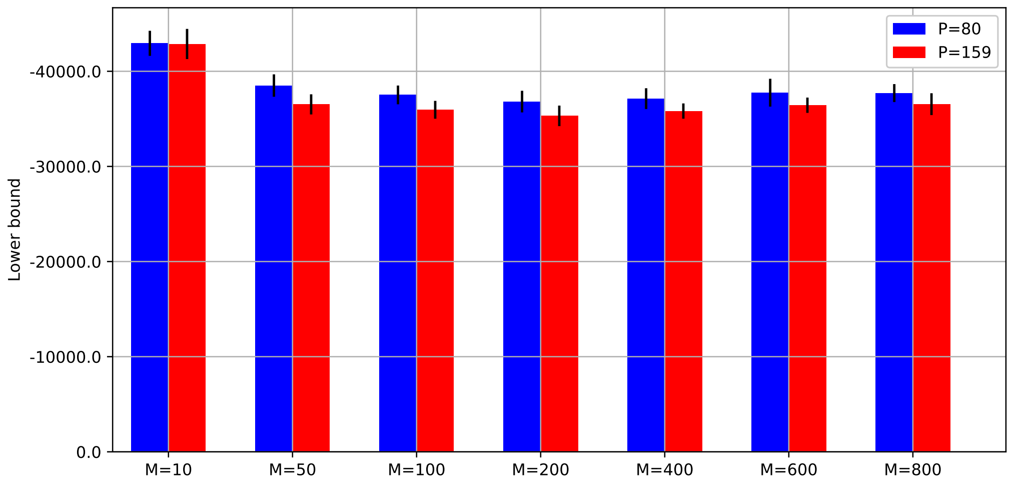


(a)

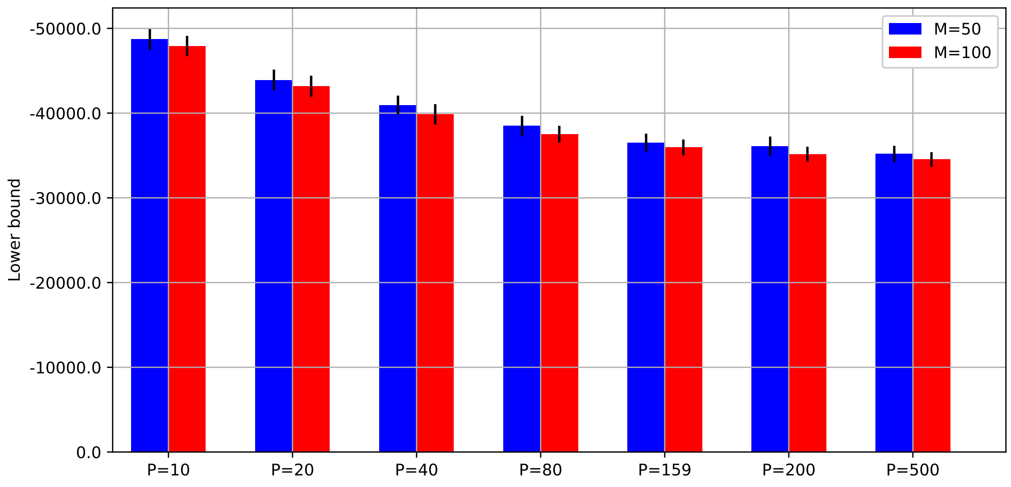


(b)

Figure 4.11: Bar plots comparing P@1 values using several pair of values (M, P) where an ARD method trained on the Bibtex dataset and run for 50 epochs. (a) shared $P = 80$, 159, and (b) shared $M = 50$, 100. Error bars of one standard deviation are also provided.



(a)



(b)

Figure 4.12: Bar plots comparing lower bound values using several pair of values (M, P) where an ARD method trained on the Bibtex dataset and run for 50 epochs. (a) shared $P = 80$, 159, and (b) shared $M = 50$, 100. The lower the bar plots are the better performance is achieved by ARD. Error bars of one standard deviation are also provided.

Method	time (seconds/epoch)
LINEAR	62
LINEAR-ℓ	62
SE	65
SE-p-ℓ	127
ARD/LINEAR	64
ARD	64
ARD-PP	110
ARD-PP-AG	648
ARD-FX	63
ARD-UN	63
ARD-S	62
ARD-S-B	63

Table 4.10: Elapsed training time comparison between different MLGPF methods, applied to Bibtex dataset using $M = |\mathcal{B}| = 500$ and $P = K = 159$.

using $P = 500$ combined by either $M = 50$ or $M = 100$ gives approximately the same P@1 value. This example also provides an interesting information; even if we use more GPs than the dimensionality of label space ($K = 159$ for Bibtex), leading to an over-determined factor loadings matrix Φ, the performance of the model improves comparing to the case of using exactly $P = K = 159$ GPs. However, in practice, we are rarely able to use same number of latent factors as K, let alone more than that. Finally, it is worth stressing the importance of having an adequate number of inducing inputs when the P is much smaller than K. This is clearly illustrated in Fig. 4.9 for the case of $P = 10$ where it is observed a significant performance boost by increasing $M = 50$ to $M = 100$, implying that for large-scale datasets we should primarily consider setting the number of inducing inputs as higher as possible. An extra illustration of those results using bar plots can be found in Fig. (b) 4.11 and (b) 4.12. Table 4.9 demonstrates the training time required for those different values of P and M used.

Time comparison

Here, we probe the training time of all the aforementioned methods, where in all experiments we used $M = |\mathcal{B}| = 500$ and $P = K = 159$. By choosing the same number of inducing inputs as like minibatch size, we guarantee that the overall complexity of computing the lower bound is $\mathcal{O}(PM^3)$ for all methods. This can be also verified by advising Tables 4.1 and 4.2. Nevertheless, experimental times presenting by Table 4.10, do not demonstrate this asymptotic behaviour for the computation of the bound. In general we see that most of the methods need almost 63 seconds to scan the whole dataset while this time is almost doubled in the case of SE-p-ℓ and ARD-PP. At a first glance this result might seem peculiar but having a closer look to those two methods we see that this time is absolutely justifiable. More specifically, those two methods are the only ones that require the computation of P Cholesky decompositions instead of just one as it is the case for the rest of the methods. This means that the computational complexity is $\mathcal{O}(PM^3 + PM^3) = \mathcal{O}(2PM^3)$. In the limit though we do not take into account any constants so the overall complexity is $\mathcal{O}(PM^3)$ but in practice we see how restrictive can be this kind of constant where not only hinders fast computations but constraints us to increase further the number of GPs P. Hence, this extra attribute makes the use of PP less appealing when it comes to this kind of applications. Finally, the most striking feature of Table 4.10 is the computational time needed for the ARD-PP-AG. This is six times slower than the corresponding method ARD-PP implemented by Tensorflow while ARD can be computed more than ten times faster than ARD-PP-AG. This is clear proof of the superiority of Tensorflow package to efficiently compute the lower bound and its gradients comparing to Autograd. Despite its appealing attribute of working in native Python, Autograd fails to achieve the same performance as Tensorflow. We argue that Tensorflow's superiority mainly stems from the fact that it has been developed to scale well on large scale neural-net-oriented applications based on the idea of computational graphs. This idea is the game changer in its performance since it allows to efficiently parallelize computations. To speed-up computations further, GPU support is also provided but we did not employ it here. All those key differences make the use of Tensorflow imperative, especially for datasets with hundred of thousands data points, labels and features.

A closer look at the size of the optimization

As we argued previously, the optimization of all the variational parameters and kernel hyperparameters of the MLGPF model, especially for the large-scale datasets, constitutes the quintessence of this chapter. In order to make the size of that problems more tangible to the reader, we present Table 4.11 which gives the number of parameters (in millions) we need to optimize by using the two different parametrizations, FP and PP, and setting the method's hyperparameters M and P as they are used in most of the experiments later. The table shows the significant storage advantage of PP over FP. Generally, the ARD-S-B (or ARD-S-B-PP) is not suitable for dealing with the extreme dimensions of those datasets since the number of parameters needed to be optimized exceeds the corresponding number for the ARD (or ARD-PP). Overall, it can be now realized how large are the optimization tasks we encounter in this chapter, where the largest one needs the optimization of more than *one billion* parameters. In the next section, we provide tangible evidence of the successful optimization of those gargantuan problems by employing the MLGPF model.

Dataset	ARD	ARD-S	ARD-S-B	M	P	$\mathcal{R}$
Bibtex	13.6	13.2	15.1	400	159	1000
Delicious	24.7	24.6	24.7	400	300	250
Mediamill	12.8	12.8	12.8	500	101	120
EUR-Lex	44.3	42.3	42.7	400	500	70
RCV1	27.5	5.1	123.2	500	30	2500
Wiki10	209.6	149.9	404.8	600	700	2000
AmazonCat	229.4	113.0	2151.9	600	550	10000
Delicious-Large	461.9	188.7	1753.9	350	700	2000
WikiLSHTC	1081.4	272.7	1081.6	500	600	500
Dataset	ARD-PP	ARD-S-PP	ARD-S-B-PP			
Bibtex	0.9	0.6	2.4	400	159	1000
Delicious	0.7	0.6	0.8	400	300	250
Mediamill	0.2	0.2	0.2	500	101	120
EUR-Lex	4.4	2.4	2.8	400	500	70
RCV1	23.8	1.4	119.4	500	30	2500
Wiki10	83.8	24.1	279.0	600	700	2000
AmazonCat	130.5	14.2	2053.0	600	550	10000
Delicious-Large	419.2	146.0	1711.2	350	700	2000
WikiLSHTC	1006.5	197.8	1006.8	500	600	500

Table 4.11: Number of parameters (in millions) to be optimized for each dataset using different parametrizations.

Dataset	P	M	epochs
Bibtex	30	500	300
Delicious	30	500	300
Mediamill	30	500	150
EUR-Lex	30	500	100
RCV1	30	500	5

Table 4.12: Parameter settings of the MLGPF model for each dataset based on the methods preseneted in Table 4.13.

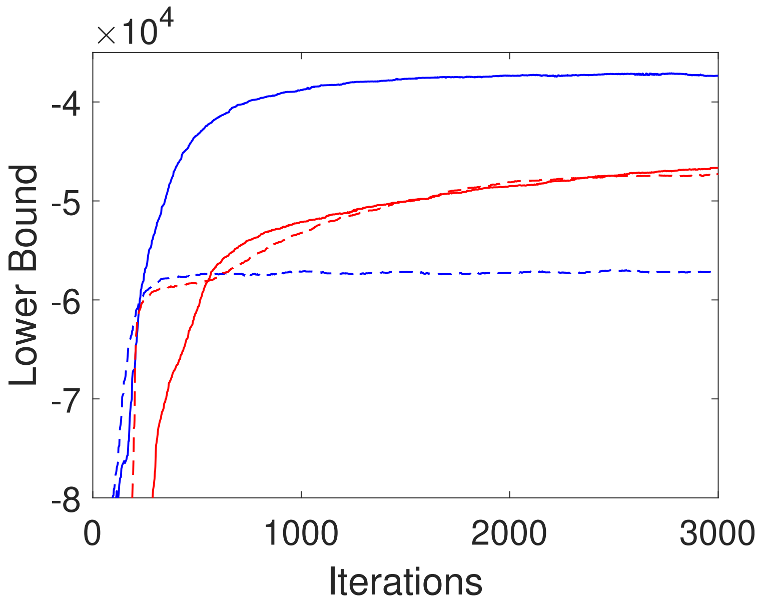
Results based on the MLGPF model

The rest of this section presents results of the MLGPF models applied to all nine different datasets of Table 4.4, by employing different combinations of methods of Section 4.5.2.

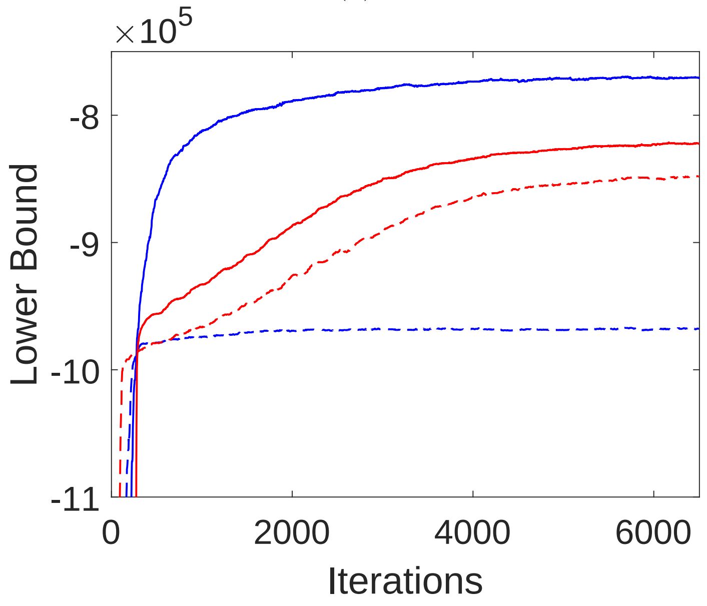
Parsimonious parametrization and Autograd results

The first results that we demonstrate are based on both the PP and the Autograd implementation. All the settings of each experiment are given by Table 4.12 while the corresponding performance results can found in Table 4.13. As you can observe, the number of GPs used was set to 30 for all datasets due to the limitations posed by Autograd as we discussed earlier. Performance is also poor comparing to results where Tensorflow is used, and the discussed problem with the poor optimization of the SE kernel is ambiguous in all tested datasets. Nevertheless, the benefits of optimizing the inducing inputs are still conspicuous in both Table 4.13 and Figures 4.13, 4.14 where the corresponding lower bounds are depicted.

Finally, the theory of Section 4.4.5 that allows to predict a label vector under partially observed labels is also applied. More specifically, for any test label vector $\mathbf{y}^{(*)}$ we set the partially observed $\mathbf{y}_o^{(*)}$ to be the first positive label (so that the size of $\mathbf{y}_o^{(*)}$ is always one). Then, we did predictions over the remaining missing vector $\mathbf{y}_m^{(*)} \in \{-1, 1\}^{K-1}$. Based on the framework in Section 4.4.5, we computed the corresponding utility vector $\bar{\mathbf{f}}^{(*)} \in \mathbb{R}^{K-1}$ using the MAP estimate of both $q(\mathbf{u}^{(*)})$ and $q(\mathbf{u}^{(*)}|\mathbf{y}_o^{(*)})$, and performed predictions based on these two alternatives. Notice that the method based on the MAP of $q(\mathbf{u}^{(*)})$ predicts without taking into account the partially observed $\mathbf{y}_o^{(*)}$. Table 4.15 compares the P@k scores of the two approaches for all datasets by using the best method from Table4.13 (typically LINEAR-PP-AG while SE-PP-AG was used only for the Mediamill dataset). It can be seen that there is a marginal (but consistent across all datasets) improvement in the predictive performance based on the MAP estimate of $q(\mathbf{u}^{(*)}|\mathbf{y}_o^{(*)})$.

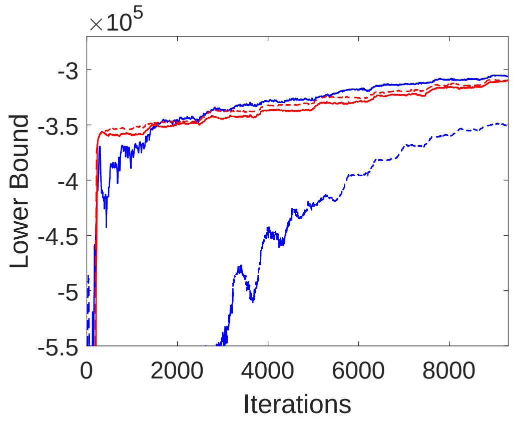


(a)

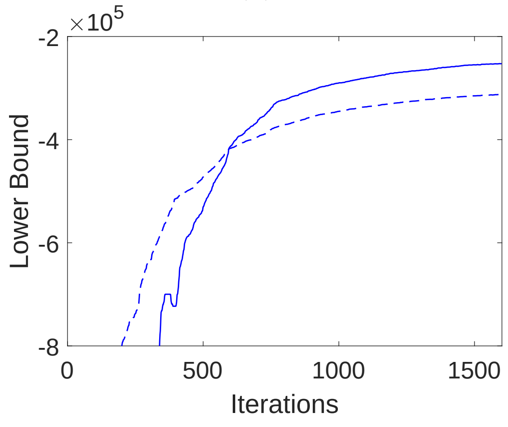


(b)

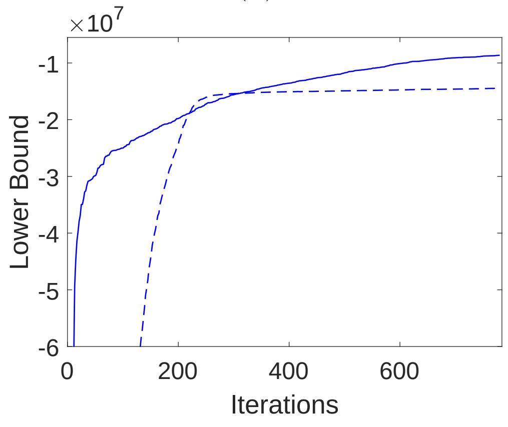
Figure 4.13: Lower bounds of (a) Bibtex and (b) Delicious. The solid lines correspond to the methods that optimized the inducing points while the dashed ones correspond to those that kept them fixed. Blue color suggests the use of linear kernel while the red one the use of SE kernel. Performance results of those methods can be found in Table 4.13.



