DISS. ETH NO. 21848

CONVEX OPTIMIZATION AS A BUILDING BLOCK FOR DIFFICULT PROBLEMS IN MACHINE LEARNING

A dissertation submitted to
ETH ZURICH

for the degree of
DOCTOR OF SCIENCES

presented by
SHARON WULFF
Master of Science in Computer Science (Waterloo, Canada)

accepted on the recommendation of
Prof. Dr. Joachim M. Buhmann, examiner
Prof. Dr. Shai Ben-David, co-examiner
Prof. Dr. Angelika Steger, co-examiner

2014

Sharon Wulff: *Convex Optimization as a Building Block for Difficult Problems in Machine Learning,* © 2014

Dedicated to my mom and dad.
For their endless love and support.

ABSTRACT

Optimization lies at the core of most machine learning problems. The machine learning community has therefore invested a great deal of effort into both adapting well-established mathematical models and techniques to better suit the learning perspective, and developing new ones.

The first theme of this thesis is the computational aspects of several learning problem. We develop accurate and efficient algorithms for the following learning problems. (i) Multiple kernel learning, where the goal is to learn a weighted combination of candidate kernels as an alternative to either averaging or guessing the most suitable combination. (ii) Inference in graphical models, particularly in situations where the graph capturing the dependencies between the random variables contains cycles, with applications ranging from vision to natural language processing. (iii) A bi-clustering problem defined over categorical valued matrices, which is a natural formulation of tasks such as gene expression profiling. These problems vary in terms of their computational complexity. While multiple kernel learning is convex in the kernel weights and classifier parameters, the inference problem in the general case is known to be NP-hard, and the NP-hardness of the bi-clustering problem is established in this work.

Convex optimization is a powerful set of of tools. With modern computational resources, the convexity of a problem is almost a certificate of tractability. However, there are many examples of learning assumptions and scenarios that lead to strictly non-convex formulations. We show that the problems we consider here can be approximated by utilizing known convex objectives. For example, by gradually shifting from a convex linear program relaxation to a non-convex, yet concise, quadratic program, we obtain an inference algorithm that offers a competitive trade-off in terms of accuracy vs. run-time.

A second line of research that is developed in this thesis is concerned with sample complexity reduction using active learning. In the active learning framework the learner can choose which examples are to be labeled based on unlabeled data and previously seen labels. A fundamental, yet still open question is: Under which conditions does

active learning reduce the sample complexity? Previously shown lower bounds of $\Omega(\frac{1}{\epsilon^2})$ imply that label complexity reduction is not possible in general; further assumptions are necessary. Using a natural data clusterability assumption, we show that it is possible to attain substantial label savings under several realistic regimes.

Finally we present the complete pipeline of designing and testing a machine learning solution for the real-world problem of vehicle detection based on a sensor network measurements. Our approach combines existing machine learning techniques with new methods developed in this thesis.

ZUSAMMENFASSUNG

Mathematische Optimierung ist der Kern von den meisten maschinellen Lernproblemen. Die Gesellschaft für maschinelles Lernen hat deshalb viele Bemühungen dafür aufgewendet bestehende mathematische Modelle und Techniken dahingehend zu verändern, dass sie die Lernperspektive besser reflektieren.

Das erste Leitmotiv dieser Dissertation ist der Berechnungsaspekt von verschiedenen Lernproblemen. Wir entwickeln genaue und effiziente Algorithmen für die folgenden Lernprobleme: (i) Mehrkernlernmethoden, wobei das Ziel das Lernen einer gewichteten Kombination von Kandidatkernen ist, als Alternative zum einfachen Durchschnitt oder auch dem Raten einer günstigen Kombination. (ii) Inferenz in graphischen Modellen, insbesondere wenn der Graph, der die Abhängigkeiten zwischen den Zufallsvariablen ausdrückt, Zyklen enthält. Dies hat Anwendungen im Bild- und Sprachverarbeitungsbereich. (iii) Ein Biclustering Problem das über Matrizen mit kategorischen Werten definiert ist und eine natürliche Formulierung für die Analyse von Genexpressionsdaten ist. Diese Probleme variieren bezüglich deren Berechnungskomplexität. Während das Mehrkernlernen ein konvexes Problem in den Kerngewichten und in den Klassifikatorparametern ist, ist das Inferenzproblem in graphischen Modellen im generellen NP-hart. Weiter wird die NP-Härte des Biclustering Problems in dieser Arbeit hergeleitet.

Konvexe Optimierung ist ein mächtiges Werkzeug. Mit modernen Berechnungsresourcen, ist die Konvexität eines Problems ein Zertifikat für die Fügsamkeit des Problemes. Es gibt aber auch viele Beispiele von Problemen und Lernannahmen und -szenarien die zu strikt nichtkonvexen Formulierungen führen. Wir zeigen, dass die Probleme die wir in dieser Arbeit betrachten, durch bekannte konvexe Funktionen approximiert werden können. Beispielsweise indem wir die Optimierungsfunktion graduell von einem konvexen linearen Programm zu einem nicht-konvexen, aber kompakten quadratischen Programm verändern, erhalten wir einen Inferenzalgorithmus der einen kompetitiven Kompromiss zwischen Genauigkeit und Laufzeit bietet.

Eine zweite Forschungsrichtung welche in dieser Dissertation verfolgt wird, beschäftigt sich mit der Beispielskomplexitätsreduktion

durch aktives Lernen. Im aktiven Lernen Szenario kann der Lernende auswählen welche Beispiele annotiert werden sollen, basierend auf nicht annotierten Beispielen und Beispielen die zuvor annotiert wurden. Eine fundamentale, aber weiterhin offene Frage ist die folgende: "Unter welchen Voraussetzungen reduziert aktives Lernen die Beispielskomplexität"? Zuvor bekannte untere Schranken von $\Omega(\frac{1}{\epsilon^2})$ implizieren, dass eine Beispielskomplexitätreduktion im allgemeinen nicht möglich ist und weitere Annahmen nötig sind. Unter einer natürlichen Gruppierungsannahme, zeigen wir, dass es möglich ist eine substanzielle Annotierungsreduktion zu erhalten; dies unter mehreren realistischen Annahmen.

Zuletzt präsentieren wir eine komplette Studie vom Design und Testen einer maschinellen Lernlösung für das Problem der Fahrzeugdetektion basierend auf den Messungen eines Sensornetzwerks. Unsere Lösung kombiniert existierende maschinelle Lerntechniken mit neuen Methoden welche in dieser Dissertation entwickelt wurden.

PUBLICATIONS

The following publications are included in parts or as an extended version in this thesis:

- S. Wulff (2008). „Computational Complexity Of Bi-clustering.“ University of Waterloo, Ontario, Canada.
- P. Pletscher and S. Wulff (2012). „LPQP for MAP: Putting LP Solvers to Better Use.“ In: *Proceedings of the 29th International Conference on Machine Learning (ICML)*.
- S. Wulff and C. S. Ong (2013). „Analytic center cutting plane method for multiple kernel learning.“ In: *Annals of Mathematics and Artificial Intelligence*.
- R. Urner, S. Wulff, and S. Ben-David (2013). „PLAL: CLuster-based active learning.“ In: *Conference on Learning Theory (COLT)*.
- S. Wulff, R. Urner, and S. Ben-David (2013). „Monochromatic Bi-Clustering.“ In: *Proceedings of the 30th International Conference on Machine Learning (ICML)*.

ACKNOWLEDGMENTS

I would like to express my utmost gratitude to my supervisor Joachim Buhmann. Throughout my PhD he had offered his advise and support, and at the same time he gave me the freedom to pursue almost any research direction I found interesting. His door was always open for professional consultation, as well as discussing personal matters. His comments were very often the missing link or alternatively the hidden flaw in the algorithm or framework in discussion. During my PhD I had the privilege of working again with Shai Ben-David, my master's degree supervisor from the university of Waterloo, Canada. Through mutual visits and Skype calls he has been nothing less than a source of inspiration to me. His sharp observations and rigorous methodology have shaped my approach to research. Finally in the list of mentors I am happy to include Cheng Soon Ong who taught me numerous very useful things; from down to earth tools usage and machine learning programming techniques, to identifying good ideas and structuring a research project and maybe most importantly: how to maintain my sanity and life balance during the ups and downs that are unavoidable through the course of a PhD. I would also like to thank my collaborator and co-author Ruth Urner, it was both fruitful and a great pleasure to work with her. I would like to thank the Shockfish (Tinynode) crew, especially Pierre Castella for a very engaging interaction over the course of our joint project. I would like to thank my colleagues and friends Sasha, David, Brian, Gabriel and Hasta for useful scientific discussions and to David for proofreading. I would like to thank our secretary Rita Klute from the bottom of my heart, her kindness and resourcefulness were proven priceless in many occasions and she had a great contribution to the general good atmosphere in the lab. I would also like to thank Denise Spicher, the informatik department secretary, for her great help sorting out all possible administrative and teaching related matters, always with a big smile on her face. I would like to thank my dear family. My parents Uri and Batia, and sisters Anat and Karin who always supported and believed in me. Their love gave me the strength to cope with everything and anything that came my way. I wish my grandmother, Lea, would have been alive to see me graduate, I

know it would have meant the world to her. Finally I wish to thank and acknowledge Patrick Pletscher, my colleague, collaborator, best friend and life partner. From extremely useful dinner-time brainstorming, to a shoulder to cry on over a rejected paper, his love and support truly made all the difference in the world.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

1

INTRODUCTION

In recent years we are experiencing an immense growth in the quantity of data that is being generated and collected for a constantly growing set of purposes. Gene sequencing, social networks and smart sensors are just a few examples of relatively new fields generating large amount of data. Developing new methods to effectively and efficiently analyze data is of great importance.

The machine learning approach to utilizing data is similar to the human learning process: Generalizing from past data to make observations and predictions on related future data. To automate this procedure, machine learners devise algorithms that use input examples to form a hypothesis about the nature of the data. The hope is that this hypothesis is then useful for explaining or predicting measures of interest in future data.

Often the predictions we wish to make take the form of an associating an outcome or a label to data. For example given the current state of the atmosphere predicting whether or not it will rain tomorrow. We use the terms training or learning to refer to the part of the procedure in which the hypothesis is formed; and testing or predicting for when it is applied to new unseen data.

There are various choices and assumptions we can make when applying this general framework in practice, each of which leads to a distinct learning problem. For example the distinction between batch and online learning corresponds to the choice between having the training and testing as two separate phases or not. Passive vs. active learning refers to the differentiation between getting labels a priori for all examples, or having the algorithm choose which examples are to be labeled based on the unlabeled data. In most practical scenarios we can not expect to form a hypothesis that correctly captures all possible inputs. Thus we also need to specify the hypothesis class that our algorithm searches over, and the loss quantifying incorrect predictions.

In the process of investigating a certain problem or learning scenario, we formulate a model and encode our choices and assumptions in terms

of mathematical objects. This procedure often results in an optimization problem. With the demand for robust and scalable methods, researchers put much effort into improving the efficiency of optimization techniques for machine learning problems. While large parts of the theory and methodology used in the machine learning community is common and even based on older fields such as statistical pattern recognition, the focus on efficiency is one of the key differences.

In fortuitous cases, the devised problem belongs to a class of problems which are considered "easy" to solve. Convex problems are the most prominent example of such a class. The property that makes the class of convex problems almost a synonym for efficiently solvable, is that every local optima is also the global one. Examples of learning problems where the resulting optimization is convex include any regularized empirical risk minimization scheme with a convex loss and a convex regularizer. For example, Logistic and Ridge regression, Support Vector Machine (SVM)s and structured output learning. However, even within this class the convergence rates differ substantially between algorithms and even more so between subclasses of convex problems such as smooth or strongly convex functions.

In less fortuitous cases, the optimization problem that best captures our understanding of a particular model does not fall under the category of any known tractable problem classes – or is even an NP-hard problem. There are many examples of NP-hard problems in machine learning, e.g. k-means clustering or finding the smallest decision tree that correctly classifies all of training examples. Approaches to solving such cases vary in terms of the guarantees they provide. Approximation algorithms output a solution which is guaranteed to be within a bounded multiplicative or additive gap compared to the optimal solution. Another type of approach modifies the original objective so that the modified problem is easier to solve. For example convexifying a non-convex objective, or searching over a subset of the original search space. The latter strategy can be seen as finding the optimal hypothesis in a restricted subset of the hypothesis class. A third approach recovers a local minimum of an NP-hard problem by iteratively improving on an initial solution. Although they do not have formal guarantees, in practice these algorithms can be very successful.

The work presented in this thesis divides naturally into the following three parts.

CONVEX OPTIMIZATION FOR MACHINE LEARNING PROBLEMS The first part of this thesis focuses on developing algorithms for the optimization problems that emerge in the context of three different learning scenarios. In Chapter 3 we consider the Multiple Kernel Learning (MKL) problem, where the goal is to learn a weighted combination of kernels for prediction with an SVM (Lanckriet et al. 2004b; Sonnenburg et al. 2006). We use a cutting plane algorithm, that approximates the objective with growing refinement by enclosing constraints on the optimal kernel parameters. To form a new constraint, we choose a query point and call an "oracle" that computes the constraint based on the query point. Interestingly, this oracle reduces to a single kernel SVM call. We show that it is advantageous to use the analytic center of the set defined by the existing constraints as the query point.

In Chapter 4 we study the Maximum-A-Posteriori (MAP) inference problem in the context of pairwise graphical models, and in Chapter 5 we analyze a variant of the bi-clustering task. While the MKL formulation we consider is convex in the kernel weights and SVM parameters, the latter two are NP-hard combinatorial problems. The NP-hardness of the bi-clustering problem is shown in Section 5.4. For both problems we leverage on a convex Linear Program (LP) relaxation over a marginal polytope (Wainwright and Jordan 2008) to derive computationally efficient algorithms. In order to find a MAP solution we combine the LP relaxation with a Quadratic Program (QP) relaxation, and solve the resulting non-convex objective by means of decomposition into two convex functions. For the proposed variant of the bi-clustering task we show an approximation algorithm with guarantees, as well as a dual decomposition algorithm (Sontag, Globerson, and Jaakkola 2011) which is a LP relaxation in the dual space, and is better suited for large problem instances.

We demonstrate the usefulness of the developed algorithms empirically by showing results on synthetic and real world data. We test the accuracy of the prediction as well as the run time.

ACTIVE LEARNING The second part of this thesis is concerned with the Active Learning (AL) settings. Consider the following examples of learning applications: predicting the presence of cancer cells in extracted tumor tissues; automatically recovering objects

in web images using crowd-sourced tags; and predicting the matching between two people based on their dating website profiles. One thing these applications have in common is that labeling training examples is costly. The cost can be quantified in terms of the medical supplies and time pathologists need to invest in order to distinguish between cancerous vs. healthy cells; the compensation paid to mechanical turk or equivalent service workers; and the time and mental effort users expend on meeting potential matches. The premise of AL is based on the observation that while labels are expensive, quite often unlabeled examples are readily available. In the AL setting the learning algorithm is presented with unlabeled training data, and can then choose which examples it would like the system or the user to label. In the image tagging example an AL algorithm will consider a batch of images and dispatch only a subset of them to mechanical turk workers. Naturally the goal of AL is to use as few labels as possible while maintaining high prediction accuracy.

Although there is much work done on developing AL algorithms, completely characterizing the scenarios in which AL reduces the sample complexity remains an open problem. In Chapter 6 we continue the line of AL research pursued in (Dasgupta and Hsu 2008) which exploits the data's cluster structure to guide the choice of examples. We analyze the sample complexity of an AL algorithm similar to the one proposed in (Dasgupta and Hsu 2008) under a natural assumption on the "smoothness" of the data labeling function.

SENSOR NETWORK BASED VEHICLE DETECTION In a joint project between ETH and a company named Tinynode[1] we developed a machine learning solution for the problem of detecting vehicles in rest areas based on earth's magnetic field sensors. The challenges of accurately predicting the presence of vehicles based on the sensors measurements include: high variability between sensors, correlated sensors measurements, as well as resources required for obtaining labeled data. In Chapter 7 we present a machine learning solution pipeline, including data representation, feature selection, choosing a suitable prediction model and an active learning approach for reducing the image annotation efforts.

1 http://www.tinynode.com

The rest of the thesis is organized as follows. A summary of our contributions is given in Chapter 1. A formal presentation of the background and concepts we use is given in Chapter 2. In Chapter 3 and Chapter 4 we derive algorithms for the multiple kernel learning problem and MAP inference respectively. Chapter 5 provides a full theoretical analysis, as well as practical algorithms for the monochromatic bi-clustering problem. In Chapter 6 we present our AL labeling procedure and sample complexity analysis. Each chapter introduces and motivates the specific setting, and finally presents results on synthetic and real-world data. In Chapter 7 we discuss our solution to the sensor-based vehicle detection application. Finally Chapter 8 concludes with a summary of our contributions and suggested future directions.

CONTRIBUTIONS

The machine learning settings described in this thesis as well as most of the techniques, were known before. Below is a list of the original contributions of this thesis.

1. We study the Multiple Kernel Learning (MKL) problem with a convex combination of kernels, as described in (Lanckriet et al. 2004b; Sonnenburg et al. 2006; Zien and Ong 2007). In our work (Wulff and Ong 2013), we cast several previously proposed MKL solutions in a cutting-plane optimization framework, with a single kernel SVM-oracle. We show that these solutions correspond to different ways of generating query points. We propose to use the analytic center as a query point, with the intuition that this query point in expectation halves the hypothesis space. We show empirically that our more "regularized" approach requires fewer iterations in many cases, and compared to other solutions it is more robust to variations in the data.

2. We present an efficient message-passing algorithm for the MAP problem in a pairwise graphical model. Our algorithm combines a QP relaxation with the widely used LP relaxation, through a Kullback-Leibler (KL) penalty term. The optimization is performed by repeatedly solving a convex sub-problem by means of a known MAP algorithm. Through experiments we demonstrate that compared with other popular LP solvers, our algorithm exhibits a favorable trade-off between running time and low energy

(cost) solutions. This behavior can very beneficial in practical applications. The results of this work are described in (Pletscher and Wulff 2012).

3. We continue the line of research into the monochromatic bi-clustering problem initiated in my master thesis (Wulff 2008). Our contributions here are two fold: We give a complete theoretical analysis of the computational complexity of the monochromatic bi-clustering with missing entries problem. This includes a new NP-hardness result and an adapted Polynomial Time Approximation Scheme (PTAS). In addition, we formulate the monochromatic bi-clustering task as a factorized energy minimization problem. This factorization gives rise to an efficient and practical algorithm which minimizes the LP relaxation in the dual space, where the high connectivity and large factors are not an obstacle. This work was partially published in (Wulff, Urner, and Ben-David 2013).

4. We extend the theoretical understanding of the data assumptions under which Active Learning (AL) provably reduces label complexity. In (Urner, Wulff, and Ben-David 2013) we use a data clusterability notion previously used in the semi-supervised learning settings, to prove that a variant of a previously proposed AL algorithm requires fewer labels compared to passive learning algorithms under the same assumption. Finally, we show that a "budgeted" version of this algorithm can be used as preprocedure to reduce image annotation efforts in the Tinynode sensors application.

5. We provide a practical machine learning solution to the real world problem of occupancy prediction based on the earth's magnetic field sensor network measurements.

2

BACKGROUND

This chapter introduces the basic notation of this thesis and outlines the foundations on which our methods are building.

2.1 THE LEARNING FRAMEWORK

Let $\mathcal{X}$ denote a domain set and $\boldsymbol{x} \in \mathcal{X}$ a data point, for example $\mathcal{X} = \mathbb{R}^d$ or $\mathcal{X} = [0,1]^d$ for some dimension $d \in \mathbb{N}$. For most parts of this thesis we consider the supervised classification settings where the task is to assign a label to each data point. Let $\mathcal{Y}$ denote some label space, and $y \in \mathcal{Y}$ a label. In binary classification the label has two possible states $\mathcal{Y} = \{-1, 1\}$, more complex settings are discussed in Section 2.3.

We consider the space of input-output pairs $\mathcal{X} \times \mathcal{Y}$ endowed with the data generating probability distribution P over $\mathcal{X} \times \mathcal{Y}$. We use $P_{\mathcal{X}}$ to denote the marginal probability of P over $\mathcal{X}$.

A hypothesis (or classifier) f is a function that maps instances to labels, $f : \mathcal{X} \to \mathcal{Y}$. A hypothesis class $\mathcal{F}$ is a set of hypotheses.

We measure the discrepancy or the error of predicting a label $\hat{y}$, when the real label is y, using a loss function $\ell : \mathcal{Y} \times \mathcal{Y} \to [0, \infty)$. Below are few important examples

1. Zero-one loss: $\ell(y, \hat{y}) = \mathbb{I}_{y \neq \hat{y}}$ where $\mathbb{I}$ is an indicator function returning 1 if the condition is satisfied and 0 otherwise.

2. Square loss: $\ell(y, \hat{y}) = (y - \hat{y})^2$

3. Hinge loss: for $t \in \mathbb{R}$ the Hinge loss is defined as $\max(0, 1 - t)$. In binary classification where the output of the classifier is given by a real valued score t, the Hinge loss is defined as $\ell(y, t) = max(0, 1 - yt)$.

Note that a loss function should satisfy $\forall y, \hat{y} \in \mathcal{Y}$: $\ell(y, \hat{y}) \geq 0$ and $\ell(y, y) = 0$. From here on we assume a fixed loss function ℓ.

For a hypothesis f we define the error of f with respect to P as the expected loss:

$$\mathrm{Err}_P(f) = \mathbb{E}[\ell(f(\boldsymbol{x}), y))]$$

For the zero-one loss $\mathrm{Err}_P(f)$ is given by $\Pr_{(\boldsymbol{x},y)\sim P}(f(\boldsymbol{x}) \neq y))$

We let $f^* \in \mathcal{Y}^{\mathcal{X}}$ denote the hypothesis which minimizes the expected loss over all possible hypotheses:

$$f^* = \operatorname*{argmin}_{f \in \mathcal{Y}^{\mathcal{X}}} \mathrm{Err}_P(f)$$

and use $f^*_{\mathcal{F}}$ to denote the hypothesis with the lowest expected error restricted to the class $\mathcal{F}$:

$$f^*_{\mathcal{F}} = \operatorname*{argmin}_{f \in \mathcal{F}} \mathrm{Err}_P(f)$$

For simplicity we assume that $f^*, f^*_{\mathcal{F}}$ are well defined and unique.

2.1.1 *Empirical Risk Minimization*

In reality we do not have access to the underlying distribution P. Instead we assume that we are given a sample S of examples, $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$, independently drawn from P. We often refer to S as the training set.

We define the empirical error of a hypothesis f as

$$\mathrm{Err}_S(f) = \frac{1}{n}\sum_{i=1}^{n}[\ell(f(\boldsymbol{x}_i), y_i)]$$

Definition 2.1 (Empirical Risk Minimization (ERM) principle)**.** Given a sample S drawn from some P and a fixed hypothesis class $\mathcal{F}$, choose the hypothesis $f \in \mathcal{F}$ which minimizes the empirical error w.r.t. S.

All of the sample based learning algorithms adhere to the ERM principle in one form or another. Often the minimizer is found with respect to an additional penalty term on the complexity of f (see discussion in Subsection 2.1.2).

We use f^*_S to denote the ERM minimizer, formally defined as

$$f^*_S = \operatorname*{argmin}_{f \in \mathcal{F}} \mathrm{Err}_S(f)$$

Following the presentation in (Duda and Hart 1973) we can now express the generalization error of a learning algorithm as

$$\begin{array}{ccccc} \mathbb{E}[\mathrm{Err}_P(f^*_{\mathcal{F}}) - \mathrm{Err}_P(f^*)] & + & \mathbb{E}[\mathrm{Err}_P(f^*_S) - \mathrm{Err}_P(f^*_{\mathcal{F}})] & = \\ \text{approximation error} & + & \text{estimation error} & \end{array} \tag{2.1}$$

Where the expectation is taken with respect to the random choice of the training set.

The approximation error is governed by the ability of functions from $\mathcal{F}$ to approximate the true underlying labeling function. This ability is often referred to as the complexity or capacity of the hypothesis class (Vapnik 1982). The well-known VC-dimension is a measure of the complexity of a hypothesis class, defined for binary predictors (Vapnik and Chervonenkis 1971; Vapnik 1982).

The estimation error is due to the minimization over the empirical risk instead of the expected risk. This error depends on the number of examples given and again on the complexity of the hypothesis class.

A choice of less complex hypothesis class leads to a lower estimation error, but a larger approximation error (Vapnik 1982).

The task of finding the ERM minimizer for a given model (often with additional assumptions) can be a computationally difficult optimization problem. Bottou and Bousquet (2008) propose to add an optimization-error term to the generalization error decomposition (2.1). In cases where the empirical risk minimizer, f^*_S, can not be recovered exactly due to large amount of examples or limited computational resources, this term is greater than zero. The authors show that due to this error, for large-scale data the generalization error trade-off differs compared to the lower sample size scenario, this can have implications on the choice of hypothesis class for example.

Using the above setup we formally define learning as[1]

Definition 2.2 (Learning)**.** A learning algorithm $\mathcal{A}$ *learns* some hypothesis class $\mathcal{F}$ over $\mathcal{D}$ with respect to a set of distributions $\mathcal{Q}$ over $\mathcal{D} \times \{-1, 1\}$, if there exists a function $m : (0,1) \times (0,1) \to \mathbb{N}$ such that for all distributions $P \in \mathcal{Q}$ with probability at least $1 - \delta$ over the choice

1 The presentation here is in terms of binary classification as it relies on the definitions of the PAC (Probably Approximately Correct) learning model. Extension of the learning theory to other settings can be found in (Natarajan 1989; Ben-David et al. 1995; Daniely et al. 2011).

of an *i.i.d.* sample S, of size at least $m(\epsilon, \delta)$ from P, $\mathcal{A}(S) \leq \mathrm{Err}_P(f^*_{\mathcal{F}}) + \epsilon$. $\mathcal{A}(S)$ denotes the output classifier of $\mathcal{A}$ on the input S.

In the agnostic PAC (Probably Approximately Correct) learning model (Valiant 1984), learnability is defined with respect to the set $\mathcal{Q}$ of all distributions over $\mathcal{D} \times \{-1, 1\}$.

Definition 2.3 (Sample complexity)**.** The smallest function that satisfies the condition in Definition 2.2 is called the *sample complexity* of algorithm $\mathcal{A}$ for learning $\mathcal{F}$ with respect to $\mathcal{Q}$.

2.1.2 *Regularized Risk Minimization*

One way to control the approximation-estimation trade-off (2.1) is via a penalty on the complexity of the classifier. For a classifier f and a sample $S = \{(\boldsymbol{x}_i, y_i)\}_{i=1}^n$ the general form of the regularized risk is given by

$$\frac{1}{n}\sum_{i=1}^{n}[\ell(f(\boldsymbol{x}_i), y_i)] + \Omega(f) \tag{2.2}$$

where $\Omega(f) : [0, \infty] \to \mathbb{R}$ is the regularization term.

The Regularized Risk Minimization (RRM) (or Regularized Loss Minimization (RLM)) classifier is defined as

$$f^*_S = \operatorname*{argmin}_{f \in \mathcal{F}} \frac{1}{n}\sum_{i=1}^{n}[\ell(f(\boldsymbol{x}_i), y_i)] + \Omega(f)$$

Typically the regularization term is convex and differentiable, thus easy to compute. In Chapter 3 we study a binary classification prediction problem using a weighted combination of kernels. The kernel coefficient are found subject to the sparsity inducing $\mathcal{L}_1$ norm (i.e. for $\mathbf{w} \in \mathbb{R}^d$, $\Omega(\mathbf{w}) = \|\mathbf{w}\|_1$) as the regularization term.

There is large body of work on RRM and generalization error bounds, however, a survey of this topic is beyond the scope of this thesis.

2.2 LINEAR CLASSIFIERS AND SUPPORT VECTOR MACHINES

Linear classifiers assume a discriminative model of the form

$$f_{\boldsymbol{w},b}(\boldsymbol{x}) = \mathrm{sgn}(\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}) \rangle + b) \tag{2.3}$$

The hypothesis class is parametrized by a weight vector $\boldsymbol{w} \in \mathbb{R}^d$, bias $b \in \mathbb{R}$ and a feature mapping $\boldsymbol{\phi}$ that maps the original features to an alternative representation. In the simplest case the feature mapping is the identity function. Different learning objectives for such a linear model exist, most of them can be written in the form of Regularized Risk Minimization (RRM):

$$f^*_{\boldsymbol{w},b} = \operatorname*{argmin}_{\boldsymbol{w},b} \sum_{i=1}^{n} \ell(\boldsymbol{w}, \boldsymbol{x}_i, y_i) + \frac{1}{2}\|\boldsymbol{w}\|^2, \tag{2.4}$$

here we assume a $\mathcal{L}_2$ regularizer on the weight vector, which is the most common approach. The loss function $\ell(\boldsymbol{w}, \boldsymbol{x}, y)$ is written using a slightly different notation, but it has the same meaning; the loss of using $\boldsymbol{w}$ for predicting a label y' for $\boldsymbol{x}$, when the true label is y. Several different choices exist for the loss as well, one widely used choice is the Hinge, or maximum-margin loss. It became popular due to its application in the Support Vector Machine (SVM) (Boser, Guyon, and Vapnik 1992; Cortes and Vapnik 1995) algorithm. For the binary classification setting, where $y \in \{-1, 1\}$ the maximum margin amounts to

$$\ell(\boldsymbol{w}, \boldsymbol{x}, y) = \max(0, 1 - y(\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}) \rangle + b)).$$

An alternative formulation of the SVM objective adds the Hinge loss to the constraints, rather than the objective and is given by

$$\begin{aligned} \min_{\boldsymbol{w},b} \quad & \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N} \xi_i \\ \text{s.t.} \quad & y_i(\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}_i) \rangle + b) \geq 1 - \xi_i \qquad \forall i, \quad \xi_i \geq 0 \end{aligned} \tag{2.5}$$

The parameter C governs the trade-off between the weight of the loss and the regularization in the minimization. This formulation is more common in the standard literature as it has a direct interpretation in terms of geometry. The Hinge loss is convex in the weight $\boldsymbol{w}$ and therefore the overall RRM problem in (2.4) is also convex in $\boldsymbol{w}$. Hence any convex solver which can deal with the non-differentiability of the Hinge loss could be used to solve the SVM problem, in practice however generic solvers are prohibitively slow at solving the problem and highly specialized SVM solvers are generally used (Chang and Lin 2011; Fan et al. 2008).

2.2.1 *Kernel Methods*

The dual of the optimization problem in (2.5) has one variable α_i for each of the n data points and can be derived by standard Lagrangian duality (Schölkopf and Smola 2001).

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{i=1}^{n} \alpha_i - \sum_{i=1}^{n} \sum_{j=1}^{n} \frac{1}{2} \alpha_i \alpha_j \langle \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_j) \rangle \\ \text{s.t.} \quad & \alpha_i \geq 0 \quad \forall i \\ & \sum_{i=1}^{n} \alpha_i y_i = 0 \quad \forall i \end{aligned} \tag{2.6}$$

Whereas the primal problem is a minimization problem with respect to $\boldsymbol{w}$, the dual in (2.6) is a maximization problem with respect to the dual variables α_i. A second important observation about the dual problem is the fact that the data points only appear through the inner product terms $\langle \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_j) \rangle$. This observation gave rise to the idea of the "kernel trick" (Schölkopf and Smola 2001), to use a kernel function $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \boldsymbol{\phi}(\boldsymbol{x}_i), \boldsymbol{\phi}(\boldsymbol{x}_j) \rangle$ directly instead of first mapping the data points through $\boldsymbol{\phi}(\cdot)$ and in a second stage computing the inner product. Intuitively, the kernel trick enables us to learn and predict in some potentially complex feature space representation of the data, using a linear classifier.

More formally a kernel function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is an inner product in some dot product or feature space $\mathcal{H}$, satisfying: $\forall \boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}$

$$k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') \rangle_{\mathcal{H}} \tag{2.7}$$

with $\boldsymbol{\phi} : \mathcal{X} \to \mathcal{H}$ as the mapping function.

Due to properties of an inner product, an equivalent definition is by means of the Gram matrix: A symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is called a positive semi definite kernel if $\forall \boldsymbol{x}_1, \ldots \boldsymbol{x}_n \in \mathcal{X}$ the $n \times n$ Gram matrix defined by

$$K(i, j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

is positive semi-definite.

This equivalence enables the kernel trick in cases where computing the mapping of the data $\boldsymbol{x} \mapsto \boldsymbol{\phi}(\boldsymbol{x})$ is computationally difficult, e.g. radial basis kernel (see definition Table 2.1) where the feature space itself is of infinite dimension (Steinwart, Hush, and Scovel 2006).

Moreover, the Moore-Aronszajn theorem (Aronszajn 1950) guarantees that a positive definite kernel is associated with a unique reproducing kernel Hilbert space, thus the inner product is well defined. We refer the reader to (Schölkopf and Smola 2001; Shawe-Taylor and Cristianini 2004) for a detailed presentation of learning with kernels, and to (Aronszajn 1950) for reproducing kernel Hilbert spaces.

Table 2.1 shows a list of kernels that are frequently used for machine learning applications. Most of these kernels have hyperparameters, such as σ in the radial basis function kernel which are crucial to set to a value that leads to good generalization properties for the problem. Multiple kernel learning discussed in Chapter 3 is one approach to solve this problem in a principled manner. Kernels are closed under

Name	Kernel $k(\boldsymbol{x}, \boldsymbol{x}')$
Linear	$\langle \boldsymbol{x}, \boldsymbol{x}' \rangle$
Polynomial	$(\langle \boldsymbol{x}, \boldsymbol{x}' \rangle + c)^d$
Radial basis function	$\exp(-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|^2}{2\sigma^2})$
Histogram intersection ($x_d \geq 0, x'_d \geq 0$)	$\sum_{d=1}^{D} \min(x_d, x'_d)$

Table 2.1: List of popular kernels in machine learning.

the following operations. Let k_1 and k_2 be valid kernels:

- *scaling*: For a positive real number a, $k(\boldsymbol{x}, \boldsymbol{x}') := ak_1(\boldsymbol{x}, \boldsymbol{x}')$ is a kernel.
- *sum*: The sum $k(\boldsymbol{x}, \boldsymbol{x}') := k_1(\boldsymbol{x}, \boldsymbol{x}') + k_2(\boldsymbol{x}, \boldsymbol{x}')$ of two kernels is a kernel.
- *linear combination*: For real positive numbers a and b, $k(\boldsymbol{x}, \boldsymbol{x}') := ak_1(\boldsymbol{x}, \boldsymbol{x}') + bk_2(\boldsymbol{x}, \boldsymbol{x}')$ is a kernel.
- *product*: The product of two kernels $k(\boldsymbol{x}, \boldsymbol{x}') := k_1(\boldsymbol{x}, \boldsymbol{x}')k_2(\boldsymbol{x}, \boldsymbol{x}')$ is a kernel.
- *exponentiated*: For a positive integer p, $k(\boldsymbol{x}, \boldsymbol{x}') := k_1(\boldsymbol{x}, \boldsymbol{x}')^p$ is a kernel
- *metric*: For $\boldsymbol{x}, \boldsymbol{x}' \in \mathbb{R}^d$, and for a positive definite matrix $\boldsymbol{M}$, $k(\boldsymbol{x}, \boldsymbol{x}') := \boldsymbol{x}^\top \boldsymbol{M} \boldsymbol{x}'$ is a kernel.

The first three operations are useful in the context of multiple kernel learning Chapter 3.

2.3 STRUCTURED MODELS

So far we discussed the prediction problem where each example $\boldsymbol{x}$ has a single label y. For example, in binary classification, the label y of an instance is in $\{-1, 1\}$. As it turns out, most of the theory is general and can be extended to structured models, models where the label is vector valued. This scenario is important in natural language processing, computational biology or computer vision. In computer vision for example, the label of an image might be a full segmentation into background and foreground pixels.

Formally, the label $\boldsymbol{y}$ is a vector of individual output variables $y_i \in \mathcal{Y}_i$. In segmentation applications $\mathcal{Y}_i$ might take K different values and the overall label domain is the product domain, $\mathcal{Y} = \mathcal{Y}_i^{|\mathcal{V}|}$, where $|\mathcal{V}|$ is the length of the sequence or the number of pixels in a computer vision application. It becomes apparent that the number of possible states of a label $\boldsymbol{y}$ becomes extremely large, and the combinatorial problem of picking a label that minimizes some cost function with dependencies among the output variables, is a difficult combinatorial optimization problem. In Chapter 4 we discuss a relaxation approach for this problem. While there exists a whole body of literature on learning structured models (Bakir et al. 2007; Nowozin and Lampert 2011; Taskar, Guestrin, and Koller 2003; Tsochantaridis et al. 2005; Pletscher 2012), this thesis will not discuss this aspect in much detail and will mostly consider the prediction aspect of the problem, i.e. given a cost function, how to choose the label that minimizes the cost function.

For the specification of the cost of a labeling in a structured output setting, graphical models and factor graphs (Koller and Friedman 2009; Kschischang, Frey, and Loeliger 2001) provide a very convenient framework, and have been shown to be extremely valuable. We associate a label with a set of $|\mathcal{V}|$ random variables, the output variables. In a graphical model representation, the relationships between these output variables, which according to the underlying assumption are not independent, are made explicit in the form of a directed or undirected graph structure. The joint distribution of the output variables, is factorized into a product of joint distributions over subsets of the variables in the graph.

In this thesis we will only discuss undirected graphical models. Furthermore, as we mostly consider non-probabilistic discriminative models, we will often not explicitly define a distribution over the output

variables, but instead consider a generic cost of an assignment of the variables, also called the energy of a configuration. This a common approach in the graphical models literature, when needed, the link to probabilities is usually done through the Gibbs distribution for a given energy.

We start by describing a class of undirected graphical models, namely the factor graph (Kschischang, Frey, and Loeliger 2001).

2.3.1 *Factor Graphs*

Given a set of random variables $\boldsymbol{y} = \{y_1, \dots y_n\}$, their factor graph representation $\mathcal{FG} = (\mathcal{V}, \mathcal{C}, \mathcal{E})$ consists of: Variable nodes $\mathcal{V}$, where for each variable y_i there is an associated node $i \in \mathcal{V}$. Edges $\mathcal{E}$, connecting a subset of dependent variables to the factor nodes. And finally factor nodes $\mathcal{C}$ that express the dependencies between the adjacent variables.

The factor graph representation is not unique. For example, the two factor graphs shown in Figure 2.1, describe the same dependencies between the variables. By convention we use circles for variable nodes and squares for factor nodes.

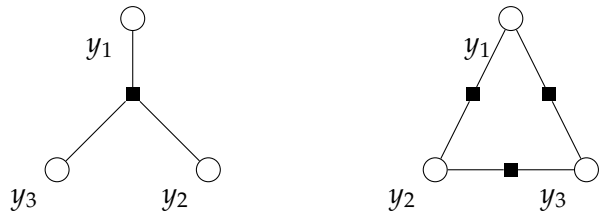


Figure 2.1: Two factor graphs representation of the same dependencies between the random variables

The cost, or energy $E(\boldsymbol{y})$ of a joint assignment $\boldsymbol{y}$ of the random variables, is factorized according to the factor graph. It is written as

$$E(\boldsymbol{y}) = \sum_{c \in \mathcal{C}} \theta_c(\boldsymbol{y}_c).$$

where $\theta_c(\boldsymbol{y}_c) : \mathcal{Y}_c \to \mathbb{R}$ is called a factor potential function, and it is used to specify the cost contribution of the joint configuration $\boldsymbol{y}_c$. The subset notation restricts the variables to only those in factor c. In a cost minimization problem, assignments with smaller energies are favorable over those that have a higher energy.

2.3.2 *Pairwise Markov Random Field*

Markov Random Field (MRF) is a particular case of a factor graph, though it is in fact an earlier undirected graphical model (Geman and Geman 1984; Kindermann and Snell 1980). A MRF does not contain any factor nodes, and the edges are directly connecting variable nodes. The potential functions in a MRF are defined over maximal cliques in the graphs.

In Figure 2.1, the factor graph representation on the right can be transformed into a MRF, by simply removing the factor nodes.

In a pairwise Markov Random Field (MRF) the maximal clique in the graph has size two. The pairwise MRF is a very popular representation, due to the wide range of applications which adhere to this form. Application where the potentials are defined over larger subsets, also called high order factors, are often more convenient to handle in the factor graph form.

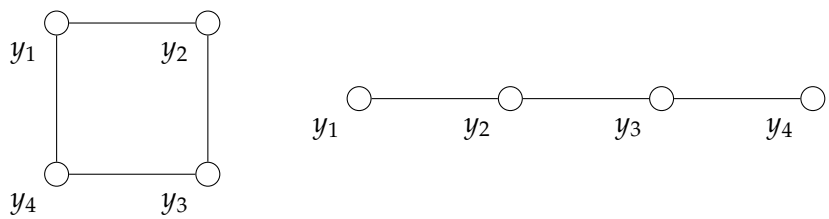


Figure 2.2: Two examples of MRF models. *Left*: Grid graph typically used in computer vision applications. *Right*: linear chain graph which is a natural dependency structure for natural language and speech applications.

The factor potential functions in a pairwise MRF $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, are defined over the edges. For an edge $(i, j) \in \mathcal{E}$ the pairwise potential function θ_{ij} essentially lists the cost $\theta_{ij}(y_i, y_j)$ of each joint state $y_i, y_j \in \mathcal{Y}_i \times \mathcal{Y}_j$. In addition, we define for each node $i \in \mathcal{V}$ a unary potential function $\boldsymbol{\theta}_i$, specifying the cost $\theta_i(y_i)$ of each state $y_i \in \mathcal{Y}_i$. In a factor graph representation, a unary potential function can be defined through a factor with a single variable node connected to it.

