Selected machine learning reductions

Anna Choromanska

Submitted in partial fulfillment of the

Requirements for the degree

of Doctor of Philosophy

in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2014

ABSTRACT

Selected machine learning reductions

Anna Choromanska

Machine learning is a field of science aiming to extract knowledge from the data. Optimization lies in the core of machine learning as many learning problems are formulated as optimization problems, where the goal is to minimize/maximize an objective function. More complex machine learning problems are then often solved by reducing them to simpler sub-problems solvable by known optimization techniques. This dissertation addresses two elements of the machine learning system 'pipeline', designing efficient basic optimization tools tailored to solve specific learning problems, or in other words optimize a specific objective function, and creating more elaborate learning tools with sub-blocks being essentially optimization solvers equipped with such basic optimization tools. In the first part of this thesis we focus on a very specific learning problem where the objective function, either convex or non-convex, involves the minimization of the partition function, the normalizer of a distribution, as is the case in conditional random fields (CRFs) or log-linear mod-

els. Our work proposes a tight quadratic bound on the partition function which parameters are easily recovered by a simple algorithm that we propose. The bound gives rise to the family of new optimization learning algorithms, based on bound majorization (we developed batch, both full-rank and low-rank, and semi-stochastic variants), with linear convergence rate that successfully compete with state-of-the-art techniques (among them gradient descent methods, Newton and quasi-Newton methods like L-BFGS, etc.). The only constraint we introduce is on the number of classes which is assumed to be finite and enumerable. The bound majorization method we develop is simultaneously the first reduction scheme discussed in this thesis, where throughout this thesis by 'reduction' we understand the learning approach or algorithmic technique converting a complex machine learning problem into a set of simpler problems (that can be as small as a single problem).

Secondly, we focus on developing two more sophisticated machine learning tools, for solving harder learning problems. The tools that we develop are built from basic optimization sub-blocks tailored to solve simpler optimization sub-problems. We first focus on the multi class classification problem where the number of classes is very large. We reduce this problem to a set of simpler sub-problems that we solve using basic optimization methods performing additive update on the parameter vector. Secondly we address the problem of learning data representation when the data is unlabeled for any classification task. We reduce this problem to a set of

simpler sub-problems that we solve using basic optimization methods, however this time the parameter vector is updated multiplicatively. In both problems we assume that the data come in a stream that can even be infinite. We will now provide more specific description of each of these problems and describe our approach for solving them.

In the multi class classification problem it is desirable to achieve train and test running times which are logarithmic in the label complexity. The existing approaches to this problem are either intractable or do not adapt well to the data. We propose a reduction of this problem to a set of binary regression problems organized in a tree structure and introduce a new splitting criterion (objective function) allowing gradient descent style optimization (bound optimization methods can also be used). A decision tree algorithm that we obtain differs from traditional decision trees in the objective optimized, and in how that optimization is done. The different objective has useful properties, such us it guarantees balanced and small-error splits, while the optimization uses an online learning algorithm that is queried and trained simultaneously as we pass over the data. Furthermore, we prove an upper-bound on the number of splits required to reduce the entropy of the tree leafs below threshold ϵ. We empirically show that the trees we obtain have logarithmic depth, which implies logarithmic training and testing running times, and significantly smaller error than random trees.

Finally, we consider the problem of unsupervised (clustering) learning of data representation, where the quality of obtained clustering is measured using a very simple, intuitive and widely cited clustering objective, k-means clustering objective. We introduce a family of online clustering algorithms by extending algorithms for online supervised learning, with access to expert predictors (which are basic sub-blocks of our learning system), to the unsupervised learning setting. The parameter vector corresponds to the probability distribution over the experts. Different update rules for the parameter vector depend on an approximation to the current value of the k-means clustering objective obtained by each expert, and model different levels of non-stationarity in the data. We show that when the experts are batch clustering algorithms with approximation guarantees with respect to the k-means clustering objective, applied to a sliding window of the data stream, our algorithms obtain approximation guarantees with respect to the k-means clustering objective. Thus simultaneously we address an open problem posed by Dasgupta for approximating k-means clustering objective on data streams. We experimentally show that our algorithms' empirical performance tracks that of the best clustering algorithm in its experts set and that our algorithms outperform widely used online algorithms.

Codes for all machine learning tools we developed are publicly released.