(a)



(b)



(c)

Figure 4.14: Lower bounds of (a) Mediamill, (b) EUR-Lex, and (c) RCV1. The solid lines correspond to the methods that optimized the inducing points while the dashed ones correspond to those that kept them fixed. Blue color suggests the use of linear kernel while the red one the use of SE kernel. Performance results of those methods can be found in Table 4.13.

Dataset		LINEAR-FX-PP-AG	LINEAR-PP-AG	SE-FX-PP-AG	SE-PP-AG
Bibtex	P@1	18.01	59.56	41.03	42.27
	P@3	10.03	36.10	23.05	23.14
	P@5	7.92	27.00	17.07	17.73
Delicious	P@1	41.92	66.78	64.33	66.03
	P@3	38.16	61.49	57.88	59.46
	P@5	35.76	57.12	52.81	54.66
Mediamill	P@1	77.12	82.62	82.34	83.02
	P@3	61.88	64.34	65.75	66.69
	P@5	42.24	49.18	50.67	51.42
EUR-Lex	P@1	67.50	77.68	-	-
	P@3	52.18	62.30	-	-
	P@5	41.87	51.24	-	-
RCV1	P@1	31.04	80.61	-	-
	P@3	24.06	64.73	-	-
	P@5	19.31	46.00	-	-

Table 4.13: Predictive Performance of the MLGPF model for the seven multi-label datasets. Those methods that have not reported results for a dataset are indicated with the "-" sign. All the experimental settings of each dataset can be found in Table 4.12.

Subspace inducing inputs results

The next experiments test the performance of the MLGPF model under the use of subspace inducing where we keep the same parametrization and implementation as before. The results are collectively demonstrated in Table 4.17 where lower bounds and experimental settings are given by Figures 4.15, 4.16 and Table 4.16. We also provide performance results for the methods LINEAR-S-FX-PP-AG, LINEAR-FX-PP-AG, SE-S-FX-PP-AG, and SE-S-FX-PP-AG, i.e. both fixed inducing and subspace inducing inputs while all the training times of each those methods can be found in Table 4.19.

Once more, Autograd fails to successfully train the model when an SE kernel is employed, apart from the case of Mediamill dataset, where the results were competitive with those of Tensorflow based methods (see Table 4.23). However, Linear kernel achieves much better predictive power in both cases where the full and the subspace inputs were considered. We also observe that the subspace-inducing-inputs based methods often attained higher P@k values than the their full counterparts. This could be interpreted in some cases that there is much redundancy in the full input space and it can effectively be approximated by a lower dimensional manifold. However, when it comes to high dimensional spaces ($D > 2000$), experiments showed that we cannot guarantee both competitive per-

Datasets	LINEAR-FX-PP-AG	LINEAR-PP-AG	SE-FX-PP-AG	SE-PP-AG
Bibtex	2.96	3.17	2.93	3.10
Delicious	8.07	8.19	8.22	8.16
EUR-Lex	11.94	13.23	-	-
Mediamill	18.53	18.71	18.69	18.66
RCV1	673.2	2431	-	-

Table 4.14: Training times (in minutes per epoch) of the MLGPF model for all used datasets based on the settings of Table 4.12.

Datasets		$q(\mathbf{u}^{(*)})$	$q(\mathbf{u}^{(*)}\|\mathbf{y}_o^{(*)})$
	P@1	34.79	35.35
Bibtex	P@3	22.00	22.65
	P@5	16.45	16.88
	P@1	66.19	66.37
Delicious	P@3	60.81	61.10
	P@5	56.19	56.39
	P@1	70.25	71.17
EUR-Lex	P@3	54.04	54.67
	P@5	42.70	43.28
	P@1	79.73	80.04
Mediamill	P@3	56.40	56.63
	P@5	41.34	41.64
	P@1	65.96	65.99
RCV1	P@3	48.92	40.95
	P@5	36.02	36.04

Table 4.15: Performance scores when predicting the partially observed label vectors based on the MAP estimate of $q(\mathbf{u}^{(*)})$ and $q(\mathbf{u}^{(*)}|\mathbf{y}_o^{(*)})$. In all datasets, except Mediamill the LINEAR-PP-AG was used. For Mediamill, the SE-PP-AG was chosen.

Dataset	P	M	epochs	$\mathcal{R}$
Bibtex	30	500	400	1000
Delicious	30	500	200	250
Mediamill	30	500	200	70
EUR-Lex	40	500	200	2500
RCV1	30	500	20	2000
AmazonCat	30	500	15	2000

Table 4.16: Parameter settings of the MLGPF model for each dataset used in this chapter. The methods presented in Table 4.17 use those settings for their parameters.

Dataset		LINEAR-S-PP-AG	LINEAR-PP-AG	SE-S-PP-AG	SE-PP-AG
Bibtex	P@1	59.31	60.20	41.89	38.68
	P@3	36.73	37.02	24.30	21.71
	P@5	27.40	27.34	18.57	16.55
Delicious	P@1	66.13	67.08	59.89	61.94
	P@3	60.38	61.50	53.80	55.91
	P@5	55.69	56.88	49.21	50.88
Mediamill	P@1	82.98	82.33	84.12	82.80
	P@3	65.62	65.25	67.17	66.14
	P@5	51.32	51.09	53.15	52.16
EUR-Lex	P@1	79.31	78.34	66.42	64.95
	P@3	64.24	63.35	50.58	49.47
	P@5	52.79	52.06	40.56	39.63
RCV1	P@1	88.74	-	25.97	-
	P@3	71.27	-	21.85	-
	P@5	51.16	-	17.13	-
AmazonCat	P@1	85.90	-	44.11	-
	P@3	64.98	-	27.18	-
	P@5	49.88	-	21.18	-

Table 4.17: Predictive Performance of the MLGPF model for the seven multi-label datasets. Those methods that have not reported results for a dataset are indicated with the "-" sign. All the experimental settings of each dataset can be found in Table 4.16

formance and fast computations, i.e. $\mathcal{R} \sim \mathcal{O}(M)$. This is exactly the reason why we set $\mathcal{R} = 2000$ for the large-scale datasets. Otherwise, the model would have had inferior predictive ability. At the same time, an extra burden emerges by the increase of $\mathcal{R}$ which is related with storage requirements of the basis $\widetilde{X}$. As we discussed in Section 2.5 of Chapter 2, the basis $\widetilde{X}$ can be computed by storing the $\mathcal{R}$ singular vectors correspond to the $\mathcal{R}$ larger singular values of the design matrix X, which is achieved by a simple application of the subset SVD algorithm. However, this approach entails the problem of storing an $\mathcal{R} \times D$ dense matrix in memory where D is extremely large and $\mathcal{R}$ is needed to be as large as possible. Thus, there is an extra constraint here, which does not let us to increase $\mathcal{R}$ as well even if we were willing to sacrifice computational speed for performance. This argument can be also verified by the long training times required for the large datasets in Table 4.19.

Results based on the Tensorflow implementation

The final and most competitive results of this section for the multi-label datasets, come from the methods implemented by Tensorflow. Apart from the implementation, an equally vital ingredient of the improved performance of the model should be attributed to the employment of D extra lengthscales as hyperparameters, regardless the choice of the kernel. Admittedly, it was in Section 4.5.3 where we firstly observed the performance superiority of ARD and LINEAR-ℓ over the rest of the methods which did not use lengthscales for each dimension.

We firstly introduce the experimental settings of each of the datasets used in this Section where this information is summarized in Table 4.20. As it can be seen, Ternoflow implementation allows us to dramatically increase the number of GPs used for our model without sacrificing significant speed or increasing considerably the memory footprint of the methods. For instance, we manage to use even 700 GPs for training our model on the Delicious-Large dataset. On top of that, it is preferred to parametrize the covariance matrices of the P variational distributions using lower triangular matrices, since PP is proven to be inferior in the previous experiments. Consequently, Table 4.23 presents the highly improved P@k values comparing to the previous experiments where in some cases there is more than 35% improvement comparing to methods in Table 4.13. Apart from that, it is the first time that we manage to apply our model on datasets of the scale of Delicious-Large and WikiLSHTC which require the optimization of more than half a billion parameters (see Table 4.11). Unfortunately, the WikiLSHTC is the only dataset that we do not succeed to run enough epochs since it requires more than four days to perform a full epoch for the given settings. Furthermore, several plots are provided in Figures 4.19 and 4.20, which demonstrate the usefulness of optimizing over the inducing inputs Z, something that has already been realized from all the previous experiments.

Since we discussed the importance of using different lengthscales for each dimen-

Dataset		LINEAR-S-FX-PP-AG	LINEAR-FX-PP-AG
Bibtex	P@1	45.12	40.31
	P@3	26.79	23.16
	P@5	20.40	17.67
Delicious	P@1	63.13	63.04
	P@3	57.04	57.03
	P@5	52.26	52.40
Mediamill	P@1	75.17	78.75
	P@3	58.88	62.06
	P@5	45.33	47.45
EUR-Lex	P@1	70.10	70.70
	P@3	53.86	54.07
	P@5	43.15	43.62
AmazonCat	P@1	43.19	-
	P@3	25.29	-
	P@5	20.66	-
		SE-S-FX-PP-AG	SE-S-FX-PP-AG
Bibtex	P@1	37.25	36.43
	P@3	20.07	19.42
	P@5	15.37	14.74
Delicious	P@1	55.65	54.44
	P@3	49.87	48.56
	P@5	45.62	44.77
Mediamill	P@1	82.99	82.69
	P@3	66.22	65.85
	P@5	52.26	51.72
EUR-Lex	P@1	57.23	31.32
	P@3	42.76	22.49
	P@5	34.08	18.06
AmazonCat	P@1	30.61	-
	P@3	19.14	-
	P@5	11.64	-

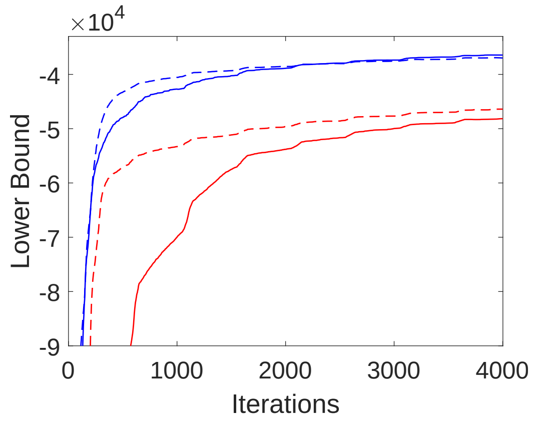
Table 4.18: Predictive Performance of the MLGPF model for five multi-label datasets. Those methods that have not reported results for a dataset are indicated with the "-" sign.

Dataset	LINEAR-S-PP-AG	LINEAR-PP-AG	SE-S-PP-AG	SE-PP-AG
Bibtex	0.94	0.97	1.27	0.99
Delicious	2.47	2.56	2.52	2.55
Mediamill	5.96	5.70	6.64	6.0
EUR-Lex	2.83	2.72	2.75	2.72
RCV1	90.0	130.0	82.1	127.5
AmazonCat	400.7	782.1	408.2	778.8

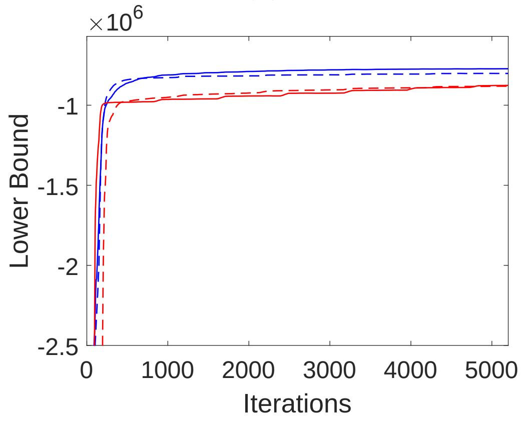
Table 4.19: Computational time (in minutes per epoch) of the MLGPF model for six multi-label datasets.

Dataset	P	M	epochs
Bibtex	159	400	150
Delicious	300	400	50
Mediamill	101	500	100
EUR-Lex	500	400	250
Wiki10	700	600	170
AmazonCat	550	600	15
Delicious-Large	700	350	50
WikiLSHTC	600	500	3

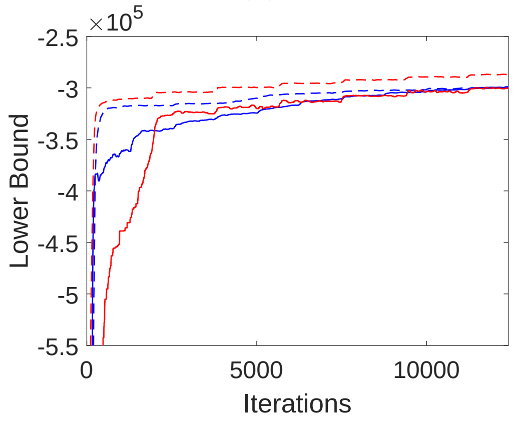
Table 4.20: Parameter settings of the MLGPF model for each dataset based on the methods preseneted in Table 4.23.



(a)

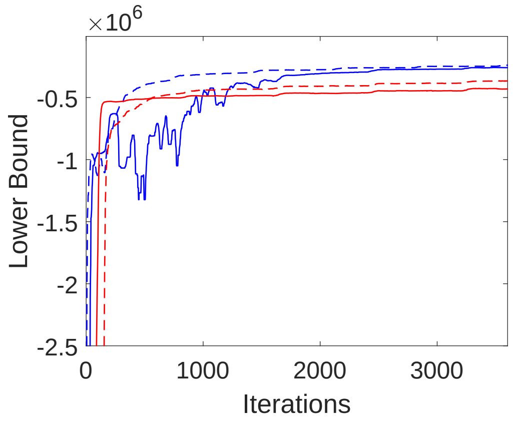


(b)

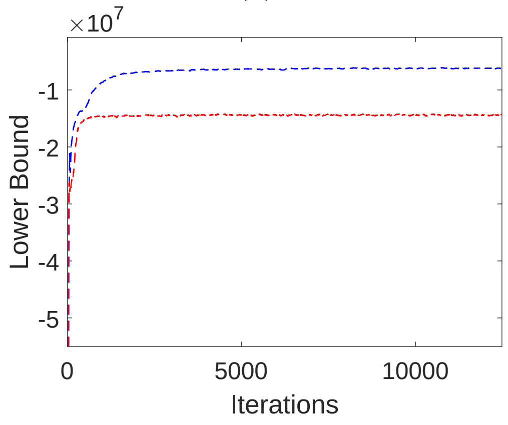


(c)

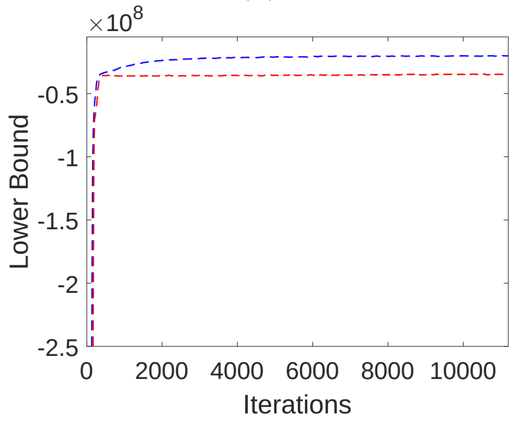
Figure 4.15: Lower bounds of (a) Bibtex, (b) Delicious, and (c) Mediamill, (d) EUR-Lex, (e) RCV1, and (f) AmazonCat. The solid lines correspond to the methods that optimized the inducing points while the dashed ones correspond to those that optimized the subspace inducing points. Blue colour represents the use of linear kernel while the red one the use of squared exponential kernel.



(a)

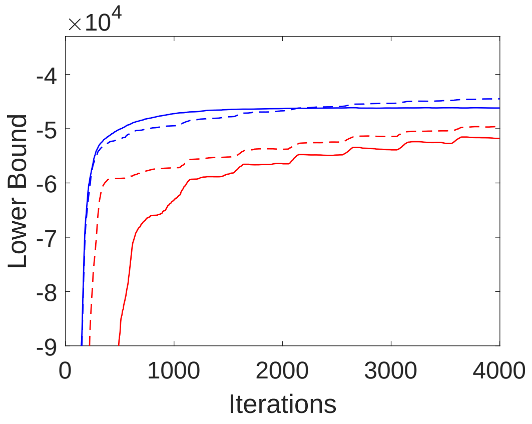


(b)

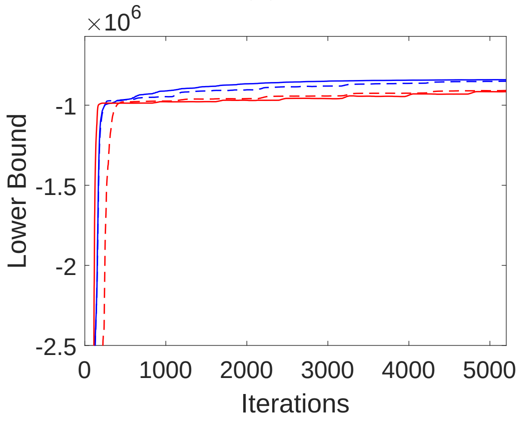


(c)

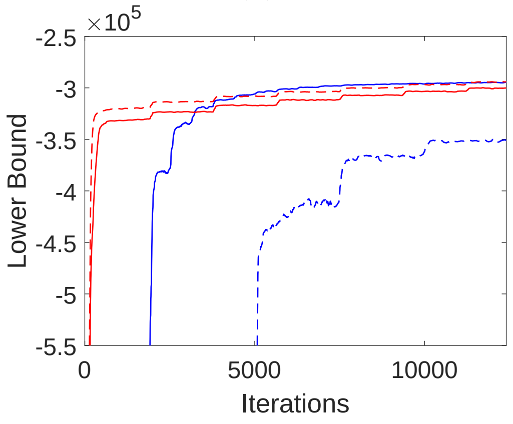
Figure 4.16: Lower bounds of (a) EUR-Lex, (b) RCV1, and (c) AmazonCat. The solid lines correspond to the methods that optimized the inducing points while the dashed ones correspond to those that optimized the subspace inducing points. Blue color suggests the use of linear kernel while the red one the use of squared exponential kernel.