Put together, the energy of a pairwise MRF has the following form

$$E(\boldsymbol{y}) = \sum_{i \in \mathcal{V}} \theta_i(y_i) + \sum_{(i,j) \in \mathcal{E}} \theta_{ij}(y_i, y_j).$$

Note that the unary as well as the pairwise potential functions, can be set to zero. If for all $(i,j) \in \mathcal{E}$ and all joint states y_i, y_j, $\theta_{ij}(y_i, y_j) = 0$, then we are back in the independent prediction case.

For the sake of clarity of presentation, we focus on pairwise MRF models for the rest of this chapter, as well as in Chapter 4.

2.3.3 *The MAP Inference Problem*

Given the potential functions of a factor graph, the Maximum-A-Posteriori (MAP) task is finding the assignment $\boldsymbol{y}^*$ with the lowest energy

$$\boldsymbol{y}^* = \operatorname*{argmin}_{\boldsymbol{y}} E(\boldsymbol{y}) = \operatorname*{argmin}_{\boldsymbol{y}} \sum_{c \in \mathcal{C}} \theta_c(\boldsymbol{y}_c). \tag{2.8}$$

Remark. The term Maximum-A-Posteriori (MAP) suggests that the task is inferring a posterior distribution, which might be confusing as the presentation here is not given in terms of distributions. In short the answer is that the MAP problem can truly be seen as maximizing a posterior distribution, if we view the potential functions as parameters of a Gibbs distribution, then

$$P(\boldsymbol{y}) = \frac{1}{Z} \exp(-E(\boldsymbol{y})) = \frac{1}{Z} \prod_{c \in \mathcal{C}} \exp(-\theta_c(\boldsymbol{y}_c))$$

where Z is a normalizing factor given by $\sum_{\boldsymbol{y} \in \mathcal{Y}} \exp(-E(\boldsymbol{y}))$. Thus

$$\boldsymbol{y}^* = \operatorname*{argmax}_{\boldsymbol{y}} P(\boldsymbol{y}) = \operatorname*{argmin}_{\boldsymbol{y}} \sum_{c \in \mathcal{C}} \theta_c(\boldsymbol{y}_c)$$

In the general case finding the MAP labeling for a given graph and potential functions without additional assumptions, was shown to be NP-hard (Cooper 1990; Roth 1996; Chandrasekaran, Srebro, and Harsha 2008).

A well studied scenario in which finding the MAP assignment of an MRF can be done efficiently, is when the graph structure is a tree. A graph is called a tree if between any pair of nodes there is only one path, where a path is defined as a sequence of the graph edges. Equivalently, a tree is a graph which does not contain cycles. The MAP problem on tree graphs can be solved exactly using the max-product algorithm (Kschischang, Frey, and Loeliger 2001). The max-product algorithm is a dynamic programming algorithm, which for

a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, requires only two iterations through the graph. The total running time is $\mathcal{O}(K^2|\mathcal{E}| + K|\mathcal{V}|)$, where $K = \max_i |\mathcal{Y}_i|$. This algorithm on general tree structured graphs, is also known as belief propagation (Pearl 1986), the Viterbi algorithm (Viterbi 1967) and the forward-backward algorithm (Rabiner 1989).

Other examples of probabilistic inference tasks which emerge in the context of learning and prediction with factor graphs, include finding the marginals or conditional probability of a configuration, or the normalizing factor Z. These questions are beyond the scope of this thesis.

There was much work done on approximating the MAP solution in the general case. In Chapter 4 we present a solution which is a combination of two relaxation approaches. A more detailed related work review is deferred to Chapter 4.

In Section 5.6 we show that a variant of the bi-clustering problem can be written as an energy minimization problem over a factor graph. In this factorization the optimal assignment of the individual high order factors can be found efficiently. We use this fact to derive a dual decomposition message passing algorithm for this problem.

2.3.4 *Conditional Random Field*

So far we assumed the potentials of a graphical model or factor graph to be given. While in some settings this might make sense, most often one would also like to estimate these potentials from actual observations, similar to learning the hyperplane in an SVM. There exist several approaches to do so, the structured SVM (Taskar, Guestrin, and Koller 2003; Tsochantaridis et al. 2005) and the Conditional Random Field (CRF) (Lafferty, McCallum, and Pereira 2001; Sutton and Mccallum 2012) are the most popular formulations. We use $\boldsymbol{x}$ to denote the input variable, or observation, $\boldsymbol{x}$ could for example be a noisy binary image that we wish to denoise, or a sentence in which we would like to label the different words according to their part-of-speech tags (e.g. noun, verb or article). Similarly as in the standard discriminative models discussed earlier, one would like to learn a classifier of the form

$$f_{\boldsymbol{w}}(\boldsymbol{x}) = \operatorname*{argmin}_{\boldsymbol{y} \in \mathcal{Y}} E(\boldsymbol{y}|\boldsymbol{x}, \boldsymbol{w}) = \operatorname*{argmin}_{\boldsymbol{y} \in \mathcal{Y}} -\langle \boldsymbol{w}, \boldsymbol{\phi}(\boldsymbol{x}, \boldsymbol{y}) \rangle.$$

Note, that the output variables $\boldsymbol{y}$ are now included in the feature map $\phi(\boldsymbol{x}, \boldsymbol{y})$. The feature map captures both, the graph structure and the exact dependence of the parameters on the states of $\boldsymbol{y}$. For fixed parameters and a given input variable the energy minimization above reduces to the standard MAP problem for given potentials discussed earlier. Here a linear parameterization of the potentials is assumed. This is convenient as for the estimation of $\boldsymbol{w}$ for observed input/output pairs $\{(\boldsymbol{x}_i, \boldsymbol{y}_i)\}_{i=1}^n$, generalizations of the logistic regression and the SVM are directly applicable, resulting in the CRF and structured SVM, respectively. For general graph structures, the problem of estimating the parameters of a structured model is NP-hard, as one needs to solve some form of energy minimization. For the special case of a tree structured graph the problem is however tractable.

In Chapter 7 we apply a linear chain Conditional Random Field (CRF) to predict the occupancy labels of a truck rest area based on sensor measurements.

2.4 BASIC CONVEX OPTIMIZATION DEFINITIONS

In this section we define few convex optimization related terms that appear a number of times in different chapters of this thesis. We make no attempt at presenting a general background in optimization and convex optimization, as this is not the focus of this work. Instead, in each chapter we introduce the relevant methods and motivate the choice of the optimization framework in the context of the specific learning problem. Classic books on convex optimization include (Bertsekas 1999; Boyd and Vandenberghe 2004; Borwein and Lewis 2005; Nocedal and Wright 2006). For a selection of machine learning specialized optimization topics we refer the reader to (Sra, Nowozin, and Wright 2011).

CONVEX SET A set $S \subseteq \mathbb{R}^d$ is a convex set if it contains the line segment joining any of its points, i.e. if $\forall \boldsymbol{x}, \boldsymbol{x}' \in S, \quad 0 \leq \lambda \leq 1$ then

$$\lambda \boldsymbol{x} + (1 - \lambda)\boldsymbol{x}' \in S.$$

POLYHEDRON, POLYTOPE A polyhedron is an example of a convex set, defined as the intersection of finite number of half-spaces.

$$\text{Polyhedron} := \{\boldsymbol{x} | \boldsymbol{a_i}^\top \boldsymbol{x} \leq b_i \quad \boldsymbol{a_i} \in \mathbb{R}^d,\ b_i \in \mathbb{R}\}_{i=1}^n$$

A bounded polyhedron is called a polytope. For example the simplex defined as:

$$\boldsymbol{x} \in \Delta^d := \left\{ \boldsymbol{x} \middle| \sum_{i=1}^{d} x_i = 1, \forall i : x_i \geqslant 0 \right\},$$

is a polytope.

CONVEX, CONCAVE FUNCTION A function $f : \mathbb{R}^d \to \mathbb{R}$ is convex if its domain S is convex and $\forall \boldsymbol{x}, \boldsymbol{x}' \in S, \quad 0 \leq \lambda \leq 1$

$$f(\lambda \boldsymbol{x} + (1-\lambda)\boldsymbol{x}') \leq \lambda f(\boldsymbol{x}) + (1-\lambda) f(\boldsymbol{x}')$$

f is concave if $-f$ is convex.

CONVEX OPTIMIZATION PROBLEM A convex optimization problem is one of the form

$$\begin{array}{ll} \text{minimize} & f_0(\boldsymbol{x}) \\ \text{subject to} & f_i(\boldsymbol{x}) \leq 0, \quad i = 1, \dots n \\ & \boldsymbol{a_j}^\top \boldsymbol{x} = b_j \quad j = 1, \dots m \end{array}$$

Here $\boldsymbol{a_j} \in \mathbb{R}^d$ and $b_j \in \mathbb{R}$. The objective and inequality constraints $f_i : \mathbb{R}^d \to \mathbb{R}$ are convex, and the equality constraints $\boldsymbol{a_j}^\top \boldsymbol{x} = b_j$ are affine.

LINEAR PROGRAM (LP) A LP is an optimization problem where the objective and all the constraints are affine.

$$\begin{array}{ll} \text{minimize} & \boldsymbol{c}^\top \boldsymbol{x} + r \\ \text{subject to} & \boldsymbol{g_i}^\top \boldsymbol{x} \leq h_i, \quad i = 1, \dots n \\ & \boldsymbol{a_j}^\top \boldsymbol{x} = b_j, \quad j = 1, \dots m \end{array}$$

Where $\boldsymbol{c}, \boldsymbol{g_i}, \boldsymbol{a_j} \in \mathbb{R}^d$ and $r, h_i, b_j \in \mathbb{R}$. LPs are convex problems.

QUADRATIC PROGRAM (QP) A QP is an optimization problem where the objective is quadratic and all the constraints are affine.

$$\begin{array}{lll} \text{minimize} & \frac{1}{2}\boldsymbol{x}^\top \boldsymbol{P}\boldsymbol{x} + \boldsymbol{q}^\top \boldsymbol{x} + r & \\ \text{subject to} & \boldsymbol{g_i}^\top \boldsymbol{x} \leq h_i, & i = 1, \dots n \\ & \boldsymbol{a_j}^\top \boldsymbol{x} = b_j, & j = 1, \dots m \end{array}$$

Where $\boldsymbol{q}, \boldsymbol{g_i}, \boldsymbol{a_j} \in \mathbb{R}^d$, $r, h_i, b_j \in \mathbb{R}$. In general $\boldsymbol{P} \subset \mathbb{R}^{d \times d}$ is a symmetric matrix. When $\boldsymbol{P}$ is positive semi-definite the QP is a convex problem.

3

ANALYTIC CENTER CUTTING PLANE METHOD FOR MULTIPLE KERNEL LEARNING

This chapter studies the Multiple Kernel Learning (MKL) problem, in the settings where the goal is to learn a convex combination of a given set of candidate kernels. As was mentioned in Subsection 2.2.1, the choice of the kernel is vital to the success of prediction using a SVM classifier. The MKL framework addresses the problem of choosing a suitable kernel by learning the optimal combination of several kernels. This allows the practitioner to suggest a set of possibly matching kernels rather than committing to a specific one.

The cutting planes approach is an alternative to interior point methods for solving convex problems. We formulate the MKL problem as a Semi-Infinite Linear Programming (SILP) in the kernel coefficients. The merit of this approach is that the generation of the planes (constraints), is done via an SVM call. We observe that many recent MKL solutions can be cast in the framework of oracle based optimization, and show that they vary in terms of query point generation. Motivated by the success of centering approaches in interior point methods, we propose the analytic center as the query point generator.

Finally we explore the settings in which MKL can be expected to improve over a straight-forward kernel combination solution. The results in this chapter are based on Wulff and Ong (2013).

3.1 INTRODUCTION

Kernel methods, are a class of algorithms that consider only the similarity between examples (Schölkopf and Smola 2001). A kernel function k implicitly maps examples $\boldsymbol{x}$ in some space, to a feature space given by a feature map Φ, via the identity $k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \langle \Phi(\boldsymbol{x}_i), \Phi(\boldsymbol{x}_j) \rangle$. It is often unclear which is the most suitable kernel for the task at hand, and hence the user may wish to try a combination of several kernels. Another scenario in which a combination of kernels is expedient, is when the data is multi-modal. For example consider a medical application, in which doctors may wish to derive a medical condition predictor

based on the patients blood test, scanned image of an extracted tissue, and the family history. Each of these information sources is likely to have an embedding in a different similarity space. One option is to construct a custom kernel which takes into account all of the sources. This is however a far more complicated approach than combining often well-studied, source-specific kernels.

One problem with simply adding kernels, is that using uniform weights is possibly not optimal. An extreme example is the case when one kernel is not correlated with the labels at all, in which case giving it positive weight just adds noise (Lanckriet et al. 2004a).

Multiple Kernel Learning (MKL) is a way of optimizing kernel weights while training an SVM. In addition to leading to good classification accuracies, MKL can also be useful for identifying relevant and meaningful features (Borgwardt et al. 2005; Lanckriet et al. 2004a; Sonnenburg, Rätsch, and Schäfer 2005; Ong and Zien 2008).

MKL finds a convex combination of kernels (Lanckriet et al. 2004b; Sonnenburg et al. 2006), that is a classifier of the form

$$f_{\mathbf{w},b,\boldsymbol{\beta}}(\boldsymbol{x},y) = \sum_{k=1}^{p} \beta_k \left\langle \mathbf{w}_k, \Phi_k(\boldsymbol{x}) \right\rangle + b$$

where $\boldsymbol{\beta}$ is a vector of length p with weights corresponding to the p candidate kernels, and $\mathbf{w}$, b are the SVM parameters. The MKL problem was first formulated as a Semidefinite Programming (SDP) problem (Lanckriet et al. 2004b). One can exploit the known structure of the MKL problem to speed up the optimization, for example using a Sequential Minimal Optimization (SMO) inspired approach (Bach, Lanckriet, and Jordan 2004). However, this requires a full reimplementation of the solver. MKL has also been shown to be equivalent to the group Least Absolute Shrinkage and Selection Operator (LASSO) (Bach 2008).

Recently, leveraging on the existence of efficient software for solving the SVM optimization problem, a Semi-Infinite Linear Programming (SILP) approach was developed in (Sonnenburg et al. 2006). Their solution is based on the concept of cutting planes. Cutting planes methods alternate between choosing query points and calling an oracle. Given a query point, the oracle returns halfspaces to be included in the current set of constraints, forming the feasible set. In the context of MKL, the oracle is an SVM solver and the query points are kernel weights. One disadvantage of the SILP is that it requires many cutting planes (and hence calls to the SVM) before convergence. This gave rise

to a sub-gradient based approach (Rakotomamonjy et al. 2008) and a bundle method (Xu et al. 2008).

Our contributions are as follows. We review the MKL problem in an oracle based optimization framework and demonstrate that they are essentially different ways of generating query points. We propose the use of the analytic center as a query point, with the intuition that this query point in expectation halves the hypothesis space. We conduct experiments comparing the performance of several MKL solutions on UCI data. We show that our more"regularized" approach often requires fewer iterations, and is more robust to variations in data. The experimental results are followed by a discussion of the correlation between the MKL objective and test accuracy.

3.2 ORACLE BASED METHODS

Oracle based methods are widely used for solving integer programming problems. Here we focus on using such methods for solving convex optimization problems (Goffin and Vial 2002). The goal of oracle based algorithms is to find a point in a convex set Z, or to determine that this set is empty (see Algorithm 3.1 for a pseudo-code description). In an optimization problem, Z is the set of ε-suboptimal points. The method does not assume any direct access to the description of Z, such as the objective and constraint functions, except through an oracle. Instead the method generates a query point q (see Subsection 3.2.2) which upon satisfying $q \notin Z$ is passed to the oracle (see Subsection 3.2.1). The oracle returns a hyperplane which separates q from the set Z. This hyperplane is called a "cut" since it eliminates a halfspace from our search, which is the reason oracle based methods are also known as cutting plane methods. Cutting planes are also sometimes called column generation methods since in the dual space, the cuts become columns in the constraint matrix.

3.2.1 *Oracles and Cuts*

As mentioned before, a particular optimization problem is defined via an oracle. For a convex optimization problem with m constraints,

$$\begin{array}{ll} \min_z & f_0(z) \\ \text{s.t.} & f_i(z) \leqslant 0, \qquad i = 1, \ldots, m \end{array} \tag{3.1}$$

where $f_0, \ldots, f_m$ are convex and differentiable, the target set Z is the optimal (or ε-optimal) set. Given a query point q, the oracle first check whether it is feasible. If q is not feasible, then we choose one of the violated constraints $f_j(q) > 0$ and form a cut

$$f_j(q) + \nabla f_j(q)^\top (z - q) \leqslant 0 \tag{3.2}$$

This cut (3.2) is called a *feasibility cut* for the problem (3.1) since it cuts away a halfspace of points known to be infeasible (they violate the jth constraint). If q is feasible, we construct the cutting plane

$$\nabla f_0(q)^\top (z - q) \leqslant 0, \tag{3.3}$$

which is called the *objective cut* for (3.1). This cuts out the halfspace

$$\{z \mid \nabla f_0(q)^\top (z - q) > 0\}$$

since all such points have an objective value larger than $f_0(q)$ and hence cannot be optimal. If q is feasible, and $\nabla f_0(q) = 0$ then q is optimal. In general, for non-differentiable problems, the gradients $\nabla f_j(z)$ can be replaced by sub-gradients.

Algorithm 3.1 Cutting plane algorithm for optimization

Require: an initial polyhedron $\mathcal{P}_0$ containing Z.
  $t = 0$
  repeat
    Generate a query point $q^{(t+1)}$ in $\mathcal{P}_t$
    Query the oracle at $q^{(t+1)}$,
    Oracle returns a cutting plane $a_{t+1}^\top z \leqslant b_{t+1}$.
    Update the constraints:
      $\mathcal{P}_{t+1} = \mathcal{P}_t \cap \{z | a_{t+1}^\top z \leqslant b_{t+1}\}$.
    $t = t + 1$
  until convergence or $\mathcal{P}_{t+1} = \emptyset$

3.2.2 *Generating Query Points*

In principle, we would like the query point $q^{(t+1)}$ corresponding to the current polyhedron $\mathcal{P}_t$ (containing the optimal set Z) to be generated such that the resulting cut reduces the size of $\mathcal{P}_{t+1}$ as much as possible.

When querying the oracle with $q^{(t+1)}$, we do not know in which direction the generated cut will exclude, but we do know that $q^{(t+1)}$ will be in the excluded halfspace. One approach is to greedily use the vertex of the current polytope which minimizes the objective, leading to the following method.

3.2.3 *Method of Kelley-Cheney-Goldstein*

For the optimization problem (3.1), the piecewise linear function

$$f^{(t)}(z) = \max_{i \leqslant t} f_0(z_i) + \nabla f_0(z_i)^\top (z - z_i)$$

is a lower approximation to $f_0(z)$. The next query point $q^{(t+1)}$ is found by solving

$$\begin{array}{ll} \min & \theta \\ \text{s.t.} & \theta \geqslant f_0(q_i) + \nabla f_0(q_i)^\top (z - q_i), \quad \forall i \leqslant t \\ & A_t z \leqslant b_t, \end{array} \tag{3.4}$$

where A_t, b_t are the set of existing cutting planes which define the current polyhedron $\mathcal{P}_t$. In the rest of this chapter we refer to the above method as the Kelley-Cheney-Goldstein (KCG).

Using $\frac{\nabla f_0(q_i)}{\|\nabla f_0(q_i)\|}$ instead of $\nabla f_0(q_i)$, in equation (3.4), results in finding the center of the largest sphere. This variant of equation (3.4) is called the Chebyshev center method. This modification, where the gradients are scaled to unit length, has significantly better convergence properties (Goffin and Vial 2002). This already shows the power of centering, which is exploited by the following method.

3.2.4 *Analytic Center Cutting Plane Method*

The analytic center is a concept which gained popularity due to interior point methods.

Given a constraint $a_i^\top z \leqslant b_i$, define the slack $s_i \in \mathbb{R}$ as $s_i = b_i - a_i^\top z$, that is, s_i is a measure of how close or how far the current solution is from the constraint. An interior point of the feasible set is a point for which all the slacks are strictly positive. The analytic center is defined as the unique maximizer of the function $f(s) = \prod_{i=1}^{t} s_i$ where $s \in \mathbb{R}^t$ is the

vector of slacks of the current set of constraints $\{a_i^\top z \leqslant b_i, i = 1, \ldots, t\}$. The geometrical interpretation of the analytic center is the point that maximizes the product of distances to all of the faces of the polytope.

Maximizing the product of the slacks (e.g. instead of sum), guarantees that every slack is strictly positive.We can rewrite the analytic center as

$$\operatorname*{argmax}_z f(s) = \operatorname*{argmax}_z \prod_{i=1}^{t} s_i = \operatorname*{argmax}_z \sum_{i=1}^{t} \log(b_i - a_i^\top z) \tag{3.5}$$

This function is also known as the logarithmic barrier, and its unique maximizer can be efficiently found using Newton iterations (Goffin and Vial 2002).

Going back to the cutting planes framework, computing a query point which is as far as possible from the border of the feasible set, intuitively results in a cut with a substantial reduction in the size of the set. The convergence of the Analytic Center Cutting Plane Method (ACCPM) in terms of number of oracle calls was theoretically analyzed in (Goffin and Vial 2002). The analysis relies on the properties of the logarithmic barrier. Other choices of centers, such as the center of gravity, center of the maximum volume ellipsoid and the volumetric center are possible. However, in this work we focus on the analytic center.

3.3 MULTIPLE KERNEL LEARNING

In this section, we briefly review MKL and derive the oracle function. We detail our approach of query point generation using the analytic center and then review recent MKL approaches in the framework of oracle based methods.

We start off by presenting a simple example in which a combination of kernels is outperforming prediction using the individual kernels.

3.3.1 *Kernel Combination Toy Example*

In general, constructing a motivating example for kernel combination with low (two, for the sake of visualization) dimensional data, is not an easy task. Since the complexity of a kernel based classifier is mostly due to the kernel function, many low dimensional datasets can be easily separated by a single suitable kernel. Furthermore, a single

Gaussian kernel is likely to perform very well on most "simple" datasets. The dataset should be such that it requires a combination of several kernels to achieve good separation, while each of the kernels yield lower accuracy when used separately.

One way to construct such a dataset is by sampling from a multimodal distribution, where each mode is separated well by one of the kernels. We use a mixture of linearly separated data, and a distribution where each class is sampled from a circle with some fixed radius and added noise. The dataset is depicted in Figure 3.1

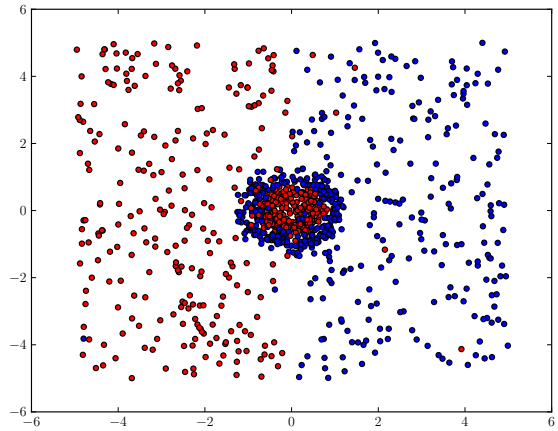
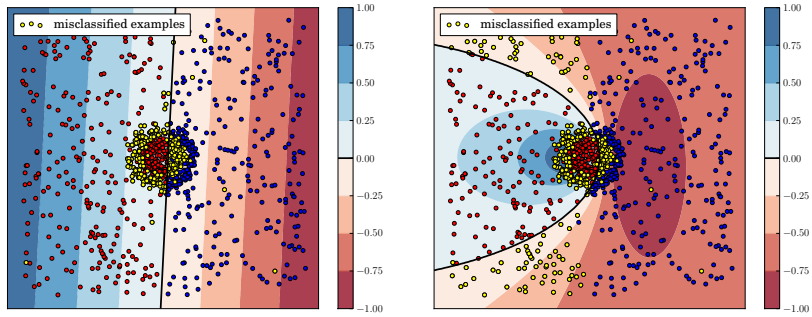


Figure 3.1: Two class dataset of 1000 examples sampled from a multimodal distribution. The colors represent the class labels

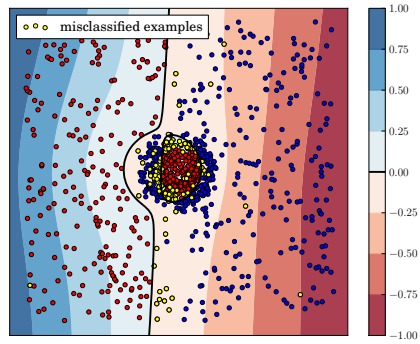
The kernels we use in the combination are a linear kernel and a polynomial kernel of degree 2. In Figure 3.2 the decision boundaries of the SVM classifiers with the different kernels are presented. The accuracy of the combined kernel is higher than any of the single kernel classifiers. As can be expected, the decision boundary of the kernel combination is capturing the class conditional distributions better than any of the single kernels.

We measure also the accuracy of the trained classifiers on a second test data, sampled from the same data distribution. As expected, the accuracy of the three classifiers is very similar on the test data. Setting non-uniform weights for the kernels in the combination, did not have much effect on the accuracy, the results were also very similar with varying values of the SVM hyper-parameter.



(a) Single linear kernel SVM. Accuracy on the training set 76.2%, on a test set 74.3%.

(b) Polynomial kernel with degree 2 SVM. Accuracy on the training set 68.3%, on a test set 68.7%.



(c) Combined kernel with uniform weights. Accuracy on the training set 88.7%, on a test set 88.3%.

Figure 3.2: Decision boundary of SVM classifiers on the training data in Figure 3.1, with single linear and polynomial kernel, vs. the combination with uniform weights.

3.3.2 *Review of Multiple Kernel Learning*

We follow the setting of finding a convex combination of kernels that performs well in a SVM binary classification task (Lanckriet et al. 2004b; Sonnenburg et al. 2006; Zien and Ong 2007), that is for a given training dataset $\{(\boldsymbol{x}_i, y_i)\}_{i=1,\ldots,n}$, find a classifier

$$f_{\mathbf{w},b,\boldsymbol{\beta}}(\boldsymbol{x}) = \sum_{k=1}^{p} \beta_k \langle \mathbf{w}_k, \Phi_k(\boldsymbol{x}) \rangle + b$$

where the kernel weights $\boldsymbol{\beta}$ and the SVM parameters $\mathbf{w}$, b are found by optimizing [1]

$$\begin{aligned} \min_{\boldsymbol{\beta},\mathbf{w},b,,} \quad & \frac{1}{2}\left(\sum_{k=1}^{p}\beta_k\|\mathbf{w}_k\|\right)^2 + C\sum_{i=1}^{n}\xi_i \\ \text{s.t. } \forall i: \quad & y_i\sum_{k=1}^{p}\beta_k\left\langle\mathbf{w}_k,\Phi_k(\boldsymbol{x})\right\rangle + b \geqslant 1-\xi_i \text{ and } \xi_i \geq 0. \end{aligned} \tag{3.6}$$

The indices i, j range over the data examples, $1 \leq i, j \leq n$. The p mixing coefficients $\boldsymbol{\beta}$ (indexed by k) should reflect the utility of the respective feature map for the classification task, and are normalized to be on the simplex, i.e.

$$\boldsymbol{\beta} \in \Delta^p := \left\{\boldsymbol{\beta}\,\middle|\,\sum_{k=1}^{p}\beta_k = 1, \forall k: \beta_k \geqslant 0\right\},$$

giving them the flavor of probabilities. This $\mathcal{L}_1$ regularizer on $\boldsymbol{\beta}$ promotes sparsity, and hence we are trying to select a subset of kernels (Bach 2008). We refer to equation (3.6) as the primal problem. The dual formulation of the above problem is given by (Lanckriet et al. 2004b; Rakotomamonjy et al. 2008) a Quadratically Constrained Quadratic Program (QCQP),

$$\begin{aligned} \min_{\boldsymbol{\alpha},\gamma} \quad & \gamma - \sum_i \alpha_i \\ \text{s.t.} \quad & \forall k: \gamma \geqslant \frac{1}{2}\left\|\mathbf{w}_k(\boldsymbol{\alpha})\right\|^2 \\ & \forall i: 0 \leqslant \alpha_i \leqslant C \\ & \textstyle\sum_i y_i\alpha_i = 0, \end{aligned} \tag{3.7}$$

where the margin term for the k'th kernel is given by

$$\left\|\mathbf{w}_k(\boldsymbol{\alpha})\right\|^2 = \sum_{i,j}\alpha_i\alpha_j y_i y_j k_k(\boldsymbol{x}_i,\boldsymbol{x}_j).$$

Following (Sonnenburg et al. 2006), we can move the sum of alphas into the constraints by changing $\gamma' = \gamma - \sum_i \alpha_i$, and then convert the

1 Equation (3.6) is not identical to the original formulation in (Lanckriet et al. 2004b; Sonnenburg et al. 2006), but has been shown to be equivalent (Zien and Ong 2007).

QCQP (3.7) by a second (partial, only w.r.t γ) dualization.The derivation w.r.t γ recovers the simplex constraint on the kernel coefficients $\boldsymbol{\beta}$.

$$\begin{array}{ll} \max\limits_{\boldsymbol{\beta}\in\Delta^p} \min\limits_{\boldsymbol{\alpha}} & \frac{1}{2}\sum\limits_k \beta_k \|\mathbf{w}_k(\boldsymbol{\alpha})\|^2 - \sum\limits_i \alpha_i \\ \text{s.t.} & \forall i : 0 \leqslant \alpha_i \leqslant C \\ & \sum_i y_i \alpha_i = 0. \end{array} \tag{3.8}$$

We further derive our solution based on equation (3.8).

Remark. [Sample complexity of MKL] Learning the kernel combination rather than committing to a fixed single kernel is advantageous in many scenarios, however this flexibility has a cost in terms of an increased sample complexity required to learn this hypothesis class. The estimation error (see Subsection 2.1.1) of a single kernel classifier with margin γ is with probability at least $1-\delta$ bounded by $O\left(\sqrt{(1/\gamma^2 - \log(\delta))/n}\right)$ where n is the number of examples (Koltchinskii and Panchenko 2002). The first bound for the MKL formulation with convex combination of weights, subject to an $\mathcal{L}_1$ norm constraint which we consider here, was given in (Lanckriet et al. 2004b). The true generalization error of a hypothesis f, is bounded by the γ-empirical error of f defined as: $\mathrm{Err}_S^\gamma(f) = \frac{1}{n}\sum\limits_{i=1}^n \mathbb{I}_{y_i f(\boldsymbol{x}_i)<\gamma}$, plus the estimation error. The bound in (Lanckriet et al. 2004b) has the following form $\mathrm{Err}_P(f) \leq \mathrm{Err}_S^\gamma(f) + O\left(\sqrt{(p/\gamma^2)/n}\right)$. which means that the error can scale multiplicatively with the number of kernels. If this bound would have been tight the MKL framework would have been limited to a small number of candidate kernels or alternatively large sample size scenarios. Later a bound in which the dependency on the number of kernels is additive was given in (Srebro and Ben-david 2006): $\tilde{O}\left(\sqrt{(p + R^2/\gamma^2)/n}\right)$. The $\tilde{O}$ notation hides logarithmic factors, and the $R^2 \in \mathbb{R}$ is an upper bound on the value of $K_k(\boldsymbol{x}, \boldsymbol{x}')$ for all $1 \leq k \leq p$ and $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{D}$. In (Cortes, Mohri, and Rostamizadeh 2010) a bound based on the Rademacher complexity of the hypothesis set was given, with a multiplicative dependency but only logarithmic on the number of kernels p: $O\left(\sqrt{((\log p)R^2/\gamma^2)/n}\right)$. In (Hussain and Shawe-Taylor 2011) an additive and logarithmic dependency on p was obtained. Their

bound which is also derived based on the Rademacher complexity is given by: $O\left(\sqrt{(\ln p + R^2/\gamma^2)/n}\right)$.

3.3.3 *MKL Solution In An Oracle Based Framework*

To remain consistent with equation (3.1) we change the objective sign, and define the following function of the kernel coefficients

$$\begin{aligned} g_0(\boldsymbol{\beta}) \quad = \quad & \max_{\boldsymbol{\alpha}} \sum_i \alpha_i - \frac{1}{2} \sum_k \beta_k \left\| \mathbf{w}_k(\boldsymbol{\alpha}) \right\|^2 \\ & \text{s.t.} \forall i : 0 \leqslant \alpha_i \leqslant C \\ & \textstyle\sum_i y_i \alpha_i = 0. \end{aligned} \tag{3.9}$$

Using the above definition, equation (3.8) becomes

$$\begin{aligned} \min_{\boldsymbol{\beta}} \quad & g_0(\boldsymbol{\beta}) \\ \text{s.t.} \quad & \boldsymbol{\beta} \in \Delta^p \end{aligned} \tag{3.10}$$

Observe that for a given $\boldsymbol{\beta}$, evaluating $g_0(\boldsymbol{\beta})$ amounts to an SVM call (Sonnenburg et al. 2006). That is, $g_0(\boldsymbol{\beta})$ is exactly the dual formulation of the SVM (with opposite sign), where the kernel is a linear combination of the p kernels weighted by the coefficients $\boldsymbol{\beta}$. This allows us to use an SVM solver as the oracle. The constraints on the dual variables $\boldsymbol{\alpha}$ in equation (3.8) are automatically satisfied.

3.3.4 *ACCPM for Multiple Kernel Learning*

Motivated by the properties of the analytic center discussed in Subsection 3.2.4, we propose to use it as the query point generation criteria.

Let $\boldsymbol{\beta}^l, \boldsymbol{\alpha}^l$ where $l = 1, \ldots, t-1$ denote the previous query points and the corresponding SVM solver solutions. At iteration t, the analytic center (equation (3.5)), is found by maximizing the following objective

$$\boldsymbol{\beta}^t = \operatorname*{argmax}_{\boldsymbol{\beta} \in \Delta^p} \sum_{l=1}^{t-1} \log \frac{1}{2} \left(\sum_k \beta_k^l \left\| \mathbf{w}_k(\boldsymbol{\alpha}^l) \right\|^2 - \sum_k \beta_k \left\| \mathbf{w}_k(\boldsymbol{\alpha}^l) \right\|^2 \right) \tag{3.11}$$

The oracle then solves the single kernel SVM problem with $\boldsymbol{\beta}^t$ as the weighting coefficients, to obtain $\boldsymbol{\alpha}^t$. Next, the oracle computes the gradient of $g_0(\boldsymbol{\beta})$, given by $-\frac{1}{2} \left\| \mathbf{w}_k(\boldsymbol{\alpha}) \right\|^2$ where we use $\boldsymbol{\alpha} = \boldsymbol{\alpha}^t$.

Note that since we restrict the generation of β's to $\beta \in \Delta^p$, the oracle only ever returns objective cuts (equation (3.3)), hence the new cut is given by

$$[\frac{1}{2}\left\|\mathbf{w}_k(\alpha^t)\right\|^2]^\top(\beta - \beta^t) \leqslant 0. \tag{3.12}$$

In the above equation, the first term is a vector of the same length as β with values $\frac{1}{2}\left\|\mathbf{w}_k(\alpha)\right\|^2$.

A pseudo-code description of the analytic center cutting planes algorithm for multiple kernel learning is given in algorithm Algorithm 3.2.

Algorithm 3.2 Analytic center cutting planes algorithm for multiple kernel learning

Require: A set of p kernels, labeled sample $\{(\mathbf{x}_i, y_i)\}$

initialize $\beta^0 = \frac{1}{p}\mathbf{1}$, $t = 0$.

repeat

Solve $\alpha^t = g_0(\beta^t)$, an SVM call with β^t as the kernel coefficients.

Compute the gradient as $-\frac{1}{2}\left\|\mathbf{w}_k(\alpha^t)\right\|^2$ for each kernel $k = 1, \ldots, p$ and the objective cut given in equation 3.12.

Solve β^{t+1} the analytic center from equation 3.11.

t = t+1

until Duality gap (subsubsection 3.3.4.2) is smaller than ϵ.

3.3.4.1 *Related Work*

In this section we review recent MKL work. We follow the order of presentation in (Goffin and Vial 2002), Section 3.2. At iteration t, the lower approximation of g_0 (equation (3.9)), calculated by the KCG method (Subsection 3.2.3) is

$$g_t(\beta) = \max_{l \leq t} g_0(\beta^l) + \nabla g_0(\beta^l)^T(\beta - \beta^l) \tag{3.13}$$

where β^l is the vector of kernel coefficients chosen in the lth iteration. Let α^l denote the maximum over α corresponding to β^l i.e.

$$\alpha^l := \operatorname*{argmax}_{\alpha} \sum_i \alpha_i - \frac{1}{2}\sum_k \beta_k^l \left\|\mathbf{w}_k(\alpha)\right\|^2$$

Using the above definitions, one can verify that $g_t(\boldsymbol{\beta})$ can be rewritten as

$$g_t(\boldsymbol{\beta}) = \max_{l \leq t} \sum_i \alpha_i^l - \frac{1}{2} \sum_k \beta_k \left\| \mathbf{w}_k(\boldsymbol{\alpha}^l) \right\|^2 \tag{3.14}$$

Therefore, the KCG query points are found by optimizing

$$\boldsymbol{\beta}^{t+1} = \operatorname*{argmin}_{\boldsymbol{\beta} \in \Delta^p} g_t(\boldsymbol{\beta}) \tag{3.15}$$

The KCG method was applied to MKL in (Sonnenburg et al. 2006; Ong and Zien 2008) where equation (3.10) was further transformed into a SILP.

A sub-gradient based approach which use the gradient with respect to the kernel weights was suggested in (Rakotomamonjy et al. 2008) (simpleMKL algorithm). One difficulty with sub-gradient methods is determining the optimal step size to be taken in the direction of the sub-gradient. They use a one dimensional line search, which involves several calls to the SVM solver. The sub-gradient method is memoryless, it does not utilize the gradient computed in previous iterations. Since previous search directions could be useful in finding a new search direction, a bundle method which projects the SILP solution to the level set was proposed (Xu et al. 2008). The extended level set bundle method has been shown to converge faster than sub-gradient.

The level-set method chooses the projection of the previous query point into a level set that is a weighted combination of the piece-wise linear lower approximation of $g_t(\boldsymbol{\beta})$ and the tightest upper bound discovered so far.

$$\begin{aligned} \boldsymbol{\beta}^{t+1} = \quad & \operatorname*{argmin}_{\boldsymbol{\beta} \in \Delta^p} \{ \left\| \boldsymbol{\beta} - \boldsymbol{\beta}^t \right\|^2 : g_t(\boldsymbol{\beta}) \leq L_t \} \\ & \text{with } L_t = \lambda \overline{f^t} + (1 - \lambda) \underline{f^t} \end{aligned} \tag{3.16}$$

where

$\overline{f^t} := \min_{l \leq t} g_0(\boldsymbol{\beta}^l)$ is the best upper bound discovered so far.
$\underline{f^t} := \operatorname*{argmin}_{\boldsymbol{\beta} \in \Delta^p} g_t(\boldsymbol{\beta})$ is the point chosen by KCG.

3.3.4.2 *Duality Gap*

The convergence of the duality gap is a natural stopping criteria for convex optimization. Recall the primal MKL formulation in equation (3.6),

for a specific value of the coefficients vector β and the corresponding dual variables α^l, the primal objective is given by

$$\frac{1}{2}(\sum_{k=1}^{p}\beta_k^l\sum_{i,j}\|\mathbf{w}_k(\alpha)\|^2)+C\sum_{i=1}^{n}\xi_i \tag{3.17}$$

The slack variables ξ_i can be retrieved from

$$\xi_i=\max(0,1-y_i(\sum_{k=1}^{p}\beta_k\sum_j\alpha_j k_k(\mathbf{x}_i,\mathbf{x}_j)))$$

The dual objective is given in equation (3.7)

$$\max_k\|\mathbf{w}_k(\alpha)\|^2-\sum_i\alpha_i^l \tag{3.18}$$

Hence the duality gap is equal to: (3.17) - (3.18).

3.3.5 *Implementation Issues*

One of the advantages of casting MKL as an oracle based method, lies in the resulting modular solution structure of an oracle-SVM component and a query point generator. Our implementation is a framework written in Python, having interfaces defining an oracle and a query point generator, such that the actual implementation can be easily replaced.

We used shogun[2] as the oracle-SVM solver and for computing the kernels. In this chapter we focus on ACCPM in the settings of binary classification. In principle, since we use shogun as the SVM solver, we would easily be able to solve other learning tasks supported by shogun. We used the software package OBOE[3] for finding the analytic center. The query points that OBOE selects are minimizers of equation (3.5) plus a proximal term. The optimization of the logarithmic barrier function is done using an infeasible start Newton method. More details of the implementation can be found in (Babonneau et al. 2007). We used SWIG[4] for creating a Python interface to OBOE (which is written in C).

For comparison purposes we implemented the KCG query point generation in our framework. We used mosek's[5] python interface to solve the LP.

2 http://www.shogun-toolbox.org
3 https://projects.coin-or.org/OBOE
4 http://www.swig.org
5 http://www.mosek.com/

3.4 COMPUTATIONAL RESULTS

The usefulness of the multiple kernel learning approach has been demonstrated in various learning applications and as a model selection framework. In this section we benchmark recent MKL implementations using some UCI[6] datasets and compare their performance. We then empirically test some more general properties of MKL optimization.

The benchmarked MKL methods are the simpleMKL (Rakotomamonjy et al. 2008), the extended level set method (Xu et al. 2008) (level-set) and our oracle based software with two different query point generators. The analytic center (ACCPM) and our KCG method implementations (KCG). The SILP (Sonnenburg et al. 2006) algorithm is another MKL solution based on the KCG method, but we do not compare directly with this implementation.