(a)

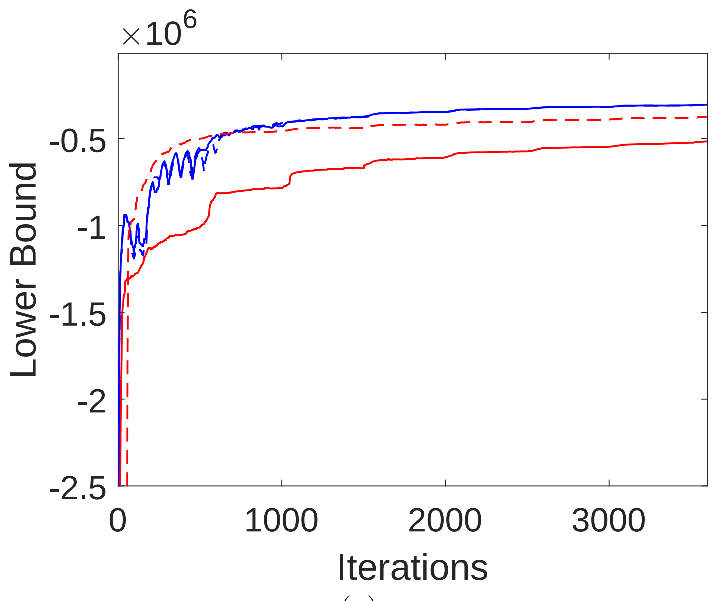


(b)

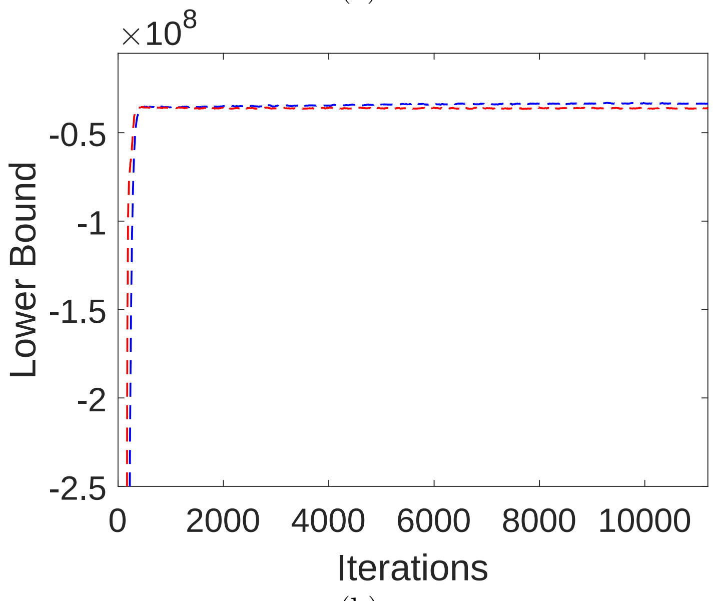


(c)

Figure 4.17: Lower bounds of (a) Bibtex, (b) Delicious, and (c) Mediamill. The solid lines correspond to the methods that kept fixed the inducing points while the dashed ones correspond to those that kept fixed the subspace inducing points. Blue color suggests the use of linear kernel while the red one the use of squared exponential kernel.



(a)



(b)

Figure 4.18: Lower bounds of (a) EUR-Lex and (b) AmazonCat. The solid lines correspond to the methods that kept fixed the inducing points while the dashed ones correspond to those that kept fixed the subspace inducing points. Blue color suggests the use of linear kernel while the red one the use of squared exponential kernel.

Jitter value		Bibtex	Delicious	Mediamill	EUR-Lex
	P@1	48.74	65.99	83.77	39.30
10^{-1}	P@3	27.52	59.75	66.55	29.19
	P@5	20.42	55.19	52.37	23.50
	$\mathcal{F}$	-578338.58	-9307386.81	-2174238.77	-46239799.31
	P@1	59.56	67.59	83.84	66.10
10^{-2}	P@3	35.08	61.66	66.80	50.66
	P@5	25.70	57.19	52.60	40.94
	$\mathcal{F}$	-579552.89	-9298837.28	-2198666.96	-46205485.58
	P@1	60.27	67.06	83.87	73.37
10^{-3}	P@3	35.94	61.76	66.94	57.72
	P@5	26.51	57.14	52.79	46.53
	$\mathcal{F}$	-580261.52	-9300034.11	-2447467.24	-46205385.34
	P@1	60.63	67.22	83.84	74.29
10^{-4}	P@3	36.12	61.74	66.91	58.67
	P@5	26.60	57.08	52.78	47.39
	$\mathcal{F}$	-580357.94	-9300225.81	-4484085.82	-46205506.37
	P@1	60.35	67.06	84.02	74.40
10^{-5}	P@3	36.09	61.84	67.12	58.63
	P@5	26.63	57.19	52.71	47.30
	$\mathcal{F}$	-580367.92	-9300245.88	-18348238.70	-46205520.13

Table 4.21: Sensitivity analysis of the jitter term used for the FP for each small-scale dataset. We run the ARD method using the same seed for our number generator in order to remove randomness from the algorithm and allow comparisons for a given dataset. Five different jitter values are tested, $\{10^i\}_{i=1}^5$, where the first 4 rows correspond to jitter=10^{-1}, the next 5th to 8th row corresponds to jitter=10^{-2} and so on. All the P@k metrics (Eq. (1.21)) and $\mathcal{F}$ are computed after training our model. For all datasets we use $M = 100$, $P = 80$, and epochs=50.

Jitter value		Bibtex	Delicious	Mediamill	EUR-Lex
	P@1	60.63	67.18	84.05	74.45
10^{-6}	P@3	36.11	61.88	66.81	58.78
	P@5	26.69	57.24	52.51	47.45
	$\mathcal{F}$	-580368.93	-9300247.89	-82464394.56	-46205521.53
	P@1	60.55	67.22	83.80	74.42
10^{-7}	P@3	36.11	61.88	66.58	58.74
	P@5	26.64	57.20	52.36	47.40
	$\mathcal{F}$	-580369.03	-9300248.09	-214513820.87	-46205521.66
	P@1	60.59	67.22	83.74	74.42
10^{-8}	P@3	36.11	61.88	66.58	58.72
	P@5	26.64	57.20	52.23	47.40
	$\mathcal{F}$	-580369.04	-9300248.11	-287236121.47	-46205521.68
	P@1	60.59	67.22	83.76	74.42
10^{-9}	P@3	36.11	61.88	66.46	58.72
	P@5	26.64	57.20	52.16	47.41
	$\mathcal{F}$	-580369.04	-9300248.12	-299410784.72	-46205521.68
	P@1	60.59	67.22	83.69	74.42
10^{-10}	P@3	36.11	61.88	66.55	58.72
	P@5	26.64	57.20	52.16	47.41
	$\mathcal{F}$	-580369.04	-9300248.12	-300721969.03	-46205521.68

Table 4.22: Sensitivity analysis of the jitter term used for the FP for each small-scale dataset. We run the ARD method using the same seed for our number generator in order to remove randomness from the algorithm and allow comparisons for a given dataset. Five different jitter values are tested, $\{10^{i+5}\}_{i=1}^{5}$, where the first 4 rows correspond to jitter=10^{-6}, the next 5th to 8th row corresponds to jitter=10^{-7} and so on. All the P@k metrics (Eq. (1.21)) and $\mathcal{F}$ are computed after training our model. For all datasets we use $M = 100$, $P = 80$, and epochs=50.

Small-scale Datasets		Bibtex	Delicious	EUR-Lex	Mediamill
LINEAR-ℓ	P@1	66.07 ± 0.09	66.72 ± 0.23	81.13 ± 0.19	83.30 ± 0.04
	P@3	40.68 ± 0.11	61.09 ± 0.16	66.83 ± 0.15	66.10 ± 0.05
	P@5	29.90 ± 0.04	55.94 ± 0.09	55.13 ± 0.05	51.47 ± 0.05
ARD	P@1	**66.46 ± 0.12**	**69.02 ± 0.16**	**82.48 ± 0.13**	**84.95 ± 0.07**
	P@3	**41.05 ± 0.08**	**63.22 ± 0.11**	**68.43 ± 0.06**	**67.90 ± 0.03**
	P@5	**30.26 ± 0.06**	**58.64 ± 0.07**	**56.54 ± 0.04**	**54.17 ± 0.05**
ARD-FX	P@1	63.04 ± 0.13	66.43 ± 0.29	75.32 ± 0.17	80.27 ± 0.07
	P@3	39.27 ± 0.11	61.24 ± 0.20	62.07 ± 0.12	66.19 ± 0.05
	P@5	29.11 ± 0.06	56.83 ± 0.11	51.21 ± 0.08	53.51 ± 0.06
Large-scale Datasets		**Wiki10**	**AmazonCat**	**Delicious-Large**	**WikiLSHTC**
ARD	P@1	83.90	92.48	42.61	28.24
	P@3	70.73	77.21	39.42	19.69
	P@5	61.00	62.16	37.42	14.15
ARD-FX	P@1	77.79	-	-	-
	P@3	64.23	-	-	-
	P@5	55.35	-	-	-

Table 4.23: Predictive Performance of the MLGPF model for eight multi-label (test) datasets. Those methods that have not been applied on a dataset are indicated with the "-" sign. Error bars (one standard deviation) are provided for the small-scale datasets. The experimental settings of each method can be found in Table 4.20.

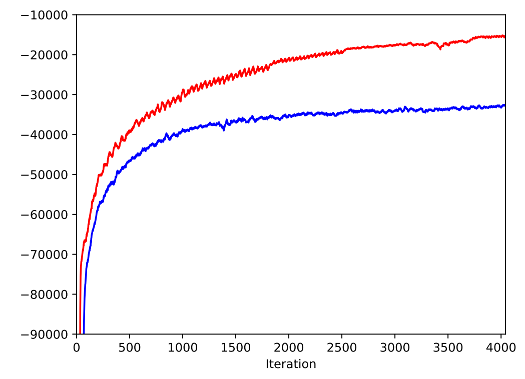
Dataset	LINEAR-ℓ	ARD-FX	ARD
Bibtex	0.63	0.62	0.63
Delicious	2.95	3.06	2.94
Mediamill	5.09	4.12	4.35
EUR-Lex	6.12	6.65	6.13
Wiki10	-	49.2	46.1
AmazonCat	-	-	2072.3
Delicious-Large	-	-	392.7

Table 4.24: Computational time (in minutes per epoch) of the MLGPF model for six multi-label (training) datasets corresponding to the methods. Those methods that have not been applied on a dataset are indicated with the "-" sign.

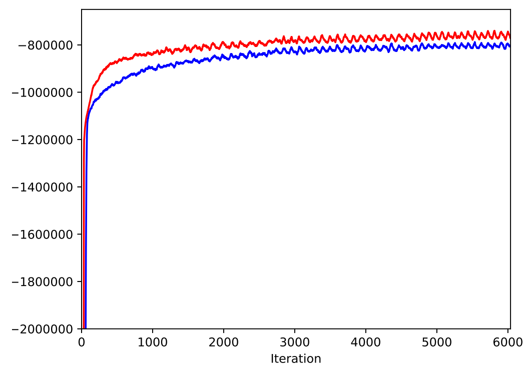
sion, the panel of figures in 4.21 shows the optimized values of the inverse length-scales w_d, for each dimension d, given in descending order. Higher values indicate higher feature importance while dimensions with values close to zero do not provide any signal to the used model. Those plots imply that there is much redundancy in all those high-dimensional datasets that hinders the predictive performance of the model, verifying the poor performance we saw to previous experiments. Training times can be also found in Table 4.24

In the final part of this section, we compare the performance of our model using the ARD method, since it is the most competitive of all the others overall, with other state-of-the-art algorithms specifically designed to cope with XML tasks. More specifically, we choose the SLEEC Bhatia et al. (2015), PFastreXML Jain et al. (2016), FastXML Prabhu and Varma (2014b), and the PD-Sparse Yen et al. (2016) method (see Section 1.2.2 of Chapter 1 for a short description of the methods). The choice of those algorithms is based on the fact that their predictive power is higher than other methods in the XML literature, for both small and large-scale datasets, therefore making comparisons more fair. Further, an extra reason of that choice is that all the P@k-based results for those methods can be found reported in Repository (2010), for almost all the datasets we use in this chapter. Moreover, those methods span a wide range of XML algorithms categories like Embedding based methods(SLEEC), tree-based methods (PFastreXML, FastXML), and methods that based to 1-vs-All strategy (PD-Sparse) where more details about those categories and their methods can be found in Section 1.2 of Chapter 1. It should be also noted that known deep learning methods such as convolutional neural networks have been succefully deployed for extreme multi-label classification (Liu et al., 2017) leading to predictive power. Nevertheless, we do not report their results here since they are heavily relied on GPU computations and their performance is not as high as the aforementioned methods for the datasets used here.

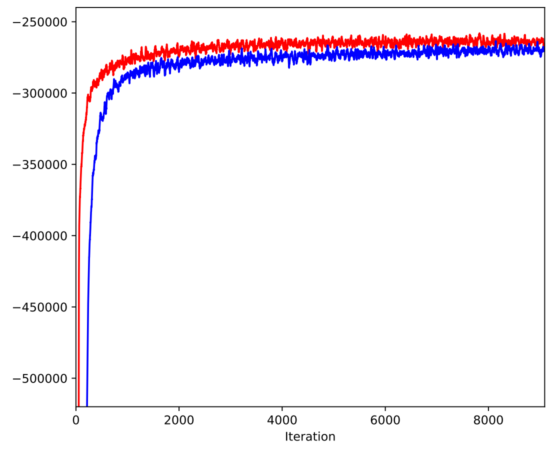
All the P@k scores for each method are collectively compared in Table 4.25 where the the top-2 P@k scores for each dataset are in bold. Nonetheless, for further comparisons with more methods, we refer the to tables 6 and 8 of the XML repository (Repository, 2010). All the scores indicate that our methods is not only equally competitive with all the chosen baselines, but in some cases the MLGPF model's performance surpasses all the other algorithms. In some other cases, the accuracy difference between our method and the chosen ones exceeded the 10%. For example, in Eurlex dataset, our method gives a score P@1 = 81.44% while FastXML could not predict better than 71.36% which is very positive for a non-parametric model under Bayesian treatment in conjunction with those prodigious dimensions. In general, we observe that our method performs very well when applied to small-scale datasets. Nevertheless, the MLGPF model exhibits similar competitive performance on large-scale datasets too, attaining scores very close to the top-1 method for each dataset. Finally, the WikiLSHTC is the only dataset



(a)

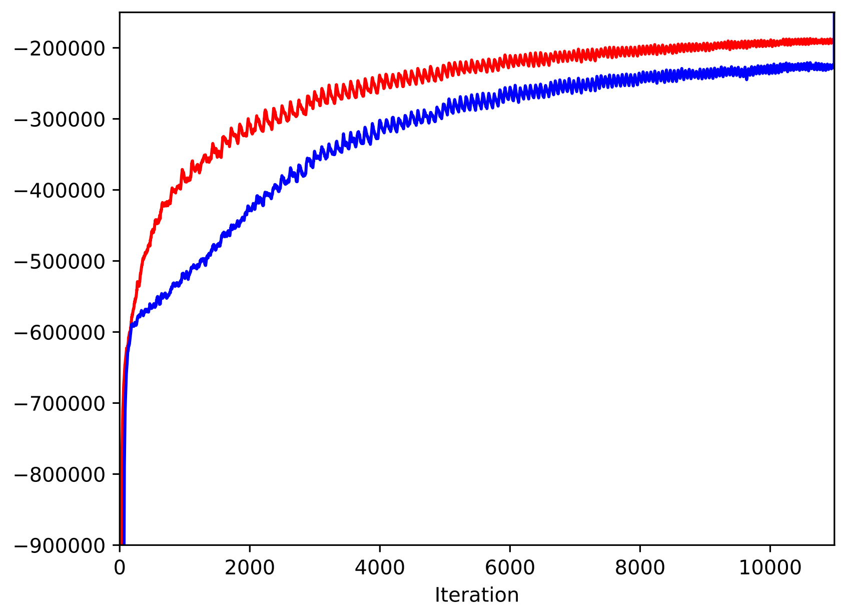


(b)

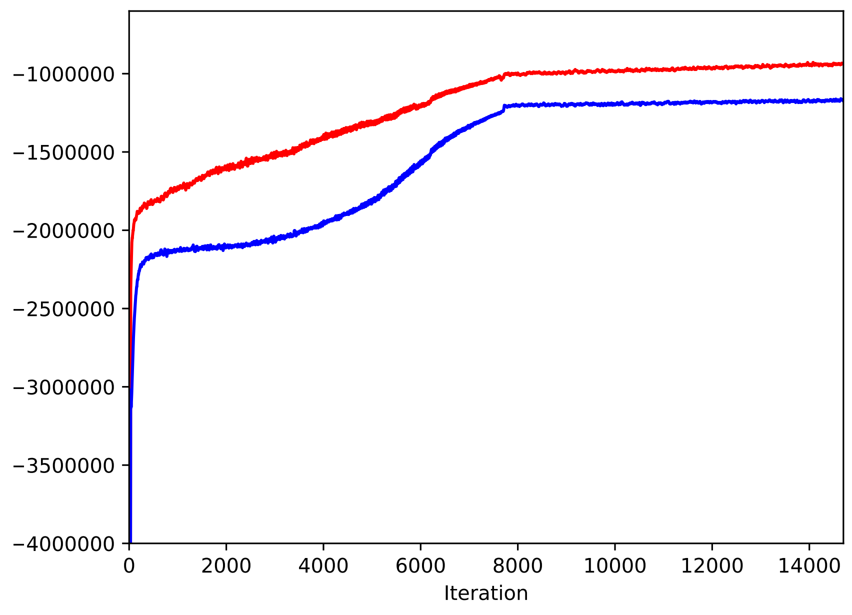


(c)

Figure 4.19: Evolution of the lower bound for (a) Bibtex, (b) Delicious, and (c) Mediamill. The blue line corresponds to maximization of the lower bound using fixed inducing inputs Z, while the red one to optimized Z. The ARD kernel is used for each dataset while all the other parameters are set as described in Table 4.20.

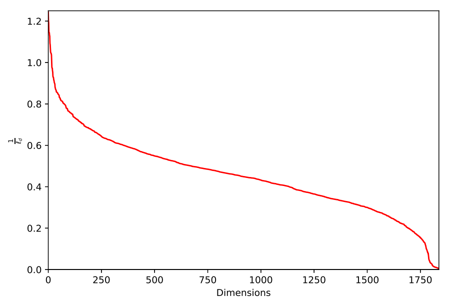


(a)

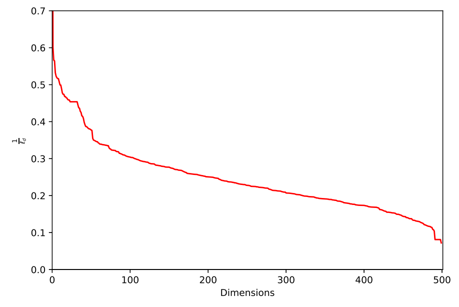


(b)

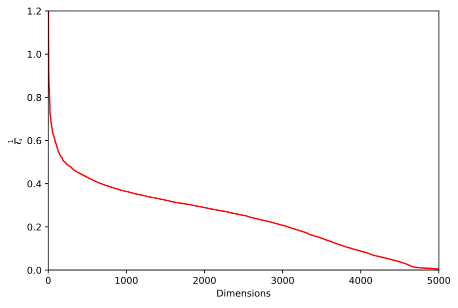
Figure 4.20: Evolution of the lower bound for (a) EUR-Lex and (b) Wiki10. The blue line corresponds to maximization of the lower bound using fixed inducing inputs Z, while the red one to optimized Z. The ARD kernel is used for each dataset while all the other parameters are set as described in Table 4.20.



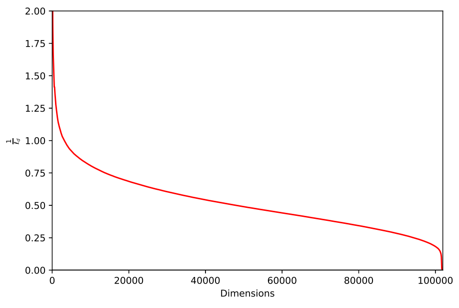
(a)



(b)



(c)



(d)

Figure 4.21: Values of the the optimized inverse lengthscales w_d , $d = 1, \cdots, D$ across all input dimensions, for (a) Bibtex and (b) Delicious, (c) Eurlex and (d) Wiki10. The optimized values obtained by the ARD method where its performance is presented in Tables 4.23.

where our method fails to compete with the predictive power of the state-of-the art methods, mainly because of the exiguous number of epochs used.

Multi-class classification results based on the MLGPF model

Having presented and discussed a multitude of experimental results produced by the MLGPF model for a series of different multi-label datasets, we would like to test how this model performs when we restrict the data points to be labelled by exactly one class each time. In other words, we would like to apply the MLGPF model to the special subcategory of multi-label classification tasks, namely the multi-class classification. For this purpose, we make use of our familiar from the previous chapter MNIST dataset and we compare results with those of Section 3.8.3, Chapter 3.

Initially, we set $P = K = 10$ while all the other parameters are kept the same as the RM-F method in Table 3.2, i.e. $M = 100$, $|\mathcal{B}| = 1000$ and epochs=100 where in all cases the same SE kernel is used. Moreover, both of the available parametrization

Dataset		MLGPF	SLEEC	PfastreXML	FastXML	PD-Sparse
Bibtex	P@1	**66.46**	**65.08**	63.46	63.42	61.29
	P@3	**41.05**	**39.64**	39.22	39.23	35.82
	P@5	**30.26**	**28.87**	29.14	28.86	25.74
Delicious	P@1	**69.02**	67.59	67.13	**69.61**	51.82
	P@3	**63.22**	61.38	62.33	**64.12**	44.18
	P@5	**58.64**	56.56	58.62	**59.27**	38.95
Mediamill	P@1	**84.95**	**87.82**	83.98	84.22	81.86
	P@3	**67.90**	**73.45**	67.37	67.33	62.52
	P@5	**54.17**	**59.17**	53.02	53.04	45.11
EUR-Lex	P@1	**82.48**	**79.26**	75.45	71.36	76.43
	P@3	**68.43**	**64.30**	62.70	59.90	60.37
	P@5	**56.54**	52.33	**52.51**	50.39	49.72
Wiki10	P@1	**83.90**	**85.88**	83.57	83.03	-
	P@3	**70.73**	**72.98**	68.61	67.47	-
	P@5	**61.00**	**62.70**	59.10	57.76	-
AmazonCat	P@1	**92.48**	90.53	91.75	**93.11**	90.60
	P@3	77.21	76.33	**77.97**	**78.20**	75.14
	P@5	62.16	61.52	**63.68**	63.41	60.69
Delicious-Large	P@1	42.61	**47.85**	41.72	**43.07**	34.37
	P@3	**39.42**	**42.21**	37.83	38.66	29.48
	P@5	**37.42**	**39.43**	35.58	36.19	27.04
WikiLSHTC	P@1	28.24	54.83	**56.05**	49.75	**61.26**
	P@3	19.69	33.42	**36.79**	33.10	**39.48**
	P@5	14.15	23.85	**27.09**	24.45	**28.79**

Table 4.25: Performance comparison between the MLGPF model using ARD and other state-of-the-art methods. The top-2 P@k ($k = 1, 3, 5$) for each dataset are in bold. “-” sign indicates absence of reported results for a dataset.

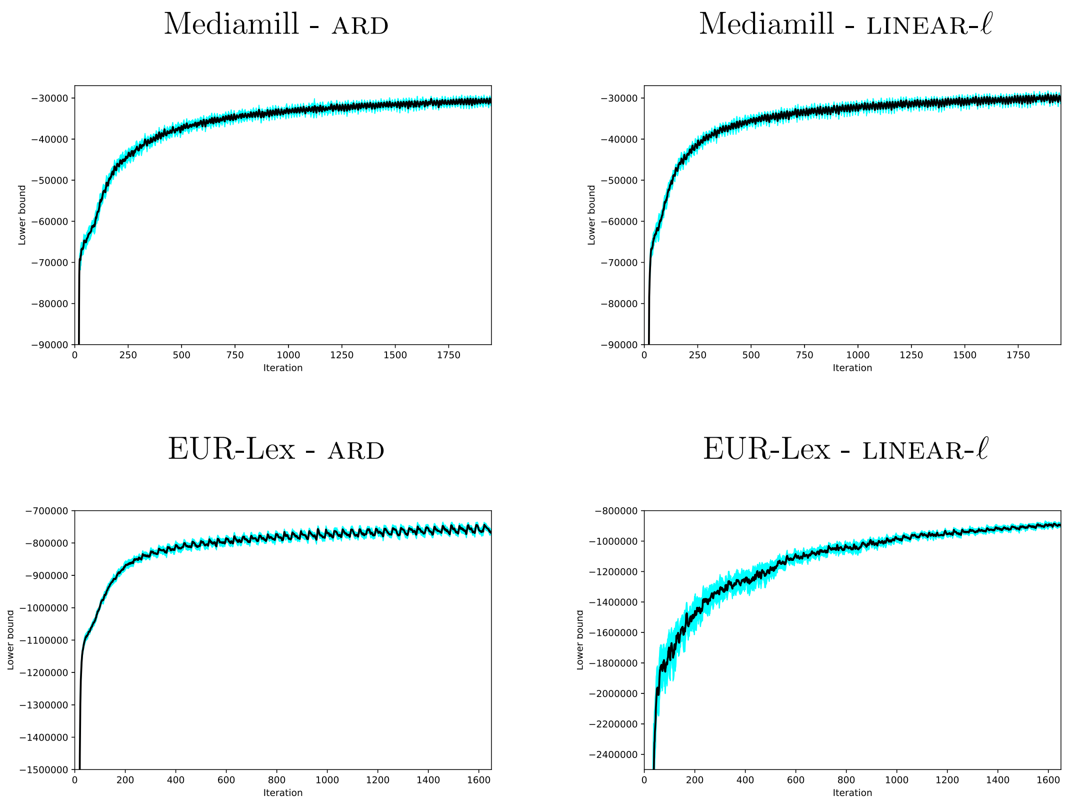


Figure 4.22: Evolution of the (mean) lower bound for Mediamill (first row) and EUR-Lex (second row). Right column corresponds to LINEAR-ℓ method while the left one to ARD. All the other parameters are set as described in Table 4.20. Shaded area depicts 68% confidence intervals.

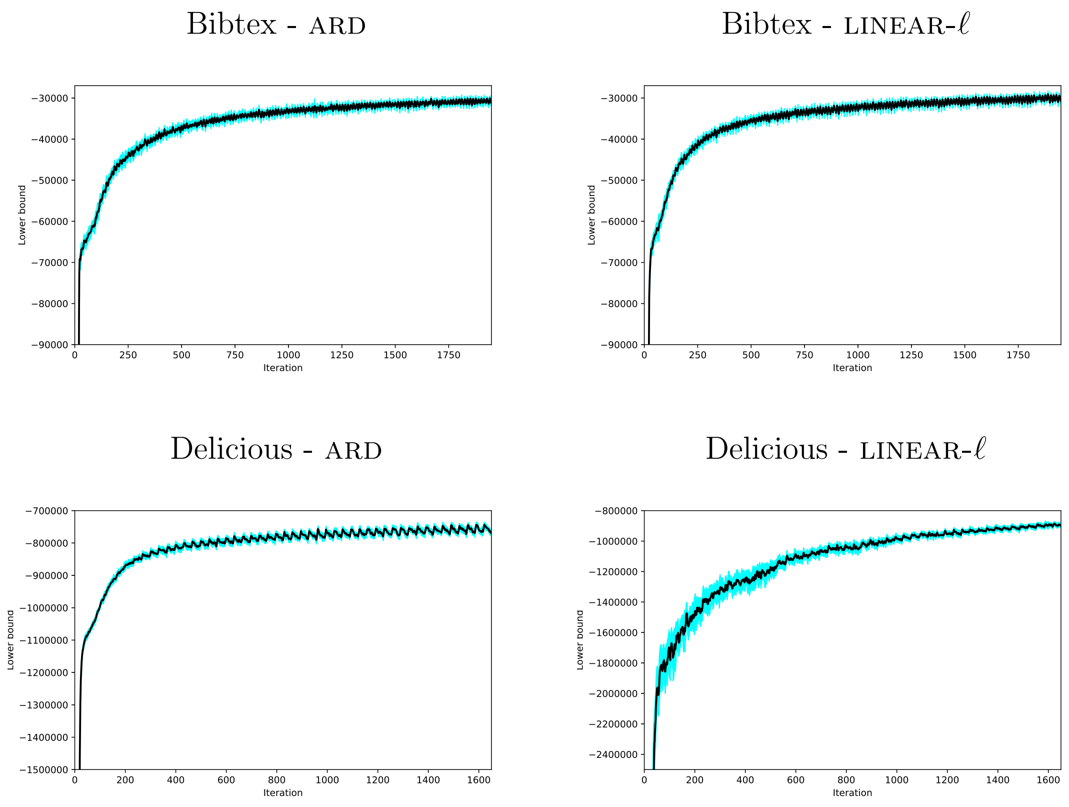
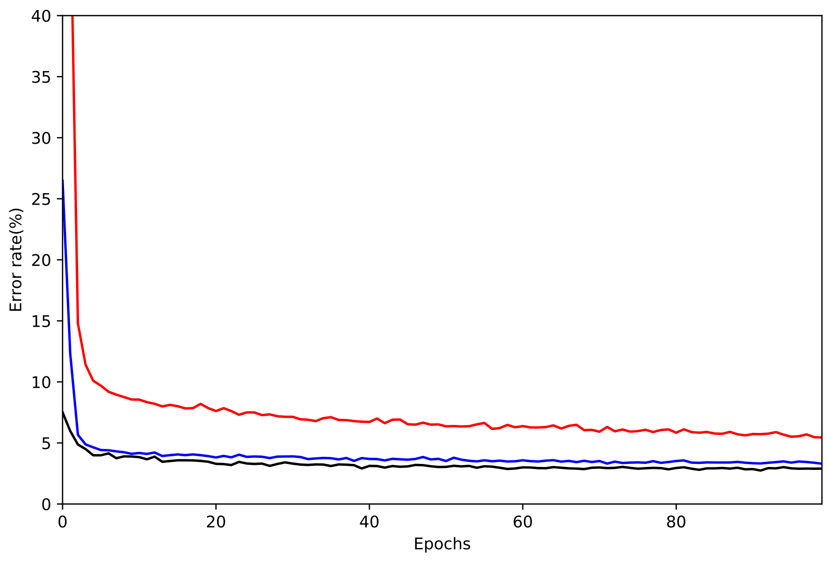
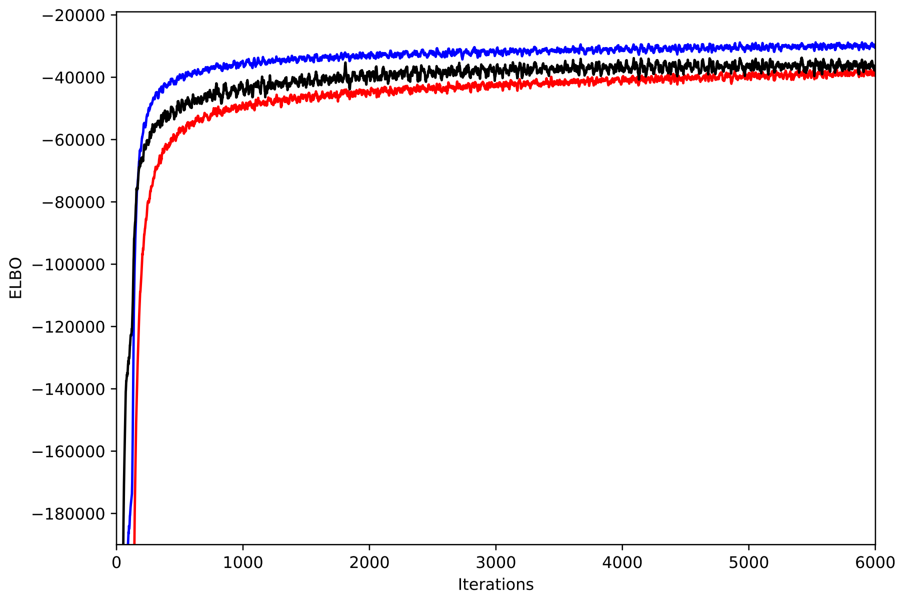


Figure 4.23: Evolution of the (mean) lower bound for Bibtex (first row) and Delicious (second row). Right column corresponds to LINEAR-ℓ method while the left one to ARD. All the other parameters are set as described in Table 4.20. Shaded area depicts 68% confidence intervals.



(a)



(b)

Figure 4.24: (a) Error rates per epoch comparison between the methods SE (blue line) and SE-PP (red line) of the MLGPF model, using $M = 100$, and RM-F (black line). On the other hand, (b) depicts the evolution of the lower bound per iteration between the two parametrizations of the MLGPF model and RM-F. Colour line description is the same as (a). $N = 60000$ and $|B| = 1000$, thus each epoch corresponds to 60 iterations.