As a point of reference for a non MKL approach, we compare with a single kernel SVM classifier. The kernel we use is the averaged sum of the candidate MKL kernels, equivalent to uniform β weights (average).

3.4.1 *Experiments Overview*

We conduct our experiments on UCI repository datasets and consider three different aspects of the solutions.

1. *Performance analysis* We compare the performance of the different implementations. We measure the number of SVM solver calls, the accuracy of the solution, the actual running-time of computing a single MKL solution and the number of kernels which are assigned non-zero weights. A similar experiment is described in (Rakotomamonjy et al. 2008), where the performance of the SILP and the simpleMKL implementations are compared. The experiment was conducted again by (Xu et al. 2008), where the extended level set method was introduced. To be consistent with these experiments, we follow the same general settings and test the performance on the same UCI datasets.

2. *Duality-gap convergence* All of the MKL methods we consider incorporate an iterative procedure while seeking the optimal solution. In every iteration a new kernel coefficients vector is computed.

6 http://archive.ics.uci.edu/ml/datasets.html

For each such intermediate solution we store the corresponding duality gap and compare the algorithm's convergence behavior.

3. *Accuracy vs. duality gap* We look at the relation between the accuracy and the duality gap at points gathered up during the run of the algorithms. While achieving high accuracy is a measure of generalization ability and is desirable in every supervised learning task, the duality gap is the mathematical objective which is actually being optimized. To our best knowledge the connection between the two measures has never been studied before.

3.4.2 *Experimental Settings*

We ran the five methods on eight UCI repository datasets: wbpc, ionosphere, sonar, bupa, pima, vote, heart and wdpc (the first five were used in (Rakotomamonjy et al. 2008) and the rest added in (Xu et al. 2008)). Similarly to (Rakotomamonjy et al. 2008) the candidate kernels were: Gaussian kernels with 10 different width values, and polynomial kernels with degrees $\{1, 2, 3\}$. All kernels were computed with respect to all features and to every single feature separately. Thus the total number of kernels per dataset is $13(d + 1)$, where d is the number of features. We determined the widths of the Gaussian kernels by taking the Euclidean distances between the training data points. We sorted the pairwise distances in an increasing order and selected 10 kernel widths which are uniformly spaced on a log scale between the 10% and 90% quantiles of this range. These values reflect the data spread and are therefore potentially good candidate kernels (note that the Gaussian kernel widths are different from (Rakotomamonjy et al. 2008), where the widths are fixed for all datasets).

All kernel matrices were normalized to have unit trace and are computed prior to the run of the algorithm. The C hyper-parameter was set to 100, this value is the same as in (Rakotomamonjy et al. 2008) and (Xu et al. 2008). The optimality of this value in terms of solution accuracy was verified by means of cross-validation. As a starting point all of the candidate kernels were assigned equal weights. The training examples were normalized to zero mean and unit variance. The evaluation technique we used is 20 random (70%,30%) splits of the data.

The stopping criteria we used is the convergence of the duality gap below a threshold of 5e-03, or the number of iterations exceeds 500. The

duality gap stopping criteria value was the lowest value for which the level-set and simpleMKL runs on all dataset permutations converged (within a reasonable time). The other two methods, ACCPM and KCG, converged even when the stopping criteria was set to a much lower value. This is visible in Figure 3.3 where the curves of the duality gap along the iterations of the algorithms are presented. A slightly higher threshold value was used in (Rakotomamonjy et al. 2008) and (Xu et al. 2008).

In the performance comparison experiment, we report the accuracy, run-time, the number of SVM solver calls and the number of chosen kernels. The accuracy is the average percentage of correct predictions made on the 20 test-sets. The running time is the average running time in seconds until convergence, without the accumulated time of the SVM solver. We subtract the SVM solver time in the comparison since the implementations use different SVM solvers. In fact, since the datasets used here are rather small, the SVM solver run-time for most of the implementation is negligible. Often MKL is used in applications where the datasets are much larger and the SVM solver time becomes the dominant run-time component. We therefore compare the average number of SVM calls used by the algorithms to reach a single solution. The reported number of kernels is the number of kernels assigned non-vanishing weights in the end of the run. These are also the kernels that take part in prediction.

All of the implementations were run on an Intel(R) Core(TM)2 2.83GHz computer running Redhat Linux.

3.4.3 *Results*

3.4.3.1 *Performance Analysis*

Table 3.1 shows the performance results obtained by the different implementations. For each dataset, n and p, denote the number of examples and number of kernels respectively.

In term of number of SVM solver calls, ACCPM achieves significantly better results on most datasets. On 6 out of 8 datasets ACCPM uses the lowest number of SVM calls, on the remaining two datasets ACCPM is either competitive or second to level-set. The deviations in the number of solver calls shows that ACCPM is more robust with respect to different

Algorithm	# SVM calls	Time (s)	# Kernels	# SVM calls	Time (s)	# Kernels
	wpbc, $n = 138$, $p = 442$			pima, $n = 537$, $p = 117$		
accpm	55.1 ± 4.1	8.4 ± 0.8	12.9 ± 2.2	28.6 ± 2.6	6.1 ± 0.6	12.7 ± 3.2
level-set	153.6 ± 25.4	40.4 ± 11.7	13.9 ± 1.7	58.6 ± 8.6	27.5 ± 4.5	11.7 ± 1.5
simpleMKL	788.2 ± 201.3	17.4 ± 1.3	13.5 ± 2.1	466.9 ± 128.8	49.8 ± 7.6	11.2 ± 1.9
kcg	109.1 ± 25.7	13.9 ± 5.8	10.3 ± 1.7	66.2 ± 14.6	13.5 ± 3.1	8.2 ± 1.1
average	1.0	0.04 ± 0.0	442	1.0	0.22 ± 0.0	117
	ionosphere, $n = 245$, $p = 455$			vote, $n = 304$, $p = 221$		
accpm	100.4 ± 9.4	58.5 ± 17.0	15.7 ± 2.6	34.75 ± 2.49	2.7 ± 0.4	14.7 ± 5
level-set	73.3 ± 31.9	25.9 ± 18.5	18.4 ± 1.8	115.3 ± 84.3	51.2 ± 92.4	10 ± 2.5
simpleMKL	1273 ± 420.6	56.0 ± 5.9	19 ± 2.3	4792 ± 7708	87.2 ± 111.7	9.2 ± 3.2
kcg	255.5 ± 54.7	108.6 ± 83.1	14.8 ± 2.4	46.6 ± 12.4	2.9 ± 0.9	5.2 ± 2.1
average	1.0	0.14 ± 0.0	455	1.0	0.06 ± 0.0	221
	sonar, $n = 145$, $p = 793$			heart, $n = 189$, $p = 182$		
accpm	133.4 ± 14.3	180.1 ± 68.6	21.7 ± 1.7	46.2 ± 4.7	3.1 ± 0.5	11.3 ± 1.3
level-set	127.5 ± 93	55.2 ± 50.5	29.2 ± 1.9	81.3 ± 34.7	8.8 ± 6.6	14.7 ± 1.6
simpleMKL	4464 ± 2211	81.2 ± 15.4	21.9 ± 2	650.2 ± 436.1	7.4 ± 2.2	13.7 ± 1.5
kcg	450.4 ± 48.3	602.7 ± 222.4	21.2 ± 2.2	116.5 ± 29.6	10.2 ± 4.7	10.5 ± 1.6
average	1.0	0.14 ± 0.0	793	1.0	0.04 ± 0.0	182
	bupa, $n = 241$, $p = 91$			wdbc, $n = 398$, $p = 403$		
accpm	25.2 ± 2.8	1.4 ± 0.1	6.6 ± 1.7	72.4 ± 4.1	24.4 ± 3.7	13.3 ± 1.4
level-set	123.9 ± 19.4	13.8 ± 3.5	7.7 ± 1.3	115.7 ± 38.1	94.9 ± 38.8	9.8 ± 1.5
simpleMKL	332.7 ± 142.6	4.3 ± 1.4	7.6 ± 1.8	1843.8 ± 1572.6	178.2 ± 34	10.5 ± 1.3
kcg	33 ± 7.2	2.1 ± 0.4	5.7 ± 0.8	119.6 ± 14.3	28.8 ± 4.2	10.4 ± 1.1
average	1.0	0.05 ± 0.0	91	1.0	0.23 ± 0.01	403

Table 3.1: Comparison of the MKL solutions ACCPM, level-set (Xu et al. 2008), simpleMKL (Rakotomamonjy et al. 2008) and KCG and the non-MKL, average, on UCI datasets using random data splits. Time is the average running time in seconds, number of SVM calls is the average number of calls made during one run and the number of kernels is the number of kernels assigned non-vanishing weights in the end of the run.

data splits which is the cause of variation in the results of the other methods. A very likely explanation is the centering approach.

We observe that the number of SVM solver calls made by simpleMKL is significantly higher compared to the other methods. This can be attributed to the fact that simpleMKL performs a line search, which involves several calls to the SVM solver during the computation of a single new solution (update of the weights). In all other implementations there is a one-to-one correspondence between an SVM call and kernel coefficients update. The SVM calls invoked during the line search can become cheap using warm start. For this reason as well as the fact

Dataset	MKL accuracy(%)	"average" accuracy(%)
wpbc	79.4 ± 6.4	78.5 ± 5.5
ionosphere	93.1 ± 1.9	82.9 ± 2.9
sonar	77.8 ± 5.5	68.8 ± 5.2
bupa	66.7 ± 3.4	58.8 ± 2.7
pima	76.1 ± 2.4	75.0 ± 2.5
vote	95.6 ± 1.8	94.2 ± 2.1
heart	83.6 ± 3.4	84.3 ± 3.1
wdbc	96.6 ± 1.2	93.9 ± 1.6

Table 3.2: Accuracy results of the MKL methods vs. the average kernel. The numbers below represent the accuracy of ACCPM, however level-set, simpleMKL and KCG method, achieve the same accuracy within the error range.

	wpbc	ionosphere	sonar	bupa
# Iterations	35.35 ± 17.09	68.4 ± 19.4	187.25 ± 87.38	25.55 ± 12.31
	pima	vote	heart	wdbc
# Iterations	39.1 ± 12.6	162.45 ± 196.39	40.35 ± 22.3	79.3 ± 55.37

Table 3.3: number of sub-gradient iterations used by simpleMKL

that the SVM implementations are different, we report the running time without the SVM computations factor.

Comparing the running time without the SVM computation time, ACCPM is the fastest on 5 out of 8 datasets and is as fast as the fastest solution, KCG, on an additional one. On the remaining two datasets level-set is the fastest algorithm. We will revisit this fact later when considering the convergence behavior. The KCG method assigns less kernels with non-vanishing coefficients on most datasets.

The accuracy results are presented in Table Table 3.2. All of the MKL solutions achieved the same accuracy within the error range, hence the table only shows results of ACCPM and the non-MKL solution"average".

Considering the accuracy results, it is clear that for some datasets e.g. wbpc, pima and heart, learning the kernel weights has no advantage over simply adding the candidate kernels. In heart dataset the MKL approach is even slightly worse than the averaging one (although still within the error range). We further discuss this point when comparing the duality gap and accuracy.

3.4.3.2 *Duality Gap Convergence*

We analyze the convergence behavior of the various methods by comparing the average duality gap at each iteration. Figure 3.3 shows the curves of changes in duality gap along the iterations.

The level-set method as well as simpleMKL, exhibit a sharp reduction in the duality gap in the beginning of the iterative procedure. However in most datasets after some iterations the duality gap remains almost at the same level or slightly increases, before it converges. It can be seen that a lower convergence threshold value than the one used here, will not be reached on some of data splits.

ACCPM on the other hand convergences in a very steady manner. Again, this can be a result of choosing the center of the set, where in each iteration, the size of the remaining feasible set is expected to be reduced by roughly half. However, the point in which the two strategies cross, is sometimes in a region of already sufficiently small duality gap, resulting in an advantage to a less regularized approach. For example in the curve depicting the run on sonar.

3.4.3.3 *Accuracy vs. Duality Gap*

The goal of this experiment is to empirically answer the question, to what extent does optimizing the MKL objective function, leads to good generalization. For each intermediate solution we computed the accuracy and the duality gap. Plots of the resulting accuracy-duality gap pairs of ionosphere and sonar are shown in Figure 3.4. We choose to present these two datasets as they demonstrate different correlation trends. We also computed the average Pearson correlation scores, over 20 different data splits. The mean correlation values are shown in Table 3.4. The results differ between the methods as the intermediate solutions are different.

The results indicate that for some datasets there is no corresponding improvement in test accuracy for a reduction in duality gap. From the ACCPM column of Table 3.4, heart has the least (absolute) correlation, and indeed according to Table 3.2, the average kernel performs better on heart dataset. For the other two datasets on which MKL yields a rather low generalization improvement (wpbc and pima), we measure small absolute correlations. It is not the whole story, as on sonar dataset, with a slightly higher correlation, MKL does improve over an average kernel.

The hope is that we can a priori (before running MKL) identify datasets where MKL does not improve accuracy. Further, we may wish to change the MKL formulation such that the objective function is a better guide to test accuracy (generalization).

Table 3.4: Mean Pearson correlation values between duality gap and accuracy, on various UCI data sets

Data set	accpm	levelset	simpleMKL	kcg
wpbc	-0.25 ± 0.5	-0.3 ± 0.3	-0.24 ± 0.3	-0.33 $\pm$ 0.3
iono	-0.84 ± 0.0	-0.53 ± 0.1	-0.74 ± 0.1	-0.7 ± 0.0
sonar	-0.3 ± 0.3	0.1 ± 0.3	-0.32 ± 0.2	-0.49 ± 0.1
bupa	-0.74 ± 0.2	-0.57 ± 0.2	-0.62 ± 0.2	-0.63 ± 0.1
pima	-0.25 ± 0.5	-0.17 ± 0.5	-0.91 ± 0.1	-0.82 ± 0.2
vote	-0.56 ± 0.4	-0.01 ± 0.1	-0.79 ± 0.1	-0.78 ± 0.2
heart	-0.05 ± 0.4	0.03 ± 0.4	-0.69 ± 0.4	-0.76 ± 0.1
wdbc	-0.75 ± 0.2	-0.43 ± 0.2	-0.58 ± 0.1	-0.89 ± 0.1

3.4.4 *Relevance of Performance Measures*

An SVM solver call is a fundamental step in the run of all of the benchmarked MKL methods. The number of solver call is an important evaluation measure since this would be the dominant computational cost when the size of the training set is larger than the number of kernels in the linear combination, which is often the case in real applications. This is simply due to the fact that SVM is quadratic in the number of examples, whereas the query point generation procedure scales with the number of kernels. Hence one may use a more expensive approach to choose the query points without adversely affecting the total computational cost.

The running time can be useful in identifying trends in the run-time behavior of the algorithms. In our experiments however, it can be misleading since the sizes of the datasets are small which distorts the cost proportions and the compared methods use different SVM solvers. To compensate for this we report the running time without including the time taken by the SVM.

In this work we compare our method with several MKL solutions, which are considered "state of the art" in this rapidly progressing field. Two approaches can be taken when conducting such a comparison. One is reimplementing all the algorithms in the same framework, and the other is reusing published code. Each approach has its pros and cons in terms of providing the fairest comparison, this is not an easy trade-off. We chose to use the original software of SimpleMKL and level-set as we believe it will be difficult to obtain an equally optimized code and wish to avoid using a poor implementation. The main disadvantage of this approach is that comparing the running-time is less meaningful.

3.5 CONCLUSIONS

In general, oracle based methods can be used to solve cone programming problems which covers a large class of machine learning tasks. From an implementation viewpoint, what is required is a decomposition of the problem such that it is easy to implement the oracle or there is already an existing implementation.

In this chapter we have shown the benefit of choosing a central point when using an alternating optimization method for multiple kernel learning. The experiments demonstrate that our more "regularized" approach often requires fewer iterations, and is more robust to variations in data. Further, empirically it has a smooth convergence curve, in contrast to previous methods. With the availability of software for computing both the oracle (shogun) and the analytic center (OBOE), we also demonstrate the synergies of software reuse and open source software.

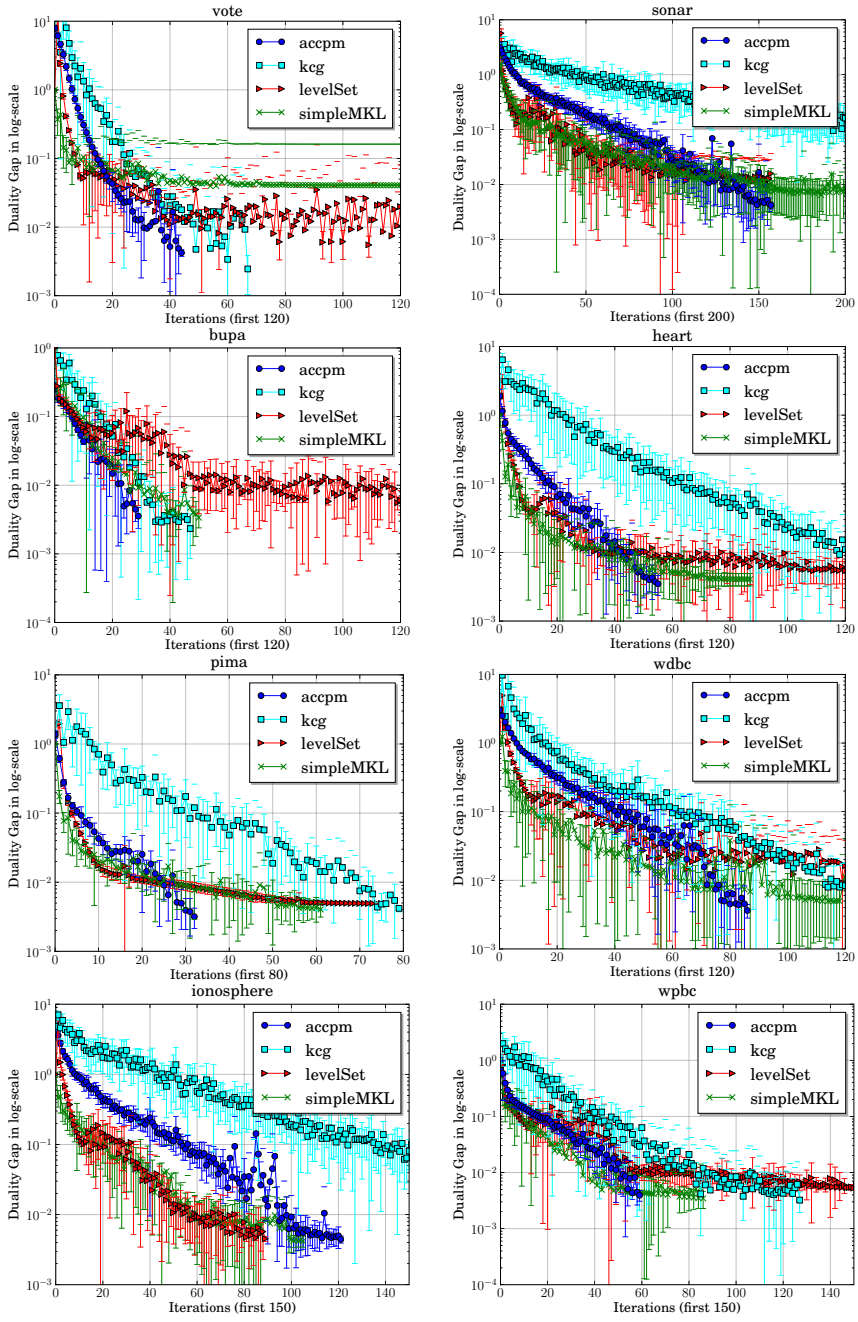


Figure 3.3: Duality gap convergence along the iterations for various datasets. The points (bars) are computed as the average (and standard deviation) value over the 20 data splits of the duality gap at a particular iteration.

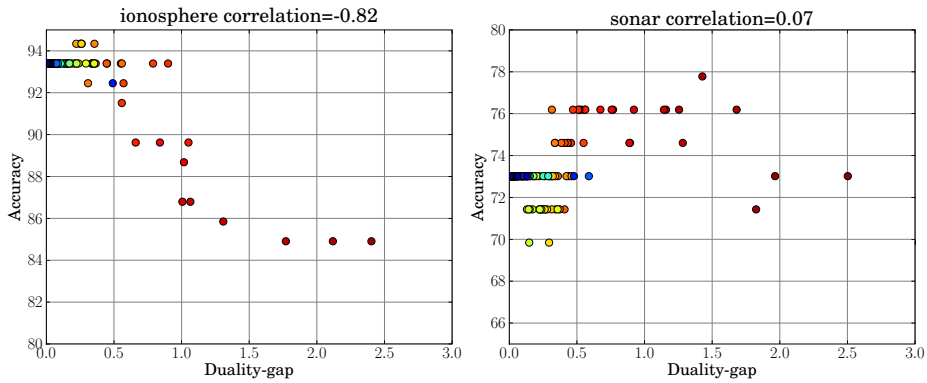


Figure 3.4: Duality-gap vs. accuracy as generated by one run of ACCPM on ionosphere (left side) and sonar (right side)

4 LPQP FOR MAP INFERENCE

In this chapter we develop a solution for the Maximum-A-Posteriori (MAP) problem, introduced and motivated in Subsection 2.3.3. One of the more popular and well-studied approaches for solving the MAP is based on a Linear Program (LP) relaxation. A different approach which was recently suggested for MAP is the Quadratic Program (QP) relaxation. Our work aims at combining these two approaches. We propose a novel MAP relaxation that penalizes the Kullback-Leibler (KL) divergence between the LP auxiliary variables, and the QP equivalent terms. The difference stems from inconsistencies in the LP formulation, which on one hand in some cases lead to poor solutions due to the misspecification of the objective, but at the same time, being a convex formulation, allow for exact solutions. By controlling the parameter that governs the penalty, we explore the range between the convex and non-convex yet more concise MAP objective.

We develop two efficient algorithms based on variants of this relaxation. The algorithms minimize the non-convex objective using belief propagation and dual decomposition as building blocks. Experiments on synthetic and real-world data show that the solutions returned by our algorithms substantially improve over the LP relaxation.

The results in this chapter are described in Pletscher and Wulff (2012).

4.1 INTRODUCTION

We study the problem of Maximum-A-Posteriori (MAP) inference in graphical models. The MAP task is to compute a minimal energy assignment of a set of dependent variables. This task arises in the context of graphical models, as explained in Subsection 2.3.3, but it can be understood in a more general context, as finding the most probable assignment problem. In the general case, MAP inference is intractable, and therefore most of the current research efforts are concentrated on finding efficient and accurate approximation algorithms. In recent years, Linear Program (LP) relaxations gained popularity due to their proven success in relevant applications. Several efficient algorithms have been

developed to solve the LP emerging from the relaxation. Despite their success, in many practical problems the solution attained by the LP relaxation is still far from the global minima.

Our work improves over the LP relaxation by leveraging on a second class of relaxations, namely the Quadratic Program (QP) relaxation. The QP formulation offers a concise and compact description of the MAP problem. We formulate a joint LP and QP MAP objective, that encourages auxiliary variables present in the LP relaxation, to agree with their counterpart in the QP relaxation, through a penalty function. Despite the non-convexity of this objective, we show that by slowly increasing the weight of the penalty, the solutions found are either competitive with, or in most cases better than the LP relaxation solutions. This is in general not the case for the few existing QP relaxation solvers.

We propose two variants of the penalty function, each leading to a different non-convex Linear and Quadratic Program relaxation (LPQP) objective. We show that the two objectives can be decomposed into a difference of convex functions, and solved in the framework of Concave-Convex Procedure (CCCP), described in Subsection 4.4.1. We are still left with two convex sub-problems that constitute the CCCP solution. One of which can be solved with the norm-product belief propagation algorithm, and for solving the other one we use the dual decomposition method.

Interestingly, the main computational task of both of the resulting LPQP algorithms, turns out to be similar to known entropy-augmented LPs. The merit of the an additional entropy term is the smoothing of the non-smooth LP objective thus gaining faster convergence rate (see Section 2.4). In this respect the LPQP algorithms have an edge, as the parameter controlling the augmented term is increased rather than decreased during the run.

Our contributions are as follows: First we introduce a combined LPQP objective, incorporating the QP constraints through a soft penalty function in the objective. We propose two alternatives for the penalty function, which differ in the way the edges in the graph are weighted. Secondly, we derive CCCP based algorithms for the LPQP objectives, and show that their core computational effort reduces to current entropy-augmented LP solvers. This demonstrates that these modern LP solvers can in some cases be utilized in a better way, leading to possibly faster convergence, as well as lower energy MAP solutions. Through experiments on various datasets, we demonstrate the performance of

the suggested LPQP MAP inference in comparison to other commonly used solvers.

4.2 BACKGROUND AND NOTATION

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ represent an undirected graph, with pairwise interactions between the variables (see Section 2.3) . Let y_i denote a discrete variable with a finite domain $\mathcal{Y}_i$[1], representing the assignment of the i-th node. The MAP problem is the following

$$\min_{\boldsymbol{y}} \sum_{i \in \mathcal{V}} \theta_i(y_i) + \sum_{(i,j) \in \mathcal{E}} \theta_{ij}(y_i, y_j). \tag{4.1}$$

Where $\theta_i(y_i)$ and $\theta_{ij}(y_i, y_j)$ are unary and pairwise potential functions associated with the node and edge assignments. Throughout this chapter we assume a given and fixed potential unary and pairwise functions.

Problem (4.1) can be expressed as an integer quadratic program using a vector notation:

$$\begin{aligned} \min_{\boldsymbol{\mu}} \quad & \sum_{i \in \mathcal{V}} \boldsymbol{\theta}_i^\top \boldsymbol{\mu}_i + \sum_{(i,j) \in \mathcal{E}} \boldsymbol{\mu}_i^\top \boldsymbol{\Theta}_{ij} \boldsymbol{\mu}_j \\ \text{s.t.} \quad & \mu_{i;k} \in \{0,1\} \quad \forall i,k \quad \text{and} \quad \sum_k \mu_{i;k} = 1 \quad \forall i. \end{aligned} \tag{4.2}$$

The pairwise and unary potentials in (4.2), are represented as a matrix $\boldsymbol{\Theta}_{ij}$ and a vector $\boldsymbol{\theta}_i$, respectively.

Variational approaches to MAP inference reformulate the combinatorial optimization problem in (4.1) as a continuous optimization problem. The next sections formally define two such approaches, namely the LP and QP relaxations. In general, the LP minimization results in a lower bound on the energy of the global minimizer, while the QP results in an upper bound.

1 For notational convenience we assume where $\mathcal{Y}_i = \{1, \ldots, K\}$, in the experiments we will however also consider settings where the domain of the variables has different size.

4.2.1 *Linear Programming Relaxation*

The LP approach (Schlesinger 1976; Wainwright and Jordan 2008) is based on a convex relaxation of (4.2), where an additional variable $\boldsymbol{\mu}_{ij}$ is included for each edge. The LP is given by

$$\min_{\boldsymbol{\mu} \in \mathcal{L}_\mathcal{G}} \sum_{i \in \mathcal{V}} \boldsymbol{\theta}_i^\mathsf{T} \boldsymbol{\mu}_i + \sum_{(i,j) \in \mathcal{E}} \boldsymbol{\theta}_{ij}^\mathsf{T} \boldsymbol{\mu}_{ij}, \tag{4.3}$$

Where $\mathcal{L}_\mathcal{G}$ is a set of summation constraints constraints which ensure consistency between the unary and pairwise variables.

$$\mathcal{L}_\mathcal{G} = \left\{ \boldsymbol{\mu} \;\middle|\; \begin{array}{ll} \sum_k \mu_{i;k} = 1 & \forall i \in \mathcal{V} \\ \sum_l \mu_{ij;kl} = \mu_{i;k} & \forall k, (i,j) \in \mathcal{E} \\ \sum_k \mu_{ij;kl} = \mu_{j;l} & \forall l, (i,j) \in \mathcal{E} \\ \mu_{ij;kl} \geq 0 & \forall k, l, (i,j) \in \mathcal{E} \end{array} \right\}.$$

To fully understand the implications of using the local marginal polytope as the constraint set, we need to describe the notion of a *marginal polytope* (Wainwright and Jordan 2003).

The marginal polytope, denoted by $\mathcal{M}$, is defined as the marginals of the factors corresponding to a valid distribution.

$$\mathcal{M} = \{ \boldsymbol{\mu} \mid \exists P(\boldsymbol{y}|\theta) \text{ with marginals } \boldsymbol{\mu} \}.$$

For a pairwise graphical model $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ the condition on valid marginals can be written as:

$$\mathcal{M}_\mathcal{G} = \left\{ \boldsymbol{\mu} \;\middle|\; \begin{array}{l} P(y_i = k) = \mu_{i;k} \; \forall i \in \mathcal{V}, \; \forall y_i \in \mathcal{Y}_i \\ P(y_i = k, y_j = l) = \mu_{ij;kl} \; \forall (i,j) \in \mathcal{E}, \; \forall y_i \in \mathcal{Y}_i, \; \forall y_j \in \mathcal{Y}_j \end{array} \right\}.$$

When the underlying graph $\mathcal{G}$ is a tree, the local polytope $\mathcal{L}_\mathcal{G}$ is a sufficient set of constraints ensuring that $\boldsymbol{\mu} \in \mathcal{M}_\mathcal{G}$. However this is not necessarily true in the general case. A simple example is given in (Wainwright and Jordan 2003). Consider a single cycle pairwise graph on three nodes with $1 \leq i \leq 3 : \mu_{i;k} = 0.5$ in the binary settings $k \in \{0, 1\}$, and $\forall i \neq j$

$$\mu_{ij} = \begin{pmatrix} \alpha_{i,j} & 0.5 - \alpha_{i,j} \\ 0.5 - \alpha_{i,j} & \alpha_{i,j} \end{pmatrix}$$

For any choice of $\alpha_{i,j} \in [0, 0.5]$, the constraints in $\mathcal{L}_\mathcal{G}$ hold, but the choice $\alpha_{1,2} = \alpha_{1,3} = 0.4$ and $\alpha_{2,3} = 0.1$ yields inconsistencies in the joint probability of the variables.

If $\mathcal{L}_\mathcal{G}$ in (4.3) is replaced by $\mathcal{M}_\mathcal{G}$, then the solution recovers the true MAP assignment. Otherwise the solution is a lower bound on the MAP energy. If we were to make the two sets identical, we would need to add an exponentially large number of summation constraints to $\mathcal{L}_\mathcal{G}$ (Wainwright and Jordan 2008). The work in (Sontag et al. 2008) proposes to tighten the polytope by including summation constraints over larger subsets of variables. This approach has been successful in identifying the global minima for some problems. However, it suffers from an increased complexity as ultimately an exponentially large set of possible constraints might need to be searched over.

A solution to an LP-based approach admits an easy to verify certificate of optimality; if the solution is integer, it is the global optimum.

4.2.2 *Quadratic Programming Relaxation*

An alternative relaxation of the integer quadratic program in (4.2) is obtained by simply dropping the integer constraints. The resulting QP is given by:

$$\begin{aligned} \min_{\boldsymbol{\mu}} \quad & \sum_{i \in \mathcal{V}} \boldsymbol{\theta}_i^\mathsf{T} \boldsymbol{\mu}_i + \sum_{(i,j) \in \mathcal{E}} \boldsymbol{\mu}_i^\mathsf{T} \boldsymbol{\Theta}_{ij} \boldsymbol{\mu}_j \\ \text{s.t.} \quad & 0 \leq \mu_{i;k} \leq 1 \quad \forall i, k \quad \text{and} \quad \sum_k \mu_{i;k} = 1 \quad \forall i. \end{aligned} \tag{4.4}$$

A major advantage of the QP relaxation, is the fact that it is tight. In this context the tightness means that the minimizer of (4.4) is also the minimizer of (4.1), as was shown in (Ravikumar and Lafferty 2006). The QP also benefits from a more compact description compared to the LP relaxation, as it requires fewer constraints and variables to formulate the *exact* MAP problem. The variable vector $\boldsymbol{\mu}$, is of size $K \cdot |\mathcal{V}| + K^2 \cdot |\mathcal{E}|$ in the LP (4.3) and only $K \cdot |\mathcal{V}|$ in the QP. The biggest drawback of the QP relaxation, is that in the general case the optimization problem is non-convex due to the edges product term. This fact renders an exact minimization difficult.

In terms of motivation, our work is similar to the QP relaxation approach. The QP formulation of the MAP problem (4.4) was introduced in (Ravikumar and Lafferty 2006), but stems from classical mean-field

approaches. Ravikumar and Lafferty (2006) solved the non-convex problem using a convex relaxation. The solution was later improved in (Kappes and Schnoerr 2008) through a Difference of convex functions (DC) formulation (see Subsection 4.4.1). Both solvers are generic in the sense that they do not exploit the graph structure. Recently Kumar and Zilberstein (2011) introduced a message-passing algorithm for solving the QP relaxation. While improving the run time over the other two algorithms, it still generally suffers from poor solutions due to local minima. The QP solvers often deal with this drawback by restarting with different initializations. We observed that our LPQP algorithms, are much more resilient with respect to the initialization. In all of the experiments we conducted, a restart was never required. We attribute this behavior to the gradual progression between the LP and QP. Finally, in concurrent work Kumar, Zilberstein, and Toussaint (2012) propose a hybrid LP and QP approach to MAP, similar to our formulation discussed in the next section. The resulting optimization problem is solved by a custom message-passing scheme. Our work on the other hand, in its essence reduces to well-known entropy-augmented LP objectives, for which efficient message-passing algorithms exist.

4.3 COMBINED LP AND QP RELAXATION

We propose to optimize an objective which is a combination of the LP and QP relaxations. We retain the auxiliary variables $\boldsymbol{\mu}_{ij}$ of the pairwise terms, but force these variables to agree with the product of the unary marginals $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_j$. The constraints, given by $\text{vec}(\boldsymbol{\mu}_i \boldsymbol{\mu}_j^\mathsf{T}) = \boldsymbol{\mu}_{ij}\ \forall (i,j) \in \mathcal{E}^2$, are enforced through a penalty function $g(\cdot)$ incorporated in the objective. The extent to which the constraint is enforced, is regulated by the parameter ρ.

We focus on the Kullback-Leibler (KL) divergence as the penalty function, due to the probabilistic nature of the compared marginal terms. For probability distributions $\boldsymbol{p}$ and $\boldsymbol{q}$ of a discrete random variable, their KL divergence is defined to be

$$D_{KL}(\boldsymbol{p}, \boldsymbol{q}) := \sum_k p_k \log\left(\frac{p_k}{q_k}\right).$$

2 Here $\text{vec}(\boldsymbol{\mu}_i \boldsymbol{\mu}_j^\mathsf{T})$ denotes the vectorized version of the outer product of $\boldsymbol{\mu}_i$ and $\boldsymbol{\mu}_j$.

A related term which we later use is the entropy of a probability distribution, defined as

$$H(\boldsymbol{p}) := -\sum_k p_k \log(p_k)$$

The general form of the combined objective reads as

$$\min_{\boldsymbol{\mu} \in \mathcal{L}_\mathcal{G}} \boldsymbol{\theta}^\mathsf{T} \boldsymbol{\mu} + \rho g(\boldsymbol{\mu}). \tag{4.5}$$

The first term is simply the LP objective (4.3), written as a scalar product between the potential function, and the concatenated unary and pairwise variables.

We consider two constructions of the penalty term. The constructions differ in the weighting of the edges.

UNIFORM WEIGHTING The KL divergence is penalized in the same way for all the edges in the graph:

$$g^{uni}(\boldsymbol{\mu}) := \sum_{(i,j)\in\mathcal{E}} D_{KL}(\boldsymbol{\mu}_{ij}, \mathrm{vec}(\boldsymbol{\mu}_i \boldsymbol{\mu}_j^\mathsf{T})). \tag{4.6}$$

TREE-BASED WEIGHTING The KL divergence is penalized uniformly within a forest-shaped sub-graph:

$$g^{tree}(\boldsymbol{\mu}) := \sum_{a\in\mathcal{A}} \eta_a \left(\sum_{(i,j)\in\mathcal{E}_a} D_{KL}(\boldsymbol{\mu}_{ij}, \mathrm{vec}(\boldsymbol{\mu}_i \boldsymbol{\mu}_j^\mathsf{T})) \right). \tag{4.7}$$

We assume that a decomposition of the original graph into acyclic subgraphs exists, and is given by

$$\mathcal{G}_a = (\mathcal{V}_a, \mathcal{E}_a), \qquad \mathcal{V} = \bigcup_{a\in\mathcal{A}} \mathcal{V}_a, \qquad \mathcal{E} = \bigcup_{a\in\mathcal{A}} \mathcal{E}_a.$$

The positive weights η_a are tree specific, and assumed to sum to one. In this work we simply used $\eta_a = 1/|\mathcal{A}|$.

For $\rho = 0$, (4.5) amounts to the standard LP relaxation. On the other extreme when $\rho \to \infty$, the constraints $\mathrm{vec}(\boldsymbol{\mu}_i \boldsymbol{\mu}_j^\mathsf{T}) = \boldsymbol{\mu}_{ij}\ \forall (i,j) \in \mathcal{E}$ are fulfilled and the QP relaxation is recovered. By successively increasing ρ during the run of our algorithms, we achieve a gradual enforcement of the constraints.

4.4 LPQP ALGORITHMS

In this section we derive two algorithms for the non-convex LPQP objective in (4.5), with the different penalty terms in (4.6) and (4.7).

4.4.1 *DC decomposition and CCCP algorithm*

The Concave-Convex Procedure (CCCP) (Yuille and Rangarajan 2003), can be applied to a constrained optimization problem, where the objective is non-convex, provided that the objective has a decomposition into a difference of convex functions (Pham Dinh and Le Thi 1998), i.e. a convex and a concave part. In our setting, we wish to find a decomposition of the form

$$\min_{\boldsymbol{\mu} \in \mathcal{L}_{\mathcal{G}}} u_\rho(\boldsymbol{\mu}) - v_\rho(\boldsymbol{\mu}),$$

where both, $u_\rho(\boldsymbol{\mu})$ and $v_\rho(\boldsymbol{\mu})$ are convex.

The CCCP algorithm in each iteration linearizes $-v_\rho$, the concave part of the objective (4.5) around a solution obtained in the previous iteration, such that $u_\rho(\boldsymbol{\mu}) - \boldsymbol{\mu}^\mathsf{T} \nabla v_\rho(\boldsymbol{\mu}^t)$ is convex in $\boldsymbol{\mu}$. The solution $\boldsymbol{\mu}^{t+1}$ is found by minimizing this convex problem subject to the original constraints. The algorithm continues generating the sequence $\{\boldsymbol{\mu}^t\}_{t=1}^{\infty}$ until some convergence criteria is met. The convex program reads:

$$\boldsymbol{\mu}^{t+1} = \operatorname*{argmin}_{\boldsymbol{\mu} \in \mathcal{L}_{\mathcal{G}}} u_\rho(\boldsymbol{\mu}) - \boldsymbol{\mu}^\mathsf{T} \nabla v_\rho(\boldsymbol{\mu}^t). \tag{4.8}$$

The sequence $\{\boldsymbol{\mu}^t\}_{t=1}^{\infty}$ was shown to be monotonically decreasing (Yuille and Rangarajan 2003) and converging to a saddle point of the original objective (Sriperumbudur and Lanckriet 2009). The CCCP is similar to a previous line of work known as difference of convex analysis (DCA) (Pham Dinh and Le Thi 1998; Le Thi and Pham Dinh 2005), considering a bit broader class of functions where the concave part is not necessarily differentiable.

The decompositions of the two objectives we consider, as well as the gradients of the concave part are given below. In the derivations we used the following identity

$$D_{KL}(\boldsymbol{\mu}_{ij}, \operatorname{vec}(\boldsymbol{\mu}_i \boldsymbol{\mu}_j^\mathsf{T})) = H(\boldsymbol{\mu}_i) + H(\boldsymbol{\mu}_j) - H(\boldsymbol{\mu}_{ij}),$$

which holds due to the marginalization constraints of the pairwise marginals (Wainwright and Jordan 2008).

UNIFORM WEIGHTING The Difference of convex functions (DC) decomposition of the combined LPQP objective in (4.5) for the uniform penalty term in (4.6) is given by:

$$u_\rho(\boldsymbol{\mu}) = \boldsymbol{\theta}^\mathsf{T}\boldsymbol{\mu} - \rho \sum_{(i,j)\in\mathcal{E}} H(\boldsymbol{\mu}_{ij})$$
$$v_\rho(\boldsymbol{\mu}) = -\rho \sum_{i\in\mathcal{V}} d_i H(\boldsymbol{\mu}_i).$$

Here d_i denotes the degree of the i-th node in the graph. The derivative of the concave part w.r.t. the unary marginals is

$$\frac{\partial v_\rho(\boldsymbol{\mu})}{\partial \mu_{i;k}} = \rho d_i (1 + \log \mu_{i;k}),$$

The derivative w.r.t. the pairwise marginals is zero.

TREE-BASED WEIGHTING The DC decomposition of the combined LPQP objective in (4.5) for the tree weighted penalty term in (4.7) is given by:

$$u_\rho(\boldsymbol{\mu}) = \boldsymbol{\theta}^\mathsf{T}\boldsymbol{\mu} - \rho \sum_{a\in\mathcal{A}} \eta_a \left(\sum_{(i,j)\in\mathcal{E}_a} H(\boldsymbol{\mu}_{ij}) - \sum_{i\in\mathcal{V}_a} (d_i^a - 1) H(\boldsymbol{\mu}_i) \right)$$
$$v_\rho(\boldsymbol{\mu}) = -\rho \sum_{a\in\mathcal{A}} \eta_a \sum_{i\in\mathcal{V}_a} H(\boldsymbol{\mu}_i).$$

Here d_i^a denotes the degree of the i-th node in the subgraph indexed by a. $\mathcal{A}(i)$ denotes the set of all trees that contain node i. The derivative of the concave part w.r.t. the unary marginals is

$$\frac{\partial v_\rho(\boldsymbol{\mu})}{\partial \mu_{i;k}} = \rho \sum_{a\in\mathcal{A}(i)} \eta_a (1 + \log \mu_{i;k}).$$

As in the uniform case, the derivative of the concave part w.r.t. to the pairwise marginals is zero.