	Error rate (%)	time (sec/epoch)
$P = 5$	4.0	104.7
$P = 10$	3.1	140.6
$P = 20$	3.0	230.2
RM-F	2.6	141.1

Table 4.26: Error rates and computational times in seconds per epoch for the MLGPF model using different number of GPs P in tandem with RM-F method's error given by Table 3.2. For all cases an SE kernel and FP are used while both number of inducing inputs and epochs are set to 100. Nevertheless, computational times are calculated by using $M = 1000$.

schemes are employed and then compared where the plots of the results can be found in Fig. 4.24. The deficiency of the PP is one more time apparent in terms of both lower bound values and error rates. On the other hand, the MLGPF SE method based on the FP, attains error rates very close to the optimal method RM-F, providing in that way an efficient alternative to the RobustMax likelihood. By observing the lower bounds in Fig. (b) 4.24, we interestingly discern that using the MLGPF likelihood leads to higher ELBO values than the RobustMax one although predictive performance of the former is inferior of the latter. Apart from that, we also tested the MLGPF model using $P = 5$ and $P = 20$ in order to investigate the potential benefits (or drawbacks) of increasing (or decreasing) the number of GPs comparing to the number of total classes K. As Table 4.26 reveals, reducing the number of used GPs deteriorates the performance while a slight decrease of the error is observed when P is doubled from 10 to 20. Moreover, regarding computational times, both RM-F and SE with $P = 10$ are almost identical while increasing P leads to an extra computational burden. Finally, by investigating the behaviour of the inducing inputs Z, before and after the optimization procedure, we obtain similar results as in Section 3.8.3 of Chapter 3, where it is observed that the optimized inducing inputs move away from their initial values and they are anchored in values which form the decision boundary.

4.6 Conclusion

In this chapter, we move to a more general category of classification tasks where multiple labels can be assigned to a given data. We propose a model based on a combination of Gaussian processes and a linear latent variable model that is scalable no matter how large the values of N, D, and K, whilst correlations between labels can be captured. These attributes pave the way for applying our model to multi-label datasets where sizes are prohibitively large for other GP-based models. As we also notice in the previous chapter, the MLGPF model using the FP leads to better performance than using the PP one, as all the results imply in Section

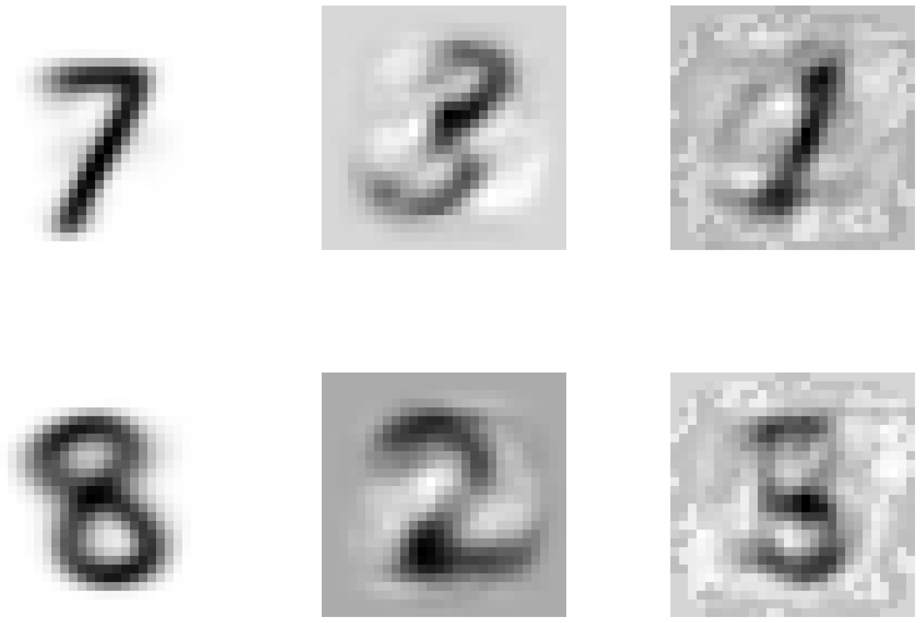


Figure 4.25: Visualization of the inducing inputs based on the results of Fig. 4.24. Each row corresponds to a different inducing input. Left: Initialized inducing input with a k-means center. Center: Optimized inducing point by using the MLGPF model with PP. Right: Optimized inducing point by using the MLGPF model with FP.

4.5.3. Moreover, we observe that our model is able to be trained using our Tensorflow implementation for extremely large datasets while its predictive performance is competitive with other state-of-the-arts methods (Table 4.25). Nevertheless, the main drawback of our method is the training time needed, especially for large scale datasets, where some of them may require a few weeks training in contrast to other state-of-the-art methods that need one or two days training at most (Bhatia et al., 2015; Prabhu and Varma, 2014b; Jain et al., 2016; Yen et al., 2016) for the same datasets. We should also stress the importance of using kernels that include lengthscales for each of the input dimensions which is supported by all the experimental results. Further to that, the methods that use subspace inducing inputs do not seem to provide any essential advantage over those that use full inducing inputs for the small-scale datasets while for the large-scale ones, this can lead to significant deterioration of their predictive power. Finally, we find that our model can be successfully used for multi-class classification tasks with predictive performance very close to the state-of-the-art method for sparse GP models for multi-class classification tasks (Hensman et al., 2015b).

Chapter 5

Conclusion

5.1 Contributions

In this thesis we study how multi-class and multi-label tasks can be approached by Gaussian process models using variational inference tools in order to be able to deal with extremely large number of training instances, labels and input dimensions. The main contributions are briefly summarized here.

- In Chapter 2 we present a new parsimonious parametrization for the variational distribution which is able to reduce the number of variational parameters and can maintain flexibility of the model. Nonetheless, empirical results in Chapters 3 and 4 indicate that the conventional full parametrization should be preferred since it provides better performance for the methods that use it while the more variational parameters required by the full parametrization does not seem to be restrictive in practice. Moreover, a representation trick of the inducing input matrix Z is introduced which is able to reduce the number of variational parameters in cases where the input dimensions are extremely large. Its usefulness is investigated in Chapter 4 where all the results indicate that it does not provide any essential benefits for the small-scale datasets while for the large-scale ones it significantly limits the performance of the MLGPF model.

- In Chapter 3 we develop a method that is based on the sparse Gaussian process variational inference framework of Hensman et al. (2015b) and a computational efficiently lower bound of Titsias (2016) on the softmax function. The method is suitable for multi-class classification problems and it can scale well for a large number of data point and classes. The experiments conducted using that method show that it can achieve performance very close to other GP-based methods in literature despite its approximating nature. Furthermore, using our own implementation based on Tensorflow for the method of Hensman et al. (2015b) with a large number of inducing inputs, we achieve the highest reported performance for GP-models without convolutional assumptions. We also investigate the case where a regression problem is transformed into a classification one and then our method is applied. This is motivated by the high scalability our method attains over the number of classes. Nevertheless, results show that using a high number of classes to transform the regression problem does not lead to better predictive performance.

- Finally, in Chapter 4 we probe the use of Gaussian processes into multi-label classification problems for the first time. More specifically, we focus on a specific category of multi-label classification tasks that involve extremely large number of data instances, labels, and input dimensions. For instance, one of those datasets has $N = 196606$ training data points, $K = 205443$ labels, and $D = 782585$ input dimensions. For dealing with this kind of datasets, we introduce a method that is based on a latent factor model and the variation inference for sparse GPs framework. A series of experiments on a wide range of multi-label datasets shows that our method achieves performance similar and sometimes higher than other state-of-the-art algorithms for the extreme multi-label classification task. This success is inextricably intertwined with the Tensorflow-based code we implement. Moreover, the experiments reveal the importance of having different lengthscales for each input dimension. We also find that our method can be deployed to tackle multi-class classification tasks effectively, giving predictive accuracy close to the method of Hensman et al. (2015b). Its main drawback is the large training time required for the large-scale datasets comparing to other more efficient algorithms in literature.

5.2 Future work

In Chapter 3 we obtain a stochastic unbiased estimate of (3.20) by using random sub-sampling over the set of classes to get a class and then we randomly choose another class ℓ from the remaining set of classes (Section 3.4). Nonetheless, we could also investigate the performance of our model by imposing a proposal distribution over the set of the remaining classes that favours classes with large latent function values in order to conduct importance sampling as it is also proposed in Titsias (2016). Moreover, we could probe the use of the subspace inducing inputs for all the methods described in that chapter for multi-class classification for datasets with larger input dimensions than MNIST.

Regarding the multi-label classification problems, a possible future research direction would be to combine tree structures that learn a label hierarchy with Gaussian processes in a similar manner as it is introduced in Prabhu et al. (2018b) where linear classifiers are used in the leaves of the trees. Furthermore, another potential research path would be to investigate the performance of our model on datasets that consist of dense inputs vectors and dense label vectors as well. Nonetheless, we would like the number of training instances to be sufficiently large in order to refer to variational approximation methods. For instance, applications from cognitive neuroscience could be considered where our model would be deployed to classify high-dimensional fMRI data to multiple attributes of the stimuli (Bobadilla-Suarez et al., 2019). A further study of the potential benefits of using more sophisticated kernel functions for these kind of problems could be also pursued in the future. Additionally, there is an idea of extending our MLGPF model to be suitable for a non-binary multi-label setting where each label can take more than two values. Possible connections and comparisons of our model with the work of Moreno-Muñoz et al. (2018) would be of interest to us too.

Finally, inspired by the encouraging results of Salimbeni et al. (2018) for sparse Gaussian process models with non-conjugate likelihoods, we would like to study the effect of using natural gradients in the optimization of the ELBO on the MLGPF model.

Bibliography

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., et al. (2016). Tensorflow: A system for large-scale machine learning. In *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*, pages 265–283.

Agrawal, R., Gupta, A., Prabhu, Y., and Varma, M. (2013). Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *Proceedings of the 22nd international conference on World Wide Web*, pages 13–24. ACM.

Alvarez, M. and Lawrence, N. D. (2009). Sparse convolved Gaussian processes for multi-output regression. In *Advances in neural information processing systems*, pages 57–64.

Álvarez, M., Luengo, D., Titsias, M., and Lawrence, N. D. (2010). Efficient multi-output Gaussian processes through variational inducing kernels. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 25–32.

Álvarez, M. A. and Lawrence, N. D. (2011). Computationally efficient convolved multiple output Gaussian processes. *J. Mach. Learn. Res.*, 12:1459–1500.

Alvarez, M. A., Rosasco, L., Lawrence, N. D., et al. (2012). Kernels for vector-valued functions: A review. *Foundations and Trends® in Machine Learning*, 4(3):195–266.

Amari, S.-I. (1982). Differential geometry of curved exponential families-curvatures and information loss. *The Annals of Statistics*, pages 357–385.

Amari, S.-I. (1998). Natural gradient works efficiently in learning. *Neural computation*, 10(2):251–276.

Azevedo-Filho, A. and Shachter, R. D. (1994). Laplace's method approximations for probabilistic inference in belief networks with continuous variables. In *Uncertainty Proceedings 1994*, pages 28–36. Elsevier.

Babbar, R. and Schölkopf, B. (2017). Dismec: Distributed sparse machines for extreme multi-label classification. In *Proceedings of the Tenth ACM International Conference on Web Search and Data Mining*, pages 721–729. ACM.

Baglama, J. and Reichel, L. (2005). Augmented implicitly restarted lanczos bidiagonalization methods. *SIAM Journal on Scientific Computing*, 27(1):19–42.

Barber, D. and Wiegerinck, W. (1999). Tractable variational structures for approximating graphical models. In *Advances in Neural Information Processing Systems*, pages 183–189.

Baroni, M., Dinu, G., and Kruszewski, G. (2014). Don't count, predict! a systematic comparison of context-counting vs. context-predicting semantic vectors. In *Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, volume 1, pages 238–247.

Bauer, M., van der Wilk, M., and Rasmussen, C. E. (2016). Understanding probabilistic sparse Gaussian process approximations. In *Advances in Neural Information Processing Systems 29*, pages 1533–1541. Curran Associates, Inc.

Bhatia, K., Jain, H., Kar, P., Varma, M., and Jain, P. (2015). Sparse local embeddings for extreme multi-label classification. In *Advances in Neural Information Processing Systems*, pages 730–738.

Bi, W. and Kwok, J. (2013). Efficient multi-label classification with many labels. In *International Conference on Machine Learning*, pages 405–413.

Billingsley, P. (2008). *Probability and measure.* John Wiley & Sons.

Bishop, C. M., Lawrence, N. D., Jaakkola, T., and Jordan, M. I. (1998). Approximating posterior distributions in belief networks using mixtures. In *Advances in neural information processing systems*, pages 416–422.

Blei, D. M., Kucukelbir, A., and McAuliffe, J. D. (2017). Variational inference: A review for statisticians. *Journal of the American Statistical Association*, 112(518):859–877.

Blei, D. M., Ng, A. Y., and Jordan, M. I. (2003). Latent Dirichlet allocation. *Journal of machine Learning research*, 3(Jan):993–1022.

Bobadilla-Suarez, S., Ahlheim, C., Mehrotra, A., Panos, A., and Love, B. C. (2019). Measures of neural similarity. *BioRxiv*, page 439893.

Bonilla, E. V., Chai, K. M., and Williams, C. (2008). Multi-task Gaussian process prediction. In *Advances in neural information processing systems*, pages 153–160.

Bouchard, G. (2007). Efficient bounds for the softmax function and applications to approximate inference in hybrid models.

Boutell, M. R., Luo, J., Shen, X., and Brown, C. M. (2004). Learning multi-label scene classification. *Pattern recognition*, 37(9):1757–1771.

Boyle, P. and Frean, M. (2005). Multiple output Gaussian process regression.

Brown, R. (1828). Xxvii. a brief account of microscopical observations made in the months of june, july and august 1827, on the particles contained in the pollen of plants; and on the general existence of active molecules in organic and inorganic bodies. *The Philosophical Magazine*, 4(21):161–173.

Cao, Y., Brubaker, M. A., Fleet, D. J., and Hertzmann, A. (2013). Efficient optimization for sparse Gaussian process regression. In *Advances in Neural Information Processing Systems*, pages 1097–1105.

Chai, K. M. A. (2012). Variational multinomial logit Gaussian process. *Journal of Machine Learning Research*, 13(Jun):1745–1808.

Chen, Y.-N. and Lin, H.-T. (2012). Feature-aware label space dimension reduction for multi-label classification. In *Advances in Neural Information Processing Systems*, pages 1529–1537.

Cireşan, D., Meier, U., and Schmidhuber, J. (2012). Multi-column deep neural networks for image classification. *arXiv preprint arXiv:1202.2745*.

Ciresan, D. C., Meier, U., Masci, J., Gambardella, L. M., and Schmidhuber, J. (2011). Flexible, high performance convolutional neural networks for image classification. In *Twenty-Second International Joint Conference on Artificial Intelligence*.

Cisse, M. M., Usunier, N., Artieres, T., and Gallinari, P. (2013). Robust bloom filters for large multilabel classification tasks. In *Advances in Neural Information Processing Systems*, pages 1851–1859.

Clare, A. and King, R. D. (2001). Knowledge discovery in multi-label phenotype data. In *European Conference on Principles of Data Mining and Knowledge Discovery*, pages 42–53. Springer.

Cressie, N. (1992). Statistics for spatial data. *Terra Nova*, 4(5):613–617.

Cunningham, J. P., Shenoy, K. V., and Sahani, M. (2008). Fast Gaussian process methods for point process intensity estimation. In *Proceedings of the 25th international conference on Machine learning*, pages 192–199. ACM.

Cutajar, K., Osborne, M., Cunningham, J., and Filippone, M. (2016). Preconditioning kernel matrices. In *International Conference on Machine Learning*, pages 2529–2538.

Dai, Z., Alvarez, M., and Lawrence, N. (2017). Efficient modeling of latent information in supervised learning using Gaussian processes. In *Advances in Neural Information Processing Systems*, pages 5131–5139.

Damianou, A., Ek, C., Titsias, M., and Lawrence, N. (2012). Manifold relevance determination. *arXiv preprint arXiv:1206.4610*.

Damianou, A. and Lawrence, N. (2013). Deep gaussian processes. In *Artificial Intelligence and Statistics*, pages 207–215.

Damianou, A. C., Titsias, M., and Lawrence, N. D. (2011). Variational Gaussian process dynamical systems. In Shawe-Taylor, J., Zemel, R., Bartlett, P., Pereira, F., and Weinberger, K., editors, *Advances in Neural Information Processing Systems 24*, pages 2510–2518.

Dezfouli, A. and Bonilla, E. V. (2015). Scalable inference for Gaussian process models with black-box likelihoods. In *Advances in Neural Information Processing Systems*, pages 1414–1422.

Duchi, J., Hazan, E., and Singer, Y. (2011). Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research*, 12(Jul):2121–2159.

Einstein, A. (1905). Über die von der molekularkinetischen theorie der wärme geforderte bewegung von in ruhenden flüssigkeiten suspendierten teilchen. *Annalen der physik*, 322(8):549–560.

Elisseeff, A. and Weston, J. (2002). A kernel method for multi-labelled classification. In *Advances in neural information processing systems*, pages 681–687.

Engel, Y., Mannor, S., and Meir, R. (2005). Reinforcement learning with Gaussian processes. In *Proceedings of the 22nd international conference on Machine learning*, pages 201–208. ACM.

Flaxman, S., Wilson, A., Neill, D., Nickisch, H., and Smola, A. (2015). Fast kronecker inference in Gaussian processes with non-Gaussian likelihoods. In *International Conference on Machine Learning*, pages 607–616.

Fox, C. W. and Roberts, S. J. (2012). A tutorial on variational bayesian inference. *Artificial intelligence review*, 38(2):85–95.

Fröhlich, B., Rodner, E., Kemmler, M., and Denzler, J. (2013). Large-scale gaussian process multi-class classification for semantic segmentation and facade recognition. *Machine vision and applications*, 24(5):1043–1053.

Gal, Y. and Turner, R. (2015). Improving the Gaussian process sparse spectrum approximation by representing uncertainty in frequency inputs.

Gal, Y., Van Der Wilk, M., and Rasmussen, C. E. (2014). Distributed variational inference in sparse Gaussian process regression and latent variable models. In *Advances in Neural Information Processing Systems*, pages 3257–3265.

Gelfand, A. E. and Smith, A. F. (1990). Sampling-based approaches to calculating marginal densities. *Journal of the American statistical association*, 85(410):398–409.

Gelfand, I. M., Silverman, R. A., et al. (2000). *Calculus of variations.* Courier Corporation.

Ghahramani, Z. (2013). Bayesian non-parametrics and the probabilistic approach to modelling. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 371(1984):20110553.

Ghamrawi, N. and McCallum, A. (2005). Collective multi-label classification. In *Proceedings of the 14th ACM international conference on Information and knowledge management*, pages 195–200. ACM.

Gibaja, E. and Ventura, S. (2014). Multi-label learning: a review of the state of the art and ongoing research. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 4(6):411–444.

Gibaja, E. and Ventura, S. (2015). A tutorial on multilabel learning. *ACM Comput. Surv.*, 47(3):52:1–52:38.

Girolami, M. and Rogers, S. (2006). Variational bayesian multinomial probit regression with Gaussian process priors. *Neural Computation*, 18(8):1790–1817.

Gittens, A. and Mahoney, M. W. (2016). Revisiting the nyström method for improved large-scale machine learning. *The Journal of Machine Learning Research*, 17(1):3977–4041.

Goovaerts, P. et al. (1997). *Geostatistics for natural resources evaluation.* Oxford University Press on Demand.

Hastings, W. K. (1970). Monte carlo sampling methods using markov chains and their applications.

Hensman, J., Durrande, N., Solin, A., et al. (2017). Variational fourier features for Gaussian processes. *Journal of Machine Learning Research*, 18(151):1–151.

Hensman, J., Fusi, N., and Lawrence, N. D. (2013). Gaussian processes for big data. In *Conference on Uncertainty in Artificial Intellegence*, pages 282–290. auai.org.

Hensman, J., Matthews, A. G., Filippone, M., and Ghahramani, Z. (2015a). Mcmc for variationally sparse Gaussian processes. In *Advances in Neural Information Processing Systems*, pages 1648–1656.

Hensman, J., Matthews, A. G. d. G., and Ghahramani, Z. (2015b). Scalable variational Gaussian process classification. In *Proceedings of the Eighteenth International Conference on Artificial Intelligence and Statistics*.

Hernández-Lobato, D. and Hernández-Lobato, J. M. (2016a). Scalable Gaussian process classification via expectation propagation. In *AISTATS*.

Hernández-Lobato, D. and Hernández-Lobato, J. M. (2016b). Scalable Gaussian process classification via expectation propagation. In *Artificial Intelligence and Statistics*, pages 168–176.

Hernández-Lobato, D., Hernández-Lobato, J. M., and Dupont, P. (2011). Robust multi-class Gaussian process classification. In *Advances in neural information processing systems*, pages 280–288.

Hinton, G., Srivastava, N., and Swersky, K. (2012). Neural networks for machine learning lecture 6a overview of mini-batch gradient descent.

Hjort, N. L., Holmes, C., Müller, P., and Walker, S. G. (2010). *Bayesian nonparametrics*, volume 28. Cambridge University Press.

Hoang, Q. M., Hoang, T. N., and Low, K. H. (2017). A generalized stochastic variational bayesian hyperparameter learning framework for sparse spectrum Gaussian process regression. In *Thirty-First AAAI Conference on Artificial Intelligence*.

Hoffman, M. D., Blei, D. M., Wang, C., and Paisley, J. (2013). Stochastic variational inference. *J. Mach. Learn. Res.*, 14(1):1303–1347.

Hsu, D. J., Kakade, S. M., Langford, J., and Zhang, T. (2009). Multi-label prediction via compressed sensing. In *Advances in neural information processing systems*, pages 772–780.

Jain, H., Prabhu, Y., and Varma, M. (2016). Extreme multi-label loss functions for recommendation, tagging, ranking & other missing label applications. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 935–944. ACM.

Jasinska, K., Dembczynski, K., Busa-Fekete, R., Pfannschmidt, K., Klerx, T., and Hullermeier, E. (2016). Extreme f-measure maximization using sparse probability estimates. In *International Conference on Machine Learning*, pages 1435–1444.

Jordan, M. I., Ghahramani, Z., Jaakkola, T. S., and Saul, L. K. (1999). An introduction to variational methods for graphical models. *Machine learning*, 37(2):183–233.

Kapoor, A., Viswanathan, R., and Jain, P. (2012). Multilabel classification using bayesian compressed sensing. In *Advances in Neural Information Processing Systems*, pages 2645–2653.

Katakis, I., Tsoumakas, G., and Vlahavas, I. (2008). Multilabel text classification for automated tag suggestion. In *Proceedings of the ECML/PKDD*, volume 18.

Keerthi, S. and Chu, W. (2006). A matching pursuit approach to sparse gaussian process regression. In *Advances in neural information processing systems*, pages 643–650.

Keysers, D., Deselaers, T., Gollan, C., and Ney, H. (2007). Deformation models for image recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(8):1422–1435.

Kim, H. and Ghahramani, Z. (2003). The em-ep algorithm for Gaussian process classification.

Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Kolmogoroff, A. (1941). Interpolation und extrapolation von stationaren zufalligen folgen. *Izvestiya Rossiiskoi Akademii Nauk. Seriya Matematicheskaya*, 5(1):3–14.

Kullback, S. and Leibler, R. A. (1951). On information and sufficiency. *The annals of mathematical statistics*, 22(1):79–86.

Kupperman, M. et al. (1958). Probabilities of hypotheses and information-statistics in sampling from exponential-class populations. *The Annals of Mathematical Statistics*, 29(2):571–575.

Lawrence, N. D. (2004). Gaussian process latent variable models for visualisation of high dimensional data. In *Advances in neural information processing systems*, pages 329–336.

Lawrence, N. D., Seeger, M., and Herbrich, R. (2002). Fast sparse Gaussian process methods: the informative vector machine. In *Neural Information Processing Systems, 13*. MIT Press.

LeCun, Y. (2019). The mnist database. `http://yann.lecun.com/exdb/mnist/`.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324.

Lewis, D. D., Yang, Y., Rose, T. G., and Li, F. (2004). Rcv1: A new benchmark collection for text categorization research. *Journal of machine learning research*, 5(Apr):361–397.

Lin, Z., Ding, G., Hu, M., and Wang, J. (2014). Multi-label classification via feature-aware implicit label space encoding. In *International Conference on Machine Learning*, pages 325–333.

Liu, H., Ong, Y.-S., Shen, X., and Cai, J. (2018). When Gaussian process meets big data: A review of scalable gps. *arXiv preprint arXiv:1807.01065*.

Liu, J., Chang, W.-C., Wu, Y., and Yang, Y. (2017). Deep learning for extreme multi-label text classification. In *Proceedings of the 40th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 115–124. ACM.

Liu, Q. and Pierce, D. A. (1994). A note on gauss—hermite quadrature. *Biometrika*, 81(3):624–629.

MacKay, D. J. (2003). *Information theory, inference and learning algorithms*. Cambridge university press.

Maclaurin, D., Duvenaud, D., Johnson, M., and Townsend, J. (2015). Autograd. `https://github.com/HIPS/autograd`.

Marimont, R. and Shapiro, M. (1979). Nearest neighbour searches and the curse of dimensionality. *IMA Journal of Applied Mathematics*, 24(1):59–70.

Matthews, A. G. d. G. (2017). *Scalable Gaussian process inference using variational methods*. PhD thesis, University of Cambridge.

Matthews, A. G. d. G., Hensman, J., Turner, R., and Ghahramani, Z. (2016). On sparse variational methods and the kullback-leibler divergence between stochastic processes. *Journal of Machine Learning Research*, 51:231–239.

Matthews, D. G., Alexander, G., Van Der Wilk, M., Nickson, T., Fujii, K., Boukouvalas, A., León-Villagrá, P., Ghahramani, Z., and Hensman, J. (2017). Gpflow: A Gaussian process library using tensorflow. *The Journal of Machine Learning Research*, 18(1):1299–1304.

McAuley, J., Targett, C., Shi, Q., and Van Den Hengel, A. (2015). Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 43–52. ACM.

Melkumyan, A. and Ramos, F. T. (2009). A sparse covariance function for exact gaussian process inference in large datasets. In *Twenty-First International Joint Conference on Artificial Intelligence*.

Mencia, E. L. and Fürnkranz, J. (2008). Efficient pairwise multilabel classification for large-scale problems in the legal domain. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 50–65. Springer.

Mercer, J. (1909). Xvi. functions of positive and negative type, and their connection the theory of integral equations. *Philosophical transactions of the royal society of London. Series A, containing papers of a mathematical or physical character*, 209(441-458):415–446.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., and Dean, J. (2013). Distributed representations of words and phrases and their compositionality. In Burges, C. J. C., Bottou, L., Welling, M., Ghahramani, Z., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc.

Milios, D., Camoriano, R., Michiardi, P., Rosasco, L., and Filippone, M. (2018). Dirichlet-based Gaussian processes for large-scale calibrated classification. In *Advances in Neural Information Processing Systems*, pages 6005–6015.

Mineiro, P. and Karampatziakis, N. (2015). Fast label embeddings for extremely large output spaces. *arXiv preprint arXiv:1503.08873*.

Minka, T. P. (2001). *A family of algorithms for approximate Bayesian inference*. PhD thesis, Massachusetts Institute of Technology.

Morariu, V. I., Srinivasan, B. V., Raykar, V. C., Duraiswami, R., and Davis, L. S. (2009). Automatic online tuning for fast Gaussian summation. In *Advances in neural information processing systems*, pages 1113–1120.

Moreno-Muñoz, P., Artés, A., and Álvarez, M. (2018). Heterogeneous multi-output Gaussian process prediction. In *Advances in Neural Information Processing Systems*, pages 6711–6720.

Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*.

Neal, R. M. (1995). *BAYESIAN LEARNING FOR NEURAL NETWORKS*. PhD thesis, Citeseer.

Nguyen, T. and Bonilla, E. (2013). Efficient variational inference for Gaussian process regression networks. In *Artificial Intelligence and Statistics*, pages 472–480.

Nguyen, T. V., Bonilla, E. V., et al. (2014). Collaborative multi-output Gaussian processes. In *UAI*, pages 643–652.

Nickisch, H. and Rasmussen, C. E. (2008). Approximations for binary Gaussian process classification. *Journal of Machine Learning Research*, 9(Oct):2035–2078.

Niculescu-Mizil, A. and Abbasnejad, E. (2017). Label filters for large scale multilabel classification. In *Artificial Intelligence and Statistics*, pages 1448–1457.

Opper, M. and Archambeau, C. (2009). The variational Gaussian approximation revisited. *Neural computation*, 21(3):786–792.

Papoulis, A. and Pillai, S. U. (2002). *Probability, random variables, and stochastic processes*. Tata McGraw-Hill Education.

Partalas, I., Kosmopoulos, A., Baskiotis, N., Artieres, T., Paliouras, G., Gaussier, E., Androutsopoulos, I., Amini, M.-R., and Galinari, P. (2015). Lshtc: A benchmark for large-scale text classification. *arXiv preprint arXiv:1503.08581*.

Prabhu, Y., Kag, A., Gopinath, S., Dahiya, K., Harsola, S., Agrawal, R., and Varma, M. (2018a). Extreme multi-label learning with label features for warm-start tagging, ranking & recommendation. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 441–449. ACM.

Prabhu, Y., Kag, A., Harsola, S., Agrawal, R., and Varma, M. (2018b). Parabel: Partitioned label trees for extreme classification with application to dynamic search advertising. In *Proceedings of the 2018 World Wide Web Conference*, pages 993–1002. International World Wide Web Conferences Steering Committee.

Prabhu, Y. and Varma, M. (2014a). Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '14, pages 263–272, New York, NY, USA. ACM.

Prabhu, Y. and Varma, M. (2014b). Fastxml: A fast, accurate and stable tree-classifier for extreme multi-label learning. In *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 263–272. ACM.

QuiÃ±onero-Candela, J., Rasmussen, C. E., Figueiras-Vidal, A. R., et al. (2010). Sparse spectrum Gaussian process regression. *Journal of Machine Learning Research*, 11(Jun):1865–1881.

Quiñonero-Candela, J. and Rasmussen, C. E. (2005a). A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6:1939–1959.

Quiñonero-Candela, J. and Rasmussen, C. E. (2005b). A unifying view of sparse approximate Gaussian process regression. *Journal of Machine Learning Research*, 6(Dec):1939–1959.

Rasmussen, C. E. and Williams, C. K. (2006). Gaussian processes for machine learning. *Gaussian Processes for Machine Learning, by CE Rasmussen and CKI Williams. ISBN-13 978-0-262-18253-9.*

Read, J., Pfahringer, B., Holmes, G., and Frank, E. (2009). Classifier chains for multi-label classification. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 254–269. Springer.

Repository, X. (2010). Xml repository. `http://manikvarma.org/downloads/XC/XMLRepository.html`.

Robbins, H. and Monro, S. (1951). A stochastic approximation method. *The annals of mathematical statistics*, pages 400–407.

Romanuke, V. V. (2016). Training data expansion and boosting of convolutional neural networks for reducing the mnist dataset error rate.

Ruiz, F. J., Titsias, M. K., Dieng, A. B., and Blei, D. M. (2018). Augment and reduce: Stochastic inference for large categorical distributions. *arXiv preprint arXiv:1802.04220*.

Salimbeni, H. and Deisenroth, M. (2017). Doubly stochastic variational inference for deep Gaussian processes. In *Advances in Neural Information Processing Systems*, pages 4588–4599.

Salimbeni, H., Eleftheriadis, S., and Hensman, J. (2018). Natural gradients in practice: Non-conjugate variational inference in gaussian process models. *arXiv preprint arXiv:1803.09151*.

Saul, L. K. and Jordan, M. I. (1996). Exploiting tractable substructures in intractable networks. In *Advances in neural information processing systems*, pages 486–492.

Schervish, M. J. (2012). *Theory of statistics*. Springer Science & Business Media.

Seeger, M., Williams, C. K. I., and Lawrence, N. D. (2003). Fast forward selection to speed up sparse Gaussian process regression. In *Ninth International Workshop on Artificial Intelligence*. MIT Press.

Shen, Y., Seeger, M., and Ng, A. Y. (2006). Fast gaussian process regression using kd-trees. In *Advances in neural information processing systems*, pages 1225–1232.

Sheth, R., Wang, Y., and Khardon, R. (2015). Sparse variational inference for generalized gp models. In *International Conference on Machine Learning*, pages 1302–1311.

Si, S., Zhang, H., Keerthi, S. S., Mahajan, D., Dhillon, I. S., and Hsieh, C.-J. (2017). Gradient boosted decision trees for high dimensional sparse output. In *Proceedings of the 34th International Conference on Machine Learning-Volume 70*, pages 3182–3190. JMLR. org.

Siblini, W., Kuntz, P., and Meyer, F. (2018). Craftml, an efficient clustering-based random forest for extreme multi-label learning.

Skolidis, G. and Sanguinetti, G. (2011). Bayesian multitask classification with Gaussian process priors. *IEEE Transactions on Neural Networks*, 22(12).

Smola, A. J. and Bartlett, P. L. (2001). Sparse greedy Gaussian process regression. In *Advances in neural information processing systems*, pages 619–625.

Smola, A. J. and Schölkopf, B. (2000). Sparse greedy matrix approximation for machine learning.

Snelson, E. and Ghahramani, Z. (2006). Sparse Gaussian processes using pseudo-inputs. In Weiss, Y., Schölkopf, B., and Platt, J. C., editors, *Advances in Neural Information Processing Systems 18*, pages 1257–1264.

Snoek, C. G., Worring, M., Van Gemert, J. C., Geusebroek, J.-M., and Smeulders, A. W. (2006). The challenge problem for automated detection of 101 semantic concepts in multimedia. In *Proceedings of the 14th ACM international conference on Multimedia*, pages 421–430. ACM.

Stein, M. L. (2012). *Interpolation of spatial data: some theory for kriging*. Springer Science & Business Media.

Stoyan, D. (1996). Hans wackernagel: Multivariate geostatistics. an introduction with applications. with 75 figures and 5 tables. springer-verlag, berlin, heidelberg, new york, 235 pp., 1995, dm 68.-isbn 3-540-60127-9. *Biometrical Journal*, 38(4):454–454.

Tagami, Y. (2017). Annexml: Approximate nearest neighbor search for extreme multi-label classification. In *Proceedings of the 23rd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 455–464. ACM.

Tan, L. S., Ong, V. M., Nott, D. J., and Jasra, A. (2016). Variational inference for sparse spectrum Gaussian process regression. *Statistics and Computing*, 26(6):1243–1261.

Teh, Y. W., Kurihara, K., and Welling, M. (2008). Collapsed variational inference for HDP. In *Advances in neural information processing systems*, pages 1481–1488.

Teh, Y. W., Newman, D., and Welling, M. (2007). A collapsed variational bayesian inference algorithm for latent Dirichlet allocation. In *Advances in neural information processing systems*, pages 1353–1360.