For both objectives, the convex part $u_\rho(\boldsymbol{\mu})$ consists of the original LP formulation, with an additional term that encourages configurations

with a large entropy. This term in the uniform weights penalty, is the entropy of the pairwise marginals, whereas in the tree-based penalty, it is the sum of tree entropies[3] given by $\sum_{(i,j)\in\mathcal{E}} H(\boldsymbol{\mu}_{ij}) - \sum_{i\in\mathcal{V}}(d_i - 1)H(\boldsymbol{\mu}_i)$. The concave part of the decompositions, v_ρ, corresponds to an entropy of the unary marginals. In the CCCP step (4.8), $\log(\boldsymbol{\mu}_i)$ is replaced by $\log(\boldsymbol{\mu}_i^t)$, the marginal from the previous iteration, resulting in an entropy approximation.

4.4.2 *Algorithm Overview*

The general scheme of the suggested LPQP algorithms is shown in Algorithm Algorithm 4.1. The algorithm consists of two loops. The inner loop solves the DC problem for a fixed penalty parameter ρ, whereas the outer loop gradually increases the value of ρ.

Algorithm 4.1 LPQP algorithm scheme for MAP.

Require: $\mathcal{G} = (\mathcal{V}, \mathcal{E}), \boldsymbol{\theta}$.
1: initialize $\boldsymbol{\mu} \in \mathcal{L}_\mathcal{G}$ uniform, $\rho = \rho_0$.
2: **repeat**
3: $\quad t = 0, \boldsymbol{\mu}^0 = \boldsymbol{\mu}$.
4: $\quad$ **repeat**
5: $\quad\quad \boldsymbol{\mu}^{t+1} = \operatorname{argmin}_{\boldsymbol{\tau}\in\mathcal{L}_\mathcal{G}} u_\rho(\boldsymbol{\tau}) - \boldsymbol{\tau}^\mathsf{T}\nabla v_\rho(\boldsymbol{\mu}^t)$.
6: $\quad\quad t = t + 1$.
7: $\quad$ **until** $\|\boldsymbol{\mu}^t - \boldsymbol{\mu}^{t-1}\|_2 \leq \epsilon_{\text{dc}}$.
8: $\quad \boldsymbol{\mu} = \boldsymbol{\mu}^t$.
9: $\quad$ increase ρ.
10: **until** $\|\boldsymbol{\mu} - \boldsymbol{\mu}^0\|_2 \leq \epsilon_\rho$.
11: **return** $\boldsymbol{\mu}$.

The main computational task is in line 5, where a particular instance of a convex optimization problem is solved. Warm-starting the problem in line 5 with the previous solution between successive calls, leads to a substantial speed-up. We choose the initial $\rho = \rho_0$ depending on the scaling of the energies, and use a multiplicative increase with a fixed value. In the experiments we use a multiplicative factor of 1.5, but the results were not very sensitive to this choice.

3 The tree entropy term is due to a factorized form of the joint tree graph probability distribution (Wainwright and Jordan 2008)

SOLUTION ROUNDING Similarly to the LP and QP relaxations, the solutions returned by the LPQP algorithms can be fractional. Since the LPQP scheme ultimately solves a variant of the QP relaxation, to attain the final integer solutions, we use the QP solution rounding scheme suggested in (Ravikumar and Lafferty 2006). Given a unary marginals vector $\boldsymbol{\mu}^*$, we assign the i-th node the label y_i^* given by

$$y_i^* = \underset{k}{\operatorname{argmin}} \left(\theta_{i;k} + \sum_{j \in \mathcal{N}(i)} \sum_{l} \theta_{i,j;k,l} \mu_{j;l}^* \right).$$

Here $\mathcal{N}(i)$ denotes the neighbors of node i. After determining the label of the i-th variable, we set $\mu_{i;y_i^*}^* = 1$ and $\mu_{i;k}^* = 0 \quad \forall\, k \neq y_i^*$, and continue until labels are assigned to all nodes. It can be verified that the rounded solution has an energy that is smaller or equal to the energy of the initial solution $\boldsymbol{\mu}^*$.

4.4.3 *Uniform Weighting*

The convex sub-problem we get in the CCCP step with the uniform weighting penalty function (4.6), is given by

$$\min_{\boldsymbol{\mu} \in \mathcal{L}_\mathcal{G}} \sum_{i \in \mathcal{V}} \tilde{\boldsymbol{\theta}}_i^\mathsf{T} \boldsymbol{\mu}_i + \sum_{(i,j) \in \mathcal{E}} \boldsymbol{\theta}_{ij}^\mathsf{T} \boldsymbol{\mu}_{ij} - \rho \sum_{(i,j) \in \mathcal{E}} H(\boldsymbol{\mu}_{ij}). \tag{4.9}$$

where $\tilde{\boldsymbol{\theta}}_i$, is a modification of the unary potentials by an additional gradient term, originating in the linearized part of the DC decomposition (4.8) [4].

$$\tilde{\boldsymbol{\theta}}_i = \boldsymbol{\theta}_i - \rho d_i \log(\boldsymbol{\mu}_i^t), \tag{4.10}$$

As a result of this unary potentials modification, configurations with small probability in the previous iteration t, are vigorously discouraged.

BELIEF PROPAGATION The convex problem in (4.9) is solved by the norm-product belief-propagation (BP) algorithm (Hazan and Shashua 2010). It is a primal-dual ascent algorithm, guaranteed to converge to the global optimum for any choice of $\rho > 0$.

4 The ρd_i term in ∇v_ρ is constant and can therefore be dropped.

The norm-product algorithm applied to (4.9) computes messages passed from node j to node i as follows

$$m_{j\to i}(y_i) \propto \left(\sum_{y_j} \psi_{ij}^{1/\rho}(y_i, y_j) \frac{\psi_j^{1/(d_j\rho)}(y_j) \prod_{s\in\mathcal{N}(j)} m_{s\to j}^{1/(d_j\rho)}(x_j)}{m_{i\to j}^{1/\rho}(y_j)} \right)^{\rho}$$

here we define $\psi_{ij}(y_i, y_j) = \exp(-\theta_{ij}(y_i, y_j))$ and $\psi_i(y_i) = \exp(-\tilde{\theta}_i(y_i))$. Upon convergence the marginals $\boldsymbol{\mu}_i$ are obtained by multiplying the incoming messages at variable i:

$$\mu_i(y_i) \propto \left(\psi_i(y_i) \prod_{j\in\mathcal{N}(i)} m_{j\to i}(y_i) \right)^{1/(d_i\rho)}.$$

Due to warm starting with the previous DC iteration solution, typically only few passes through the graph are needed for the messages to converge in the later stages of the run.

4.4.4 *Tree-based Weighting*

The convex sub-problem corresponding to the CCCP step with the tree-based weighting penalty (4.7) is,

$$\min_{\boldsymbol{\mu}\in\mathcal{L}_\mathcal{G}} \sum_{i\in\mathcal{V}} \tilde{\boldsymbol{\theta}}_i^\mathsf{T} \boldsymbol{\mu}_i + \sum_{(i,j)\in\mathcal{E}} \boldsymbol{\theta}_{ij}^\mathsf{T} \boldsymbol{\mu}_{ij} - \rho \sum_{a\in\mathcal{A}} \eta_a H_{\text{tree}}^a(\boldsymbol{\mu}). \tag{4.11}$$

Where we define the entropy of a tree by

$$H_{\text{tree}}^a(\boldsymbol{\mu}) := \left(\sum_{(i,j)\in\mathcal{E}_a} H(\boldsymbol{\mu}_{ij}) - \sum_{i\in\mathcal{V}_a} (d_i^a - 1) H(\boldsymbol{\mu}_i) \right).$$

As before, the linearization of the concave part in the CCCP step, results in a modification of the unaries

$$\tilde{\boldsymbol{\theta}}_i = \boldsymbol{\theta}_i - \rho \sum_{a\in\mathcal{A}(i)} \eta_a \log(\boldsymbol{\mu}_i^t). \tag{4.12}$$

DUAL DECOMPOSITION The dual decomposition framework (Bertsekas 1999; Komodakis, Paragios, and Tziritas 2007), can be applied to an optimization problem provided that the objective can be decomposed into several sub-problems, also known in the literature as the

slave problems. The global variables, $\boldsymbol{\mu}$ in our case, are replaced with local copies in each slave problem, denoted here by $\boldsymbol{\nu}^a$, such that the minimization of the slave problems can be carried out independently. To enforce the local variables corresponding to the same original variables to assume the same value, a designated constraint is introduced. The optimization of the sum of slave problems, subject to these constraints, is called the master problem.

We consider the following master problem

$$\begin{aligned} & \sum_{a \in \mathcal{A}} \min_{\boldsymbol{\nu}^a \in \mathcal{L}_{\mathcal{G}_a}} s_a(\boldsymbol{\nu}^a) \qquad (4.13) \\ \text{s.t.} \quad & \boldsymbol{\nu}_i^a = \frac{1}{|\mathcal{A}(i)|} \sum_{a' \in \mathcal{A}(i)} \boldsymbol{\nu}_i^{a'} \quad \forall i, a \in \mathcal{A}(i) \\ & \boldsymbol{\nu}_{ij}^a = \frac{1}{|\mathcal{A}(i,j)|} \sum_{a' \in \mathcal{A}(i,j)} \boldsymbol{\nu}_{ij}^{a'} \quad \forall (i,j), a \in \mathcal{A}(i,j). \end{aligned}$$

Where the slave problems are defined as

$$s_a(\boldsymbol{\nu}) := \sum_{i \in \mathcal{V}_a} \bar{\boldsymbol{\theta}}_i^{\mathsf{T}} \boldsymbol{\nu}_i + \sum_{(i,j) \in \mathcal{E}_a} \bar{\boldsymbol{\theta}}_{ij}^{\mathsf{T}} \boldsymbol{\nu}_{ij} - \rho \eta_a H_{\text{tree}}^a(\boldsymbol{\nu}).$$

Note that since we now include the unary and pairwise terms in the summation over the trees, the corresponding potentials should be adjusted accordingly

$$\bar{\boldsymbol{\theta}}_i = \frac{\tilde{\boldsymbol{\theta}}_i}{|\mathcal{A}(i)|}, \qquad \bar{\boldsymbol{\theta}}_{ij} = \frac{\boldsymbol{\theta}_{ij}}{|\mathcal{A}(i,j)|}.$$

We use the idea from Domke (2011) who formulates the constraint on the replicated marginal variables to agree with the mean. This is simpler than the traditional constraints:

$$\begin{aligned} \boldsymbol{\nu}_i^a &= \boldsymbol{\mu}_i \quad \forall i, a \in \mathcal{A}. \\ \boldsymbol{\nu}_{ij}^a &= \boldsymbol{\mu}_{ij} \quad \forall (i,j), a \in \mathcal{A}. \end{aligned}$$

We can write the Lagrangian and rearrange to get

$$\begin{aligned} & L(\boldsymbol{\nu}^1, \ldots, \boldsymbol{\nu}^{|\mathcal{A}|}, \boldsymbol{\lambda}) = \\ & \qquad \sum_{a \in \mathcal{A}} \min_{\boldsymbol{\nu}^a \in \mathcal{L}_{\mathcal{G}_a}} \left(s_a(\boldsymbol{\nu}^a) + \sum_{i \in \mathcal{V}_a} \boldsymbol{\theta}_i^a(\boldsymbol{\lambda}) \boldsymbol{\nu}_i^a + \sum_{(i,j) \in \mathcal{E}_a} \boldsymbol{\theta}_{ij}^a(\boldsymbol{\lambda}) \boldsymbol{\nu}_{ij}^a \right), \end{aligned}$$

with

$$\boldsymbol{\theta}_i^a(\boldsymbol{\lambda}) = \boldsymbol{\lambda}_i^a - \frac{1}{|\mathcal{A}(i)|} \sum_{a' \in \mathcal{A}(i)} \boldsymbol{\lambda}_i^{a'}$$
$$\boldsymbol{\theta}_{ij}^a(\boldsymbol{\lambda}) = \boldsymbol{\lambda}_{ij}^a - \frac{1}{|\mathcal{A}(i,j)|} \sum_{a' \in \mathcal{A}(i,j)} \boldsymbol{\lambda}_{ij}^{a'}.$$

The Lagrange multipliers vector $\boldsymbol{\lambda}$ is of the same length as all the $\boldsymbol{\nu}$ concatenated together, where for variables that are only replicated once, the corresponding Lagrange multiplier can be dropped. We can think of the potentials as being a function of $\boldsymbol{\lambda}$ and thus the dual problem of (4.13) is given by

$$\max_{\boldsymbol{\lambda}} \sum_{a \in \mathcal{A}} \min_{\boldsymbol{\nu}^a \in \mathcal{L}_{\mathcal{G}_a}} s_a(\boldsymbol{\nu}^a, \boldsymbol{\lambda}).$$

Where $s_a(\boldsymbol{\nu}^a, \boldsymbol{\lambda})$ is defined as

$$\sum_{i \in \mathcal{V}_a} \bar{\boldsymbol{\theta}}_i^a(\boldsymbol{\lambda})^\mathsf{T} \boldsymbol{\nu}_i + \sum_{(i,j) \in \mathcal{E}_a} \bar{\boldsymbol{\theta}}_{ij}^a(\boldsymbol{\lambda})^\mathsf{T} \boldsymbol{\nu}_{ij} - \rho \eta_a H_{\text{tree}}^a(\boldsymbol{\nu}),$$

and the modified potentials are given by

$$\bar{\boldsymbol{\theta}}_i^a(\boldsymbol{\lambda}) = \bar{\boldsymbol{\theta}}_i + \boldsymbol{\theta}_i^a(\boldsymbol{\lambda})$$
$$\bar{\boldsymbol{\theta}}_{ij}^a(\boldsymbol{\lambda}) = \bar{\boldsymbol{\theta}}_{ij} + \boldsymbol{\theta}_{ij}^a(\boldsymbol{\lambda}).$$

All the slave computations can be carried out exactly using the sum-product algorithm, in two passes over the tree. For the maximization w.r.t. $\boldsymbol{\lambda}$ we use the fast iterative shrinkage-thresholding (FISTA) algorithm (Beck and Teboulle 2009), a modern variant of Nesterov's traditional fast gradient method (Nesterov 1983). Our dual decomposition approach is very similar to (Savchynskyy et al. 2011), with two key differences. First, the authors study only a specific choice of the decomposition for the 4-connected grid graph in which each node marginal is replicated twice and each edge is only considered once. Second, we are also interested in settings of $\rho \gg 0$, which is not meaningful in the context of the standard LP relaxation.

The algorithm we used for solving the master problem is given in Algorithm Algorithm 4.2. It is based on FISTA descent as described in (El Ghaoui 2012; Vandenberghe 2012).

Algorithm 4.2 FISTA ascent for the master problem.

initialize $\boldsymbol{\lambda}^{(0)} = \boldsymbol{v}^{(0)}$, $k = 1$.
repeat
  $\theta_k = 2/(k+1)$.
  $\boldsymbol{y} = (1-\theta_k)\boldsymbol{\lambda}^{(k-1)} + \theta_k \boldsymbol{v}^{(k-1)}$.
  $\boldsymbol{u} = \boldsymbol{y} + t_k \nabla f(\boldsymbol{y})$, perform line-search for t_k.
  ensure ascent for $\boldsymbol{\lambda}^{(k)}$:

$$\boldsymbol{\lambda}^{(k)} = \begin{cases} \boldsymbol{u} & f(\boldsymbol{u}) \geq f(\boldsymbol{\lambda}^{(k-1)}) \\ \boldsymbol{\lambda}^{(k-1)} & \text{otherwise.} \end{cases}$$

  $\boldsymbol{v}^{(k)} = \boldsymbol{\lambda}^{(k-1)} + \frac{1}{\theta_k}(\boldsymbol{u} - \boldsymbol{\lambda}^{(k-1)})$.
  $k = k+1$.
until converged.
return $\boldsymbol{\lambda}^{(k-1)}$.

4.4.5 *Convergence of the LPQP Algorithms*

Claim 4.1. *The concave-convex procedure in Algorithm 4.1 converges to a stationary point of the LPQP objective in (4.5) with $\rho = \rho_{final}$, the parameter value reached when the marginals do not change further.*

Proof. It was shown in (Sriperumbudur and Lanckriet 2009) that the CCCP with a convex constraint set converges to a stationary point of the objective. In the last DC iteration, a CCCP is solved with $\rho = \rho_{\text{final}}$. □

4.4.6 *Entropy-augmented LP Solvers*

Recently, several works (Jojic, Gould, and Koller 2010; Savchynskyy et al. 2011) proposed to smooth the LP objective by adding a term that favors entropic marginals. The merit of this additional term is in overcoming the non-smoothness of the objective. In order to ultimately solve the original LP, these entropy-augmented solvers progressively lower the entropy term. Naturally, the convergence of these algorithms is fairly fast in the beginning. This line of research originates in Nesterov's work on fast gradient methods (Nesterov 1983).

The proposed LPQP solvers have the opposite behavior with respect to the smoothness of the objective. The influence of the entropy term is

rather increased through the progression of the algorithm, leading to favorable convergence properties.

4.5 EXPERIMENTS

We use LPQP-U to refer to the implementation of the uniform weighting penalty, and LPQP-T for the tree-based weighting. In the experiments where the graph did not have a natural decomposition, we used a depth-first search algorithm to construct a tree decomposition in a greedy fashion for LPQP-T.

BENCHMARKED METHODS We compare the performance of LPQP-U and LPQP-T with the widely used MAP algorithms, Sequential Tree-Reweighted Message Passing (TRWS) (Kolmogorov 2006) and Max-Product Linear Programming (MPLP) (Sontag et al. 2008), both of which are LP relaxations. For both algorithms we used the implementation made available by the authors. These algorithms represent different trade-offs in performance. TRWS is a highly efficient message-passing algorithm for the standard LP relaxation. It is much faster than the MPLP, especially on large instances where the MPLP convergence is pretty slow. MPLP on the other hand, initially solves the LP relaxation over the local polytope, and in later iterations includes additional summation constraints over sets of three or four variables. This strategy naturally leads to lower (better) energy solutions, on instances where the LP relaxation is not tight. The MPLP was shown to identify the global optimum for some problems.

PERFORMANCE MEASURES In this work we mainly compared the quality of the solutions, which in the MAP setting is most naturally measured by the energy associated with an assignment (4.1). Strictly comparing energy values is problematic for two reasons. The values lack proper scaling required for quantitative comparison of different results on the same problem instance, and are not comparable across instances. We therefore exercise the following scoring procedure. Let $e_1, \dots, e_J$ denote the energies of the compared solutions, we set

$$s_i = \frac{\max\limits_{1 \le j \le J}(e_j) - e_i}{\max\limits_{1 \le j \le J}(e_j) - \min\limits_{1 \le j \le J}(e_j)} \tag{4.14}$$

as the score of the i-th method. This scheme assigns the worst and the best methods, scores of zero and one respectively. The remaining methods get a fraction relative to their value between the best and the worst result. This procedure is not flawless since the scores are still computed relative to the worst energies. It was most often the case though, that TRWS was the lowest scoring method. Being an often used algorithm with provable merits, using it as a normalizing measure is in our opinion a sensible choice. In experiments where the optimal value is known, we use this value instead of $\min_{1\leq j\leq J} e_j$. In addition to comparing the quality of the solution, we comment about the trends in the efficiency (run-time) of the various methods.

4.5.1 *Synthetic Potts Model Data*

We follow a similar experimental setup as in (Ravikumar, Agarwal, and Wainwright 2010). The graph is a 4-nearest neighbor grid of varying size. We used $M = 60, 90, 120$ where M is the grid side-length, and M^2 is the overall number of variables. We used $K = 2$ and $K = 5$ for the number of states. The unary potentials were randomly set to $\theta_{i;k}(y_i) \sim \mathit{Uniform}(-\sigma, \sigma)$, and for σ we used values in $\{0.05, 0.5\}$. Note that the problem instance gets harder for small values of σ, this parameter can be understood as the signal-to-noise ratio. The pairwise potentials $\theta_{ij}(y_i, y_j)$, were set to penalize agreements or disagreements of the labels, by an amount $\alpha_{ij} \sim \mathit{Uniform}(-1, 1)$, chosen at random. We set $\theta_{ij}(y_i, y_j) = 0$ if $y_i \neq y_j$ and α_{ij} otherwise. In this experiment we choose the graph decomposition for the LPQP-T solution as the vertical and horizontal split of the grid edges. The two trees have all the original nodes in common, but no overlapping edges.

The results of the comparison using the performance measure given in (4.14), are presented in Table 4.1. For each choice of parameters, we averaged the scores of 5 runs. Furthermore, Figure 4.1 shows the progress of the objective during a run of the LPQP-U algorithm.

In terms of running time, TRWS was always first to output a solution, followed by the LPQP algorithms. MPLP was always slower and on the larger instances did not converge within a predefined maximal time. We therefore restricted the number of tightening iterations of MPLP to a maximum of 1000. A tightening iteration includes additional constraints into the local marginal polytope. Even after this change,

M (size)	60		90		120	
K (# states)	2	5	2	5	2	5
	$\sigma = 0.05$					
MPLP	0.71	0.99	0.51	0.96	0	0.95
LPQP-U	0.97	0.99	0.97	1	0.98	1
LPQP-T	1	0.97	1	0.98	1	0.98
TRWS	0	0	0	0	0.39	0
	$\sigma = 0.5$					
MPLP	1	1	1	1	1	0.99
LPQP-U	0.99	0.92	0.99	0.91	1	0.94
LPQP-T	0.99	0.95	0.99	0.94	0.99	0.96
TRWS	0	0	0	0	0	0

Table 4.1: Averaged scores achieved by the MAP solvers on the synthetic grid data. The scores, computed according to (4.14), assign in each run 1 and 0 to the best and the worst objective values. The remaining algorithms get a fractional score reflecting their relative objective value.

MPLP was still considerably slower than the other algorithms. Between the LPQP algorithms, the LPQP-U was most often faster than LPQP-T.

As we expect, TRWS returned the worst assignment on almost all configurations. The energies obtained by LPQP-U, LPQP-T and MPLP were in general very close. We observe that both of the LPQP algorithms, returned slightly better solutions in comparison to the MPLP, when the potentials were sampled with lower signal-to-noise ratio σ.

The run time of LPQP-T seems to be mostly influenced by the structure of the decomposition. In later experiments where the decomposition consisted of a larger number of trees with more variables in common, the LPQP-T was significantly slower compared to the LPQP-U. In terms of the energy of the solutions, the two algorithms were very similar. For this reason we report from now on the LPQP-U only. The LPQP-T can still be beneficial in settings where the computations are done on a distributed system.

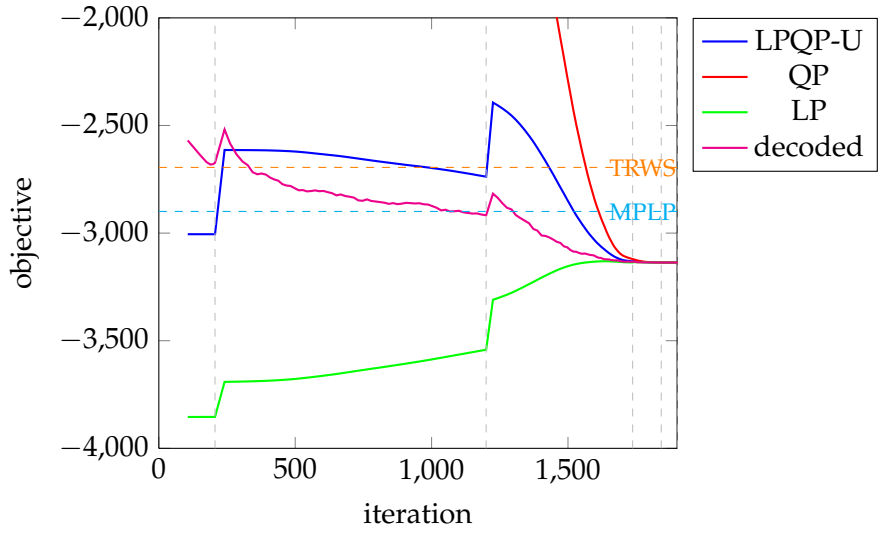


Figure 4.1: Development of the different objectives (for the same μ) during a run of the LPQP-U. The decoded objective refers to the current solution independently rounded to integer values. The vertical lines show iterations where ρ was increased. The horizontal lines show the energy of the solution found by TRWS and MPLP, respectively.

4.5.2 *Protein Design & Prediction*

The protein inference problem discussed in (Yanover, Meltzer, and Weiss 2006), consists of two tasks: protein side-chain prediction and protein design. For the protein prediction task, it was shown in (Yanover, Meltzer, and Weiss 2006) that only for 30 out of the 370 protein prediction instances, the LP relaxation is not tight. For 28 of them, the true MAP was computed using general integer programming techniques, Figure 4.2 visualizes the results on these instances. The LPQP found the global minimum of roughly 2/3 of these more difficult instances. On the remaining 340 instances, the LP is tight. The LPQP found the global optimum in all but three cases (results are not shown). MPLP was applied to this task in (Sontag et al. 2008), and achieved the global optimum on all instances.

The protein design task consists of 97 instances. We used MPLP to compute the global optimum, but for one of the instances, MPLP did

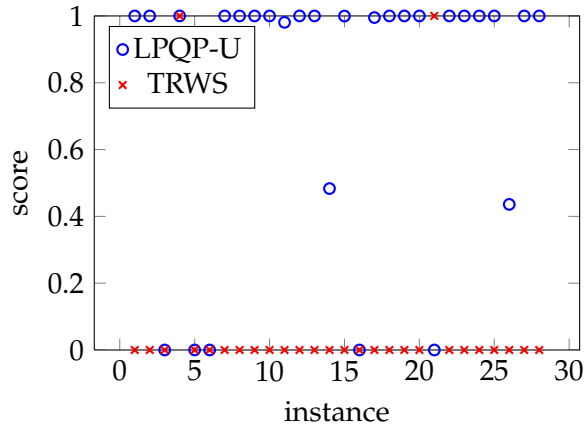


Figure 4.2: Protein prediction results for instances where the LP is not tight. LPQP-U improves on TRWS in all but one cases. For 20 of the 28 instances LPQP finds the true MAP.

not finish within a time-budget of 7 days. The average scores for the remaining 96 instances are as follows. LPQP-U: 0.93, MPLP: 1 and TRWS: 0.03. The average energies are: LPQP-U: -184.06, MPLP: -184.60, TRWS: -173.55. The QP message-passing algorithm in (Kumar and Zilberstein 2011), was tested on this task as well. The evaluation criteria used in this work was the average (across the 97 instances) percentage of the optimal value. While the reported average value in (Kumar and Zilberstein 2011) is 97.7%, our solution achieves 99.7% percentage of the optimal value on average.

4.5.3 *Decision Tree Fields*

As a last experiment we apply our LPQP algorithm to the "hard discrete energy minimization instances" dataset (Nowozin et al. 2011), available on the authors webpage. The task is to fill in, or inpaint, a blanked out area in a binary image of Chinese handwritten characters, see Figure 4.3. The dataset consists of 100 energy minimization instances, and comes with approximate MAP solutions obtained using Simulated Annealing (SA) inference, which was found to work better than TRWS. For 43 instances the LPQP algorithm obtained better solutions than the previously best known solutions. Figure 4.3 visualizes some of the instances where the LPQP algorithm leads to a better solution. We ob-

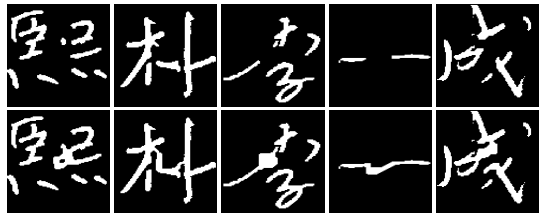


Figure 4.3: Results for the Chinese character inpainting dataset. *Top*: results obtained by LPQP-U. *Middle*: solutions from (Nowozin et al. 2011) obtained by simulated annealing. *Bottom*: Energy difference between the simulated annealing solution and the LPQP solution, the larger the value is, the better the LPQP solution is.

served that the SA solutions seem to hallucinate too much regularity which is not supported by the underlying energy. The scoring of the three algorithms is as follows. LPQP-U: 0.84, SA: 0.74 and TRWS: 0.21. We failed to apply MPLP as the tightening operation did not succeed.

4.6 SUMMARY

This chapter introduces a novel formulation for MAP inference in graphical models, combining the LP and QP relaxation terms through a KL divergence measure. The resulting problem, albeit being non-convex, gives rise to efficient algorithms built upon known LP solvers. Empirical results show that the LPQP algorithms, especially in the uniform weighting version, offers a favorable trade-off between the quality of the solutions and the run-time.

5 MONOCHROMATIC BI-CLUSTERING

This chapter introduces a cost function for the bi-clustering task with categorical valued input matrices, the *monochromatic cost*. Various applications can be casted in this framework, for example the analysis of social networks, functional grouping of biological agents based on their pairwise interactions and others. We analyze the computational aspects of the resulting problem. We provide a NP-hardness proof, as well as approximation algorithms with various levels of guarantees.

The results presented in this chapter are based on Wulff, Urner, and Ben-David (2013).

5.1 INTRODUCTION

Common clustering tasks take as input a dataset and a similarity (or distance) function over it, with the aim of finding a partition of the data into groups of mutually similar elements. Bi-clustering is a variant of this general task, in which the input data comes from two domain sets, and instead of having a distance function over its elements, the input is some relation over these sets. For example, a set of documents and a set of words and the relation indicating the membership of words in the documents. In this setting, the bi-clustering task is to find partitions of each of the two domain sets into groups, such that words in the same group appear in the same groups of documents and documents that share a group are likely to contain words from the same groups of words. Namely, the relation values in each of the resulting blocks (i.e., the product of two groups, one from each domain set) are as homogeneous as possible.

Similarly to clustering, bi-clustering is an unsupervised method for detecting meaningful structure in data. While common clustering tasks require some user-defined notion of similarity between data points, the bi-clustering task we analyze is fully determined by a matrix of an empirically measured pairwise relation (or interaction) between the domain points. The only additional learner's prior knowledge

it requires, is the number of row and column groups. In this sense bi-clustering can be viewed as a more "objective" method.

Bi-clustering is not a new framework. Biologists apply bi-clustering techniques to detect groups of similar genes based on their gene expression levels over a set of different treatments. In recommender systems it is used to determine groups of similar customers based on the matrix of their preferences for a set of products. Bi-clustering is also applied in text categorization and other diverse data mining settings. In spite of its wide scope of applications, bi-clustering tasks have hardly been analyzed in terms of their computational and sample complexity. Many popular bi-clustering paradigms are only defined through the algorithms used to carry them out. Furthermore, the various tasks that are defined as optimization problems with respect to a clear objective function, lack proven bounds on their computational complexity.

We propose the *monochromatic* cost-function for the bi-clustering problem. This cost is a natural formalization of the goal of detecting structure in the form of label-homogeneous sub-matrices, (which is shared by most existing bi-clustering works). Unlike many existing cost-functions, the monochromatic cost can easily handle missing values in the input matrix.

We analyze the computational complexity of the resulting optimization problem; we prove that finding an exact solution is NP-hard, by reducing the monochromatic bi-clustering to the maximum cut problem (see definition in Section 5.4). This work is a continuation of my master thesis study (Wulff 2008), in which we presented a randomized Polynomial Time Approximation Scheme (PTAS) for a similar problem formulation. Here we present an adaptation of this algorithm to a bi-clustering problem in a more broad settings, where the input matrix can have missing entries.

While the run time of the approximation algorithm is polynomial in the input matrix size, for high precision and large number of row and column partitions, it can be too slow. We show that by introducing auxiliary variables for the bi-clusters labels, the monochromatic cost can be phrased as an energy minimization problem (see Section 2.3). Based on an Linear Program (LP) relaxation of the factorized energy minimization objective, we derive a dual-decomposition algorithm as well as an annealing scheme with a Gibbs sampling solution. These methods are faster and therefore more suitable for limited resource scenarios.

5.2 THE MONOCHROMATIC BI-CLUSTERING COST FUNCTION

Given an input matrix over some fixed finite domain $\mathcal{D}$ of values, and integers K and L, the monochromatic bi-clustering task is to find a partition of the rows of the matrix into K groups and its columns into L groups, such that the resulting matrix blocks are as homogeneous as possible. We define the cost of a partition as the fraction of matrix entries that reside in blocks in which they are not the majority value entries.

5.2.1 *Motivation*

As an example consider gene expression profiling. DNA micro-arrays are designed to simultaneously measure the expression level of many genes within a particular mRNA sample, under various clinical conditions. The result of such an experiment is a matrix with rows corresponding to genes, columns to conditions and entries to the measured expression. Often the genes are not homogeneous under the performed tests, and thus one of the main challenges in the analysis of such datasets, is to discover local patterns of sets of genes that exhibit coherent expression across subsets of experimental conditions. The expression values are often discretized or even threshold-ed such that the post-processed data has $\{0,1\}$ entries. The raw micro-array data may contain missing values caused by various factors such as, insufficient resolution, image corruption, or even a systematic robotic failure in the generation process. These missing entries do not carry any information and thus should be disregarded for the purpose of gene grouping. In this work we assign missing entries the symbol $\star$ and design our cost function such that these values do not affect the ranking of the different solutions.

5.2.2 *Formal Definition*

Formally, for some finite domain $\mathcal{D}$, let $\boldsymbol{M} \in (\mathcal{D} \cup \{\star\})^{m \times n}$ be an input matrix (in sections Section 5.5 and Section 5.6, we consider a binary domain $\mathcal{D} = \{0,1\}$ for concreteness), and let $R = [m]$ and $C = [n]$ denote the set of $\boldsymbol{M}$'s row indices and column indices respectively. Given integers K and L, let $\mathcal{P} = (\mathcal{P}_{\mathcal{R}} = \{R_1, \ldots R_K\}, \mathcal{P}_{\mathcal{C}} = \{C_1, \ldots C_L\})$

denote a partition of R into K subsets, and of C into L subsets. We use $\boldsymbol{M}[R', C']$ for a sub-matrix defined by a subset of rows $R' \subseteq R$ and subset of columns $C' \subseteq C$.

The monochromatic cost of a partition is defined as

$$Mon_{K,L}(\boldsymbol{M}, \mathcal{P}) := \frac{1}{|\boldsymbol{M}|} \sum_{k \in [K], l \in [L]} \phi(\boldsymbol{M}[R_k, C_l]) \tag{5.1}$$

where $\phi : \{\mathcal{D} \cup \star\}^{s \times t}$ is a function returning the number of the non-$\star$ entries in a (sub) matrix that differ from the majority, non-$\star$ value. We use $|\boldsymbol{M}| := mn$ to denote the input matrix size. Formally, for a matrix $A \in (\mathcal{D} \cup \{\star\})^{s \times t}$, let d_{max} denote the majority non-$\star$ entry in A, then

$$\phi(A) = |\{(i,j) : A[i,j] \neq d_{max} \text{ and } A[i,j] \neq \star\}|$$

The majority value here refers to the most frequent value among the non-$\star$ entries, the majority value does not necessarily occur in more than half of the entries.

We formally define the monochromatic bi-clustering problem as:

Definition 5.1. Given an input matrix $\boldsymbol{M} \in (\mathcal{D} \cup \{\star\})^{m \times n}$, the K, L-MonochromaticBiclustering (K, L-MCBC) problem is finding a partition $\mathcal{P}$ of $\boldsymbol{M}$ that minimizes the monochromatic cost.

We also consider the version of the monochromatic bi-clustering in which instead of penalizing non-homogeneous bi-clusters, we promote bi-cluster homogeneity (or agreement). We refer to this version as the *monochromatic agreement*. The monochromatic agreement of an input matrix $\boldsymbol{M}$ and a partition $\mathcal{P}$ is simply $1 - Mon_{K,L}(\boldsymbol{M}, \mathcal{P})$ (naturally a matrix partition that optimizes one of these costs also optimizes the other, but there is a difference when it comes to measuring the approximation ratio of a close-to-optimal partitioning).

Remark. In this work we assume that the number of row and column clusters are known to the user. In practice determining the number of clusters or bi-clusters is a common issue in many clustering schemes, and is often treated separately from finding the clustering solution. Naturally if the objective function does not directly penalize the number of clusters, K, some external criteria needs to be set. Otherwise increasing K leads to artificially lower objective costs, up to a cost of zero for the singleton solution. There exist various approaches to estimating the number of clusters, most of which can be easily adapted to the

bi-clustering case. For example information criteria approaches such as Akaike information criterion (AIC) or the Bayesian information criterion (BIC). Recently a model selection approach based on the trade-off between the robustness and informativeness of the underlying solutions was proposed in (Buhmann 2011).

5.2.3 *Monochromatic Compression Scheme*

One can think of the monochromatic bi-clustering problem as a compression of an input $m \times n$ matrix into a $K \times L$ blocks matrix, typically, $K \ll m$ and $L \ll n$. The compressed representation requires $KL + m\log(K) + n\log(L)$ bits (the majority label of each block, augmented by the cluster index of each row and each column), instead of the straightforward $m \times n$ bit representation. The monochromatic cost is essentially the error (or the information loss) of such a compressed representation. For an error-free compression, we need to add an encoding of the outliers in each block, adding $\sum_{k\in[K],l\in[L]} \phi(\boldsymbol{M}[R_k, C_l])\log(|\boldsymbol{M}|)$ to the length of the encoding. This shows that (for fixed $\boldsymbol{M}$,K and L) the length of our encoding corresponds to the monochromatic cost.

5.3 RELATED WORK

Much of the more practically oriented work on bi-clustering discusses algorithms that lack an explicit objective function as an optimization goal. This line of research is less relevant to our work and we refer the reader to the survey by Tanay, Sharan, and Shamir (2006) for further details.

The first definition of bi-clustering as an optimization problem over some well defined objective function is probably due to Hartigan (1972). Hartigan considers real valued matrices and proposes several objective functions, including the sum of block's variances. Hartigan also proposes a heuristic for finding a low-cost bi-clustering, however, no guarantees are proven for its performance.

Cheng and Church (2000), were the first to introduce bi-clustering to gene expression analysis. They formally define a cost function, called the *low mean squared residue*, which can be viewed as a variant of Hartigan's minimum variance cost. They propose an iterative greedy search algorithm which may converge to a local minima. Other notable objec-

tive functions used in bioinformatics include *loss in mutual information* in (Dhillon, Mallela, and Modha 2003) and the *square residue*, (Cho et al. 2004). In all of these papers, neither optimization quality guarantees nor computational complexity bounds are proven.

The Infinite Relational Model (IRM), described in (Kemp and Tenenbaum 2006; Kemp and Tenenbaum 2008) is a Bayesian model for the automated discovery of structure in complex data. The approach assumes a generative model, in which the entries of the matrix $\boldsymbol{M}$ are generated by a Binomial distribution. The number of bi-clusters is automatically adjusted by a Chinese restaurant process. Unlike the IRM, the work presented here takes a combinatorial view of the problem of detecting homogeneous structure. It is more objective in the sense of not making any assumptions about the generating process. Their approach on the other hand is more general, and is suitable for more types of input data. Their work contains an extensive experimental setup, but it does not provide formal guarantees.

Various studies (Chakrabarti et al. 2004; Papadimitriou et al. 2008; Hirai, Chou, and Suzuki 2011) propose cost functions that are inspired by the minimum description length (MDL) principle. These cost functions try to minimize a variation of the Shannon entropy, and justify this by its relation to encoding length. However, this relation is of asymptotic nature, it holds only in the limit, when instance sizes go to infinity. Minimizing this measure for a specific input matrix does not necessarily yield an actual short description of the matrix. In contrast to this, the monochromatic cost function does correspond to the length of a compressed representation (see the discussion in Subsection 5.2.3), and can hence be also viewed as implementing the MDL principle. Those studies do not provide theoretical analysis of the approximation quality of their outcome or of the computational complexity of the resulting bi-clustering optimization task [1].

The algorithms and complexity community analyzed the computational complexity of the related problem of correlation clustering (the monochromatic bi-clustering cost can be viewed as a generalization of correlation clustering (see a detailed discussion in (Wulff 2008)). The problem of correlation clustering with a fixed number of clusters has been studied in (Giotis and Guruswami 2006), yielding a PTAS for the

1 Both (Chakrabarti et al. 2004) and (Papadimitriou et al. 2008) claim that the task is NP-hard. However, a closer look reveals that their argument is based on intuition and does not yield an actual proof of the claim.

minimization problem for all K, and a PTAS for the maximization version along with an NP-hardness result. Our approximation algorithm is inspired by the PTAS for Max k-CUT by Goldreich, Goldwasser, and Ron (1998).

5.4 NP-HARDNESS

We show that the K, L-MCBC problem is computationally hard (NP-hard) for input matrices over $\mathcal{D} \cup \{\star\}$.

Theorem 5.1. *The K, L-MCBC is NP-hard for every finite domain set $\mathcal{D}$ of size ≥ 2 and any (K, L) such that $K \geq 2$ and $K \leq L \leq |\mathcal{D}|^{K-1}$, or $L \geq 2$ and $L \leq K \leq |\mathcal{D}|^{L-1}$.*

PROOF SKETCH We start off by constructing a reduction from MaxCut to the 2, 2-MCBC. A *cut* in a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is a partition of the vertex set into $\mathcal{V}_1$ and $\mathcal{V}_2$. The size of the cut is the number of edges in $\mathcal{E}$ that connect vertices from $\mathcal{V}_1$ to vertices from $\mathcal{V}_2$.

Definition 5.2. The decision version of MaxCut is:

Input	A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, integer r.
Question	Is there a cut of $\mathcal{G}$ of size at least r?

Given instance $(\mathcal{G} = (\mathcal{V}, \mathcal{E}), r)$ of MaxCut we construct an instance $\boldsymbol{M}_{\mathcal{G}}$ of 2, 2-MCBC and prove that this constitutes a reduction by showing:

Lemma 5.1. *$\mathcal{G}$ has a cut of size at least r if and only if $\boldsymbol{M}_{\mathcal{G}}$ has a 2, 2-bi-clustering of monochromatic cost at most $\frac{2(|\mathcal{E}|-r)}{|\boldsymbol{M}_{\mathcal{G}}|}$.*

Finally, we extend the hardness result to the K, L-MCBC problem for larger values of K and L, by showing a reduction from K, L-MCBC to 2, 2-MCBC. For the sake of concreteness, we focus on the case where $\mathcal{D} = \{0, 1\}$. The proofs readily generalize to arbitrarily domain $\mathcal{D}$.

Note that for K and L such that $K > |\mathcal{D}|^L$, K, L-MCBC is the same as $|\mathcal{D}|^L, L$-MCBC, since for L column blocks, there are at most $|\mathcal{D}|^L$ possible patterns for the rows. Thus our current reductions misses only few relevant combinations for K and L. Theorem 5.1 further implies that over an infinite domain, K, L-MCBC is NP-hard for *all* combinations of $K \geq 2$ and $L \geq 2$. We conjecture that this holds for finite

domains as well. Furthermore, our NP-hardness result for fixed K and L implies that the MCBC problem, where K and L are part of the input, is NP-hard.

5.4.1 *NP-Hardness of* 2, 2*-MCBC*

Let $s(\mathcal{V}_1, \mathcal{V}_2)$ denote the size of the cut for $\mathcal{V}_1, \mathcal{V}_2 \subset \mathcal{V}$ such that $\mathcal{V}_1 \cap \mathcal{V}_2 = \emptyset$ and $\mathcal{V}_1 \cup \mathcal{V}_2 = \mathcal{V}$. An equivalent definition for the size of a cut is the total number of edges, $|\mathcal{E}|$, minus the edges within $\mathcal{V}_1$ and the edges within $\mathcal{V}_2$. For a subset $\mathcal{U} \subseteq \mathcal{V}$, and a vertex $v \in \mathcal{V}$ let $v[\mathcal{U}]$ denote the number of neighbors of t in $\mathcal{U}$. We can write the size of the cut as

$$s(\mathcal{V}_1, \mathcal{V}_2) = \frac{1}{2}\left(\sum_{v\in\mathcal{V}} v[\mathcal{V}] - \sum_{v\in\mathcal{V}_1} v[\mathcal{V}_1] - \sum_{v\in\mathcal{V}_2} v[\mathcal{V}_2]\right) \tag{5.2}$$

Thus maximizing the size of a cut is equivalent to minimizing the *cost* of the cut $c(\mathcal{V}_1, \mathcal{V}_2)$ defined as

$$c(\mathcal{V}_1, \mathcal{V}_2) := \frac{1}{2}\left(\sum_{v\in\mathcal{V}_1} v[\mathcal{V}_1] + \sum_{v\in\mathcal{V}_2} v[\mathcal{V}_2]\right) \tag{5.3}$$

The MaxCut question Definition 5.2 can be reformulated as: Is there a cut of $\mathcal{G}$ of cost at most r?

Given an instance $\mathcal{G} = (\mathcal{V}, \mathcal{E})$, and a cost r, we construct an instance $\boldsymbol{M}$, with a 2, 2-MCBC cost $\frac{2r}{|\boldsymbol{M}|}$. The construction is shown in Figure 5.1.

We start by defining the "left half" of the matrix $\boldsymbol{M}$. For every vertex $v \in \mathcal{V}$ we introduce $n = |\mathcal{V}|$ rows $r_1^v \dots r_n^v$ and n columns $c_1^v \dots c_n^v$. We set the entries of $\boldsymbol{M}$ corresponding to rows and columns of the same vertex (the "diagonal blocks"), to 1, i.e. $\forall v,\ 1 \le i, j \le n : \quad \boldsymbol{M}[r_i^v, c_j^v] = 1$. Let $\mathcal{V} = \{v_1 \dots v_n\}$ be an ordering of the vertices of $\mathcal{G}$, if vertices v_i and v_j of $\mathcal{G}$ are connected by an edge, we set the entry $\boldsymbol{M}[r_j^{v_i}, c_i^{v_j}]$ and $\boldsymbol{M}[r_i^{v_j}, c_j^{v_i}]$ to 0. All other entries of the left half are set to $\star$.

The "right side" of $\boldsymbol{M}$ is an $n^2 \times n^2$ matrix as well, where the diagonal $n \times n$ blocks are set to 0 and the rest to $\star$. More formally, we introduce another set of n columns $0_1^v, \dots 0_n^v$ for each vertex $v \in \mathcal{V}$. We set $\forall v,\ 1 \le i, j \le n : \quad \boldsymbol{M}[r_i^v, 0_j^v] = 0$. The remaining right half of $\boldsymbol{M}$ is set to $\star$. We refer to these columns as the 0-columns of a vertex t, and

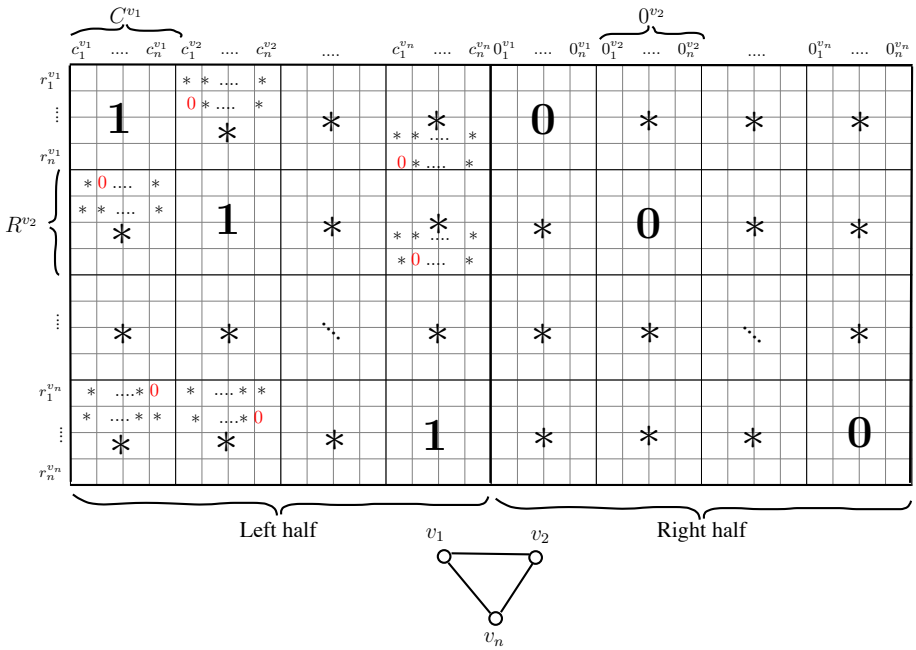


Figure 5.1: Construction of the reduction from MaxCut to 2,2-MCBC. *Bottom*: The graph instance (3 out of n vertices are displayed)

use O^v for the set $\{0_1^v, \dots 0_n^v\}$. Similarly we refer to the set of columns $\{c_1^v \dots c_n^v\}$ as the 1-columns of t, and denote it as C^v (note that while the 0 columns contain only 0 and $\star$ entries, the 1-columns consist mostly of 1 and $\star$, but also contain a few 0 entries, corresponding to edges of the graph). Finally we use R^v to refer to the set of rows $\{r_1^v \dots r_n^v\}$ associated with vertex t.

The NP-hardness of 2,2-MCBC now follows directly from the NP-hardness of MaxCut and the following lemma which is a reformulation of Lemma 5.1.

Lemma 5.2. *$\mathcal{G}$ has a cut of cost at most r if and only if $\boldsymbol{M}$ has a 2,2-bi-clustering of monochromatic cost at most $\frac{2r}{|\boldsymbol{M}|}$.*

Proof. We first show that a cut of cost at most r induces a solution of the 2,2-bi-clustering of cost $\frac{2r}{|\boldsymbol{M}|}$. Let $\mathcal{V}_1, \mathcal{V}_2$ be a cut of $\mathcal{G}$ of cost at most r. We define a partition $\mathcal{P}_\mathcal{R} = \{R_1, R_2\}$ of the rows and $\mathcal{P}_\mathcal{C} = \{C_1, C_2\}$ of the columns as follows: For all $v \in \mathcal{V}_1$ we put R^v in R_1, C^v in C_1 and O^v in C_2. For all $v \in \mathcal{V}_2$ we do the opposite, put R^v in R_2, C^v in C_2 and O^v in C_1. The partition is depicted in Figure 5.2.

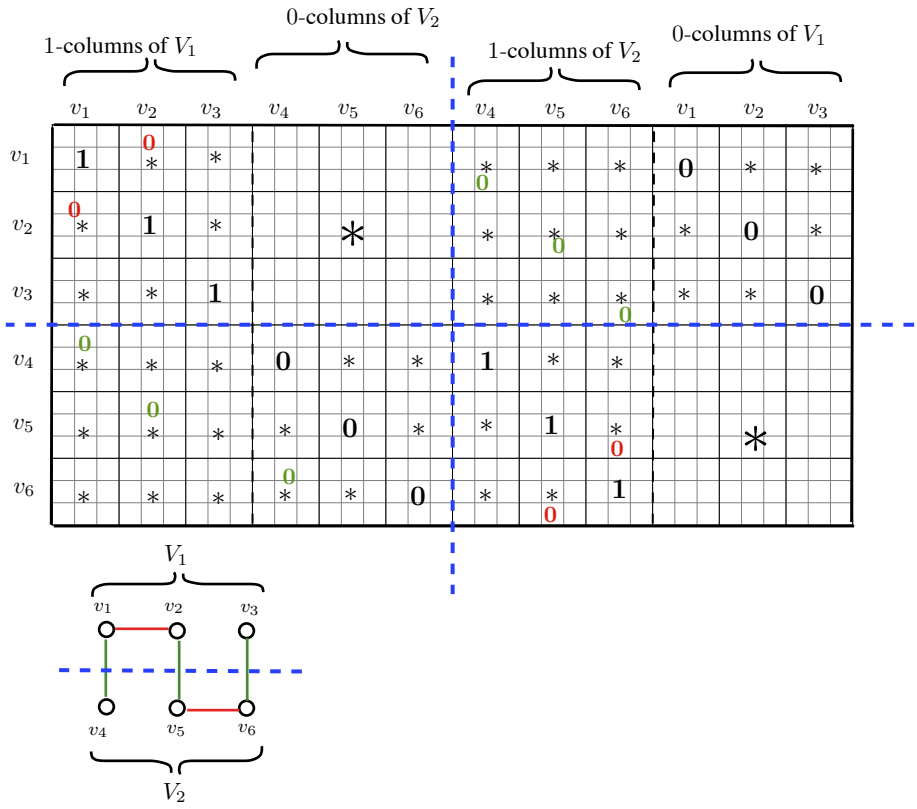


Figure 5.2: Optimal 2, 2-MCBC solution for the reduction matrix corresponding to the graph at the bottom. The zero entries highlighted in red correspond to edges that do not cross the cut, and incur a monochromatic cost respectively.

This results in a 2×2 bi-clustering with a majority of 1 in the diagonal blocks (upper left and lower right) and a majority of 0 is the off-diagonal ones (upper right and lower left). The off-diagonal blocks consist of only 0 and $\star$ entries, such that the monochromatic cost due to these blocks is 0. The diagonal blocks consist of: $n^2|\mathcal{V}_t|$ 1-entries, and $\sum_{v \in \mathcal{V}_t} v[\mathcal{V}_t]$ 0-entries, for $t \in \{1, 2\}$. The remaining entries are $\star$. Clearly there is a majority of 1 in these blocks, such that their total monochromatic cost amounts to exactly $2c(\mathcal{V}_1, \mathcal{V}_2) = 2r$ (see (5.3)). Normalizing by the size of the matrix yields the required cost.

Next we show that if there is a bi-clustering of $\boldsymbol{M}$ with cost at most r, then there is cut of $\mathcal{G}$ of cost $\frac{|\boldsymbol{M}|r}{2}$. Note that if there is any bi-clustering

solution of cost at most r, then the optimal one has cost at most r, thus we consider a bi-clustering of optimal cost. For the sake of the proof we start by considering an optimal $n^2, 2$-bi-clustering, namely a bi-clustering where every row is a set of the row partition (thus, we are only interested in the resulting 2-partition of the columns). We will then argue, that this optimal $n^2, 2$-bi-clustering is actually a $2, 2$-bi-clustering and therefore also the optimal $2, 2$-bi-clustering.

We can assume that in an optimal solution identical columns are in the same cluster. Similarly, we can assume that two columns that are "inverse" of each other (one can be obtained from the other by replacing each 0 with a 1 and each 1 with a 0) are in different clusters. Thus, for any t, all columns in O^v are in one cluster. Consider a vertex t and its corresponding set of rows R^v. Without loss of generality let us assume that the 0-columns of t are in C_2. Each of the rows $r_2^v, \ldots, r_n^v$ contains n entries 1 and at most one entry 0 in the left half. The row r_1^v contains no 0. Equally the column c_1^v contains no 0 entries. Therefore, we can assume that c_1^v is in C_1 (it is the inverse of the columns in O^v). By way of contradiction, assume that not all columns of C^v are in C_1, say c_i^v is in C_2 for some $i \geq 2$. As at least c_1^v is in C_1, all columns in O^v are in C_2 and the rows $r_1^v, \ldots, r_n^v$ contain at most one 0 entry in the left half, we can assume that all these rows have a 1-block for C_1 and a 0-block for C_2. Having a column c_i^v in C_2 incurs a cost of at least n by its 1 entries, however moving it into C_1 can incur a cost of at most 1 as the column has at most one 0 entry and all its 1 entries do not contribute to the cost anymore. Thus, all columns in C^v are in C_1.

We showed that for every vertex, all its 0 columns are in one cluster and all its 1-columns are in the other cluster (which group is in which cluster may vary). Every row therefore has a block pattern $1, 0$ or $0, 1$, and the only cost that it incurs per row is that of a 0 entry of the left half, which ended up in the 1-block of the row. Grouping all the rows with pattern $1, 0$ into R_1, and all the rows with pattern $0, 1$ in R_2, leads to a $2, 2$-bi-clustering of the same cost. As this cost is optimal for an $n^2, 2$-bi-clustering, it is also optimal for $2, 2$-bi-clustering (if there was a $2, 2$-solution of lower cost, separating the rows into singleton sets for an $n^2, 2$-bi-clustering would lead to a lower cost solution for this as well).

If we set $\mathcal{V}_1$ to be the vertices whose 1-columns are in C_1 and $\mathcal{V}_2$ the vertices whose 1-columns are in C_2, we obtain a cut. The only matrix entries that contribute to the cost of the bi-clustering are the 0 coming from edges within one of these sets. □

5.4.2 *NP-Hardness for Larger K, L*

We complete the proof of Theorem 5.1 by reducing the 2, 2-MCBC problem to the K, L-MCBC problem.

Given K and L as in Theorem 5.1, without loss of generality, we assume $K \leq 2^{L-1}$, and an input matrix $\boldsymbol{M}$ to the 2, 2-MCBC problem, we construct matrix $\boldsymbol{N}$ such that an optimal K, L-MCBC partitioning of $\boldsymbol{N}$, induces an optimal 2, 2-MCBC partitioning of $\boldsymbol{M}$.

REDUCTION OUTLINE Let m and n be the number of rows and columns of $\boldsymbol{M}$, respectively. The matrix $\boldsymbol{N}$ consists of $(K-1) \times (L-1)$ blocks, each of size $m \times n$. The top left corner block of $\boldsymbol{N}$ is the input matrix $\boldsymbol{M}$. All other blocks are either all-zero matrices or all-one matrices. All the blocks in the top row of blocks, except the left-most block, are all-zero matrices. The blocks indexed $(i, 1)$, for $2 \leq i \leq 2^{L-2} - 1$ (and $i \leq K - 1$) (blocks that reside below the top left corner block) are also all-zero matrices. The blocks indexed $(i, 1)$, for $2^{L-2} \leq i \leq K - 1$ (also residing below the top left corner block) are all-one matrices.

Finally, for every $2 \leq i \leq 2^{L-2}$ let v_i be vectors in $\{0, 1\}^{L-2}$ such that for all $i \neq j$, $v_i \neq v_j$ and none is the all-zero vector (i.e. let the set of the v_i be all vectors in $\{0, 1\}^{L-2}$ except the all-zero). Now let the (i, j) block, for $2 \leq i \leq 2^{L-2}$ (and $i \leq K - 1$) and $2 \leq j \leq L - 1$, be a homogeneous matrix all of whose entries equal the $j - 1$ entry of the vector v_i. Finally, set the entries of the $(2^{L-2} + i, j)$ block equal to those of the (i, j) block for all $1 \leq i \leq 2^{L-2} - 1$ (and $i \leq K - 1$) and $2 \leq j \leq L - 1$.

It is easy to see that the optimal K, L-MCBC partition of $\boldsymbol{N}$ induces an optimal 2, 2-MCBC partition over $\boldsymbol{M}$.

5.4.3 *Instances with Arbitrary Fraction of $\star$ Entries*

Adding blocks of $\star$-entries to a matrix does not make it any easier to find its optimal bi-clustering. Therefore, our hardness result applies to arbitrarily sparse inputs as well. However, for instances with a large fraction of $\star$-entries, approximating the monochromatic agreement is easy: any bi-clustering has relatively low cost, yielding a trivial approximation algorithm for such instances. We thus give an NP-hardness proof for the case of input matrices with small fraction of

$\star$-entries. We conjecture that the K, L-MCBC is NP-hard even restricted to input matrices without $\star$-entries.

Theorem 5.2. *For any fixed $K \geq 2$, $L \geq 2$, satisfying the condition of Theorem 5.1, and any $\epsilon \geq 0$, the $K+1, L+1$-MCBC problem restricted to instances with at most ϵ-fraction of missing $\star$ entries, is NP-hard.*

Proof. Given an $\epsilon > 0$, and K and L, we construct a reduction from K, L-MCBC, to $K+1, L+1$-MCBC, restricted to input matrices containing at most an ϵ fraction of $\star$-entries.

Given a matrix $\boldsymbol{M}$, we construct matrix $\boldsymbol{N}$ as follows: Add $|M|\frac{1}{\epsilon}$ rows and columns to $\boldsymbol{M}$ such that the upper left block of $\boldsymbol{N}$ is identical to $\boldsymbol{M}$. Set the entries of the upper right and lower left blocks to 1 and the entries to the lower right block to 0. Now $\boldsymbol{N}$ has at most an ϵ-fraction of $\star$ entries. Further, it is easy to see that the optimal $K+1, L+1$-MCBC partition of $\boldsymbol{N}$ induces an optimal K, L-MCBC solution to $\boldsymbol{M}$.

Using induction the result holds for K, L. □

5.5 POLYNOMIAL TIME APPROXIMATION ALGORITHM

In this section we present a randomized Polynomial Time Approximation Scheme (PTAS) for solving the K, L-MCBC problem. Given an accuracy parameter ϵ and confidence threshold δ, for a binary input matrix $\boldsymbol{M} \in \{0, 1, \star\}^{m \times n}$, the algorithm outputs a bi-clustering that with probability $\geq 1 - \delta$, has an agreement score within a multiplicative $(1 - \epsilon)$ factor of the score of the best possible bi-clustering for that matrix. The run time of the algorithm is polynomial in the size of the input matrix ($|\boldsymbol{M}| = mn$).

5.5.1 *Algorithm Overview*

There are two key ideas underlying the algorithm. To introduce the first concept, we need to define the monochromatic cost with respect to a pattern. The output of the monochromatic bi-clustering is a partition of the original matrix into $K \times L$ sub-matrices. We define the pattern of the solution as the resulting $K \times L$ binary matrix with entries corresponding to the majority values in each sub-matrix of the partition. One can also consider the reverse operation, where the pattern is fixed in advance, and the cost of a partition is computed with respect to the values

defined by the pattern rather than the true majorities emerging from the partition.

Formally, let $\boldsymbol{A} \in \{0,1\}^{K \times L}$ be a pattern matrix, we define the monochromatic cost with respect to $\boldsymbol{A}$ as

$$Mon_{\boldsymbol{A}}(\boldsymbol{M}, \mathcal{P}) := \frac{1}{|\boldsymbol{M}|} \sum_{k \in [K], l \in [L]} \phi_{\boldsymbol{A}[k,l]}(\boldsymbol{M}[R_k, C_l]) \tag{5.4}$$

where, for a matrix $\boldsymbol{M}$ and a value $a \in \{0,1\}$, we define

$$\phi_a(\boldsymbol{M}) = |\{(i,j) : \boldsymbol{M}[i,j] \neq a \text{ and } \boldsymbol{M}[i,j] \neq \star\}|$$

Note that in the above definition the parameters K and L are given implicitly by the dimensions of $\boldsymbol{A}$.

Our algorithm iterates over all (discarding symmetries) possible patterns, and for each pattern finds the best (approximated) partition. Naturally, the pattern of the best partition will be considered, and it is easy to see that the partition minimizing the cost with respect to the optimal pattern, is the optimal monochromatic solution.

Claim 5.1. *For a fixed pattern matrix $\boldsymbol{A}$, and a fixed partition $\mathcal{P}_{\mathcal{R}}$ of the rows of $\boldsymbol{M}$, it is possible to find the partition of the columns that yields the lowest monochromatic pattern cost, i.e.* $\operatorname{argmin}_{\mathcal{P}_{\mathcal{C}}'} Mon_{\boldsymbol{A}}(\boldsymbol{M}, (\mathcal{P}_{\mathcal{R}}, \mathcal{P}_{\mathcal{C}}'))$, *in polynomial time.*

According to Claim 5.1, if we knew the optimal partition of the rows, and conditioning on the pattern, we could efficiently recover the optimal partition of the columns. The claim is stated with respect to the rows but holds just the same for the columns

The second component in the derivation of the PTAS, is showing that it is sufficient to know the optimal partition of only a (fixed-size) uniformly sampled subset of the rows, to obtain a partition of the columns which is guaranteed to yield an "ϵ-close" agreement score to the optimal partition, with high probability (over the sample). The main technical part of the proof is showing that a sample size depending only on ϵ and δ (but independent of $|\boldsymbol{M}|$) suffices to guarantee the generation of an ϵ-approximation with high probability.

Putting the two ideas together, the approximation algorithm performs the following steps; For each $K \times L$ pattern matrix $\boldsymbol{A}$, the algorithm randomly picks a sample of the rows and a sample of the columns. For each partition of the rows sample into K groups, it computes

the implied partition of the entire set of columns, having the optimal monochromatic cost with respect to $\boldsymbol{A}$. Similarly, for every partition of the columns sample, it computes the corresponding optimal partition of the entire set of rows (w.r.t. the pattern $\boldsymbol{A}$). Next, for every resulting partitioning of the rows and columns, it computes the monochromatic cost of the full partition, and, finally, the solution with minimal cost is returned. A pseudo-code of the monochromatic approximation algorithm is given in Algorithm 5.1.

Algorithm 5.1 Monochromatic Approximation Algorithm

Input: $\boldsymbol{M} \in \{0,1,\star\}^{m\times n}$, $K, L \in \mathbb{N}_{\geq 1}$, $\epsilon, \delta > 0$.
Initialize $t = \frac{1}{2\epsilon^2} \log \frac{KL}{\delta\epsilon}$.
for each pattern matrix $\boldsymbol{A} \in \{0,1\}^{K\times L}$ **do**
  Sample t rows and columns: R^S, C^S
  for each partition $\mathcal{P}_{\mathcal{R}}^{\mathcal{S}} = \{R_1^S, \dots R_K^S\}$ of R^S and $\mathcal{P}_{\mathcal{C}}^{\mathcal{S}} = \{C_1^S, \dots C_L^S\}$ of C^S **do**
    for each row $i \in R$ and column $j \in C$ **do**
      $\text{assignment}(i) = \operatorname{argmin}_{1\leq k\leq K} Err(i, k|\boldsymbol{A}, \mathcal{P}_{\mathcal{C}}^{\mathcal{S}})$
      $\text{assignment}(j) = \operatorname{argmin}_{1\leq l\leq L} Err(j, l|\boldsymbol{A}, \mathcal{P}_{\mathcal{R}}^{\mathcal{S}})$
    end for
    compute the cost of the partition
  end for
end for
Return the partition with the minimal cost

The following additional notation is used in the pseudo-code and the analysis of the algorithm. Let $S = (R^S, C^S)$ denote the sample of the rows and columns and $t = |R^S| = |C^S|$. We use $\mathcal{P}^S = (\mathcal{P}_{\mathcal{R}}^{\mathcal{S}}, \mathcal{P}_{\mathcal{C}}^{\mathcal{S}})$ to denote the partition of the sample, where $\mathcal{P}_{\mathcal{R}}^{\mathcal{S}} = \{R_1^S, \dots R_K^S\}$ and $\mathcal{P}_{\mathcal{C}}^{\mathcal{S}} = \{C_1^S, \dots C_L^S\}$.

Given a partition of the rows $\mathcal{P}_{\mathcal{R}} = \{R_1, \dots, R_K\}$, and a pattern $\boldsymbol{A}$, we define the following error function for each column $j \in C$ and a column block $1 \leq l \leq L$,

$$Err(j, l|\boldsymbol{A}, \mathcal{P}_{\mathcal{R}}) = \frac{1}{t} \sum_{k\in[K]} \sum_{i\in R_k} |\boldsymbol{M}[i, j] - \boldsymbol{A}[k, l]| \tag{5.5}$$

The sum above ignores rows i for which $\boldsymbol{M}[i,j] = \star$. This is essentially the error with respect to the pattern $\boldsymbol{A}$ incurred by placing the column j in the l block, when the partition of the rows is given by $\mathcal{P}_{\mathcal{R}}$. Similarly $Err(i,k|\boldsymbol{A},\mathcal{P}_{\mathcal{C}})$ is defined for each row i.

5.5.2 *Algorithm Analysis*

Next we show that taking a sample size of $O(\frac{1}{\epsilon^2})$ suffices to approximate $OPT(\boldsymbol{M})$, within an additive factor of 4ϵ with high probability. OPT here denotes the optimal monochromatic solution cost for an input matrix $\boldsymbol{M}$. For the homogeneity maximization version, using a lower bound on the monochromatic agreement, we show that this is equivalent to a multiplicative approximation of $OPT'(1-\epsilon)$, where $OPT' = (1 - OPT)$ is the optimal agreement score. With a sample size is $O(\frac{1}{\epsilon^2})$, the run time of the basic step of our algorithm, is $O(\exp(\frac{1}{\epsilon^2}))$.

The main technical claim we prove is that given parameters $\epsilon > 0$ and $\delta > 0$, there exists some sample size, $t = t(\epsilon, \delta, K, L)$, for which for every input matrix $\boldsymbol{M}$ (of any size), if R^S and C^S are samples of size t drawn uniformly and independently from the set of rows and columns of $\boldsymbol{M}$, then for any fixed $K \times L$ bi-clustering pattern $\boldsymbol{A}$, there exists a partition of R^S into K groups and of C^S into L groups such that the $Mon_{\boldsymbol{A}}$ cost of the induced solution is ϵ-close to the $Mon_{\boldsymbol{A}}$ cost of the optimal partition with probability exceeding $1 - \delta$. With this, we get:

Theorem 5.3. *There is a randomized algorithm for the monochromatic bi-clustering problem, that given an input matrix $\boldsymbol{M}$ and an accuracy parameter ϵ, runs in time $|\boldsymbol{M}|^2 \exp\left(\frac{c}{\epsilon^2}\right)$ (for some constant c) and with high probability outputs a partition $\mathcal{P}$ of $\boldsymbol{M}$ with $Mon(\boldsymbol{M},\mathcal{P}) \leq OPT + 4\epsilon$.*

Proof. We begin by proving the following

Theorem 5.4. *On input $\boldsymbol{M}$, $\boldsymbol{A}$, K, L, ϵ, δ, with probability at least 1 - δ the monochromatic approximation algorithm (given in Algorithm 5.1), lines $(4) - (11)$ outputs a partition $\mathcal{P}$ of $\boldsymbol{M}$ such that $Mon_{\boldsymbol{A}}(\boldsymbol{M},\mathcal{P}) \leq OPT_{\boldsymbol{A}} + 4\epsilon$ where $OPT_{\boldsymbol{A}}$ is the minimal monochromatic cost with respect to $\boldsymbol{A}$ (see Equation 5.4).*

Proof. We start by analyzing the partition of the columns in Algorithm 5.1. Let $\mathcal{P}^{\star}_{\mathcal{R}} = \{R^{\star}_1, \ldots R^{\star}_K\}$ denote the optimal partition of the rows of $\boldsymbol{M}$ (with respect to pattern $\boldsymbol{A}$). We say that a sample $R^S \subset R$ is

good, if there exists a partition $\mathcal{P}^S_{\mathcal{R}} = \{R^S_1, \dots R^S_K\}$ of R^S such that, for all columns $j \in C$, except for at most $\epsilon|C|$ columns, the following holds:

$$\forall\, 1 \leq l \leq L: \quad \parallel Err(j, l|\boldsymbol{A}, \mathcal{P}^\star_{\mathcal{R}}) - Err(j, l|\boldsymbol{A}, \mathcal{P}^S_{\mathcal{R}}) \parallel \leq \epsilon \tag{5.6}$$

Namely, that in every column block l, the difference in the number of errors between the placement of the column j in the l'th block under the optimal partition of the rows, and the partition $\mathcal{P}^S_{\mathcal{R}}$ of the sample of rows, is bounded by ϵ.

Let R^S be a good sample of the rows, and let $\mathcal{P}^S_{\mathcal{R}} = \{R^S_1, \dots R^S_K\}$ be the partition of the sample for which all but a fraction of ϵ of the columns in C satisfy (5.6). Consider a column $j \in C$,

1. If j satisfies (5.6), then placing j in a greedy manner with respect to $\mathcal{P}^S_{\mathcal{R}}$ and the pattern $\boldsymbol{A}$, yields a cost increase compared to the optimal partition which is bounded by $t\epsilon \leq m\epsilon$. The number of columns is n, and therefore the cost increase in this case compared to the optimal solution is bounded by $mn\epsilon$.

2. If j does not satisfy (5.6), we can still bound the number of entries of j disagreeing with the pattern, by m, the overall number of entries. Since we assumed that R^S is a good sample, the number of such columns is bounded by $n\epsilon$. This implies a cost increase compared to the optimal solution, of $mn\epsilon$.

Altogether the number of errors incurred by the partition of the columns is bounded by $2mn\epsilon$. We can carry out the same analysis for the partition of the rows, assuming we have a good sample of the columns. The overall increase compared to the optimal solution is then bounded by $4mn\epsilon$.

We defined the monochromatic pattern cost (5.4) as a fraction of the number of mistakes made by the partition divided by the size of the matrix (mn), we can therefore conclude that the algorithm yields a solution which is at most OPT+4ϵ.

Now it suffices to show the following.

Lemma 5.3. *With probability at least 1 - δ over the random sampling, the rows and columns samples picked by the algorithm are good w.r.t the the optimal solution.*

Proof. As before we use R^S to denote a sample of the rows of size t, let $\mathcal{P}^{\star\mathcal{S}}_{\mathcal{R}}$ denote the restriction of the optimal partition $\mathcal{P}^{\star}_{\mathcal{R}}$ to the sample R^S, that is

$$\forall\, 1 \leq k \leq K \quad R^{\star S}_k = R^{\star}_k \cap R^S$$

We can focus our analysis on this specific partition of the sample of rows since the approximation algorithm is going over all possible partitions, and is therefore guaranteed to consider this one.

Let $j \in C$ be a column, we define an indicator random variable ξ^l_i for each row index $i \in R^S$ and column block index $l \in [L]$ in the following way

$$\xi^l_i = \begin{cases} 1 & \text{if } \boldsymbol{M}[i,j] \neq \star \text{ and } \boldsymbol{M}[i,j] \neq \boldsymbol{A}[\mathcal{P}^{\star\mathcal{S}}_{\mathcal{R}}(i), l] \\ 0 & \text{otherwise} \end{cases} \tag{5.7}$$

Where for a row index i, $\mathcal{P}^{\star\mathcal{S}}_{\mathcal{R}}(i)$ denotes its row block assignment in the partition $\mathcal{P}^{\star\mathcal{S}}_{\mathcal{R}}$. The variable ξ^l_i is simply the error associated with the i entry in the column j, if we place it in the column block l, given that the partition of the rows is given by $\mathcal{P}^{\star\mathcal{S}}_{\mathcal{R}}$ and the target pattern is $\boldsymbol{A}$.

It is easy to see that the random variable corresponding to the sum over ξ^l_i, for all i in the sample R^S, is simply the error function defined in (5.5).

$$\frac{1}{t}\sum_{i \in R^S} \xi^l_i = Err(j, l | \boldsymbol{A}, \mathcal{P}^{\star\mathcal{S}}_{\mathcal{R}}) \tag{5.8}$$

Using Chernoff additive bound we can guarantee that with a sample size of $t = \frac{1}{2\epsilon^2} \log \frac{L}{\delta\epsilon}$ (5.6) holds for a certain column j and column block l with probability at least $1 - \frac{\epsilon\delta}{L}$:

$$Pr(\parallel Err(j, l | \boldsymbol{A}, \mathcal{P}^{\star}_{\mathcal{R}}) - Err(j, l | \boldsymbol{A}, \mathcal{P}^{\mathcal{S}}_{\mathcal{R}}) \parallel > \epsilon) \quad \leq \quad \exp(-2\epsilon^2 t) \tag{5.9}$$

We apply the Markov's inequality to get that for all columns $j \in C$ except for not more than $\epsilon|C|$, for a specific column block j (5.6) holds with probability at least $1 - \frac{\delta}{L}$. Finally we get a guarantee of (5.6) for all blocks l with probability at least $1 - \delta$.

The same analysis applies for the approximated partition of the rows determined by a sufficiently large sample of the columns. □

□

According to Theorem 5.4, the monochromatic approximation algorithm for a given pattern $\boldsymbol{A}$, computes a partition $\mathcal{P}$ such that

$Mon_{\boldsymbol{A}}(\boldsymbol{M}, \mathcal{P}) \leq OPT_{\boldsymbol{A}} + 4\epsilon$. This in turn translates into a bound on the agreement of $1 - Mon_{\boldsymbol{A}}(\boldsymbol{M}, \mathcal{P}) \geq OPT_{\boldsymbol{A}} - 4\epsilon$. Since the algorithm goes over all possible patterns, and finally picks the pattern and partition with the lowest overall cost, the optimal pattern will be considered as well and thus the returned partition is guaranteed to have a cost $\leq OPT + 4\epsilon$ or agreement $\geq OPT - 4\epsilon$. The run time increase due to the iteration over the patterns is exponential in K, L but is constant in $|\boldsymbol{M}|$. □

Theorem 5.3 implies:

Corollary 5.1. *For every K, L there exists a randomized polynomial time approximation scheme for the $K \times L$ monochromatic bi-clustering agreement maximization problem.*

Proof. There is always a trivial solution to the monochromatic bi-clustering problem with an agreement score of at least $\frac{1}{2}$. This is simply assigning all of the rows and all of the columns to the same cluster. Note that the presence of missing entires implies that the agreement score of this trivial solution is even strictly larger than $\frac{1}{2}$. Therefore an additive 4ϵ approximation translates into a relative $(1-\epsilon)OPT$ bound on the agreement of the solution, with a fixed increase of the sample size and therefore the running time.

$$\begin{aligned} 1 - Mon(\boldsymbol{M}, \mathcal{P}) \geq \quad & OPT - 4\epsilon = OPT(1 - \tfrac{4\epsilon}{OPT}) \\ & \text{use } OPT \geq \tfrac{1}{2} \\ & \geq OPT(1 - 8\epsilon) \\ & \text{substitute } \epsilon' = 8\epsilon \end{aligned}$$

The corollary now follows from Theorem 5.3. □

Our algorithm can be viewed as generating a sample-dependent hypothesis space (the collection of "hypothesis partitions") and then picking the best behaving hypothesis in that space. As opposed to common learning procedures, in which generalization guarantees are a major concern, here we evaluate each hypothesis on the full data set (namely, the input matrix) and the focus of our analysis is on the approximation ability, or coverage, of the hypothesis spaces we work with.

Remark. The randomized PTAS for the monochromatic bi-clustering is similar to the PTAS given in Goldreich, Goldwasser, and Ron (1998)

for the dense MaxCut problem. Note that our reduction from MaxCut to the 2, 2-MCBC (see Section 5.4) applies to the hardness of finding the optimal solution. The objective value between the two instances is not preserved and thus we can not devise a PTAS for MaxCut using the PTAS for the monochromatic bi-clustering. In fact it is known that that the MaxCut problem in general graphs is APX-hard (Karp 1972; Goemans and Williamson 1995; Håstad 2001).

5.6 ENERGY MINIMIZATION FORMULATION

Factor graphs offer a convenient framework to represent dependencies between discrete random variables, provided that the global cost function can be decomposed over the factors (see Section 2.3). We show that the monochromatic bi-clustering can be written as an energy minimization problem over a factor graph. Finding the global optima of the monochromatic bi-clustering is then equivalent to the Maximum-A-Posteriori (MAP) solution (see Subsection 2.3.3 and Chapter 4). The merit of this approach is that we can leverage on existing techniques to derive efficient approximation algorithms.

Often several decompositions of the dependencies graph into factors are possible. The fundamental requirement of a factorization of the graph is that the cost function defined over the entire set of variables, can be written as a sum of factor potential functions. A basic step in the inference procedure is computing the optimal assignment of the factor variables. In general it is advantageous to have larger factors, i.e. containing a larger number of variables, as long as the optimal assignment for the factors can be found efficiently (Sontag, Globerson, and Jaakkola 2011).

We present two factorization of the monochromatic bi-clustering, the first is more intuitive and serves the purpose of establishing a solution in this framework. In contrast, the second one utilizes a smaller number of larger factors and thus allows for a more efficient solution.

For relatively large values of K and L, the run-time of the approximation algorithm presented in section Section 5.5 (PTAS), which is polynomial in the input size, but exponential in K, L, can be too high, making it impractical for most real-world bi-clustering applications. In Subsection 5.6.3 we derive a dual decomposition algorithm, which exploits the fact that in the factorized version of the problem, each subproblem can be solved efficiently. It was shown in (Sontag, Globerson,

and Jaakkola 2011) that the dual decomposition algorithm minimizes the Linear Program (LP) relaxation of the MAP problem. Solving the primal LP directly is very tricky when the factors correspond to a large number of variables, as the LP variables directly represent all possible joint configurations. In addition, in Subsection 5.6.4 we present a heuristic based on an annealing scheme with Gibbs sampling. Albeit not having any guarantees, this is a very fast method which is suitable for large datasets, or as an initialization procedure.

NOTATION Rather than thinking of a clustering as a partition, here we represent a clustering as an assignment of the rows and columns to clusters. Given an input matrix $\boldsymbol{M} \in \{0, 1, \star\}^{m \times n}$, we associate random variables with the rows and columns of $\boldsymbol{M}$. Let r_i and c_j denote random variables corresponding to the assignment of the i-th row and the j-th column. Hence, $r_i \in \{1, \ldots, K\}$ and $c_j \in \{1, \ldots, L\}$. In addition, we define auxiliary variables $b_{k,l} \in \{0, 1\}$ for $1 \leq k \leq K$ and $1 \leq l \leq L$ corresponding to the majority of the bi-cluster indexed by k, l. Note that here we use a different variable names notation (previously y_i), in order to differentiate between the row, column and block variables, which have a different semantic meaning. It is sometimes convenient to consider the aggregated set of variables, for this we use $\boldsymbol{y}$. To address the variables in the set, without differentiating them, we use y and the index t to jointly range over the variables, e.g. $y_t \in \boldsymbol{y}$.

In a factor graph $\mathcal{FG} = (\mathcal{V}, \mathcal{C}, \mathcal{E})$ representation, there are two types of nodes, variable nodes $\mathcal{V}$, and factor nodes $\mathcal{C}$. For each variable $y_t \in \boldsymbol{y}$ there is an associated node in $\mathcal{V}$. In this notation the joint optimization problem has the following form

$$\operatorname*{argmin}_{\boldsymbol{y}} \sum_{c \in \mathcal{C}} \theta_c(\boldsymbol{y}_c) \tag{5.10}$$

where the $\boldsymbol{y}_c$ is the set of variables associated with factor $c \in \mathcal{C}$, and $\theta_c(\boldsymbol{y}_c) : \mathcal{Y}_c \to \mathbb{R}$ are the factor potential functions (see Subsection 2.3.1).

In the following we define the factors and factor potential functions such that (5.10) is equivalent to the monochromatic bi-clustering definition in Definition 5.1.

5.6.1 *Decomposition with mnKL Factors*

DECOMPOSITION Here the set of factors $\mathcal{C}$ is the Cartesian product of the set of row, column and block majority variables:

$$\mathcal{C} = \{(r_i, c_j, b_{k,l})\}_{i\in[m],j\in[n],k,l\in[K]\times[L]}.$$

This implies that $|\mathcal{C}| = mnKL$, and each $\boldsymbol{y}_c$ consists of 3 variables with altogether $2KL$ configurations.

POTENTIAL FUNCTION

$$\theta_c(\boldsymbol{y}_c = (r_i, c_j, b_{k,l})) = \begin{cases} 1, & \text{if} \quad r_i = k, c_j = l \ \text{ and } \ \boldsymbol{M}[i,j] \neq b_{k,l} \\ 0, & \text{otherwise} \end{cases}$$

The definition applies to entries where $\boldsymbol{M}[i,j] \neq \star$. In fact we can omit the factors $\boldsymbol{y}_c = (r_i, c_j, b_{k,l})$, for which $\boldsymbol{M}[i,j] = \star$ for all $b_{k,l}$, from the set $\mathcal{C}$, since these entries have no effect on the assignment of the variables.