Teh, Y. W., Seeger, M., and Michael, J. (2005). Semiparametric latent factor models. In *Workshop on Artificial Intelligence and Statistics 10*.

Titsias, M. (2016). One-vs-each approximation to softmax for scalable estimation of probabilities. In *Advances in Neural Information Processing Systems*, pages 4161–4169.

Titsias, M. and Lawrence, N. D. (2010). Bayesian Gaussian process latent variable model. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 844–851.

Titsias, M. and Lázaro-Gredilla, M. (2014). Doubly stochastic variational bayes for non-conjugate inference. In *International conference on machine learning*, pages 1971–1979.

Titsias, M. K. (2009). Variational learning of inducing variables in sparse Gaussian processes. In *International Conference on Artificial Intelligence and Statistics*, pages 567–574.

Titsias, M. K. and Lázaro-Gredilla, M. (2011). Spike and slab variational inference for multi-task and multiple kernel learning. In *Advances in neural information processing systems*, pages 2339–2347.

Tsoumakas, G. and Katakis, I. (2007). Multi label classification: An overview. *International Journal of Data Warehouse and Mining*, 3(3):1–13.

Tsoumakas, G., Katakis, I., and Vlahavas, I. (2008). Effective and efficient multilabel classification in domains with large number of labels. In *Proc. ECML/PKDD 2008 Workshop on Mining Multidimensional Data (MMD'08)*, volume 21, pages 53–59. sn.

Tsoumakas, G. and Vlahavas, I. (2007). Random k-labelsets: An ensemble method for multilabel classification. In *European conference on machine learning*, pages 406–417. Springer.

Tsoumakas, G. and Zhang, M.-L. (2009). Learning from multi-label data.

Uhlenbeck, G. E. and Ornstein, L. S. (1930). On the theory of the brownian motion. *Physical review*, 36(5):823.

Van der Wilk, M., Rasmussen, C. E., and Hensman, J. (2017). Convolutional gaussian processes. In *Advances in Neural Information Processing Systems*, pages 2849–2858.

Wainwright, M. J., Jordan, M. I., et al. (2008). Graphical models, exponential families, and variational inference. *Foundations and Trends® in Machine Learning*, 1(1–2):1–305.

Wenzel, F., Galy-Fajou, T., Donner, C., Kloft, M., and Opper, M. (2019). Efficient Gaussian process classification using pòlya-gamma data augmentation. In

Proceedings of the AAAI Conference on Artificial Intelligence, volume 33, pages 5417–5424.

Weston, J., Bengio, S., and Usunier, N. (2011). Wsabie: Scaling up to large vocabulary image annotation. In *Proceedings of the International Joint Conference on Artificial Intelligence, IJCAI*.

Weston, J., Makadia, A., and Yee, H. (2013). Label partitioning for sublinear ranking.

Wetzker, R., Zimmermann, C., and Bauckhage, C. (2008). Analyzing social bookmarking systems: A del. icio. us cookbook. In *Proceedings of the ECAI 2008 Mining Social Data Workshop*, pages 26–30.

Wiener, N. (1949). Extrapolation, interpolation and smoothing of stationary. *Time Series, with Engineering Appli cations*.

Wilder, K. J. (1998). *Decision tree algorithms for handwritten digit recognition*. PhD thesis, UMass Amherst.

Williams, C. K. and Barber, D. (1998). Bayesian classification with Gaussian processes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20(12):1342–1351.

Williams, C. K. and Rasmussen, C. E. (1996). Gaussian processes for regression. In *Advances in neural information processing systems*, pages 514–520.

Wilson, A. and Nickisch, H. (2015). Kernel interpolation for scalable structured Gaussian processes (kiss-gp). In *International Conference on Machine Learning*, pages 1775–1784.

Wilson, A. G., Knowles, D. A., and Ghahramani, Z. (2011). Gaussian process regression networks. *arXiv preprint arXiv:1110.4411*.

Xu, C., Tao, D., and Xu, C. (2016). Robust extreme multi-label learning. In *Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining*, pages 1275–1284. ACM.

Yen, I. E., Huang, X., Dai, W., Ravikumar, P., Dhillon, I., and Xing, E. (2017). Ppdsparse: A parallel primal-dual sparse method for extreme classification. In *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 545–553. ACM.

Yen, I. E.-H., Huang, X., Ravikumar, P., Zhong, K., and Dhillon, I. (2016). Pd-sparse: A primal and dual sparse approach to extreme multiclass and multilabel classification. In *International Conference on Machine Learning*, pages 3069–3077.

Yen, I. E.-H., Kale, S., Yu, F., Holtmann-Rice, D., Kumar, S., and Ravikumar, P. (2018). Loss decomposition for fast learning in large output spaces. In *International Conference on Machine Learning*, pages 5626–5635.

Yu, H.-F., Jain, P., Kar, P., and Dhillon, I. (2014). Large-scale multi-label learning with missing labels. In *International Conference on Machine Learning*, pages 593–601.

Zaragoza, J. C., Sucar, E., Morales, E., Bielza, C., and Larranaga, P. (2011). Bayesian chain classifiers for multidimensional classification. In *Twenty-second international joint conference on artificial intelligence*.

Zeiler, M. D. (2012). Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*.

Zhang, M. and Zhou, Z. (2013). A review on multi-label learning algorithms. *Knowledge and Data Engineering, IEEE Transactions on*, PP(99):1.

Zhang, W., Yan, J., Wang, X., and Zha, H. (2018). Deep extreme multi-label learning. In *Proceedings of the 2018 ACM on International Conference on Multimedia Retrieval*, pages 100–107. ACM.

Zubiaga, A. (2012). Enhancing navigation on wikipedia with social tags. *arXiv preprint arXiv:1202.5469*.

Appendix A

Useful formulas for the multivariate Gaussian distribution

We present here some of the Gaussian identities that we widely use throughout this thesis.

A.1 Marginal and conditional Gaussian distribution

Let

$$\begin{bmatrix}\mathbf{f}\\ \mathbf{y}\end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix}\boldsymbol{\mu}_\mathbf{f}\\ \boldsymbol{\mu}_\mathbf{y}\end{bmatrix}, \begin{bmatrix}\Sigma_\mathbf{f} & \Sigma_\mathbf{fy}\\ \Sigma_\mathbf{yf} & \Sigma_\mathbf{y}\end{bmatrix}\right) \tag{A.1}$$

where $\mathbf{f} \in \mathbb{R}^{D_\mathbf{f}}$ and $\mathbf{y} \in \mathbb{R}^{D_\mathbf{y}}$ and cross-covariance matrix $\Sigma_\mathbf{fy} = \Sigma_\mathbf{yf}^\top \in \mathbb{R}^{D_\mathbf{f} \times D_\mathbf{y}}$. The marginals of this Gaussian distribution are given by,

$$p(\mathbf{f}) = \int p(\mathbf{f}, \mathbf{y}) d\mathbf{y} = \mathcal{N}(\boldsymbol{\mu}_\mathbf{f}, \Sigma_\mathbf{f}), \tag{A.2}$$

$$p(\mathbf{y}) = \int p(\mathbf{f}, \mathbf{y}) d\mathbf{f} = \mathcal{N}(\boldsymbol{\mu}_\mathbf{y}, \Sigma_\mathbf{y}), \tag{A.3}$$

while the conditionals are

$$p(\mathbf{f}|\mathbf{y}) = \mathcal{N}(\boldsymbol{\mu}_\mathbf{f} + \Sigma_\mathbf{fy}\Sigma_\mathbf{y}^{-1}(\mathbf{y} - \boldsymbol{\mu}_\mathbf{y}), \Sigma_\mathbf{f} - \Sigma_\mathbf{fy}\Sigma_\mathbf{y}^{-1}\Sigma_\mathbf{yf}), \tag{A.4}$$

$$p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(\boldsymbol{\mu}_\mathbf{y} + \Sigma_\mathbf{yf}\Sigma_\mathbf{f}^{-1}(\mathbf{f} - \boldsymbol{\mu}_\mathbf{f}), \Sigma_\mathbf{y} - \Sigma_\mathbf{yf}\Sigma_\mathbf{f}^{-1}\Sigma_\mathbf{fy}). \tag{A.5}$$

Moreover, assuming now that $\mathbf{f}$ represents some latent variables while $\mathbf{y}$ are the observed ones. Given an arbitrary $D_{\mathbf{y}} \times D_{\mathbf{f}}$ matrix A, a $D_{\mathbf{y}}$-dimensional real vector $\mathbf{b}$ and the model

$$p(\mathbf{f}) = \mathcal{N}(\mu, \Sigma), \tag{A.6}$$
$$p(\mathbf{y}|\mathbf{f}) = \mathcal{N}(A\mathbf{f} + \mathbf{b}, \Lambda), \tag{A.7}$$

we can re-write Eq. (A.1) as

$$\begin{bmatrix} \mathbf{f} \\ \mathbf{y} \end{bmatrix} \sim \mathcal{N}\left(\begin{bmatrix} \boldsymbol{\mu} \\ A\boldsymbol{\mu} + \mathbf{b} \end{bmatrix}, \begin{bmatrix} \Sigma & \Sigma A^\top \\ A\Sigma & \Lambda + A\Sigma A^\top \end{bmatrix} \right), \tag{A.8}$$

which can be used to find the posterior $p(\mathbf{f}|\mathbf{y})$ and the marginal likelihood $p(\mathbf{y})$ using equations (A.4) and (A.3), respectively.

A.2 KL-Divergence between Normal distributions

The Kullback-Leibler divergence between two D-dimensional Gaussian distributions $\mathcal{N}(\mu_1, \Sigma_1)$ and $\mathcal{N}(\mu_2, \Sigma_2)$ is given by

$$\begin{aligned} \mathrm{KL}[\mathcal{N}(\mu_1, \Sigma_1) \mid\mid \mathcal{N}(\mu_2, \Sigma_2)] = & \frac{1}{2}\big(\ (\mu_2 - \mu_1)^\top \Sigma_2^{-1} (\mu_2 - \mu_1) \\ & + \mathrm{tr}(\Sigma_2^{-1}\Sigma_1) + \log|\Sigma_2| - \log|\Sigma_1| - D \ \big) \end{aligned} \tag{A.9}$$

Appendix B

Matrix-related identities

B.1 Matrix calculus identities

We present here useful identities for matrix differentiation used in Chapter 4 to prove Proposition 1. Let two symmetric and positive definite $N \times N$ matrices Σ and A. The following equalities hold,

$$\frac{\partial \log |\Sigma|}{\partial \Sigma} = \Sigma^{-1} \tag{B.1}$$

$$\frac{\partial \operatorname{tr}(A\Sigma)}{\partial \Sigma} = A \tag{B.2}$$

B.2 The Woodbury matrix identity

For two invertible matrices, $A \in \mathbb{R}^{N\times N}$ and $C \in \mathbb{R}^{M\times M}$, and two arbitrary matrices $U \in \mathbb{R}^{N\times M}$ and $V \in \mathbb{R}^{M\times N}$, the Woodbury matrix identity is given by

$$(A + UCV)^{-1} = A^{-1} - A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \tag{B.3}$$

Appendix C

Code Snippets

We present a few code snippets for the implementation of the ELBO for both multi-class and multi-label classification cases using different parametrizations. For completeness, we also provide a code snippet based on the Autograd package which was used at the beginning of this PhD before moving to Tensorflow choice. This will make more clear the efficiency of the Tensorflow-based code over the Sutograd-based one. For all cases we consider here an ARD kernel is employed.

C.1 Autograd-based Python Code Snippet

The following Autograd-based code implements the bound in Eq. (4.32) for the multi-label classification case, with $\mathcal{N}_i = \mathcal{L}_i$, while the parsimonious parametrization is used. Notice that Autograd requires us to define a function that takes as inputs the variables (”var_par” in the code) that we want to find the partial derivatives of this function with respect to them. There is also a second input called ”args” in the code that provides other values that our function will not be differentiated over them and they are needed for computing the funcion.

```
'''
K: number of labels
M: number of inducing inputs
P: number of GPs used
D: input dimensionality
N: number of training data points
S_GH: number of G-H quadrature points
x_new: a vector of the precomputed  positions of the G-H
    qudrature (usually 10-dimensional)
w_new: a vector of the corresponding weights of the G-H
   qudrature
Xtr: the N x D design matrix
'''
```

```

import autograd.numpy as np

X_dot_X_diag = np.array((Xtr.multiply(Xtr)).sum(axis=1))
   .reshape(Xtr.shape[0])


def objective_fun(var_par, *args):

    idx_list_obj = args[0] # the batch of chosen indices
   from training dataset X_tr with size "minibatch"
    mask_labels = args[1] # a K x minibatch matrix with
   {-1,1} values; "-1" indicates positive label while
   "1" begatve one
    lik_factor = args[2]

    ell_sq_obj = np.exp(2.*var_par[-1,0]) # lengthscales

    Phi_obj = var_par[:K, :P] # K x P
    mu_p_mtr_obj = var_par[K:K+M, :P] # M x P
    Sigma_p_mtr_obj = var_par[K+M:K+2*M, :P] # M x P
    bias_obj = var_par[:K, -1].reshape(K,1) # K x 1
    Z_opt_obj = var_par[K+2*M:K+3*M, :D] # M x D

    Z_dot_Z_np = np.dot(Z_opt_obj, Z_opt_obj.T) # M x M
    diag_ZZ_np = np.diag(Z_dot_Z_obj) # (M, )
    A_p = diag_ZZ_np[:, None] + diag_ZZ_tf[None, :] - 2.*
   Z_dot_Z_np
    K_mm = np.exp(-( 0.5/(ell_sq_obj) ) * A_p)

    X_dot_Z_np = np.dot(Xtr[idx_list_obj].toarray(),
   Z_opt_obj.T) # minibatch x M
    A_k2_tmp = X_dot_X_diag[idx_list_obj][:, None] - 2.*
   X_dot_Z_np + diag_ZZ_tf
    K_nm = np.exp(-( 0.5/(ell_sq_obj) ) * A_k2) #
   minibatch x M

    # KL computation
    all_s_p_list = []
    all_m_p_mtr_arr = np.dot(K_nm, mu_p_mtr_obj) #
   minibatch x P
    KL_div = 0.0
    for p_prime in range(P):
```

```python
        mu_p_prime = mu_p_mtr_obj[:, p_prime]
        Sigma_p_prime = Sigma_p_mtr_obj[:, p_prime]

        K_mmlus_Sigma_p_prime = K_mm + np.diag(Sigma_p_prime.T)
        L_K_mmlus_Sigma_p_prime = np_lin.cholesky(K_mmlus_Sigma_p_prime) # M x M

        invL_K_mmlus_Sigma_p_prime = sc_lin.solve_triangular(L_K_mmlus_Sigma_p_prime, eye_M, lower=True)

        invK_mmlus_Sigma_p_prime = np.dot(invL_K_mmlus_Sigma_p_prime.T, invL_K_mmlus_Sigma_p_prime) # M x M

        KL_div = KL_div + 0.5 * ( np.dot( np.transpose(mu_p_prime) , np.dot(K_mm, mu_p_prime)) - np.sum(invK_mmlus_Sigma_p_prime*K_mm)
        + 2*np.sum(np.log(L_K_mmlus_Sigma_p_prime[row_col, row_col])) - np.sum(np.log(Sigma_p_prime)) )

        y_p = sc_lin.solve_triangular(L_K_mmlus_Sigma_p_prime, K_nm.T, lower=True) # M x minibatch
        all_s_p_list.append((sf2_obj) - np.sum(y_p**2, axis=0)) # minibatch x 1

    all_s_p_arr = np.array(all_s_p_list)

    # Marginal mean and variance computation
    dot_phi = (np.dot(Phi_obj, all_m_p_mtr_arr.T) + bias_obj)*mask_labels # K x minibatch
    dot_phi_sqrt = np.sqrt(np.dot(Phi_obj**2, all_s_p_arr))*mask_labels # K x minibatch

    # Expectations computation
    E_q_all = 0.0
    for x_new_i in range(S_GH):
        E_q_all =  E_q_all + w_new[x_new_i]*np.sum(np.log1p(np.exp(dot_phi - x_new[x_new_i]*sum_dot_phi_sqrt) ))

    return lik_factor*E_q_all + KL_div
```