Claim 5.2. *There is a bijection between the partitions in* (5.1)*, to the solutions of* (5.10)*, restricted to the variables* $\{r_i\}_{i\in[m]} \cup \{c_j\}_{j\in[n]}$*, such that if we set the value of each* $b_{k,l}$ *to the majority of* $\boldsymbol{M}[R_k, C_l]$*, where* $R_k = \{i : r_i = k\}$*,* $C_l = \{j : c_j = l\}$*, then the monochromatic cost of a partition in* (5.1)*, is equal to the value of its mapped solution in* (5.10)*.*

Claim 5.3. *Let* $\boldsymbol{y}^* = \{r_i^*\}_{i\in[m]} \cup \{c_j^*\}_{j\in[n]} \cup \{b_{k,l}^*\}_{k,l\in[K]\times[L]}$ *be the minimizer of* (5.10)*, then* $b_{k,l}$ *is equal to the majority of* $\boldsymbol{M}[R_k, C_l]$ $\forall k \in [K], l \in [L]$*, where* $R_k = \{i : r_i = k\}$*,* $C_l = \{j : c_j = l\}$

Claim 5.2 is pretty straightforward, every partition can be easily transformed into an assignment and vice versa. Claim 5.3 says that in an optimal assignment, the value of the block majority variables have to agree with the majority computed using the rows and columns assignments. By way of contradiction, suppose that this is not the case and let $b_{k,l}$ be a block violating the claim. Without loss of generality we assume that $b_{k,l} = 0$ where the true majority value is 1. We can write (5.10) as

$$\operatorname*{argmin}_{\boldsymbol{y}} \sum_{\substack{c\in\mathcal{C} \\ r_i\neq k \text{ or } c_j\neq l}} \theta_c(\boldsymbol{y}_c) + \sum_{\substack{c\in\mathcal{C}, \\ r_i=k \text{ and } c_j=l}} \theta_c(\boldsymbol{y}_c)$$

By keeping the rows and columns assignments the same, and changing the value of $b_{k,l}$ to 1, the first sum remains the same, while if we consider the potential function definition, it is easy to see that fewer terms in the second sum evaluate to 1. We thus get a lower energy solution in contradiction to the optimality assumption.

Corollary 5.2. *Given the optimal solution for* (5.10)*, the minimizer of the monochromatic cost in* (5.1) *can be found.*

5.6.2 *Decomposition with* $\min(m,n)KL$ *Factors*

Here we show a factorization into nKL (or mKL) factors, which despite having $2K^mL$ (or $2KL^n$) possible assignments, the optimal assignment can be found efficiently.

DECOMPOSITION The set of factors is as follows:

$$\mathcal{C} = \{(r_1, \dots, r_m, c_j, b_{k,l})\}_{j\in[n],k,l\in[K]\times[L]}$$

Therefore $|\mathcal{C}| = nKL$, and each $\boldsymbol{y}_c$ consists of $m+2$ variables, with $2K^mL$ possible configurations.

POTENTIAL FUNCTION

$$\theta_c(\boldsymbol{y}_c = (r_1, \dots, r_m, c_j, b_{k,l})) = \begin{cases} \sum_{i\in[m]} \mathbf{1}_{(r_i=k \text{ and } \boldsymbol{M}[i,j]\neq b_{k,l})}, & \text{if} \quad c_j = l \\ 0, & \text{otherwise} \end{cases}$$

We assume without loss of generality that $m > n$, this naturally has an effect on the number of factors.

5.6.3 *Dual Decomposition Algorithm*

The dual decomposition approach (Bertsekas 1999; Komodakis, Paragios, and Tziritas 2007) uses Lagrange multipliers to break up the original problem into smaller subproblems, which can be solved exactly and efficiently. The Lagrange multipliers have the rule of inciting the subproblems to agree on the assignment of the variables they share (this technique was used in Subsection 4.4.4 to solve the tree weighting of the LPQP algorithm). In our case the factor variables assignment $\boldsymbol{y}_c$

are the subproblems, and the Lagrage multipliers modify the potential function θ_c so that the local maximizing assignments agree across the subproblems. The resulting dual function then gives a lower bound on the original energy.

Sontag, Globerson, and Jaakkola (2011) discuss applying the dual decomposition framework to the MAP problem and present efficient algorithms for solving it. The only required adaptation of the presented framework to the monochromatic bi-clustering, is the factor optimization. After a brief overview of the dual function derivation, we give a detailed description of the factor computation in subsubsection 5.6.3.1. We do not present the algorithms here, as it is fairly straightforward given the factor computation.

We start with the problem in (5.10) and introduce local variables z_t^c, denoting the copy of variable y_t in factor c. We use $\boldsymbol{z}_c = \{z_t^c\}_{t \in c}$ to denote the set of local variables copies belonging to factor c, and $\boldsymbol{z}^{\mathcal{C}} = \{\boldsymbol{z}_c\}_{c \in \mathcal{C}}$ the set of all variables copies.

We can re-write (5.10) as

$$\begin{aligned} &\operatorname*{argmin}_{\boldsymbol{y}, \boldsymbol{z}^{\mathcal{C}}} \sum_{c \in \mathcal{C}} \theta_c(\boldsymbol{z}_c) \\ &\text{s.t. } z_t^c = y_t \quad \forall c, t \in c \end{aligned} \tag{5.11}$$

We use Lagrange multipliers to shift the constraints into the objective. This will then allow us to minimize each factor independently. Let $\boldsymbol{\delta} = \{\delta_{c,t}(z_t^c)\}_{c \in \mathcal{C}, t \in c, z_t^c}$ be the set of Lagrange multipliers, then

$$\begin{aligned} &L(\boldsymbol{\delta}) = \min_{\boldsymbol{y}, \boldsymbol{z}^{\mathcal{C}}} L(\boldsymbol{\delta}, \boldsymbol{y}, \boldsymbol{z}^{\mathcal{C}}) = \\ &\sum_{t \in \mathcal{V}} \min_{y_t} \sum_{c : t \in c} \delta_{c,t}(z_t^c) + \sum_{c \in \mathcal{C}} \min_{\boldsymbol{z}_c} \left(\theta_c(\boldsymbol{z}_c) - \sum_{t \in c} \delta_{c,t}(z_t^c) \right) \end{aligned} \tag{5.12}$$

The Lagrange multiplier $\delta_{c,t}(z_t^c)$ can be interpreted as the message that factor c sends to variable t about its state z_t^c.

It can be shown that the value of $L(\boldsymbol{\delta})$ in the dual problem (5.12) is always a lower bound on the monochromatic cost in (5.10). When the subproblems agree on the state of all the common variables, then the assignment is guaranteed to be the optimal one.

5.6.3.1 *Efficiently Minimizing The Subproblems*

Sontag, Globerson, and Jaakkola (2011) present sub-gradient descent as well as block coordinate descent algorithms for the solution of (5.12).

The main challenge in the implementation of both algorithms is efficiently finding a solution for the subproblems, i.e.

$$\min_{\boldsymbol{z}_c} \left(\theta_c(\boldsymbol{z}_c) - \sum_{t \in c} \delta_{c,t}(z_t^c) \right). \tag{5.13}$$

Finding the optimal assignment of (5.13) in the first factorization presented in Subsection 5.6.1 is easy. Simply modify the truth table of size $2KL$ by subtracting from each entry the three values $\delta_{c,t}(z_t^c)$ corresponding to the states of the three variables in the assignment. Then the optimal assignment is the one corresponding to the minimal truth table entry.

For the factorization given in Subsection 5.6.2 finding the optimal assignment according to (5.13) can not be done in a straightforward manner, as the number of possible assignments is exponential. Fortunately for most of the joint assignments, the single variable assignments are independent. We can divide the joint assignments into three groups, each of which allows for an efficient computation of the best assignment in the group by combining the best assignment for the single nodes. The overall optimal assignment is the minimal cost assignment out of the three groups.

For $\boldsymbol{z}_c = \{r_1^c, \ldots, r_m^c, c_j^c, b_{k,l}^c\}$:

CASE 1: $c_j^c \neq l$ For all such assignments, $\theta_c(\boldsymbol{z}_c) = 0$ such that the optimal assignment is simply $\forall t \in c :$ $\operatorname{argmin} -\delta_{c,t}(z_t^c)$. Except for the column variable c_j^c with the optimal assignment being $\operatorname{argmin}_{l' \neq l} -\delta_{c,j}(z_{j,l}^c)$.

CASE 2: $c_j^c = l$ AND $b_{k,l}^c = 0$ In this scenario, the assignment of the column and block majority is fixed. For each of the row variables r_i^c, in case $\boldsymbol{M}[i,j] = 0$, its optimal assignment is the same as in case 1. Otherwise it is given by

$$r_i^c = \begin{cases} \operatorname{argmin}_{k' \neq k} -\delta_{c,i}(z_{i,k'}^c) & \text{if } \min_{k' \neq k} -\delta_{c,i}(z_{i,k'}^c) < -\delta_{c,i}(z_{i,k}^c) + 1 \\ k & \text{otherwise .} \end{cases} \tag{5.14}$$

where the $+1$ for setting $r_i^c = k$, is the cost added due to the potential function $\theta_c(\boldsymbol{z}_c)$.

CASE 3: $c_j^c = l$ AND $b_{k,l}^c = 1$ This case is analogous to case 2, only here for row variable r_i^c, if $\boldsymbol{M}[i,j] = 1$ it has the same assignment as in case 1 and otherwise it is given by (5.14).

Example 5.1 (Optimal factor computation (5.13))**.** Here $\boldsymbol{M} \in \{0,1\}^{8\times 5}$, $K = 3$, $L = 3$, the input matrix and dual variables are given by:

$$\boldsymbol{M}: \begin{pmatrix} 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$\delta_{c,t(r_1^c)}$:	-1.21	1.01	1.47
$\delta_{c,t(r_2^c)}$:	-0.09	1.22	0.24
$\delta_{c,t(r_3^c)}$:	-0.18	1.22	-0.03
$\delta_{c,t(r_4^c)}$:	-1.63	2.03	1.22
$\delta_{c,t(r_5^c)}$:	-0.39	1.59	-0.93
$\delta_{c,t(r_6^c)}$:	1.32	1.16	0.71
$\delta_{c,t(r_7^c)}$:	-0.81	-1.46	-0.25
$\delta_{c,t(r_8^c)}$:	0.30	0.68	1.85

$\delta_{c,t(c_1^c)}$:	1.33	0.58	1.37

$\delta_{c,t(b_{1,3^c})}$:	-0.24	-0.02

We compute (5.13) for factor $\boldsymbol{z}_c = \{r_1^c, \ldots, r_8^c, c_1^c, b_{1,2}^c\}$

CASE 1: $c_1^c \neq 2$ The optimal assignment is found independently, we get $r_1^c, \ldots, r_8^c = (3,2,2,2,2,1,3,3)$, $c_1^c = 3$ and $b_{1,2}^c = 1$. The overall cost is -11.76, which is simply the sum of all the individual assignments.

CASE 2: $c_1^c = 2$ AND $b_{1,2}^c = 0$ Out of the rows that are assigned to cluster 2 in case 1, rows 2 and 3 have $1 (\neq b_{1,2}^c)$ in $\boldsymbol{M}[:,1]$. For row 2 the additional cost for violating the cluster majority, makes the optimal assignment change from 2 to 3. This is not the case for row 3 which icurs the loss but the assignment remains the same. Altogether the rows assignment is $r_1^c, \ldots, r_8^c = (3,3,2,2,2,1,3,3)$ and the cost is -8.45

CASE 2: $c_1^c = 2$ AND $b_{1,2}^c = 1$ Here rows 4,5 which were assigned to cluster 2 in case 1, incur an additional loss of 1. For row 4 this causes to change the assignment to cluster 3. Thus $r_1^c, \ldots, r_8^c = (3,2,2,3,2,1,3,3)$ and the total energy is -9.17

The optimal assignment and cost for this factor is then given in case 1.

Remark (Relation to Linear Program (LP) relaxation). As discussed in (Sontag, Globerson, and Jaakkola 2011), the Lagrangian relaxation in (5.12) is the convex dual of a LP relaxation of the original problem (see Subsection 4.2.1 for a detailed explanation of the LP approach). The connection to the LP is important as it allows us to understand the theoretical framework of the solution. Optimality is guaranteed when all the subproblems agree, which is equivalent to having an integer solution to the LP. Otherwise when the LP is not tight, the Lagrangian relaxation provides a lower bound. Note that solving the LP directly involves enforcing the local polytope constraints. For a row variable r_i the constraints are

$$\mathcal{L}_{\mathcal{G}} = \left\{ \boldsymbol{\mu} \;\middle|\; \begin{array}{ll} \sum_k \mu_{i;k}(r_i) = 1 & \\ \sum_{\boldsymbol{y}_{c \setminus i}} \mu_c(\boldsymbol{y}_c) = \mu_{i;k}(r_i) & \forall k,\ c : i \in c \end{array} \right\}.$$

which at least in the decomposition into larger factors, involves an exponential number of summation constraints.

5.6.4 *Annealing Scheme Solution With Gibbs Sampler*

We present a solution to (5.10) by means of an annealing scheme with sampling from a Gibbs distribution. This approach was intially introduced for MAP inference in (Geman and Geman 1984), in the context of an image restoration application. It was since then successfully applied to many difficult combinatorial optimization problems in various fields (Buhmann and Puzicha 2001).

The annealing scheme alternates between updating the three blocks of variables: row assignments $r_1, \dots r_m$, column assignments $c_1, \dots c_n$ and the bi-cluster majorities $b_{1,1}, \dots, b_{k,l}$.

Keeping the assignment of the columns and block majorities fixed, the rows assignment is sampled according to the following probabilities:

$$\begin{aligned} P(r_i = k | c_1, \dots c_n,\ b_{1,1}, \dots, b_{k,l}) &\propto \exp(-\frac{1}{T} \sum_{c : i \in c} (\boldsymbol{y}_c)) \\ &= \exp(-\frac{1}{T} \sum_{j \in [n]} |\boldsymbol{M}[i,j] - b_{k,c_j}|) \end{aligned}$$

The column probabilities are given as follows:

$$\begin{aligned} P(c_j = l | r_1, \dots r_m,\ b_{1,1}, \dots, b_{k,l}) &\propto \exp(-\frac{1}{T} \sum_{c: j \in c} (\boldsymbol{y}_c)) \\ &= \exp(-\frac{1}{T} \sum_{i \in [m]} |\boldsymbol{M}[i,j] - b_{r_i,l}|) \end{aligned}$$

and for $v \in \{0,1\}$, the bi-cluster majority probabilities are

$$\begin{aligned} P(b_{k,l} = v | r_1, \dots r_m,\ c_1, \dots c_n) &\propto \exp(-\frac{1}{T} \sum_{c: t=(k,l) \in c} (\boldsymbol{y}_c)) \\ &= \exp(-\frac{1}{T} \sum_{x_i = k, y_j = l} |\boldsymbol{M}[i,j] - v|). \end{aligned}$$

T is a temperature parameter that is decreased during the run of the algorithm in order to eventually only sample low-cost configurations. In case entries in $\boldsymbol{M}$ are missing, the corresponding elements do not contribute to the costs.

5.7 EXPERIMENTS

In this section we empirically evaluate the dual decomposition and Gibbs annealing scheme solution approaches on synthetic data. We use DD to denote the dual decomposition algorithm described in Subsection 5.6.3, and GS for the Gibbs annealing algorithm presented in Subsection 5.6.4.

Our DD solver minimizes the dual objective given in (5.12), using a simple gradient descent scheme described in (Sontag, Globerson, and Jaakkola 2011). It essentially amounts to computing (5.6.3.1) for each factor at each iteration.

In Subsection 5.7.1 we explain the experimental setup and the data generation procedure, and in Subsection 5.7.2 we present and discuss the results.

5.7.1 *Data and Experimental Setup*

For a given matrix size (m, n), parameters (K, L), a bias term β and a noise-level σ, we generate a data instance $\boldsymbol{M}_{\beta,\sigma}$ in the following way:

1. We generate a binary pattern, or block majorities matrix of size (K, L) by sampling each entry from a Bernoulli distribution with parameter β.

2. After fixing the pattern, for each row and column we choose uniformly at random an assignment to a row cluster $1 \leq k \leq K$, and column cluster $1 \leq l \leq L$ respectively.

3. Finally, for each entry $\boldsymbol{M}_{\beta,\sigma}[i, j]$, we choose the value matching the assignment of i and j with probability $1 - \sigma$, and with probability σ we choose the other value.

We report the results on increasing dataset sizes: $(m, n) = \{(32, 32), (75, 75), (120, 120), (154, 154)\}$ with number of clusters: $K = L = \{4, 5, 6, 7\}$ respectively. We use two bias values $\beta = \{0.3, 0.5\}$, and 8 values in the interval $[0.01, 0.3]$ for the noise-level σ.

We point out the following:

- The smallest instance has a solution space of size $\sim 2^{120}$, thus the cost of the optimal solution for each instance can only be approximated by the noise level σ we introduce.

- Smaller bias values result in more difficult instances. Intuitively, when the bias is low the input matrix as well as the pattern are more sparse (with respect to either 0 or 1), and therefore wrong assignments can be much more costly.

- In terms of the noise level, we expect to be able to find more similarly scoring solutions as the noise level increases.

A step-size update scheme has to be set for the gradient descent DD algorithm. There exist many heuristics for this choice, the basic step size requirements, denoted here as α_t, are: $\alpha_t > 0$, $\sum_{t=0}^{\infty} \alpha_t = \infty$ and $\lim_{t \to \infty} \alpha_t = 0$ (Komodakis, Paragios, and Tziritas 2011; Sontag, Globerson, and Jaakkola 2011). In our experiments, at iteration t we use the heuristic: $\alpha_t = \frac{\alpha_0}{\sqrt{t}}$ mentioned in (Komodakis, Paragios, and Tziritas 2011). For the initial step-size α_0 we do a grid search over the values: $\{0.005, 0.01, 0.25, 1\}$.

For running the GS solution, we need to set the initial parameter T and the update rule. We did a grid search over $\{0.005, 0.01, 0.1, 1, 5\}$

for the initial $1/T$ parameter, with an additive update rule, using the values $\{0.005, 0.01, 0.05, 0.1, 0.5, 1, 5\}$.

We ran both algorithms until convergence (determined by a fixed number of iterations with no change to cost) with a limit of 8 hours.

Finally for each dataset parameters we generated 5 data instances and we report the average of the best DD and GS solution, over the algorithms parameters, for each instance.

5.7.2 *Experimental Results*

The monochromatic solution cost of the DD and GS solvers are plotted in Figure 5.3, on datasets with increasing values of noise-level. The plots on the left column correspond to datasets generated using a bias value of $\beta = 0.3$, and the ones on the right using $\beta = 0.5$. The rows are ordered according to increasing dataset sizes and number of clusters.

Both solvers attain almost equally good solutions on the "easier" instances. On the more difficult instance. e.g. with lower bias, lower noise and larger dataset sizes, the DD solver is finding better solutions compared to the GS algorithm. In terms of run-time, the GS algorithm is significantly faster. The GS algorithm is however more sensitive to the choice of parameters, such that multiple runs are required. Still, altogether the GS is about two order of magnitude faster.

Since the DD algorithm is solving the LP relaxation of the monochromatic objective, we can at least empirically conclude that the LP objective is tight on most of the instances (see the remark at the end of subsubsection 5.6.3.1).

5.8 CONCLUSIONS

We introduce a formulation of bi-clustering which is both natural, in the sense that it captures the aim of most existing bi-clustering works, and offers a convenient framework to analyze the bi-clustering task from a theoretical point of view. We show that the resulting optimization problem is NP-hard but can be approximated up to any multiplicative factor in polynomial time. We also provide two efficient solvers, based on a novel energy minimization formulation. The nature of the bi-clustering task as an unsupervised learning method, makes it hard to compare different approaches aiming at optimizing different cost

functions. As future work we would like to apply the monochromatic solvers to real-data and perhaps use the emerging patterns as basis for comparison.

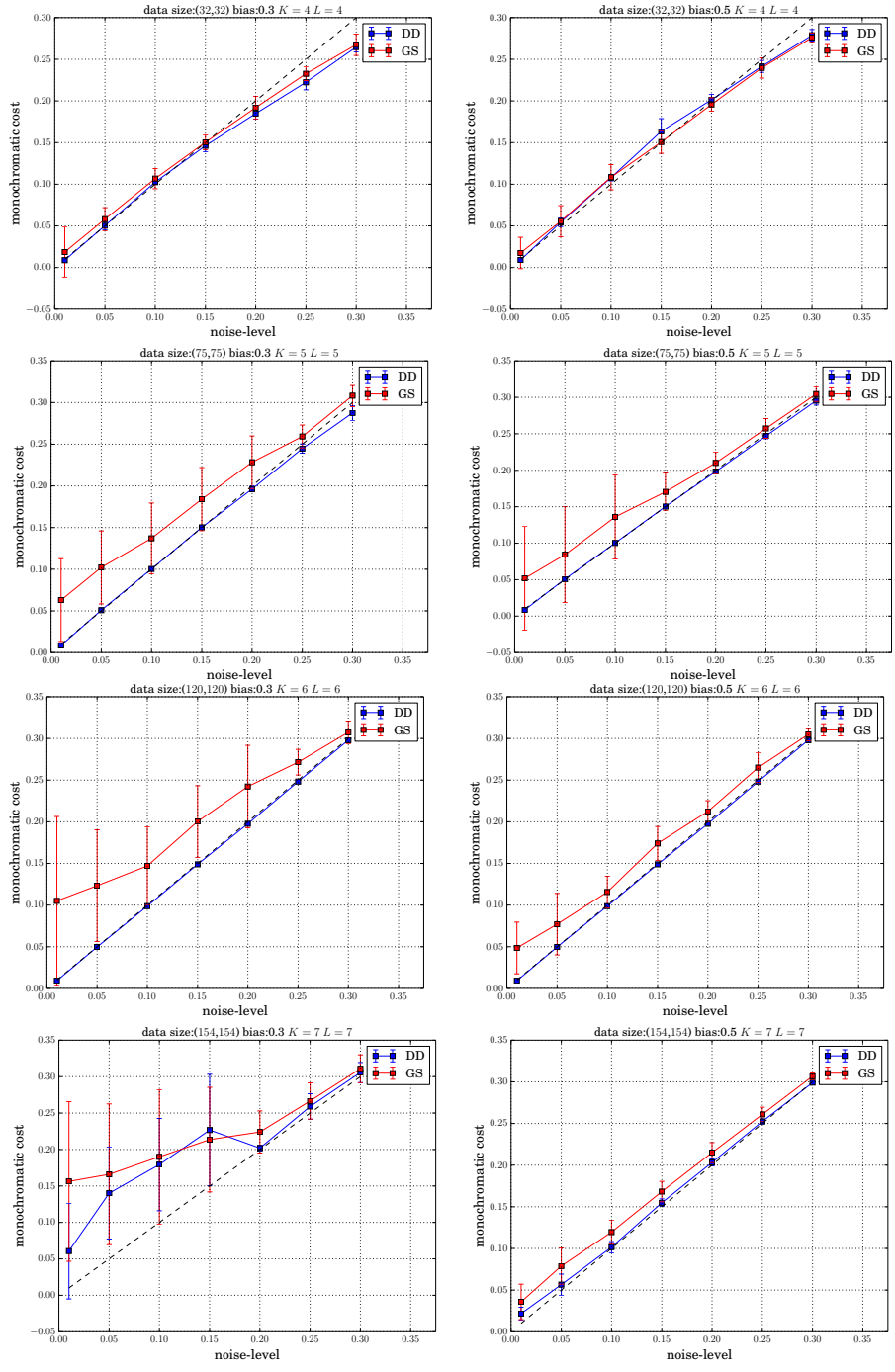


Figure 5.3: Comparison of the DD and GS solvers average monochromatic solution costs on synthetic data. The data generation procedure is explained in detail in Subsection 5.7.1.

6

PLAL: CLUSTER BASED ACTIVE LEARNING

As opposed to fully supervised learning, in Active Learning (AL) the input to the learning algorithm is an unlabeled data sample. The goal of AL is to find a classifier with low error, while minimizing the number of required labels. In many applications acquiring labels for the training examples is costly. For example in the sensor based vehicle detection problem (see Chapter 7), the labels for the sensor's measurements are obtained by annotating low resolution images, where an automatic detection does not provide the necessary accuracy. An AL algorithm which given the unlabeled sensor readings, presents only a fraction of the corresponding images to a human annotator, is very useful.

In this chapter we introduce the PLAL algorithm, an AL algorithm that exploits data clusterability. We present an analysis of its sample complexity in terms of a notion called Probabilistic Lipschitzness (PL). This chapter presents some of the results published in Urner, Wulff, and Ben-David (2013), extensions of this work appear also in Urner (2013). In Section 7.5 we apply the PLAL algorithm to the vehicle detection problem data and show that it can be used to reduce the image annotation efforts.

6.1 INTRODUCTION

Active Learning (AL) paradigms are successful in practice and there is also a variety of theoretical studies analyzing the possibilities and limitations of AL. However, several studies have shown that, under worst-case scenarios, AL algorithms are bound to require as many labeled sample points as their "passive" fully supervised counterparts (Dasgupta 2005; Kääriäinen 2006; Beygelzimer, Dasgupta, and Langford 2009). Those negative results set the frame for most of the theoretical research on AL. Rather than trying to show that active choice of label queries can *always* reduce the number of training labels, one aims to identify properties of the learning task under which an AL paradigm is beneficial.

For many label prediction tasks, there is a significant correlation between the (marginal) distribution over the data points and the labels. Under a suitable data representation, or feature choice, we expect that the closer two instances are, the less likely they are to have different labels. Probabilistic Lipschitzness (PL) is a measure that quantifies this correlation. It can also be viewed as a way to model the cluster assumption, which is often invoked in the context of semi-supervised learning.

Our algorithm (PLAL) follows a paradigm proposed by Dasgupta and Hsu (2008) for exploiting cluster structure for active learning. While most previous work on the theory of active learning focused on an efficient version space reduction for learning a hypothesis class, (Dasgupta and Hsu 2008) suggest a labeling procedure based on a hierarchical clustering of the training data. The authors show that assuming that the learner is given a "good" hierarchical clustering, an unlabeled sample can be labeled almost correctly with relatively few label queries. They suggest to then feed the now labeled sample to any standard learning procedure. In this paper we analyze a version of their approach under the assumption of PL. This condition is weaker than the availability of a "good" clustering tree in that we only need it for the analysis of our procedure (as opposed to the need for a successful preprocessing step that finds the good clustering tree). We believe that cluster-based active learning is an important research direction that has not received enough attention from the learning theory community so far.

Our main results show upper bounds for the sample complexity of PLAL-based active learning and lower bounds for the sample complexity of standard (passive) learning under similar assumptions. In particular, we show that under polynomial rates of PL, PLAL significantly reduces the sample complexity of some VC-classes.

In Section 6.3 we explain and discuss the notion of PL. The sample labeling procedure, PLAL is presented in Section 6.4. An overview of the theoretical analysis of the PLAL algorithm is given in Subsection 6.4.2. Finally we present experimental results on artificial data in Section 6.5.

6.2 RELATED WORK

The main contributions of this work are in the context of the theoretical analysis of AL sample complexity. For a general survey of trends and algorithms in AL we refer the reader to (Settles 2009).

The survey "Two faces of active learning" by Dasgupta (2011), contrasts two general approaches for active learning: Using label queries to more efficiently search through a hypothesis space and exploiting cluster structure in data. Almost all of the theoretical work so far has focused on the first setup. Starting with (Dasgupta 2004) there is a large body of work that analyzes these paradigms in the realizable case and under separability with a margin assumption ((Balcan, Broder, and Zhang 2007; Balcan, Hanneke, and Vaughan 2010; Gonen, Sabato, and Shalev-Shwartz 2011)).

There have been extensive efforts to generalize the positive results for active learning from the realizable to the agnostic case. Lower bounds of $\Omega(\frac{1}{\epsilon^2})$ by Kääriäinen (2006) and Beygelzimer, Dasgupta, and Langford (2009) imply that improvements in label complexity for learning a hypothesis class are not possible in general.

Thus research focuses on identifying parameters that characterize learning tasks where active learning is beneficial. So far the most prominent such parameter is the disagreement coefficient, introduced in (Hanneke 2007). It was used to bound the label complexity of various querying strategies (Hanneke 2007; Dasgupta, Hsu, and Monteleoni 2008; Beygelzimer et al. 2010; Beygelzimer, Dasgupta, and Langford 2009). However, the bounds on the number of label queries in these papers all involve the approximation error of the hypothesis class. They become relevant only when the approximation error is small.

A first approach at exploiting cluster structure by active learning was presented in (Dasgupta and Hsu 2008). As mentioned earlier this work proposes a labeling strategy for an unlabeled dataset, where the learner is also given a hierarchical clustering of the data. A bound on the number of label queries provided in this work depends on the depth of the effectively used clustering tree; however, it is unclear how to control this parameter.

We provide an analysis of the label reduction under the assumption of PL. A version of the PL parameter was introduced by Steinwart and Scovel (2007) under the name geometric noise exponent. (Steinwart and Christmann 2008) show that when such a parameter (here called margin

exponent) is combined with bounds on the noise rate and marginal distribution near the decision boundaries of data, it can be used to bound the approximation error of Gaussian kernels for that data. The definition of PL used in this work was introduced in (Urner, Ben-David, and Shalev-Shwartz 2011), where it was used to formally establish the merits of unlabeled data for semi-supervised learning.

A framework, where an unlabeled sample is labeled by a preliminary, active labeling procedure and then fed to a standard learner was introduced in (Hanneke 2012). Assuming the data is realizable by a vc-class, the author presents a labeling procedures based on repeated computations of the shatter function of version spaces and shows how this labeling procedure reduces the label complexity of the original standard learner. Our PLAL procedure achieves label complexity reduction results for data with bounded PL, which we believe is a more realistic assumption, and is substantially simpler and easy to implement.

6.3 DEFINITIONS AND NOTATIONS

For concreteness we focus in this chapter on domains $\mathcal{D} = [0,1]^d$ for some dimension d, and on the binary classification settings, i.e the label set is $\{0,1\}$. We use P to denote the set of distributions over $\mathcal{D} \times \{0,1\}$, and $P_{\mathcal{D}}$ the marginal distributions. We assume that there exist a deterministic labeling function $l : \mathcal{D} \to \{0,1\}$.

6.3.1 *Active Learning*

An active learner receives an unlabeled sample $S = (x_1, \ldots, x_n)$ generated *i.i.d.* by $P_{\mathcal{D}}$. The active learner can then sequentially query labels for points in S, i.e. the learner chooses indices $i_1, \ldots, i_m \in \{1, \ldots n\}$ and receives the labels $l(x_{i_1}), \ldots, l(x_{i_m})$. At each step, the choice of each i_j can depend on S and the labels seen so far. Based on the unlabeled sample S and the queried labels, the learner outputs a hypothesis.

Recall the definition of learning given in Definition 2.2 , we say that an algorithm $\mathcal{A}$ actively learns some hypothesis class H over $\mathcal{D}$ with respect to a set of distribution $\mathcal{Q}$ over $\mathcal{D} \times \{0,1\}$, if there exist functions $n_u : (0,1) \times (0,1) \to \mathbb{N}$, $n_l : (0,1) \times (0,1) \to \mathbb{N}$, such that, for all $\epsilon, \delta \in (0,1)$, for all distributions $P \in \mathcal{Q}$, with probability at least $1 - \delta$ over an *i.i.d.* unlabeled $P_{\mathcal{D}}$-generated sample S of size $n_u(\epsilon, \delta)$,

the algorithm $\mathcal{A}$ queries at most $n_l(\epsilon, \delta)$ members of S for their labels and $\mathrm{Err}_P(\mathcal{A}(S)) \leq \mathrm{Err}_P(H) + \epsilon$. Given a function n_u for the size of the unlabeled sample, we say that $\mathcal{A}$ has labeled sample complexity or label complexity n_l with respect to n_u for the smallest function n_l such that the pair of functions (n_l, n_u) that satisfies the above condition. The minimum such function n_l for which there exists a function n_u such that the pair of functions (n_l, n_u) satisfies the above condition is called the labeled sample complexity of $\mathcal{A}$ for actively learning H with respect to $\mathcal{Q}$ and denoted by $n^{\mathrm{active}}[\mathcal{A}, H, \mathcal{Q}]$.

We investigate the sample complexity as a function of $\frac{1}{\epsilon}$. Whenever we use Landau-notation to denote some function growth behavior, this function is considered as a function ϵ only (we consider the asymptotic behavior as ϵ tends to 0), and we omit log-factors.

6.3.2 *Probabilistic Lipschitzness*

Probabilistic Lipschitzness (PL) can be viewed as a way of formalizing the clusterability assumption of the data, an assumption that is often made to account for the success of semi-supervised learning. It implies that the data can be divided into clusters that are almost label-homogeneous and are separated by low-density regions.

Definition 6.1 (Probabilistic Lipschitzness)**.** Let $\phi : \mathbb{R} \to [0, 1]$. We say that $f : \mathcal{D} \to \mathbb{R}$ is ϕ-Probabilistic Lipschitz with respect to a distribution $P_{\mathcal{D}}$ over $\mathcal{D}$ if for all $\lambda > 0$:

$$\Pr_{x \sim P_{\mathcal{D}}} \left[\Pr_{x' \sim P_{\mathcal{D}}} \left[|f(x) - f(x')| \; > \; \frac{1}{\lambda} \|x - x'\| \right] \; > \; 0 \right] \; \leq \; \phi(\lambda)$$

If, for some $P = (P_{\mathcal{D}}, l)$, the labeling function l is ϕ-Lipschitz, then we say that P satisfies the ϕ-Probabilistic Lipschitzness. We denote the set of all such distributions over $[0, 1]^d$ by $\mathcal{Q}^d_\phi$. Given some PL-function ϕ and some ϵ, we let $\phi^{-1}(\epsilon)$ denote the smallest λ, such that $\phi(\lambda) \geq \epsilon$.

If a distribution P is ϕ-Lipschitz for some function ϕ, then there always exists a non-decreasing function $\phi' \leq \phi$ (pointwise) such that P is also ϕ'-Lipschitz. We will thus implicitly assume that ϕ is non-decreasing for all PL-functions ϕ considered in this work.

If a distribution $P = (P_{\mathcal{D}}, l)$ is ϕ-Lipschitz, then the weight of points x that have a positive mass of points of opposite label in an λ-ball

around them, is bounded by $\phi(\lambda)$. This definition relaxes the standard definition of Lipschitzness. Namely, for points x and x' at distance smaller than λ with opposite labels, the standard Lipschitz condition for Lipschitz constant $\frac{1}{\lambda}$ is violated as

$$|l(x) - l(x')| = 1 > \frac{1}{\lambda}\|x - x'\|$$

Thus, if the labeling function l of a distribution is L-Lipschitz then it satisfies the PL with the function

$$\phi(\lambda) = \begin{cases} 1 & \text{if } \lambda \geq \frac{1}{L} \\ 0 & \text{if } \lambda < \frac{1}{L} \end{cases}$$

Examples of PL distributions can be found in (Steinwart and Christmann 2008) or (Urner, Ben-David, and Shalev-Shwartz 2011).

We analyze the label complexity of AL assuming that the distribution has a bounded PL.

6.4 THE PLAL LABELING PROCEDURE

We begin with describing the general framework of the algorithm, suggested in (Dasgupta and Hsu 2008), assuming that a hierarchical clustering (cluster tree) of the unlabeled data is given.

The idea is to descend down the tree, starting from the root (corresponding to the entire dataset), and at each level estimate the current cluster's label homogeneity. The estimation is done by choosing data points uniformly at random and obtaining their true labels. If based on the sample a cluster can be considered label homogeneous with sufficiently high confidence, all remaining unlabeled points within the cluster are assigned the majority label, and no more points from this cluster will be queried.

If based on the sample the cluster appears to be label heterogeneous, the procedure continues descending down its children in the cluster tree. Since the cluster tree is fixed before any of the labels were seen, the algorithm can reuse labels from the parent cluster (the induced subsample can be considered a sample that was chosen uniformly at random from the points in the child-cluster) without introducing any sampling bias. A nice overview of this procedure can be found in (Dasgupta 2011).

The PLAL algorithm analyzed here follows the same general procedure, with a fixed spatial tree as the hierarchical clustering (see Definition 6.2). To obtain a concrete algorithm from the general framework, we also need to specify, how many points to query per cluster and in which order to choose the clusters. We describe our version of this labeling procedure in the next subsection.

The analysis of the algorithm in (Dasgupta and Hsu 2008) assumes that the given clustering has a label homogeneous pruning, consisting of a relatively small number of tree-node clusters. Our analysis depends on the rate in which the diameters of the clusters shrink. Using the PL assumption, we can turn such cluster-diameter bounds into error bounds and label query bounds of the procedure. The rate in which cluster diameters shrink was analyzed for spatial trees in (Verma, Kpotufe, and Dasgupta 2012).

6.4.1 *The algorithm*

Definition 6.2. A spatial tree T is a binary tree, where each node, also called here a cell, corresponds to a subset of the domain. The root, $\text{Root}(T)$, corresponds to the entire domain, e.g $\mathcal{D} = [0,1]^d$. In addition, for each node (cell) C, the children $\text{Left}(C)$ and $\text{Right}(C)$ form a 2-partition of C.

According to the definition, for each level k (distance from the root), the nodes at this level form a 2^k-partition of the space. For a sample S, a spatial tree induces a hierarchical clustering of S with clusters $S \cap C$ for the nodes C in the tree.

A pseudo-code of the PLAL algorithm is given in Algorithm 6.1. The input to the algorithm is an *i.i.d.* sample $S \in \mathcal{D}$. The algorithm maintains a partition of the space into active and inactive cells. Initially there is only one active cell, containing all the sample and corresponding to the root of the spatial tree T, which is also the entire unit cube $[0,1]^d$.

Each iteration of the PLAL algorithm corresponds to a level of the spatial tree. For each level the algorithm queries sufficiently many labels from the S points in each of the active cells, to detect if the cell is label heterogeneous (the next paragraph gives a detailed explanation of this method $C.query()$). A label homogenous cell, for which all of the queried points within the cell have the same label, is declared inactive and the remaining sample points in the cell are assigned that label. For

Algorithm 6.1 PLAL labeling procedure

Input: unlabeled sample $S = \{x_1, \ldots, x_n\}$, spatial tree T, parameters ϵ, δ
level $= 0$
$active\_cells[0].append(\text{Root}(T))$
while $active\_cells[\text{level}]$ not empty **do**
  $q_{\text{level}} = \frac{\text{level} \cdot 2 \cdot \ln(2) + \ln(1/\delta)}{\epsilon}$
  for all C **in** $active\_cells[\text{level}]$ **do**
    $C.query(q_{\text{level}})$
    if all labels seen in C are the same **then**
      label all points in $C \cap S$ with that label *(cell C now becomes inactive)*
    else
      if there are unqueried points in $C \cap S$ **then**
        $active\_cells[\text{level} + 1].append(\text{Right}(C), \text{Left}(C))$
      end if
    end if
  end for
  level $=$ level $+ 1$
end while
Return: labeled sample $\{(x_1, y_1) \ldots, (x_n, y_n)\}$

a label heterogeneous cell, the children of the cell in T are added to the list of active cells for the next round, if they still contain unlabeled points.

For a cell C, method $C.query(q)$ queries the labels of the first q sample points in the cell. For this, it reuses labels of points that were queried in earlier rounds (i.e. does not actually query those). If the cell contains fewer than q sample points, the labels of all unlabeled points among these are queried and the cell is declared inactive. In this case, it is not important whether the cell is label homogeneous or label heterogeneous, as the algorithm does not infer labels for any of the points and thus all the labels of points in such cells are correct labels. Note that "declaring a cell inactive" is implicit in the code shown in Algorithm 6.1. Only cells that are heterogeneous and contain unlabeled points, their children are added to the list of active cells for the next iteration.

At the end of the procedure all sample points in S are labeled. Each point was either queried, or given an induced label by the homogeneous

declared cell it resides in. Only in the latter case, a point might possibly have obtained an erroneous label.

6.4.2 *Overview of the Error and Sample Complexity Bounds*

In this section we provide an overview of the analysis and label complexity results of the proposed PLAL algorithm. For a formal presentation of the theorem and proofs, we refer the reader to (Urner, Wulff, and Ben-David 2013).

The analysis of the PLAL algorithm includes:

1. An error bound, showing that by choosing the query numbers $q_k = \frac{k \cdot 2 \cdot \ln(2) + \ln(1/\delta)}{\epsilon}$, at level k, with probability at least $(1-\delta)$, the PLAL algorithm labels at least $(1-\epsilon)n$ many points from S correctly.

2. A general bound on the number of queries the algorithm makes on an unlabeled sample of size n under the PL assumption. This bound is given in terms of ϵ, δ and the "deepest" level to which the algorithm has to descend in the partition. Our analysis links this level to the PL regime.

3. Specific bounds on the expected number of queries for polynomial ($\phi(\lambda) = \lambda^t$) and exponential ($\phi(\lambda) = e^{\frac{-1}{\lambda}}$) PL regimes, using dyadic trees in the partition. In addition, by giving a lower bound on the sample complexity of "passive" learning algorithms under the same PL regimes, our analysis shows that using PLAL as a pre-procedure leads to label savings.

In (Urner, Wulff, and Ben-David 2013) there is a further analysis of the robustness of ERM and RLM learning algorithms to a labeled sample with a bounded error. The analysis shows that the final classification error does not increase by much compared to having the correct labels for all sample points. Therefore using PLAL before a second round of classification is "safe".

6.5 EXPERIMENTS

We designed experiments on synthetic data to empirically evaluate two aspects of the PLAL labeling framework. The first is how well

the PLAL algorithm performs in terms of maintaining low prediction error while reducing the number of required labels. We compare the prediction error induced by a supervised classifier trained on the subset of the data requested by the PLAL, to the error induced by a classifier trained on randomly sampled subset of equal size. The second aspect and perhaps the more interesting one, is how the reduction in labeled sample size relates to the (empirical) Probabilistic Lipschitzness (PL) of the data. To address this question without assuming access to the true data generating distribution, we use an estimator which adheres to a PL definition in which the probability of finding a λ-close point with different label, is bounded (but is not necessarily zero).

6.5.1 *Synthetic Data Description*

We generated 3 datasets, each consisting of 2000 samples from a mixture of multivariate Gaussian distributions. The distributions included 4 dense Gaussian, as well as 4 sparse Gaussian, with the same parameters governing the density of each group, but each dataset having different sets of values. We used a different label for the samples associated with each Gaussian, resulting in a multi-label classification task with 8 labels. While the covariance matrix parameters ("dense" and "sparse"), of the each dataset were fixed, we varied the dimensionality of the generated data in the different experiments. We always sampled the dense Gaussian means close to the "corners" of the space, whereas the means of the sparse ones we sampled uniformly at random. This procedure essentially allowed us to create datasets exhibiting different empirical PL behaviors, by varying the covariances of the dense and sparse Gaussian of each dataset. We used diagonal covariance matrices to avoid extra noise and sampled 80% of the points from the dense Gaussian, and the remaining 20% from the sparse ones. The datasets are denoted by A,B, and C corresponding to: A-0.1 dense variance and 1 sparse variance, B-.01 dense variance and .1 sparse variance, and C-.001 dense variance and .1 sparse variance. With this choice of parameters the datasets can be intuitively casted as the most clusterable being C to the least clusterable, or least separable, being A.

6.5.2 *Empirical Probabilistic Lipschitzness*

We plotted the empirical PL of the datasets A,B, and C with dimensions 5, 15 and 25. The λ values range between $0-10$, for each λ value we calculated the empirical $\phi(\lambda)$ as the percentage of data points having at least λ close neighbor with a different label. The results are shown in Figure 6.1.

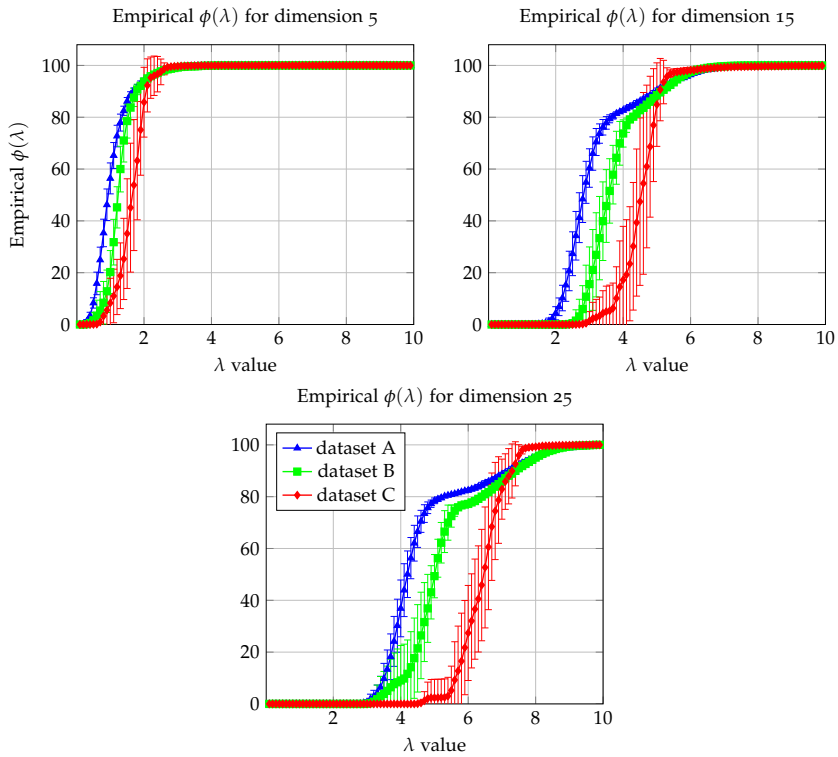


Figure 6.1: The empirical $\phi(\lambda)$ as a function of λ in the range $0-10$ for datasets A,B, and C described in Subsection 6.5.1

6.5.3 *Classification*

In this experiment we varied ϵ in the range $(0.01, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3)$, and for each ϵ value we computed the PLAL queries. We sampled uniformly at random an equal number of points to serve as a benchmark. We used a K Nearest Neighbor (NN) classifier to compute predictions

on a test set using the PLAL queries as well as the randomly sampled ones. We used K values in the range $(1, 3, 5, 10)$, and chose the best K for every run. We repeated this procedure 5 times and we report the average values for each configuration. We computed the prediction error as the percentage of labels which differ between the predictions and the true ones. The results on the datasets A,B, and C with dimensions 5, 15 and 25 are shown in Figure 6.2.

The average number of queries requested by PLAL is plotted in red and is denoted as $\% - \mathit{queries}$. The prediction error of the NN classifier with PLAL queries is denoted as % NN-PLAL-error whereas the prediction error of the NN classifier with random queries is denoted as % NN-random-error.

The plots confirm the intuition that the PLAL labeling framework will save the most labels on datasets which are more clusterable. The empirical PL behavior of the datasets matches the clusterability classification. Dataset A for all choices of dimensions, exhibits the fastest increase of $\phi(\lambda)$ whereas dataset C is the slowest. It is evident in the plots that the PLAL algorithm is more sensitive than random sampling to very small sample sizes. On such instances the overall error is close to the bound.

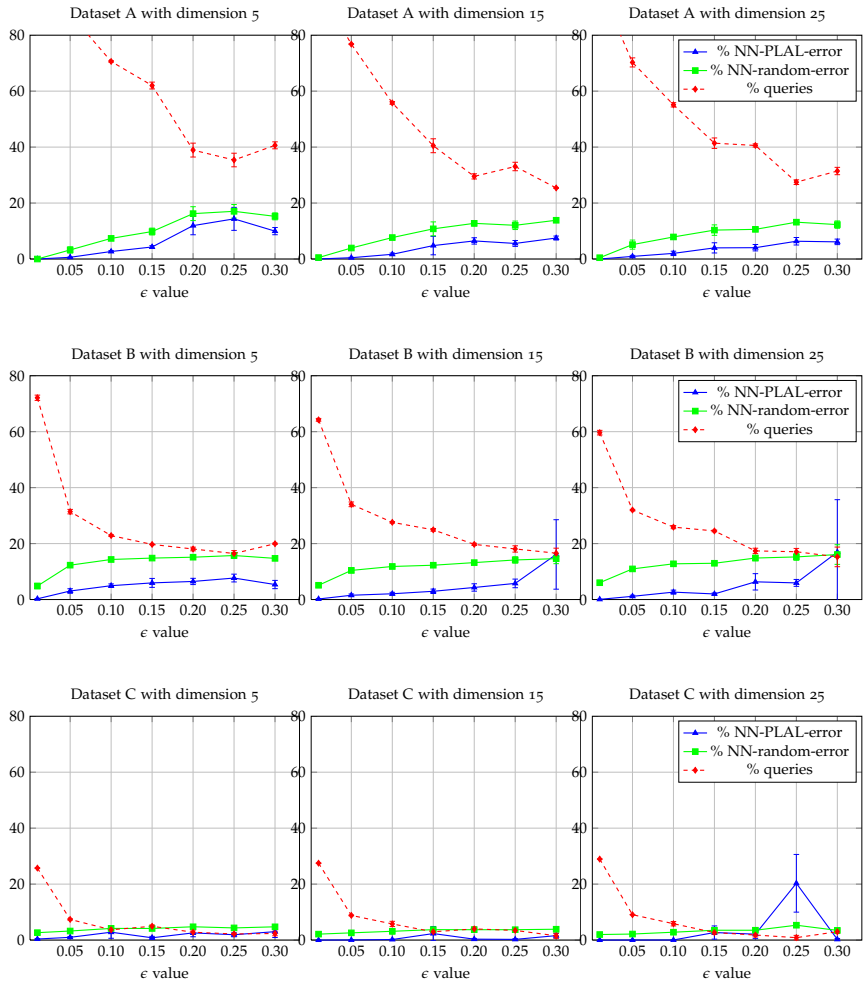


Figure 6.2: The average number of queries requested by PLAL on datasets A, B and C for different values of ϵ is denoted as $\% - queries$. The average prediction error of the NN classifier with PLAL queries is denoted as % NN-PLAL-error and the prediction error of the NN classifier with random queries is denoted as % NN-random-error

7 SENSOR NETWORK BASED VEHICLE DETECTION

This chapter presents a machine learning solution for the problem of vehicle detection based on a sensor network measurements in outdoor parking lots. This work was conducted as part of a joint SNF[1] project between ETH and a company named *Tinynode*[2]. Tinynode designs wireless sensor networks that measure the earth's magnetic field.

7.1 INTRODUCTION

The problem we address can be generally stated as: n sensors are deployed in some fixed physical layout. At time t, each sensor measures a signal that we assume is indicative of a local event. The task is to predict the aggregated occurrences of the events in every time point. In our case the time series signal is the earth's magnetic field, which is greatly affected by the presence of a close by vehicle due to its metal mass. The measurements are sent to a shared processing unit, which enables devising a centralized solution taking into account the state of the entire network. The sensors are deployed in each parking slot of an outdoor rest area, as illustrated in Figure 7.1a. Figure 7.1b is an image of the Tinynode sensor deployed in the ground.

(a) Truck rest area scheme (b) Sensor installed in the ground

Figure 7.1: Tinynode installation

The main motivation for using a magnetic sensors based system stem from the environmental conditions of an outdoor parking lot. For indoor parking spaces various solutions exist. For example, using motion sensors, ultrasonic sensors or even simply a camera. These

1 NCCR-MICS grant #51NF40-111400

2 http://www.tinynode.com

solutions do not carry through in an outdoor settings, where bad weather, visibility conditions and even theft are a major concern.

The Tinynode system was deployed in a parking lot in Germany prior to the beginning of the project, with the plan of expanding to other parking lots. The data used in this project was mainly collected from this initial parking lot, towards the end a second installation in France was ready, and could serve as a benchmark for the proposed solution. The detection algorithm prior to the collaboration was an engineered solution, based on the physical properties of the sensors and measurements performed in lab settings. The detection rate however, was not sufficient for commercial purposes, and the hope was that a machine learning based solution can perform better.

Below we list the challenges of obtaining a machine learning solution for the Tinynode application.

OBTAINING LABELS Sufficient amount of labeled data is vital for prediction in a machine learning framework. Since manned collection of examples in this setting is not scalable or reliable enough, a camera has to be stationed. In many places however, this is considered a privacy violation and is therefore forbidden. For the purpose of this project Tinynode could place one camera in the Germany parking lot, capturing about 10-11 sensors. The main problem with this solution is that it is not trivial to generalize from the data collected using one sensor to another, and even more for sensors in different rest areas. The small number of sensors in the sample is therefore far from being ideal. This generalization issue will be discussed extensively throughout this chapter. A secondary problem is annotating the images for the extraction of the occupancy labels. For this purpose we developed a semi-automatic annotating software for extracting the labels, a screen-shot is shown in Figure 7.2. Once a camera is placed, there is no resource limitation in the collection of the images, but the annotation still requires quite a bit of manual effort. In Section 7.5 we describe an Active Learning (AL) approach for choosing the images to be annotated, based on the respective unlabeled signal measurements.

GENERALIZING OVER PARKING LOTS One of the fundamental learning assumptions is that the distribution from which the training

data is sampled, is the same as the test distribution (in this context we sometimes use domain instead of a distribution). In our settings since the sensors are sensitive to presence of metal mass, the inherent background noise due to metal objects is likely to differ between parking lots.

Ideally we would collect data from each target parking lot for a fixed amount of time, and train a specific prediction model replacing the camera. Obviously this approach suffers from a high initial placement costs for each new parking lot installation. Also as mentioned earlier, placing a camera is often prohibited due to privacy protection rules. Thus the proposed solution is required to generalize well across parking lots.

NON *i.i.d.* SENSORS It became clear in the initial phase of the project that even within the same parking-lot the sensors exhibit differences in the conditional measurements distribution. Unlike between different rest areas where the differences are inherent, the differences between sensors are probably due to mis-alignment during installation time.

SPATIAL DEPENDENCE (NEIGHBORING EFFECT) Often the signal measured by a sensor is affected by vehicles parked in adjacent parking slots. This phenomenon essentially couples the measurements of neighboring sensors and poses difficulties in prediction.

TIME DEPENDENCIES There exist coupling in the time domain as well, being a time series the probability of an event (the existence of a vehicle) occurring at time t, is not independent of whether an event happened at time $t-1$. Unlike the previous points, this is a valuable information that can be exploited to enhance the prediction accuracy.

Several aspects of sensor networks have been a focus of learning based research in the recent years. For example routing protocols, minimization of energy consumption and communication cost (Le Borgne, Santini, and Bontempi 2007) as well as optimized placements (Krause et al. 2011), (Golovin, Faulkner, and Krause 2010) and enhanced detection (Faulkner et al. 2011). While these studies consider more general properties of sensor networks, here we describe the steps towards a solution for a specific network and application.

(a) Day time (b) Night time

Figure 7.2: Parking-lot images annotation software.

7.2 DATA AND FEATURES DESCRIPTION

The physical layout of the sensors is fixed during the installation time. Here we assume that the sensors are always installed in a chain like manner, as depicted in Figure 7.1a.

The data collected and sent by the sensors consist of the magnetic field readings and the time stamp. The magnetic field is a 3 dimensional vector $\boldsymbol{x} = (x_x, x_y, x_z) \in \mathbb{R}^3$. The prediction task is to infer the occupancy state of each parking spot, equipped with one sensor. We ignore the differences between vehicles (e.g. truck vs. car), such that we get a time series binary classification problem: at time t, the i'th sensor has an associated label $y_i \in \{0, 1\}$.

The data representation is perhaps the most important part in the application of every learning algorithm. Wrong data representation leads to high approximation error (see Subsection 2.1.1). This topic is also the main focus of the *ICLR*[3] learning representations conference.

In Subsection 7.2.1 we discuss the data collection and list the datasets we use in this study. In Subsection 7.2.2 we discuss the features we can extract based on the measurements and time-stamp, and in Subsection 7.2.3 we conduct an initial study of the "strength" of each feature.

Remark (Non *i.i.d.* sensors distributions). As we mentioned in the introduction, the basic assumption in the statistical learning framework is that the train and test data are generated by the same underlying distribution. In the Tinynode problem settings, this assumption means that the joint distribution over readings and labels, should be the same

3 https://sites.google.com/site/representationlearning2014/

across the sensors. Formally, for two sensors s and s', we would like to have $P_s(\boldsymbol{x}, y) = P_{s'}(\boldsymbol{x}, y)$.

Since we mostly consider discriminative classification models, it is important that the conditional distribution is similar across sensors, i.e. $P_s(y|\boldsymbol{x}) = P_{s'}(y|\boldsymbol{x})$. Also, since we can easily collect unlabeled data, a difference in the marginal distributions, i.e. $P_s(\boldsymbol{x}) \neq P_{s'}(\boldsymbol{x})$ is easier to detect (compared to the conditional distributions) and perhaps account for in the model.

7.2.1 *Datasets*

The main part of this study was done based on measurements collected from a parking lot in Germany. Throughout the collaboration the dataset changed several times, mostly due to the ongoing efforts on the company's side to reduce the noise in the readings. Once during the study the sensors were completely replaced.

We refer to the initial dataset, before the sensors were replaced as TGI-1 (first Tinynode Germany installation). In this dataset some pre-processing steps were required in order to overcome the sensors mis-alignment problem, without which generalizing from one sensor to another, was very difficult. We refer to the dataset after the sensors had been replaced as TGI-2 (second Tinynode Germany installation). Unless otherwise stated, the reported results were obtained using TGI-2.

The raw data exhibits high redundancy. The readings are collected at the rate of once every 2 minutes, which is much higher than the rate in which the occupancy state, even over the entire parking lot changes. We therefore dilute the data according to the following protocol: we keep a reading if any of the sensors changed its occupancy state since the previously stored one, or if the norm of the difference between the current reading and the previously stored one, is higher than a threshold value of 5 (though the threshold value did not effect the results substantially).

The data in dataset TGI-1 was collected in February-March 2010, it contains 11 sensors, numbered $1 - 11$. Sensor 7 was malfunctioning during the collection period, and it therefore omitted . Dataset TGI-2 was collected in April-May 2011, it contains the measurements of the 11 replaced sensors. During this period of time sensors 2 and 3 were malfunctioning and are therefore omitted from the dataset.

	# readings	Occupied (%)	Tinynode accuracy (%)
sensor 1	4087	41.28	90.29
sensor 2	4087	52.12	84.24
sensor 3	4087	55.81	92.17
sensor 4	4087	34.45	89.48
sensor 5	4087	41.13	84.56
sensor 6	4087	28.11	95.11
sensor 8	4087	31.76	95.30
sensor 9	4087	28.97	83.80
sensor 10	4087	48.89	97.70
sensor 11	4087	46.78	72.77
Average	4087	40.06 ± 9.34	88.46 ± 7.12

Table 7.1: Dataset TGI-1: number of readings, class ratios and the accuracy of the Tinynode algorithm breakdown according to the sensors.

Towards the end of the project the installation in another parking lot in France was ready. The dataset based on this parking lot contains 124 sensors, but very little amount of supervised data, since the labels were obtained by means of manned observations. We denote this dataset TFI (Tinynode France installation). The characteristics of the three datasets are given in Table 7.1, Table 7.2, Table 7.3 respectively. For space reasons, for dataset TFI instead of listing all the sensors we list them in categories according to the occupancy state in 10% intervals.

7.2.2 *Features*

PER MEASUREMENT FEATURES

RADIUS, ANGLES Although a Cartesian representation of vectors in $\mathbb{R}^d$ for some dimension d is more common, a spherical representation can be beneficial in some applications. A reading $\boldsymbol{x} = (x_x, x_y, x_z) \in \mathbb{R}^3$ is represented in spherical coordinate system using a radius and two angles. The conversion is given by

$$r = \sqrt{x_x^2 + x_y^2 + x_z^2} \quad \theta = \arccos(\frac{x_z}{r}) \quad \phi = \arctan 2(x_y, x_x)$$

	# readings	Occupied (%)	Tinynode accuracy (%)
sensor 1	3736	23.61	97.19
sensor 4	3736	20.48	96.09
sensor 5	3736	25.00	92.91
sensor 6	3736	20.74	98.80
sensor 7	3736	36.59	96.90
sensor 8	3736	18.39	97.59
sensor 9	3736	16.57	95.02
sensor 10	3736	53.91	99.38
sensor 11	3736	43.15	76.82
Average	3736	28.71 ± 12.16	94.52 ± 6.52

Table 7.2: Dataset TGI-2: number of readings, class ratios and the accuracy of the Tinynode algorithm breakdown according to the sensors.

Figure 7.3 shows the different features for 2 sensors. It can be easily seen from the plots that the radius has a strong correlation with the labels, while the angles exhibit quite a bit of variation, even when the parking lot is completely empty (time points where the labels of all the sensors are blue).

NEIGHBOR SENSORS READINGS/FEATURES Due to the influence of vehicles parked on neighboring sensors on the readings, the features of the neighboring sensors can supply valuable information and enhance the prediction model's accuracy.

TINYNODE PREDICTION The predictions of the original algorithm can be used as a feature as well. The hope is that the new prediction model can succeed on inputs that are difficult for the original algorithm.

TIME-STAMP It is evident from the labels that the distributions of vehicles in the parking lot is not uniform across 24 hours a day, and there is a difference between a week day and a weekend. The main caveat in using this information is that the occupancy pattern can differ from one parking lot to another.

FEATURE OPERATIONS

occupancy (%) range	average	# sensors	# readings	Tinynode accuracy (%)
0-10	5.4 ± 3.4	21	43.0 ± 12.8	93.3 ± 5.7
10-20	14.4 ± 2.8	35	48.3 ± 7.8	90.7 ± 5.6
20-30	25.9 ± 2.2	11	48.1 ± 5.7	89.2 ± 5.1
30-40	35.5 ± 3.2	14	49.6 ± 3.4	88.1 ± 6.1
40-50	46.2 ± 2.5	15	49.7 ± 2.8	84.2 ± 8.1
50-60	54.3 ± 2.6	15	49.7 ± 3.2	87.0 ± 8.4
60-70	65.3 ± 2.3	8	50.8 ± 0.7	80.5 ± 9.6
70-80	73.7 ± 3.5	3	49.7 ± 1.9	75.4 ± 11.2
80-90	80.4 ± 0.0	1	51.0 ± 0.0	80.4 ± 0.0
90-100	91.8 ± 0.0	1	49.0 ± 0.0	55.1 ± 0.0
Total	30.8 ± 21.5	124	48.1 ± 7.6	88.1 ± 8.5

Table 7.3: Dataset TFI: number of sensors, class ratios, number of readings and the accuracy of the Tinynode algorithm, divided into categories according to the occupancy percentage.

AVERAGE FEATURE To smooth out fluctuations we can use the average reading over a window of t last readings.

max − min FEATURE Similarly to the average feature, we can also consider the difference between the maximal and the minimal reading in a window of t readings.

7.2.3 *Feature Analysis*

Figure 7.3 is showing the different features in a time series manner, for two sensors, for the datasets TGI-1 (top) and TGI-2 (bottom). It is evident from the plots that at least visually, features such as the radius and the Tinynode-label have a high correlation to the labels, while other features such as the angles θ and ϕ are less directly correlated. The x_z axis seems the most correlated out of the three original axis.

The difference in distributions can be seen in the plots by comparing (for individual axis at least) similar feature values with different corresponding labels. This is even more evident in the TGI-1 dataset. The neighboring effect is also somewhat visible, these are the spikes in one

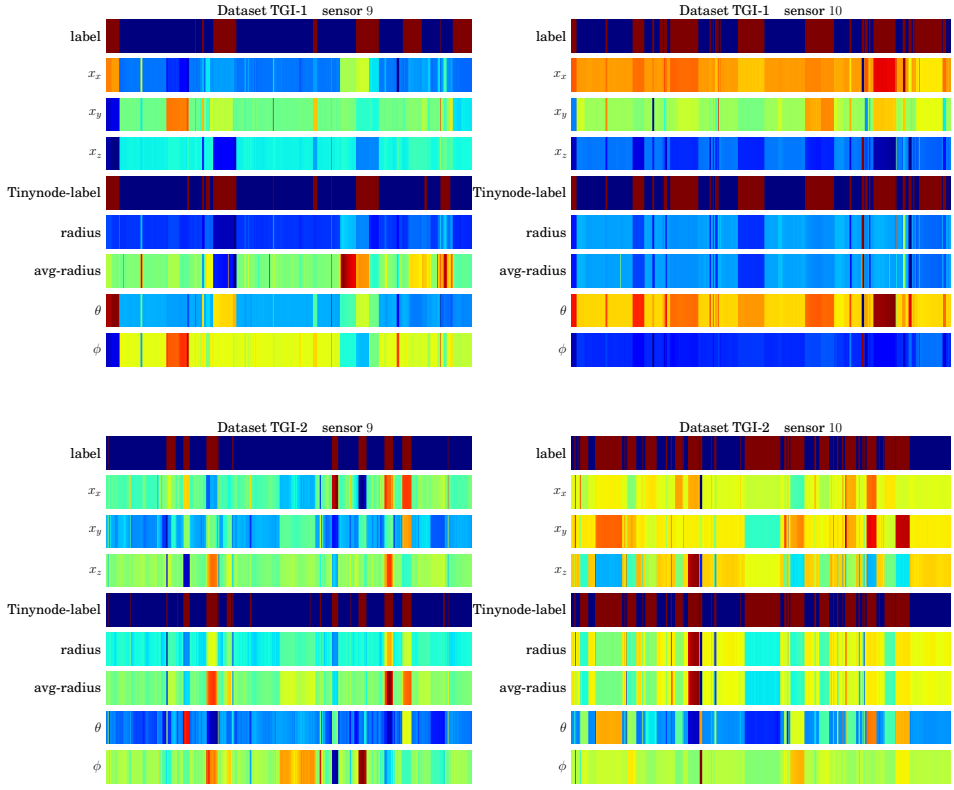


Figure 7.3: Features of sensors 9 and 10 extracted from dataset TGI-1 (top) and TGI-2(bottom). The top most row in each sensor's plot is the label, red color corresponds to the label "occupied".

sensor's features, at a time point of a label change in the other sensor (the shown features belong to neighboring sensors).

One way to quantify the correlation between individual features and the target, i.e. the labels, is by measuring the Spearman's correlation coefficient. The Spearman's correlation coefficient measures the monotonic relationship between two datasets. The output varies between -1 and $+1$ with 0 implying no correlation. For prediction, the larger the absolute value of the coefficient the better. The p-value roughly measures the probability of an uncorrelated system producing datasets that have an absolute Spearman correlation larger or equal to the one computed from these datasets. Table 7.4 shows the correlation coefficient computed for TGI-2. Again, the radius and the Tinynode label

stand out as having the highest correlation. The radius feature strength can perhaps be explained by the fact that it combines the features x_x, x_y, x_z. At least in this measure there is almost no difference between an average feature over a time window, and the feature itself. The angle θ has a higher correlation coefficient compared to ϕ. This can be due to the fact that θ is computed based on the radius and x_z, while ϕ is computed based on x_x, x_y

feature	mean correlation	mean p value
Tinynode-label	0.87 ± 0.12	0.00 ± 0.00
radius	0.72 ± 0.07	0.00 ± 0.00
avg-radius	0.70 ± 0.07	0.00 ± 0.00
θ	0.25 ± 0.14	0.00 ± 0.01
avg-θ	0.24 ± 0.14	0.03 ± 0.08
x_x	0.07 ± 0.21	0.08 ± 0.19
x_y	0.01 ± 0.20	0.10 ± 0.24
ϕ	0.01 ± 0.19	0.00 ± 0.00
avg-ϕ	0.01 ± 0.19	0.00 ± 0.00
x_z	-0.35 ± 0.26	0.00 ± 0.00

Table 7.4: Spearman's correlation coefficients between the different features and the labels, averaged over the sensors in dataset TGI-2

The final feature selection was done based on the performance of the different features in conjunction with the various models we considered (see Section 7.3). To get a feel for the accuracy using different feature combinations, we trained SVM classifiers with different parameters (kernels, kernel parameters, hyperparameter) on different combinations of features. The accuracy is computed in a Leave One Out (LOO) manner (see Subsection 7.3.2) averaged over the sensors. In Table 7.5 the ranking and accuracy of different feature sets are presented, under two criteria. On the left two columns we compare the best rank and accuracy, which is the maximal accuracy for each feature set over all of the classifier parameters we used. On the right columns the ranking is based on the accuracy averaged over all of the classifiers. This measure is robust against overfitting in identifying good features.

	Best accuracy		Avg. over models	
Feature-set	rank	accuracy(%)	rank	accuracy(%)
x_x, x_y, x_z, radius, n-label	1	97.23	6	95.61
x_y, x_z, radius	2	96.86	10	95.34
x_x, x_y, x_z, radius, ϕ	4	96.80	5	95.62
x_z, radius	5	96.71	1	95.70
x_x, x_y, x_z, radius	6	96.68	2	95.69
x_x, x_y, x_z, radius, ϕ, θ	8	96.63	14	94.59
x_x, x_y, x_z, radius, θ	10	96.57	4	95.65
x_x, x_y, x_z, radius, n-radius	11	96.55	9	95.37
x_x, x_y, x_z, radius, ϕ, θ, n-radius	14	96.42	16	94.45
x_x, x_y, x_z, Tinynode, θ	15	96.40	23	93.84
x_x, radius	17	95.94	25	93.73
x_x, x_y, x_z, Tinynode, radius	18	95.18	19	94.11
x_x, x_y, x_z, avg-radius	31	93.34	39	84.45

Table 7.5: Feature selection based on the classification error averaged over the sensors for dataset TGI-2. The features denoted as n-radius and n-label are the neighboring sensor's radius and label respectively.

It is not surprising that using the neighbors labels in training improves the performance. This feature is useful in identifying the neighboring effect which causes errors in prediction. This information is naturally not available during prediction, such that we can only use proxies to estimate its label. In this experiment it is the neighbor's radius, which does not seem to lead to same improvement. The radius as well as the z-axis,x_z, seem to be informative as well as robust features, the x_x reading seem less valuable on its own.

7.3 OCCUPANCY PREDICTION AND MODEL SELECTION ON DATASET TGI-2

In this section we describe the experimental setup we used to identify the most prominent classification model for the Tinynode data on the TGI-2 dataset. In this context the model includes the classifier type

(algorithm), parameters and data representation (features). In this section we sometimes use model and classifier interchangeably.

The performance for each sensor is measured as the normalized sum of the zero-one prediction loss over all of the sensor's readings. If the extracted features and true labels of sensor s are $(x_1^s, y_1^s), \ldots (x_n^s, y_n^s)$ and the predictions made by a trained classifier c are given by $y'^s_1, \ldots y'^s_n$ then

$$\text{accuracy}(c, s) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{I}_{y_i^s = y'^s_i}$$

As mentioned in Section 7.2, one of the difficulties in finding a suitable prediction model for this dataset, is that the readings are not really *i.i.d.* across sensors. Since the goal is to find a solution which generalizes over sensors, and rest areas, we select a model which corresponds to the classifier c^* with the largest accuracy averaged over all the sensors. If $\mathcal{S}$ is the set of sensors then

$$c^* = \underset{c}{\text{argmin}} \frac{1}{|\mathcal{S}|} \sum_{s \in \mathcal{S}} \text{accuracy}(c, s)$$

After having chosen a model, in Section 7.4 we use the classifier c^* trained on TGI-2 to predict on the dataset TFI.

7.3.1 *Candidate Classifiers*

We considered various classification algorithms, the two algorithms with the highest performance were an SVM and a CRF. For both algorithms we used all the features described in Subsection 7.2.3 in the experiments. The main difference between the algorithms is that CRF directly models the time series aspect.

SUPPORT VECTOR MACHINE (SVM) The SVM is explained in detail in Section 2.2. The parameters specified for the training of the SVM are the hyperparameter C governing the trade-off between training error and the margin, the choice of kernel and for most kernels also a kernel parameter. As the *i.i.d.* across sensors assumption is not fully realized, and since the number of different sensors for which we have supervised data is small, overfitting is a concern, especially with complex models. We therefore considered only few "simple" choices of kernels, including a linear, and polynomial kernel with degrees 2, 3 (see definitions in Table 2.1).

The SVM was trained on a feature representation of the readings, where each reading was considered one sample point regardless of the measurement time. We used the software scikit-learn[4] for the SVM implementation.

CONDITIONAL RANDOM FIELD (CRF) For a given sensor, instead of treating each measurement separately, we can also consider the entire sequence of readings (between a start and an end point) and try to jointly predict a matching sequence of labels. Such an approach is beneficial when the label of one reading is not independent of the labels of the previous (and following) readings. In our settings this is indeed the case, the probability of having an occupied (free) parking slot is higher if the slot was occupied (free) at the previous time point. The CRF (Lafferty, McCallum, and Pereira 2001; Sutton and Mccallum 2012) is a discriminative model which is suitable for such structured prediction tasks (see Section 2.3). The graphical representation of the model is the following: Each label corresponds to a node in the graph, and there is an edge connecting each label to the previous and to the following label (time-wise). This model is often referred to as a linear chain CRF, since the graph structure is chain like, where each vertex has a degree of 1 or 2. The reason we can use this representation, is that the label of each reading depends really *only* on the two closest readings (time-wise), due to the nature of the parking pattern, in which a single truck does not disappear and reappear again.

For the training of the CRF model we need to specify a regularization hyperparameter, similarly to the SVM. In addition, we can use a kernel like feature representation, here again we consider a linear and polynomial of degree 2 representation. For both the SVM and CRF we perform a grid search for finding the hyperparameters and kernels, according to a procedure described in the next subsection. For modeling the transition we use the same features as for the unary potentials but subtract the features of the neighboring time points from each other and additionally also include a constant feature, to capture the pure co-occurrence of the four different label configurations of two neighboring labels. We use maximum-marginal prediction, where we first jointly com-

4 http://scikit-learn.org/stable/

pute the marginal probabilities for the labels at the different time points and in a second stage choose the label with the maximum probability for each of them independently. This is the optimal approach if one considers the Hamming loss as the evaluation measure. For the learning as well as inference we use the Grante toolbox[5].

COMBINED CLASSIFIERS It was repeatedly shown in the machine learning literature (Koren 2009; Polikar 2009), that combining diverse models often leads to an improved accuracy. Here we consider a combination of the two models SVM and CRF by means of a second round of training. We add to the feature representation of each sensor, its prediction using the best SVM and CRF model, and then re-train using in this case an SVM. We use *Combination* to refer to this procedure in Subsection 7.3.3

7.3.2 *Experimental Setup*

In a machine learning experimental design it is important to follow the validation principles of not using the test data for the training, or for the model selection and parameters tuning. Ideally the model (features, classifier, parameters, etc.) is chosen using cross-validation over splits of the training data, and then the model's prediction on the test data is used to evaluate it's performance.

In the Tinynode application one approach would be to hold out a subset of the sensors, train and choose the model on the rest, and then report the accuracy on the hold out sensors. The problem with this procedure is that the performance would substantially differ depending on the choice of subset.

We use the following approach: For each sensor $s_j \in \mathcal{S}$, we do a LOO cross-validation among the remaining $\{\mathcal{S} \setminus s_j\}$ sensors. This procedure amounts to iterating over the models, and for each $c \in$ models train on all $\{\mathcal{S} \setminus \{s_j, s_{j'}\}\}$ where $j' \neq j$, and evaluate the accuracy of c as $\frac{1}{|\mathcal{S}|-1} \sum_{s_{j'} \in \{\mathcal{S} \setminus s_j\}} \text{accuracy}(c, s_{j'})$. Then we train the best model c^* on $\{\mathcal{S} \setminus s_j\}$, and report the performance as: $\text{accuracy}(c^*, s_j)$.

While this procedure is suitable for reporting the performance of our solution, it can potentially result in as many as $|\mathcal{S}|$ "best" performing

5 http://www.nowozin.net/sebastian/grante

models. In order to choose the one classifier that will eventually replace the company's existing algorithms, we used a LOO split with respect to all the sensors in TGI-2. E.g. we chose the model c^* with the best average accuracy(c, s_j) across all $s_j \in \mathcal{S}$, where for s_j, c was trained using $\{\mathcal{S} \setminus s_j\}$.

Finally we validate the error of our single c^* classifier on the TFI data, as this data was obviously not part of the training or model selection. The accuracy on both TGI-2 and TFI is reported in Subsection 7.3.3.

We searched over the following parameters:

1. Features (based on Subsection 7.2.3). We always included x_y, x_z and radius, we checked in addition x_x, ϕ, θ and the adjacent sensor's radius.

2. Kernels. For the SVM we consider a linear and polynomial kernels of degrees $2, 3$. For the CRF we use linear, and a polynomial of degree 2 feature like representation.

3. Regularization parameters. For both methods we search over the values $\{0.1, 1, 10, 100, 1000\}$.

7.3.3 *Experimental Results*

In terms of model parameters, for both the SVM and CRF it was sufficient to use the x_x, x_y, x_z and radius features, adding more features did not increase the performance. The polynomial kernel of degree 2 was the best choice for both models, with regularization parameters of 1000 for SVM and 10 for the CRF. As for the combined classifier, the performance slightly increased when using the ϕ, θ and the neighbors radius in the feature representation.

Table 7.6 shows the accuracy breakdown into sensors and methods. The SVM and the CRF perform very similarly, the combination has a slightly higher average performance, but on some of the sensors the accuracy even decreases. In comparison to Tinynode, on average over all the sensors the learned classifiers perform better. However this is mostly due to few sensors.

Remark (Small performance range). When comparing the accuracy of prediction, it is important to pay attention to the dynamic range in addition to the absolute numbers. For prediction tasks this range can be thought of as the difference between the baseline method and the

	Tinynode	SVM	CRF	Combination
sensor 1	97.2	97.2	97.1	97.0
sensor 4	96.1	95.7	95.7	96.0
sensor 5	92.9	96.5	95.9	95.2
sensor 6	98.8	99.0	99.0	98.9
sensor 7	96.9	96.9	96.9	97.4
sensor 8	97.6	98.3	97.1	98.0
sensor 9	95.0	96.7	96.7	96.7
sensor 10	99.4	99.4	99.4	99.4
sensor 11	76.8	91.7	89.6	94.0
Average	94.5 ± 6.5	96.8 ± 2.1	96.4 ± 2.7	97.0 ± 1.6

Table 7.6: Comparison of the prediction accuracy of different classifiers on TGI-2

highest achievable result. In this application the range corresponds to the error of the engineered solution, since ultimately the goal is to develop a machine learning solution that outperforms this solution. Towards the end of the project and as a result of the Tinynode engineered solution also improving, this range reached to as low as $5 - 7$ percent. In this settings, an additive improvement of 1 percent already amounts to about one sixth of the dynamic range.

7.4 OCCUPANCY PREDICTIONS ON DATASET TFI

On average across the 124 sensors in TFI, the SVM trained on TGI-2, achieved an accuracy of 93.2 ± 6.4. In comparison the Tinynode engineered solution has an average accuracy of 88.1 ± 8.5. The best accuracy on TFI, not restricted to the model parameters with the highest performance on TGI-2 is 94.2 ± 4.9.

A detailed comparison between the learned SVM classifier's accuracy and Tinynode, per sensor is given in Figure 7.4.

In generalizing the solution from TGI-2 to TFI, both the learned solution and the Tinynode engineered solution suffer from a degradation in performance. This can be attributed to the sensors and parking lot in TFI being different again from TGI-2, but also to overfitting. Since the Tinynode solution changed after the beginning of the project as a

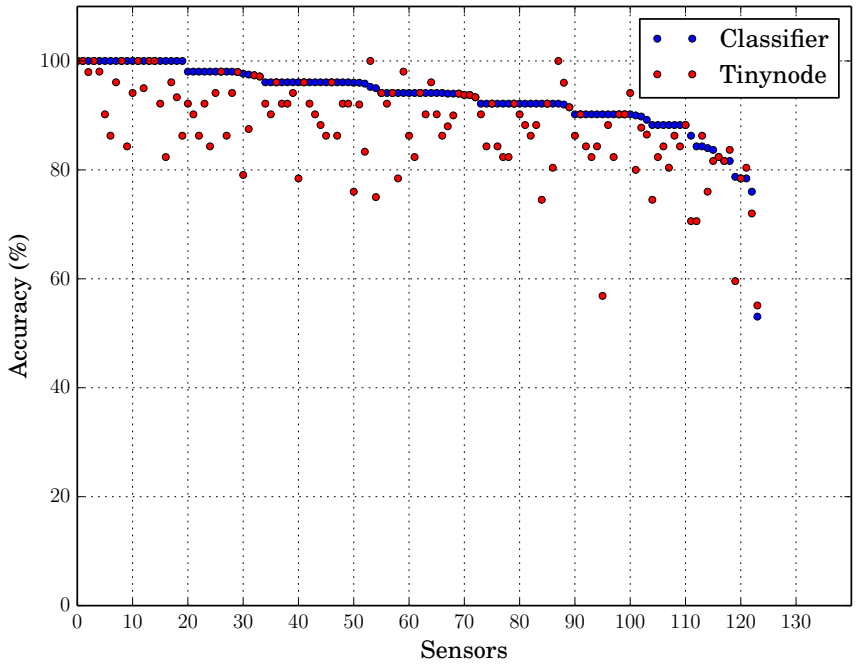


Figure 7.4: Comparison of the number of sensors with certain prediction accuracy on TFI data, achieved by the Tinynode engineered solution and a classifier trained on TGI-2

result of having a large labeled dataset in hand, it is also not immune against overfitting (the Tinynode performance on the older dataset is given in Table 7.1).

Comparing the solution learned on TGI-2 with the Tinynode solution, the learned solution is outperforming on average as well as for most sensors, in a sensor-wise comparison.

7.5 ACTIVE LEARNING APPROACH FOR LABELS EXTRACTION

For the parking lot where a camera could be placed, the labels were obtained based on the images. We wrote a software which presents the images to the an annotator, and the matches the labels with the measurements. The usage still requires manual assignment of labels, as a fully automatic detection is likely not to be reliable enough, especially during night time and low visibility conditions (e.g. Figure 7.2). Instead

of annotating all the images, we can run an AL algorithm, and reduce the amount of manual labor required.

7.5.1 *Experimental Setup*

ACTIVE LEARNING ALGORITHM For this application we use a budget version of the PLAL algorithm described in Chapter 6 to actively extract the labels for the Tinynode data in TGI-2. In the budget version, in addition to the input ϵ (the error bound), we give the algorithm an allowed budget. The algorithm is operating in the same manner described in Section 6.4, i.e. descends down a dyadic tree from the root to the leafs, only in this version the procedure is stopped once the number of queries reaches the budget. The extension from the queried points to the rest of the un-queried examples in non-homogeneous clusters can be done by using an additional classifier trained on the query points.

While the PLAL error bound guarantee (see Subsection 6.4.2) does not hold for the budget version, for practical purposes it is more intuitive; It allows us to explore and compare the accuracy in a budget range which is only achieved by the original algorithm for very specific ϵ values, and the variability in the results in this range is very high (see the discussion in the end of this section).