C.2 Tensorflow-based Python Code Snippet

The first code snippet demonstrates how the ELBO in Eq. (3.36) is implemented using the Tensorflow package for the multi-class classification case. Here the parsimonious parametrization is used.

```
'''
    Multu-label Classification - One-vs-Each

****************************************
***   Parsimonious Parametrization   ***
****************************************

K: number of classes
M: number of inducing inputs
'''

# Define Placeholders
X_batch = tf.placeholder(float_type, shape=[None, D]) # A batch of the training input data
ind_k_l = tf.placeholder(tf.int32, [2, ]) # the two chosen classes
lik_factor = tf.placeholder(float_type, ()) # the likelihood constant tha makes the stochastic estimation unbiased

# Define constants
x_new_tf_2d = tf.constant(x_new[:, np.newaxis], dtype=float_type) # x_new: the precomputed  positions of the G-H qudrature
w_new_tf = tf.constant(w_new, dtype=float_type) # w_new: the weights of the G-H qudrature
tf_M_ones = tf.ones([M, 1], dtype=float_type)
eye_M_tf = tf.constant(np.eye(M), dtype=float_type)

# Define variables
q_mu_tf = tf.Variable(0.5*np.ones((K, M)), dtype=float_type) # the stored variational means
Sigma_p_tf = tf.Variable(np.log(np.exp(.5) - 1)*np.ones((K, M)), dtype=float_type) # the stored diagonals
```

```
    for the variational covariances
Z_all = tf.Variable(Z_all_0, dtype=float_type) # a K x M
     x D tensor of the stored inducing inputs for each
    class
ells_tf = tf.Variable(np.log(np.exp(.5) - 1)*np.ones(D),
     dtype=float_type) # D lengthscales
sigma_f = tf.Variable(np.log(np.exp(.5) - 1), dtype=
    float_type) # kernel variance

# Choose the right variables
Z_k = Z_all[ind_k_l[0]]
Z_l = Z_all[ind_k_l[1]]
q_mu_k = q_mu_tf[ind_k_l[0]]
q_mu_l = q_mu_tf[ind_k_l[1]]
Sigma_k = Sigma_p_tf[ind_k_l[0]]
Sigma_l = Sigma_p_tf[ind_k_l[1]]

# Positive variables
Sigma_k = tf.nn.softplus(Sigma_k) + 1e-6
Sigma_l = tf.nn.softplus(Sigma_l) + 1e-6
ells_tf = tf.nn.softplus(ells_tf) + 1e-6
sigma_f = tf.nn.softplus(sigma_f) + 1e-6

Z_ells_T_k = tf.transpose(Z_k*ells_tf) # D x M
Z_ells_T_l = tf.transpose(Z_l*ells_tf) # D x M

Xb_Lambda = X_batch*ells_tf

Z_dot_Z_obj_tf_k = tf.matmul(Z_ells_T_k, Z_ells_T_k,
    transpose_a=True) # M x M
Z_mul_sum_obj_tf_k = tf.matrix_diag_part(
    Z_dot_Z_obj_tf_k) # (M, )
Z_dot_Z_obj_tf_l = tf.matmul(Z_ells_T_l, Z_ells_T_l,
    transpose_a=True) # M x M
Z_mul_sum_obj_tf_l = tf.matrix_diag_part(
    Z_dot_Z_obj_tf_l) # (M, )

A_p_tf_k = Z_mul_sum_obj_tf_k[None, :] +
    Z_mul_sum_obj_tf_k[:, None] - 2.*Z_dot_Z_obj_tf_k # M
     x M
A_p_tf_l = Z_mul_sum_obj_tf_l[None, :] +
    Z_mul_sum_obj_tf_l[:, None] - 2.*Z_dot_Z_obj_tf_l # M
     x M
```

```

X_dot_Z_obj_tf_k = tf.matmul(Xb_Lambda, Z_ells_T_k) #
   minibatch x M
X_dot_Z_obj_tf_l = tf.matmul(Xb_Lambda, Z_ells_T_l) #
   minibatch x M

tmp_xxT = tf.reduce_sum(tf.square(Xb_Lambda), 1)

A_k2_k = tmp_xxT - 2.*Z_dot_X_obj_tf_k +
   Z_mul_sum_obj_tf_k[:, None] # M x minibatch
A_k2_l = tmp_xxT - 2.*Z_dot_X_obj_tf_l +
   Z_mul_sum_obj_tf_l[:, None] # M x minibatch

K_mm_k = sigma_f*tf.exp(-A_p_tf_k) # M x M
K_mm_l = sigma_f*tf.exp(-A_p_tf_l) # M x M

K_mn_k = sigma_f*tf.exp(-A_k2_k) # M x minibatch
K_mn_l = sigma_f*tf.exp(-A_k2_l) # M x minibatch

# KL computation
L_Kmm_Sigma_k = tf.cholesky(K_mm_k + tf.matrix_diag(
   Sigma_k)) # M x M
invL_Kmm_k_Sigma = tf.matrix_triangular_solve(
   L_Kmm_Sigma_k, eye_M_tf, lower=True) # M x M
invK_Z_plus_Sigma_tf = tf.matmul(invL_Kmm_k_Sigma,
   invL_Kmm_k_Sigma, transpose_a=True) # M x M
Lq_diag_k = tf.matrix_diag_part(L_Kmm_Sigma_k)

KL_div_tf = 0.5*(tf.reduce_sum(q_mu_k*K_mm_k*q_mu_k[:,
   None])
                     - tf.reduce_sum(
   invK_Z_plus_Sigma_tf*K_mm_k)
                     - tf.reduce_sum(tf.log(Sigma_k))) +
    tf.reduce_sum(tf.log(Lq_diag_k))

# Marginal mean and variance computation
L_Kmm_Sigma_l = tf.cholesky(K_mm_l + tf.matrix_diag(
   Sigma_l)) # M x M

y_k = tf.matrix_triangular_solve(L_Kmm_Sigma_k, K_mn_k,
   lower=True) # M x minibatch
y_l = tf.matrix_triangular_solve(L_Kmm_Sigma_l, K_mn_l,
   lower=True) # M x minibatch
```

```

fvar_k = sigma_f - tf.reduce_sum(tf.square(y_k), axis=0)
    # minibatch
fvar_l = sigma_f - tf.reduce_sum(tf.square(y_l), axis=0)
    # minibatch

fmean = tf.matmul(K_mn_l, q_mu_l[:, None], transpose_a=
   True) - tf.matmul(K_mn_k, q_mu_k[:, None],
   transpose_a=True) # minibatch
fvar = fvar_k + fvar_l # minibatch

# Expectations computation
sum_dot_phi_tf = tf.squeeze(fmean) # minibatch
sum_dot_phi_sqrt_tf = tf.sqrt(fvar) # minibatch
sum_E_q_all_tf = tf.reduce_sum(w_new_tf * tf.reduce_sum(
   tf.nn.softplus(sum_dot_phi_tf - x_new_tf_2d*
   sum_dot_phi_sqrt_tf[None, :]), 1))

#Lower Bound
lower_bound = lik_factor*sum_E_q_all_tf + KL_div_tf
```

Moreover, the next two code snippets implement the ELBO in Eq. (4.32) for the multi-label classification case, with $\mathcal{N}_i = \mathcal{L}_i$. The parsimonious parametrization is used for the first snippet while the second one implements the full parametrization.

```
'''
    Multu-label Classification

****************************************
***   Parsimonious Parametrization   ***
****************************************

K: number of labels
M: number of inducing inputs
P: number of GPs used
x_new: a vector of the precomputed  positions of the G-H
    qudrature (usually 10-dimensional)
w_new: a vector of the corresponding weights of the G-H
   qudrature
'''

import numpy as np
import tensorflow as tf

```

```python
# Define Placeholders
X_batch = tf.sparse_placeholder(float_type, shape=[None, D]) # Sparse input tensor for a batch of training data
mask_labels = tf.placeholder(float_type, [K, None]) # a K x minibatch tensor with {-1,1} values; "-1" indicates positive label while "1" begatve one
lik_factor = tf.placeholder(float_type, ()) # the likelihood constant tha makes the stochastic estimation unbiased

# Define constants
x_new_tf_3d = tf.constant(x_new[:, np.newaxis, np.newaxis], dtype=float_type)
w_new_tf = tf.constant(w_new, dtype=float_type)
tf_P_ones_3d = tf.ones([P, 1 ,1],  dtype=float_type)

# Define variables
Phi = tf.Variable(np.random.randn(K, P)/np.sqrt(P), dtype=float_type) # the factor loadings matrix
q_mu_tf = tf.Variable(0.5*np.random.randn((M, P)), dtype=float_type) # variational mean vectors
Sigma_p = tf.Variable(np.log(np.exp(np.random.rand(K, M)) - 1), dtype=float_type) # variational covariance matrices
Z_tf = tf.Variable(A_0, dtype=float_type) # inducing inputs
bias_tf = tf.Variable(np.random.randn(K, 1)/np.sqrt(P), dtype=float_type) # bias term
ells_tf = tf.Variable(np.log(np.exp(1.0) - 1)*np.ones(D), dtype=float_type) # lengthscales

# Tranform Variables
Sigma_p_tf = tf.nn.softplus(Sigma_p) + 1e-6
ells_tf = tf.nn.softplus(ells_tf) + 1e-6
Z_tf = tf.nn.l2_normalize(Z_tf, 1)

Z_tf_ells_T = tf.transpose(Z_tf*ells_tf) # D x M
Xb_Lambda = X_batch.__mul__(ells_tf)

Z_dot_Z_tf = tf.matmul(Z_tf_ells_T, Z_tf_ells_T, transpose_a=True) # M x M
diag_ZZ_tf = tf.matrix_diag_part(Z_dot_Z_tf) # (M, )
```

```python
A_p_tf = diag_ZZ_tf[:, None] + diag_ZZ_tf[None, :] - 2.*Z_dot_Z_tf

X_dot_Z_obj_tf = tf.sparse_tensor_dense_matmul(Xb_Lambda, Z_tf_ells_T) # minibatch x M
A_k2_tmp = tf.sparse_reduce_sum(tf.square(Xb_Lambda), 1) - 2.*tf.transpose(X_dot_Z_obj_tf) # M x minibatch
A_k2 = tf.transpose(A_k2_tmp) + diag_ZZ_tf # minibatch x M

K_mm = tf.exp(-A_p_tf) # M x M

K_mn = tf.exp(-A_k2) # minibatch x M
K_mn = tf.transpose(K_mn) # M x minibatch

# KL computation
K_mm_Sigma = K_mm[None, :, :] + tf.matrix_diag(Sigma_p_tf)
L_Kmm_Sigma = tf.cholesky(K_mm_Sigma) # P x M x M
invL_Kmm_Sigma = tf.matrix_triangular_solve(L_Kmm_Sigma, eye_M_tf, lower=True) # P x M x M
invK_Z_plus_Sigma_tf = tf.matmul(invL_Kmm_Sigma, invL_Kmm_Sigma, transpose_a=True) # P x M x M
Lq_diag = tf.matrix_diag_part(L_Kmm_Sigma) # P x M
mahanalobis_term = tf.matmul(q_mu_tf, K_mm) # P x M
mahanalobis_term = mahanalobis_term*q_mu_tf

KL_tf = 0.5*( tf.reduce_sum(mahanalobis_term) - tf.reduce_sum(invK_Z_plus_Sigma_tf*K_mm[None, :, :])
             - tf.reduce_sum(tf.log(Sigma_p_tf)) ) + tf.reduce_sum(tf.log(Lq_diag))

# Marginal mean and variance computation
fmean = tf.matmul(K_mn, q_mu_tf, transpose_a=True, transpose_b=True) # minibatch x P
y_all = tf.matrix_triangular_solve(L_Kmm_Sigma, tf_P_ones_3d*K_mn[None, :, :], lower=True) # P x M x minibatch
fvar = 1. - tf.reduce_sum(tf.square(y_all), axis=1) # P x minibatch

# Expectations computation
dot_phi_tf = (tf.matmul(Phi, fmean, transpose_b=True) +
    bias_tf)*mask_labels # K x minibatch
dot_phi_sqrt_tf = tf.sqrt(tf.matmul(tf.square(Phi), fvar
    ))*mask_labels # K x minibatch
dot_phi_sqrt_tf = tf.expand_dims(dot_phi_sqrt_tf, 0) # 1
     x K x minibatch
E_q_all_tf = tf.reduce_sum(w_new_tf * tf.reduce_sum(tf.
    nn.softplus(dot_phi_tf - x_new_tf_3d*dot_phi_sqrt_tf)
    , [1, 2]))

#Lower Bound
lower_bound = lik_factor*E_q_all_tf + KL_div_tf
```

```
'''
    Multu-label Classification

********************************
***   Full Parametrization   ***
********************************

K: number of labels
M: number of inducing inputs
P: number of GPs used
x_new: a vector of the precomputed  positions of the G-H
     qudrature (usually 10-dimensional)
w_new: a vector of the corresponding weights of the G-H
    qudrature
'''

import numpy as np
import tensorflow as tf

# Define Placeholders
X_batch = tf.sparse_placeholder(float_type, shape=[None,
     D]) # Sparse input tensor for a batch of training
    data
mask_labels = tf.placeholder(float_type, [K, None]) # a
    K x minibatch tensor with {-1,1} values; "-1"
    indicates positive label while "1" begatve one
lik_factor = tf.placeholder(float_type, ()) # the
    likelihood constant tha makes the stochastic
    estimation unbiased

# Define constants
```

```
jitter_eye_tf = tf.constant(1e-6*np.eye(M), dtype=
   float_type)
tf_P_ones_3d = tf.ones([P, 1 ,1],  dtype=float_type)
tf_M_ones = tf.ones([M, 1], dtype=float_type)
tf_P_ones_2d = tf.ones([P, 1],  dtype=float_type)
indices_tf = tf.constant([list(i) for i in list(zip(*np.
   tril_indices(M)))], dtype=tf.int64)
const_PM = tf.constant(P*M, dtype=float_type)

def vec_to_tri_vector(vector):
    return tf.scatter_nd(indices=indices_tf, shape=[M, M
   ], updates=vector)

# Define variables
Phi = tf.Variable(np.random.randn(K, P)/np.sqrt(P),
   dtype=float_type) # the factor loadings matrix
q_mu_tf = tf.Variable(0.5*np.random.randn((M, P)), dtype
   =float_type) # variational mean vectors
L_vecs = tf.Variable(0.2*L_vecs_0, dtype=float_type) #
   variational covariance matrices
Z_tf = tf.Variable(A_0, dtype=float_type) # inducing
   inputs
bias_tf = tf.Variable(np.random.randn(K, 1)/np.sqrt(P),
   dtype=float_type) # bias term
ells_tf = tf.Variable(np.log(np.exp(1.0) - 1)*np.ones(D)
   , dtype=float_type) # lengthscales

# Tranform Variables
q_sqrt_tf = tf.map_fn(vec_to_tri_vector, L_vecs) # P x M
    x M
ells_tf = tf.nn.softplus(ells_tf) + 1e-6
Z_tf = tf.nn.l2_normalize(Z_tf, 1)

Z_tf_ells_T = tf.transpose(Z_tf*ells_tf) # D x M
Xb_Lambda = X_batch.__mul__(ells_tf)

Z_dot_Z_tf = tf.matmul(Z_tf_ells_T, Z_tf_ells_T,
   transpose_a=True) # M x M
diag_ZZ_tf = tf.matrix_diag_part(Z_dot_Z_tf) # (M, )
A_p_tf = diag_ZZ_tf[:, None] + diag_ZZ_tf[None, :] - 2.*
   Z_dot_Z_tf

X_dot_Z_obj_tf = tf.sparse_tensor_dense_matmul(Xb_Lambda
, Z_tf_ells_T) # minibatch x M
A_k2_tmp = tf.sparse_reduce_sum(tf.square(Xb_Lambda), 1) - 2.*tf.transpose(X_dot_Z_obj_tf) # M x minibatch
A_k2 = tf.transpose(A_k2_tmp) + diag_ZZ_tf # minibatch x M

K_mm = tf.exp(-A_p_tf) + jitter_eye_tf # M x M

K_mn = tf.exp(-A_k2) # minibatch x M
K_mn = tf.transpose(K_mn) # M x minibatch

# KL computation
Lp = tf.cholesky(K_mm) # M x M
alpha = tf.matrix_triangular_solve(Lp, q_mu_tf, lower=True) # M x P
Lq_diag = tf.matrix_diag_part(q_sqrt_tf) # P x M
Lp_full = tf_P_ones_3d*Lp[None, :, :] # P x M x M
LpiLq = tf.matrix_triangular_solve(Lp_full, q_sqrt_tf, lower=True) # M x P
sum_log_sqdiag_Lp = tf.reduce_sum(tf.log(tf.square(tf.matrix_diag_part(Lp))))
KL_div_tf = 0.5*(P*sum_log_sqdiag_Lp + tf.reduce_sum(tf.square(alpha)) - const_PM - tf.reduce_sum(tf.log(tf.square(Lq_diag))) + tf.reduce_sum(tf.square(LpiLq)))

# Marginal mean and variance computation
A = tf.matrix_triangular_solve(Lp, K_mn, lower=True) # M x minibatch
fvar = 1. - tf.reduce_sum(tf.square(A), 0) # minibatch
fvar = tf_P_ones_2d*fvar[None, :] # P x minibatch
A = tf.matrix_triangular_solve(tf.transpose(Lp), A, lower=False) # M x minibatch
fmean = tf.matmul(A, q_mu_tf, transpose_a=True) # minibatch x P - Marginal mean
A = tf_P_ones_3d*A[None, :, :] # P x M x minibatch
LTA = tf.matmul(q_sqrt_tf, A, transpose_a=True) # P x M x minibatch
fvar = fvar + tf.reduce_sum(tf.square(LTA), 1) # P x minibatch - Marginal variance

# Expectations computation
dot_phi_tf = (tf.matmul(Phi, fmean, transpose_b=True) + bias_tf)*mask_labels # K x minibatch
```

```
dot_phi_sqrt_tf = tf.sqrt(tf.matmul(tf.square(Phi), fvar
   ))*mask_labels # K x minibatch
dot_phi_sqrt_tf = tf.expand_dims(dot_phi_sqrt_tf, 0) # 1
    x K x minibatch
E_q_all_tf = tf.reduce_sum(w_new * tf.reduce_sum(tf.nn.
   softplus(dot_phi_tf - x_new[:, None, None]*
   dot_phi_sqrt_tf), [1, 2]))

#Lower Bound
lower_bound = lik_factor*E_q_all_tf + KL_div_tf
```