DATA SPLITS Again we conducted experiments in a LOO manner, which means that for each sensor we use the concatenation of the data of all other sensors as the initially unlabeled training data. Using the labels returned by the PLAL budget algorithm, we train a classifier and use it to predict on the hold out sensor

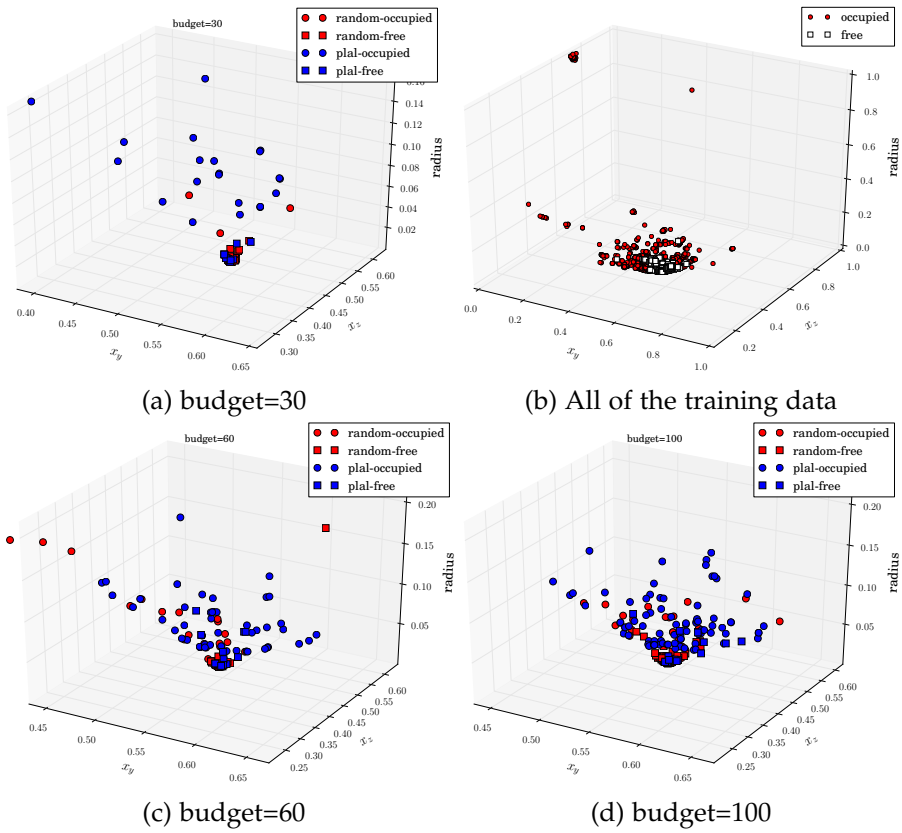
PERFORMANCE MEASURE We consider two performance measures. The first is the transductive accuracy, which is the accuracy of the labels returned by the PLAL compared to the real labels. The other measure is an inductive accuracy, which is the accuracy of the classifier trained on the PLAL labels, on the hold out sensor data.

BENCHMARK We use a random selection of query points as a benchmark to the PLAL budget algorithm. It has been noted in several works published in the field of AL that it is often difficult to find a strategy to query an oracle for labels, which is better than random

sampling. For each budget we randomly query unlabeled example points for their labels, and then use an additional classifier to extend the predictions.

7.5.2 *Experimental Results*

7.5.2.1 *Qualitative Comparison of a Single Run*



(a) budget=30 (b) All of the training data

(c) budget=60 (d) budget=100

Figure 7.5: PLAL vs. random selection of TGI-2 training data. For visualization purposes the data is projected on the $x_y, x_z, radius$ axis.

Figure 7.5 shows one run of the transductive settings for sensor 4, using budget values of 30, 60 and 100. For visualization purposes we use 3 features for plotting data, the radius, x_y and x_z. The entire training data is shown in the upper right corner. It is interesting to

note that the readings which correspond to a non-occupied state are somewhat clustered, whereas the occupied readings are much more scattered, forming many smaller clusters. The choices of query points made by the budget PLAL and random selection, are shown in the remaining plots for different budget values. The plots validate the intuition that the PLAL algorithm is better at exploring the space compared to random sampling, in which the queries are concentrated on packed regions. Another indication of this behavior, is the skewed class proportions in the set of query points chosen by PLAL, compared to the true proportions, and the random sampling proportions. The percentages of occupied readings are shown in Table 7.7.

	Budget=30	Budget=60	Budget=100	Entire dataset
PLAL	70%	80%	84%	27.43%
random	10%	33.33%	28%	

Table 7.7: Class proportions in the set of examples chosen by PLAL vs. a random choice.

7.5.2.2 *Transductive and Inductive Results*

In Figure 7.6 the transductive accuracy of three of the sensors, as well as the averaged over sensors accuracy, are shown for different budget values. To account for the randomization, we used the averaged value over 10 runs. For all of the sensors, below a certain budget threshold the budget version of PLAL always achieves higher accuracy, compared to random sampling. The amount by which it improves over random sampling differs between the sensors. We chose sensors that show different behavior trends.

The inductive results are shown in Figure 7.7. On average, the transductive and inductive results are similar, as we would expect.

For sensor 11 and partially sensor 8, the performance gap between the two methods increases from the transductive to the inductive settings. Since the transductive accuracy is measured on the same training data, and we later use the same classifier with the same parameters to train on this data, the difference must be a result of each method failing to extend the queries on different subsets of the training data (instead of

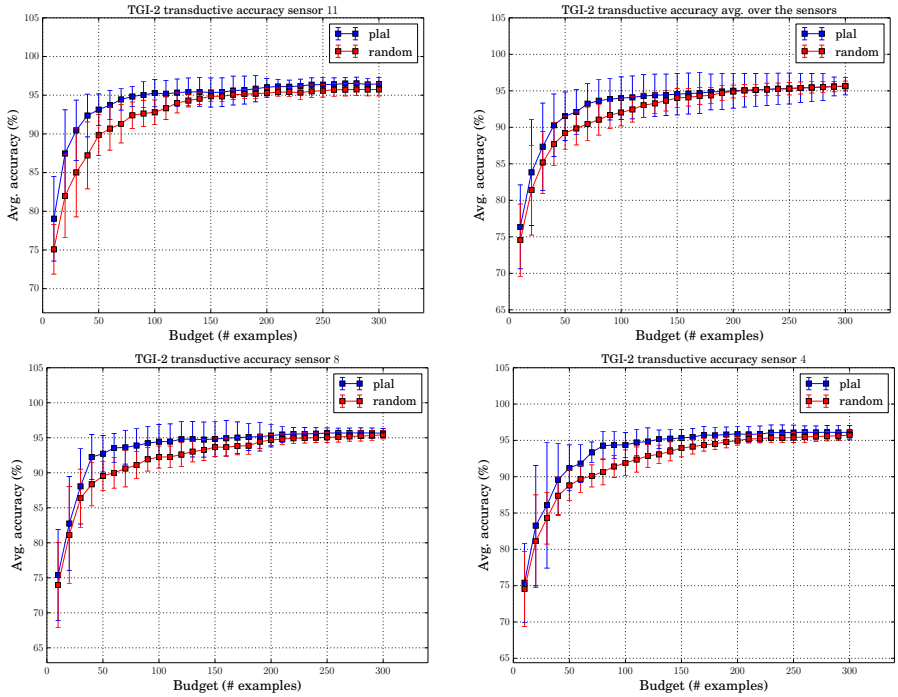


Figure 7.6: Comparison of the transductive accuracy on TGI-2 training data, with increasing budgets. All the plots show an average accuracy over 10 runs for each sensor and budget value. The top right plot shows the accuracy averaged over the sensors and the runs.

the the errors of PLAL being a subset of the errors made by the random choice).

In Figure 7.8 the budget vs. increasing values of ϵ is shown. The values are computed as an average over the sensors and 10 runs for each sensor and ϵ value. It is evident from the plot together with the performance plots, that at least for this dataset, the number of points requested by the algorithm to achieve a certain error bound, is a very conservative assessment. In reality it is possible to reach a much higher accuracy with fewer labeled examples.

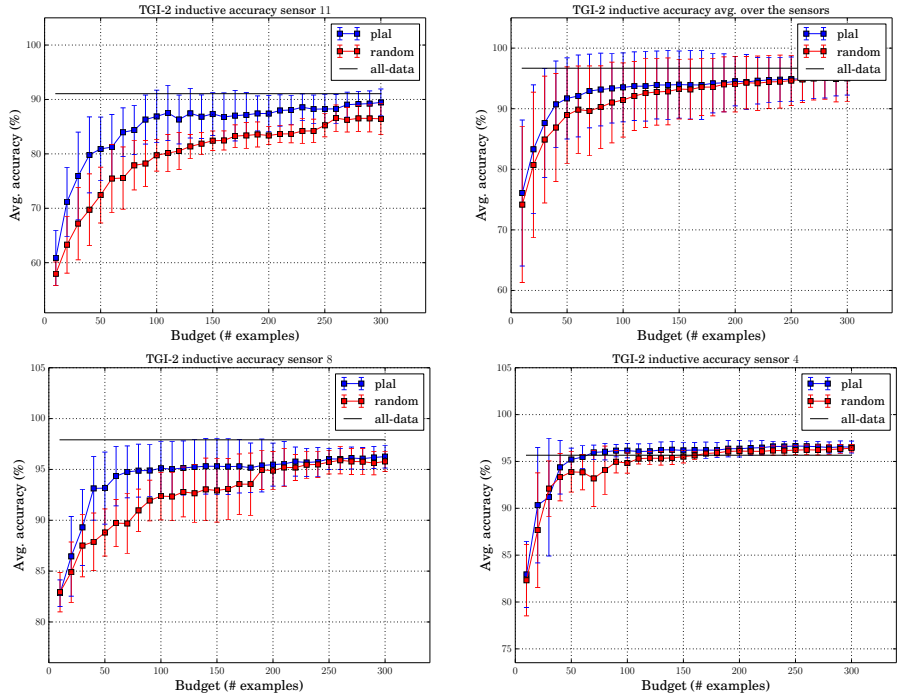


Figure 7.7: Comparison of the inductive accuracy on TGI-2 training data, with increasing budgets. All the plots show an average accuracy over 10 runs for each sensor and budget value. The top right plot shows the accuracy averaged over the sensors and the runs.

7.6 CONCLUSIONS

This chapter covers the complete pipeline of a machine learning solution to the problem of vehicle detection in rest areas, based on magnetic sensor readings. This application is an example of a case where data cleaning, as well as choosing the right data representation leads to high prediction accuracy using existing machine learning models such as SVM and CRF.

On the downside, since this application exhibits a low dynamic range where improvements can be made, it is difficult to establish superiority of certain models with a desired statistical significance. The performance on the "fresh" dataset TFI, confirms that the learning approach is useful in this settings.

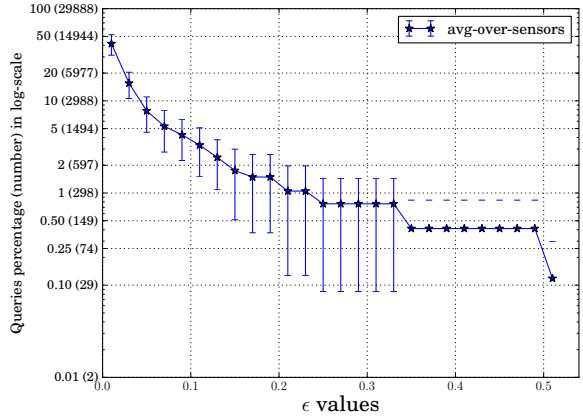


Figure 7.8: Log scale plot of the percentage of queries vs. ϵ values, averaged over the sensors and 10 runs.

8

DISCUSSION AND FUTURE WORK

This thesis presents several efficient algorithms for convex as well as NP-hard problems in machine learning. We study the Multiple Kernel Learning (MKL) problem and show that the "centering" approach results in a desirable convergence behavior. Leveraging on the convex Linear Program (LP) relaxation of the Maximum-A-Posteriori (MAP) problem, we derive efficient algorithms for the NP-hard inference and bi-clustering tasks. In addition, our work contributes to the theoretical understanding of scenarios in which Active Learning (AL) is useful in reducing the sample complexity. Finally our work provides practical insights on how to approach a real-world prediction problem with machine learning tools.

Below we list a few directions for further research.

- In Chapter 3 we presented a cutting plane algorithm for the MKL problem. The algorithm uses the analytic center as a query point, or reference within the remaining search space for constructing the new cut. The effort spent on computing the analytic center grows with each iteration, as new constraints are added to the polyhedron. Redundancy check for each constraint is costly, as it involves solving a linear program. However there exist less costly methods that can determine the redundancy of some of the constraints (Boyd and Vandenberghe 2007), as well as heuristics which are not guaranteed to be correct, but in practice work quite well. This seems like an interesting direction for reducing the per-iteration cost of our algorithm.

- One observation that we had in the context of a weighted combination of kernels, is that if S is the subset of kernels with greater than zero weights, often setting the weights to $1/|S|$ yields equally good prediction performance. In terms of optimization, the $\mathcal{L}_1$ norm is preferable over indicator variables. It would be useful to fully understand what makes the weight in this case equivalent to membership.

- Exploiting the clusterability of the data seems like a promising approach for Active Learning (AL). The solution presented here as well as the previous work along these lines by Dasgupta and Hsu (2008), do not offer an ordering or ranking of the chosen examples. A more "aggressive" approach which relies on some notion of data clusterability can be very useful in situations where we have a limited budget of examples we can label.

- A domain adaptation approach can be useful for the Tinynode application. As discussed in Chapter 7 the sensors measurements and label distributions differ within parking-lots, and even more so across parking-lots. In domain adaptation we use the source labeled data, as well as (usually unlabeled) data from the target domain to construct a target specific classifier. In the Tinynode application the unlabeled data from the target parking lots can be easily collected. In terms of supervised target data, we can collect the measurements during the installation time before any vehicles are allowed in. Learning based on positive and unlabeled data was previously used for text categorization (Liu, Lee, and Li 2002; Lee and Liu 2003; Li et al. 2009). In our context it could be combined with source labeled data.

BIBLIOGRAPHY

Aronszajn, N. (1950). „Theory of Reproducing Kernels.“ In: *Transactions of the American Mathematical Society* 68, pp. 337–404 (cit. on p. 13).

Babonneau, F., C. Beltran, A. Haurie, C. Tadonki, and J.-P. Vial (2007). „Proximal-ACCPM: A Versatile Oracle Based Optimization Method.“ In: *Optimisation, Econometric and Financial Analysis*. Ed. by E. J. Kontoghiorghes and C. Gatu (cit. on p. 34).

Bach, F. R. (2008). „Consistency of the Group Lasso and Multiple Kernel Learning.“ In: *J. Mach. Learn. Res* 9, pp. 1179–1225 (cit. on pp. 22, 29).

Bach, F. R., G. Lanckriet, and M. I. Jordan (2004). „Multiple Kernel Learning, Conic Duality, and the SMO Algorithm.“ In: *ICML* (cit. on p. 22).

Bakir, G., T. Hofmann, B. Schölkopf, A. J. Smola, B. Taskar, and S. V. N. Vishwanathan (2007). *Predicting Structured Data*. MIT Press (cit. on p. 14).

Balcan, M.-F., A. Z. Broder, and T. Zhang (2007). „Margin Based Active Learning.“ In: *COLT*, pp. 35–50 (cit. on p. 101).

Balcan, M.-F., S. Hanneke, and J. W. Vaughan (2010). „The true sample complexity of active learning.“ In: *Machine Learning* 80.2-3, pp. 111–139 (cit. on p. 101).

Beck, A. and M. Teboulle (2009). „A Fast Iterative Shrinkage-Thresholding Algorithm for Linear Inverse Problems.“ In: *SIAM Journal on Imaging Sciences* 2.1 (cit. on p. 58).

Ben-David, S., N. Cesa-Bianchi, D. Haussler, and P. M. Long (1995). „Characterizations of Learnability for Classes of {0, ..., n}-Valued Functions.“ In: *J. Comput. Syst. Sci.* 50, pp. 74–86 (cit. on p. 9).

Bertsekas, D. P. (1999). *Nonlinear Programming*. Belmont, MA: Athena Scientific (cit. on pp. 19, 56, 89).

Beygelzimer, A., S. Dasgupta, and J. Langford (2009). „Importance weighted active learning.“ In: *ICML*, p. 7 (cit. on pp. 99, 101).

Beygelzimer, A., D. Hsu, J. Langford, and T. Zhang (2010). „Agnostic Active Learning Without Constraints.“ In: *NIPS*, pp. 199–207 (cit. on p. 101).

Borgwardt, K. M., C. S. Ong, S. Schönauer, S. V. N. Vishwanathan, A. J. Smola, and H.-P. Kriegel (2005). „Protein Function Prediction via Graph Kernels." In: *Proceedings of the International Conference on Intelligent Systems for Molecular Biology* (cit. on p. 22).

Borwein, J. M. and A. S. Lewis (2005). *Convex Analysis and Nonlinear Optimization: Theory and Examples*. Springer (cit. on p. 19).

Boser, B. E., I. M. Guyon, and V. N. Vapnik (1992). „A training algorithm for optimal margin classifiers." In: *Proceedings of the fifth annual workshop on computational learning theory (COLT)*, pp. 144–152 (cit. on p. 11).

Bottou, L. and O. Bousquet (2008). „The Tradeoffs of Large Scale Learning." In: *Advances in Neural Information Processing Systems*. Vol. 20, pp. 161–168 (cit. on p. 9).

Boyd, S. and L. Vandenberghe (2004). *Convex Optimization*. New York, NY, USA: Cambridge University Press (cit. on p. 19).

Boyd, S. and L. Vandenberghe (2007). „Localization and Cutting-Plane Methods. Unpublished lecture notes." URL: http://www.stanford.edu/class/ee392o/localization-methods.pdf (cit. on p. 137).

Buhmann, Joachim. (2011). „Context Sensitive Information: Model Validation by Information Theory." In: *Pattern Recognition*. Lecture Notes in Computer Science. Springer Berlin Heidelberg, pp. 12–21 (cit. on p. 71).

Buhmann, Joachim. and Jan Puzicha (2001). „Basic Principles of Annealing for Large Scale Non-Linear Optimization." In: *Online Optimization of Large Scale Systems*. Springer Berlin Heidelberg, pp. 749–777 (cit. on p. 93).

Chakrabarti, D., S. Papadimitriou, D. S. Modha, and C. Faloutsos (2004). „Fully automatic cross-associations." In: *KDD*, pp. 79–88 (cit. on p. 72).

Chandrasekaran, V., N. Srebro, and P. Harsha (2008). „Complexity of Inference in Graphical Models." In: *Proceedings of the 24th Conference on Uncertainty in Artificial Intelligence (UAI)*, pp. 70–78 (cit. on p. 17).

Chang, C.-C. and C.-J. Lin (2011). „LIBSVM: A library for support vector machines." In: *ACM Transactions on Intelligent Systems and Technology* 2 (3). Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm, 27:1–27:27 (cit. on p. 11).

Cheng, Y. and G. M. Church (2000). „Biclustering of Expression Data." In: *International Conference on Intelligent Systems for Molecular Biology (ISMB)*, pp. 93–103 (cit. on p. 71).

Cho, H., I. S. Dhillon, Y. Guan, and S. Sra (2004). „Minimum Sum-Squared Residue Co-clustering of Gene Expression Data." In: *Proceedings of the Fourth SIAM International Conference on Data Mining*. SIAM, pp. 114–125 (cit. on p. 72).

Cooper, G. F. (1990). „The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks." In: *Artificial Intelligence* 42, pp. 393–405 (cit. on p. 17).

Cortes, C. and V. N. Vapnik (1995). „Support-Vector Networks." In: *Machine Learning* 20.3, pp. 273–297 (cit. on p. 11).

Cortes, Corinna, Mehryar Mohri, and Afshin Rostamizadeh (2010). „Generalization Bounds for Learning Kernels." In: *ICML*, pp. 247–254 (cit. on p. 30).

Daniely, A., S. Sabato, S. Ben-David, and S. Shalev-Shwartz (2011). „Multiclass Learnability and the ERM principle." In: *COLT*, pp. 207–232 (cit. on p. 9).

Dasgupta, S. (2004). „Analysis of a greedy active learning strategy." In: *NIPS* (cit. on p. 101).

Dasgupta, S. (2005). „Coarse sample complexity bounds for active learning." In: *NIPS* (cit. on p. 99).

Dasgupta, S. (2011). „Two faces of active learning." In: *Theor. Comput. Sci.* 412.19, pp. 1767–1781 (cit. on pp. 101, 104).

Dasgupta, S. and D. Hsu (2008). „Hierarchical sampling for active learning." In: *ICML*, pp. 208–215 (cit. on pp. 4, 100, 101, 104, 105, 138).

Dasgupta, S., D. Hsu, and C. Monteleoni (2008). „A General Agnostic Active Learning Algorithm." In: *ISAIM* (cit. on p. 101).

Dhillon, I. S., S. Mallela, and D. S. Modha (2003). „Information-theoretic co-clustering." In: *KDD*, pp. 89–98 (cit. on p. 72).

Domke, J. (2011). „Dual Decomposition for Marginal Inference." In: *AAAI* (cit. on p. 57).

Duda, R. O. and P. E. Hart (1973). *Pattern Classification and Scene Analysis*. John Willey & Sons (cit. on p. 9).

El Ghaoui, L. (2012). *Proximal gradient method*. Lecture notes (cit. on p. 58).

Fan, R.-E., K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin (2008). „LIBLINEAR: A Library for Large Linear Classification." In: *Journal of Machine Learning Research* 9, pp. 1871–1874 (cit. on p. 11).

Faulkner, M., M. Olson, R. Chandy, J. Krause, K. M. Chandy, and A. Krause (2011). „The Next Big One: Detecting Earthquakes and other

Rare Events from Community-based Sensors." In: *Proc. ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)* (cit. on p. 115).

Geman, S. and D. Geman (1984). „Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images." In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 6.6, pp. 721–741 (cit. on pp. 16, 93).

Giotis, I. and V. Guruswami (2006). „Correlation Clustering with a Fixed Number of Clusters." In: *Theory of Computing* 2.13, pp. 249–266 (cit. on p. 72).

Goemans, Michel X. and David P. Williamson (1995). „Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming." In: *J. ACM* 42.6 (cit. on p. 86).

Goffin, J. and J. Vial (2002). „Convex Nondifferentiable Optimization: A Survey Focused on the Analytic Center Cutting Plane Method." In: *Optimization Methods and Software* 17.5, pp. 805–867 (cit. on pp. 23, 25, 26, 32).

Goldreich, O., S. Goldwasser, and D. Ron (1998). „Property testing and its connection to learning and approximation." In: *Journal of the ACM* 45.4, pp. 653–750 (cit. on pp. 73, 85).

Golovin, D., M. Faulkner, and A. Krause (2010). „Online Distributed Sensor Selection." In: *Proc. ACM/IEEE International Conference on Information Processing in Sensor Networks (IPSN)* (cit. on p. 115).

Gonen, A., S. Sabato, and S. Shalev-Shwartz (2011). „Active Learning Halfspaces under Margin Assumptions." In: *CoRR* abs/1112.1556 (cit. on p. 101).

Hanneke, S. (2007). „A bound on the label complexity of agnostic active learning." In: *ICML*, pp. 353–360 (cit. on p. 101).

Hanneke, S. (2012). „Activized Learning: Transforming Passive to Active with Improved Label Complexity." In: *Journal of Machine Learning Research (JMLR)*, pp. 1469–1587 (cit. on p. 102).

Hartigan, J. A. (1972). „Direct clustering of a data matrix." In: *Journal of the American Statistical Association* 67.337, pp. 123–129 (cit. on p. 71).

Håstad, Johan (2001). „Some Optimal Inapproximability Results." In: *J. ACM* 48.4, pp. 798–859 (cit. on p. 86).

Hazan, T. and A. Shashua (2010). „Norm-Product Belief Propagation: Primal-Dual Message-Passing for Approximate Inference." In: *IEEE TIT* 56.12, pp. 6294–6316 (cit. on p. 55).

Hirai, H., B.-H. Chou, and E. Suzuki (2011). „A Parameter-Free Method for Discovering Generalized Clusters in a Network.“ In: *Discovery Science*, pp. 135–149 (cit. on p. 72).

Hussain, Zakria and John Shawe-Taylor (2011). „Improved Loss Bounds For Multiple Kernel Learning.“ In: *AISTATS*, pp. 370–377 (cit. on p. 30).

Jojic, V., S. Gould, and D. Koller (2010). „Accelerated dual decomposition for MAP inference.“ In: *ICML* (cit. on p. 59).

Kääriäinen, M. (2006). „Active Learning in the Non-realizable Case.“ In: *ALT*, pp. 63–77 (cit. on pp. 99, 101).

Kappes, J. H. and C. Schnoerr (2008). „MAP-Inference for Highly-Connected Graphs with DC-Programming.“ In: *DAGM* (cit. on p. 50).

Karp, Richard M. (1972). „Reducibility Among Combinatorial Problems.“ In: *Complexity of Computer Computations*. The IBM Research Symposia Series. Plenum Press, New York, pp. 85–103 (cit. on p. 86).

Kemp, C. and J. B. Tenenbaum (2006). „Learning systems of concepts with an infinite relational model.“ In: *In Proceedings of the 21st National Conference on Artificial Intelligence* (cit. on p. 72).

Kemp, C. and J. B. Tenenbaum (2008). „The discovery of structural form.“ In: *Proceedings of the National Academy of Sciences of the United States of America* (cit. on p. 72).

Kindermann, R. and J. L. Snell (1980). *Markov Random Fields and Their Applications*. AMS (cit. on p. 16).

Koller, D. and N. Friedman (2009). *Probabilistic Graphical Models: Principles and Techniques*. MIT Press (cit. on p. 14).

Kolmogorov, V. (2006). „Convergent tree-reweighted message passing for energy minimization.“ In: *PAMI* (cit. on p. 60).

Koltchinskii, V. and D. Panchenko (2002). „Empirical Margin Distributions and Bounding the Generalization Error of Combined Classifiers.“ In: *Ann. Statist.* 30.1, pp. 1–50 (cit. on p. 30).

Komodakis, N., N. Paragios, and G. Tziritas (2007). „MRF optimization via dual decomposition: Message-passing revisited.“ In: *IEEE 11th International Conference on Computer Vision (ICCV)* (cit. on pp. 56, 89).

Komodakis, N., N. Paragios, and G. Tziritas (2011). „MRF Energy Minimization and Beyond via Dual Decomposition.“ In: *IEEE Trans. Pattern Anal. Mach. Intell.* 33, pp. 531–552 (cit. on p. 95).

Koren, Y. (2009). *The BellKor Solution to the Netflix Grand Prize* (cit. on p. 126).

Krause, A., R. Rajagopal, A. Gupta, and C. Guestrin (2011). „Simultaneous Optimization of Sensor Placements and Balanced Schedules." In: *IEEE Transactions on Automatic Control* 56.10, pp. 2390–2405 (cit. on p. 115).

Kschischang, F. R., B. J. Frey, and H.-A. Loeliger (2001). „Factor graphs and the sum-product algorithm." In: *IEEE Transactions on Information Theory* 47.2, pp. 498–519 (cit. on pp. 14, 15, 17).

Kumar, A. and S. Zilberstein (2011). „Message-Passing Algorithms for Quadratic Programming Formulations of MAP Estimation." In: *UAI* (cit. on pp. 50, 64).

Kumar, A., S. Zilberstein, and M. Toussaint (2012). „Message-Passing Algorithms for MAP Estimation Using DC Programming." In: *AISTATS* (cit. on p. 50).

Lafferty, J., A. McCallum, and F. Pereira (2001). „Conditional Random Fields: Probabilistic Models for Segmenting and Labeling Sequence Data." In: *Proceedings of the Eighteenth International Conference on Machine Learning (ICML)*, pp. 282–289 (cit. on pp. 18, 125).

Lanckriet, G., T. De Bie, N. Cristianini, M. I. Jordan, and W. Stafford Noble (2004a). „A statistical framework for genomic data fusion." In: *Bioinfomatics* 20.16, pp. 2626–2635 (cit. on p. 22).

Lanckriet, G., N. Cristianini, P. Bartlett, L. El Ghaoui, and M. I. Jordan (2004b). „Learning the Kernel Matrix with Semi-Definite Programming." In: *J. Mach. Learn. Res* 5, pp. 27–72 (cit. on pp. 3, 5, 22, 28–30).

Le Borgne, Y., S. Santini, and G. Bontempi (2007). „Adaptive Model Selection for Time Series Prediction." In: *in Wireless Sensor Networks", Signal Processing (Elsevier publisher), Volume: 87, Issue: 12, Pages*, pp. 3010–3020 (cit. on p. 115).

Le Thi, H. A. and T. Pham Dinh (2005). „The DC (Difference of Convex Functions) Programming and DCA Revisited with DC Models of Real World Nonconvex Optimization Problems." In: *Annals of Operations Research* 133.1–4, pp. 23–46 (cit. on p. 52).

Lee, Wee Sun and Bing Liu (2003). „Learning with Positive and Unlabeled Examples Using Weighted Logistic Regression." In: *ICML* (cit. on p. 138).

Li, Xiao-Li, Philip S. Yu, Bing Liu, and See-Kiong Ng (2009). „Positive Unlabeled Learning for Data Stream Classification.“ In: *SIAM International Conference on Data Mining*, pp. 257–268 (cit. on p. 138).

Liu, Bing, Wee Sun Lee, and Philip S. Yu Xiaoli Li (2002). „Partially Supervised Classification of Text Documents.“ In: *ICML* (cit. on p. 138).

Natarajan, B. K. (1989). „On learning sets and functions.“ In: *Machine Learning* 4, pp. 67–97 (cit. on p. 9).

Nesterov, Y. (1983). „A method of solving a convex programming problem with convergence rate $o(1/k^2)$.“ In: *Soviet. Math. Dokl.* 27, pp. 372–376 (cit. on pp. 58, 59).

Nocedal, J. and S. J. Wright (2006). *Numerical optimization*. Springer (cit. on p. 19).

Nowozin, S. and C. H. Lampert (2011). „Structured Learning and Prediction in Computer Vision.“ In: *Foundations and Trends in Computer Graphics and Vision* 6.3-4, pp. 185–365 (cit. on p. 14).

Nowozin, S., C. Rother, S. Bagon, T. Sharp, B. Yao, and P. Kohli (2011). „Decision tree fields.“ In: *ICCV*, pp. 1668–1675 (cit. on pp. 64, 65).

Ong, C. S. and A. Zien (2008). „An Automated Combination of Kernels for Predicting Protein Subcellular Localization.“ In: *WABI*. Ed. by K. A. Crandall and J. Lagergren. Springer, pp. 186–197 (cit. on pp. 22, 33).

Papadimitriou, S., J. Sun, C. Faloutsos, and P. S. Yu (2008). „Hierarchical, Parameter-Free Community Discovery.“ In: *ECML/PKDD (2)*, pp. 170–187 (cit. on p. 72).

Pearl, J. (1986). „Fusion, Propagation, and Structuring in Belief Networks.“ In: *Artificial Intelligence* 29.3, pp. 241–288 (cit. on p. 18).

Pham Dinh, T. and H. A. Le Thi (1998). „DC optimization algorithms for solving the trust region subproblem.“ In: *SIAM Journal on Optimization* 8.2, pp. 476–505 (cit. on p. 52).

Pletscher, P. (2012). „Towards Accurate Structured Output Learning and Prediction.“ PhD thesis. Switzerland: ETH Zurich (cit. on p. 14).

Pletscher, P. and S. Wulff (2012). „LPQP for MAP: Putting LP Solvers to Better Use.“ In: *Proceedings of the 29th International Conference on Machine Learning (ICML)* (cit. on pp. ix, 6, 45).

Polikar, R. (2009). „Ensemble learning.“ In: *Scholarpedia* 4.1, p. 2776 (cit. on p. 126).

Rabiner, L. R. (1989). „A tutorial on hidden Markov models and selected applications in speech recognition.“ In: *Proceedings of IEEE*. Vol. 77, pp. 257–286 (cit. on p. 18).

Rakotomamonjy, A., F. R. Bach, S. Canu, and Y. Grandvalet (2008). „SimpleMKL.“ In: *J. Mach. Learn. Res* 9, pp. 2491–2521 (cit. on pp. 23, 29, 33, 35–38).

Ravikumar, P., A. Agarwal, and M. J. Wainwright (2010). „Message-passing for Graph-structured Linear Programs: Proximal Methods and Rounding Schemes.“ In: *J. Mach. Learn. Res.* 11, pp. 1043–1080 (cit. on p. 61).

Ravikumar, P. and J. Lafferty (2006). „Quadratic Programming Relaxations for Metric Labeling and Markov Random Field MAP Estimation.“ In: *ICML* (cit. on pp. 49, 50, 55).

Roth, D. (1996). „On the hardness of approximate reasoning.“ In: *Artificial Intelligence* 82, pp. 273–302 (cit. on p. 17).

Savchynskyy, B., J. H. Kappes, S. Schmidt, and C. Schnörr (2011). „A study of Nesterov's scheme for Lagrangian decomposition and MAP labeling.“ In: *CVPR*, pp. 1817–1823 (cit. on pp. 58, 59).

Schlesinger, M. I. (1976). „Syntactic analysis of two-dimensional visual signals in noisy conditions.“ In: *Kibernetika* 4, pp. 113–130 (cit. on p. 48).

Schölkopf, B. and A. J. Smola (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press (cit. on pp. 12, 13, 21).

Settles, B. (2009). *Active Learning Literature Survey*. Computer Sciences Technical Report 1648. University of Wisconsin–Madison (cit. on p. 101).

Shawe-Taylor, J. and N. Cristianini (2004). *Kernel Methods for Pattern Analysis*. New York, NY, USA: Cambridge University Press (cit. on p. 13).

Sonnenburg, S., G. Rätsch, and C. Schäfer (2005). „A general and efficient multiple kernel learning algorithm.“ In: *Neural Information Processings Systems* (cit. on p. 22).

Sonnenburg, S., G. Rätsch, C. Schäfer, and B. Schölkopf (2006). „Large Scale Multiple Kernel Learning.“ In: *J. Mach. Learn. Res* 7, pp. 1531–1565 (cit. on pp. 3, 5, 22, 28, 29, 31, 33, 35).

Sontag, D., A. Globerson, and T. Jaakkola (2011). „Introduction to Dual Decomposition for Inference.“ In: *Optimization for Machine Learning*.

Ed. by S. Sra, S. Nowozin, and S. J. Wright. MIT Press (cit. on pp. 3, 86, 90, 93–95).

Sontag, D., T. Meltzer, A. Globerson, Y. Weiss, and T. Jaakkola (2008). „Tightening LP Relaxations for MAP using Message-Passing." In: *Proceedings of the 24th Conference in Uncertainty in Artificial Intelligence (UAI)*, pp. 503–510 (cit. on pp. 49, 60, 63).

Sra, S., S. Nowozin, and S. J. Wright (2011). *Optimization for Machine Learning*. Neural information processing series. MIT Press (cit. on p. 19).

Srebro, Nathan and Shai Ben-david (2006). „Learning bounds for support vector machines with learned kernels." In: *In Annual Conference On Learning Theory (COLT*. Springer, pp. 169–183 (cit. on p. 30).

Sriperumbudur, B. and G. Lanckriet (2009). „On the Convergence of the Concave-Convex Procedure." In: *NIPS*, pp. 1759–1767 (cit. on pp. 52, 59).

Steinwart, I. and A. Christmann (2008). *Support Vector Machines*. 1st. Springer Publishing Company, Incorporated (cit. on pp. 101, 104).

Steinwart, I., D. Hush, and C. Scovel (2006). „An Explicit Description of the Reproducing Kernel Hilbert Spaces of Gaussian RBF Kernels." In: *Information Theory, IEEE Transactions on* 52.10, pp. 4635–4643 (cit. on p. 12).

Steinwart, I. and C. Scovel (2007). „Fast Rates for Support Vector Machines." In: 35.2, pp. 575–607 (cit. on p. 101).

Sutton, C. and A. Mccallum (2012). „An Introduction to Conditional Random Fields." In: *Foundations and Trends in Machine Learning* 4.4, pp. 267–373 (cit. on pp. 18, 125).

Tanay, A., R. Sharan, and R. Shamir (2006). „Biclustering Algorithms: A Survey." In: *Handbook of Computational Molecular Biology* (cit. on p. 71).

Taskar, B., C. Guestrin, and D. Koller (2003). „Max-Margin Markov Networks." In: *Advances in Neural Information Processing Systems 16 (NIPS)* (cit. on pp. 14, 18).

Tsochantaridis, I., T. Joachims, T. Hofmann, and Y. Altun (2005). „Large Margin Methods for Structured and Interdependent Output Variables." In: *Journal of Machine Learning* 6, pp. 1453–1484 (cit. on pp. 14, 18).

Urner, R. (2013). „Learning with non-Standard Supervision." PhD thesis. Waterloo, Canada: University of Waterloo (cit. on p. 99).

Urner, R., S. Ben-David, and S. Shalev-Shwartz (2011). „Unlabeled data can Speed up Prediction Time.“ In: *ICML* (cit. on pp. 102, 104).

Urner, R., S. Wulff, and S. Ben-David (2013). „PLAL: CLuster-based active learning.“ In: *Conference on Learning Theory (COLT)* (cit. on pp. ix, 6, 99, 107).

Valiant, L. G. (1984). „A Theory of the Learnable.“ In: 27, pp. 1134–1142 (cit. on p. 10).

Vandenberghe, L. (2012). *Fast proximal gradient methods*. Lecture notes (cit. on p. 58).

Vapnik, V. N. (1982). *Estimation of Dependences Based on Empirical Data: Springer Series in Statistics (Springer Series in Statistics)*. Springer-Verlag New York, Inc. (cit. on p. 9).

Vapnik, V. N. and A. Y. Chervonenkis (1971). „On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities.“ In: *Theory of Probability and its Applications* 16, pp. 264–280 (cit. on p. 9).

Verma, N., S. Kpotufe, and S. Dasgupta (2012). „Which Spatial Partition Trees are Adaptive to Intrinsic Dimension?“ In: *CoRR* abs/1205.2609 (cit. on p. 105).

Viterbi, A. (1967). „Error bounds for convolutional codes and an asymptotically optimum decoding algorithm.“ In: *IEEE Transactions on Information Theory* 13.2, pp. 260–269 (cit. on p. 18).

Wainwright, M. J. and M. I. Jordan (2003). *Variational inference in graphical models: The view from the marginal polytope* (cit. on p. 48).

Wainwright, M. J. and M. I. Jordan (2008). *Graphical Models, Exponential Families, and Variational Inference* (cit. on pp. 3, 48, 49, 53, 54).

Wulff, S. (2008). „Computational Complexity Of Bi-clustering.“ University of Waterloo, Ontario, Canada (cit. on pp. ix, 6, 68, 72).

Wulff, S. and C. S. Ong (2013). „Analytic center cutting plane method for multiple kernel learning.“ In: *Annals of Mathematics and Artificial Intelligence* (cit. on pp. ix, 5, 21).

Wulff, S., R. Urner, and S. Ben-David (2013). „Monochromatic Bi-Clustering.“ In: *Proceedings of the 30th International Conference on Machine Learning (ICML)* (cit. on pp. ix, 6, 67).

Xu, Z., R. Jin, I. King, and M. R. Lyu (2008). „An Extended Level Method for Efficient Multiple Kernel Learning.“ In: *NIPS* (cit. on pp. 23, 33, 35–38).

Yanover, C., T. Meltzer, and Y. Weiss (2006). „Linear Programming Relaxations and Belief Propagation – An Empirical Study.“ In: *JMLR* (cit. on p. 63).

Yuille, A. L. and A. Rangarajan (2003). „The concave-convex procedure.“ In: *Neural Computation* (cit. on p. 52).

Zien, A. and C. S. Ong (2007). „Multiclass Multiple Kernel Learning.“ In: *ICML* (cit. on pp. 5, 28, 29).

ACRONYMS

MAP	Maximum-A-Posteriori
ERM	Empirical Risk Minimization
RRM	Regularized Risk Minimization
RLM	Regularized Loss Minimization
MRF	Markov Random Field
CRF	Conditional Random Field
SVM	Support Vector Machine
MKL	Multiple Kernel Learning
SMO	Sequential Minimal Optimization
LASSO	Least Absolute Shrinkage and Selection Operator
LP	Linear Program
QP	Quadratic Program
CCCP	Concave-Convex Procedure
DC	Difference of convex functions
SA	Simulated Annealing
ACCPM	Analytic Center Cutting Plane Method
SDP	Semidefinite Programming
SILP	Semi-Infinite Linear Programming
KCG	Kelley-Cheney-Goldstein
QCQP	Quadratically Constrained Quadratic Program
GS	Gibbs Sampler

DD	Dual Decomposition
PTAS	Polynomial Time Approximation Scheme
KL	Kullback-Leibler
MPLP	Max-Product Linear Programming
TRWS	Sequential Tree-Reweighted Message Passing
LPQP	Linear and Quadratic Program relaxation
LPQP-U	LPQP with Uniform Penalty
LPQP-T	LPQP with Tree-weighted Penalty
LOO	Leave One Out
AL	Active Learning
PL	Probabilistic Lipschitzness
NN	Nearest Neighbor
TGI-1	Tinynode Germany Installation 1
TGI-2	Tinynode Germany Installation 2
TFI	Tinynode France Installation

CURRICULUM VITAE

Name	Sharon Wulff
Date of birth	August 2, 1980 in Tel-Aviv, Israel
2001 -2005	Bsc. Computer Engineering and Math, Hebrew University, Jerusalem, Israel.
2005 - 2007	Software engineer at CheckPoint Technologies, Tel-Aviv, Israel.
2007-2008	Master of Computer Science, Machine Learning, University of Waterloo, Waterloo, ON, Canada
2008 - 2014	Doctoral studies, Machine Learning, ETH Zürich, Switzerland