Table of Contents

List of Tables vi

List of Figures viii

Acknowledgments x

Chapter 1: Introduction 1

1.1 Selected machine learning optimization methods for log-linear models 3

1.1.1 Partition function 4

1.1.2 Bound majorization methods 5

1.1.3 First- and second-order generic optimization methods 6

1.2 Online multi class classification problem 10

1.2.1 From multi class to binary classification setting 11

1.2.2 Optimization with statistical and computational constraints 12

1.3 Online k-means clustering problem 13

1.3.1 Learning with expert advice 15

1.3.2 k-means clustering objective 18

1.3.3 Optimization methods for k-means clustering objective . . . 19

1.4 Outline of contributions 21

Chapter 2: Quadratic bound majorization for partition function optimization 25

2.1 Partition Function Bound 27

2.2 Bound in convex and non-convex learning problems 31

2.2.1 Maximum entropy problem 31

2.2.2 Conditional Random Fields and Log-Linear Models 32

2.2.3 Latent Conditional Likelihood 37

2.2.4 Graphical Models for Large n 40

2.3 Low-Rank Bounds for Large d 44

2.4 Batch bound method versus generic optimization methods: empirical evaluation 46

2.4.1 ℓ_2-regularized logistic regression 46

2.4.2 Markov CRFs 48

2.4.3 Maximum latent conditional likelihood problems 50

2.5 Conclusion 51

Chapter 3: Semi-stochastic partition function bound majorization 53

3.1 Quadratic bound methods 55

3.2 Theoretical analysis of the semi-stochastic quadratic bound majorization method 58

3.2.1 General Convergence Theory 59

3.2.2 Rates of Convergence for Logistic Regression 63

3.3 Semi-stochastic bound method versus generic optimization methods: empirical evaluation 65

3.3.1 Efficient inexact solvers 66

3.3.2 Mini-batches selection scheme 68

3.3.3 Step size 69

3.3.4 Comparator methods 69

3.3.5 Experiments 71

3.4 Conclusion 73

Chapter 4: Online multi class partition trees for logarithmic time predictions **75**

4.1 Splitting criterion 77

4.1.1 Formulation of the objective function 77

4.2 Purity and balancing factors 80

4.2.1 Balancing factor 82

4.2.2 Purity factor 84

4.3 Boosting statement 87

4.4 Algorithm 92

4.5 Empirical evaluation of the splitting criterion 95

4.6 Conclusion 97

Chapter 5: Online *k*-means clustering with expert advice **99**

5.1 From supervised to unsupervised learning setting 101

5.1.1 Preliminaries 102

5.1.2 Algorithms 103

5.2 Performance guarantees 106

5.2.1 Regret bounds 106

5.2.2 Approximation guarantees 110

5.3 Experiments 113

5.4 Conclusion 120

Chapter 6: Conclusion **121**

Chapter 7: Appendix **139**

7.1 Appendix A 139

7.2 Appendix B 148

7.3 Appendix C 150

List of Tables

2.1 Convergence time in seconds under various regularization levels for a) Bupa b) Wine c) Heart d) Ion and e) Hepatitis datasets. 47

2.2 Time in seconds and number of effective passes through the data required to obtain within ϵ of the L-BFGS solution (where $\epsilon = 10^{-4}$) for logistic regression problems on SRBCT, Tumors, Text and SecStr datasets. 48

2.3 Time in seconds and number of effective passes through the data required to obtain within ϵ of the L-BFGS solution (where $\epsilon = 10^{-4}$) for Markov CRF problems on CoNLL and PennTree datasets. λ was set to $\lambda = 10^{1}$. 49

2.4 Comparison of the results obtained by us ([**1**]) with the results from [**2**] for Markov CRF problem on CoNLL dataset. 49

2.5 Test latent log-likelihood at convergence for different values of $m \in \{1, 2, 3, 4\}$ on ion, bupa, hepatitis, wine and SRBCT data-sets. . . . 50

4.1 Testing error for online multi class partition trees and random trees on artificial datasets. 96

4.2 Testing error for online multi class partition trees and random trees on MNIST and RCV1 datasets. 96

5.1 Mean and standard deviation, over the sequence, of k-means cost on points seen so far. $k = 25$ for Gaussians, $k = 15$ otherwise. The best expert and the best 2 scores of the algorithms, per experiment, are bold. Below the triple lines, 3 more experts are added to the ensemble. 115

List of Figures

1.1 Bound majorization scheme. 5

1.2 Two balanced linear splits: red and green. Red split induces larger classification error than the green one. 13

1.3 The learner maintains the distribution over n experts. 15

1.4 A generalized Hidden Markov Model (HMM) of probability of the next observation, given past observations, and the current best expert. 16

1.5 Learn-α setting. 17

2.1 Convergence of test latent log-likelihood on ion, hepatitis, SRBCT, wine and bupa datasets. 51

3.1 Comparison of optimization strategies for l_2-regularized logistic regression. From left to right: training excess cost, testing cost and testing error. From top to bottom: *rcv1* ($\alpha_{\text{SGD}} = 10^{-1}$, $\alpha_{\text{ASGD}} = 1$, $\alpha_{\text{SQB}} = 10^{-1}$), *adult* ($\alpha_{\text{SGD}} = 10^{-3}$, $\alpha_{\text{ASGD}} = 10^{-2}$, $\alpha_{\text{SQB}} = 1$), *sido* ($\alpha_{\text{SGD}} = 10^{-3}$, $\alpha_{\text{ASGD}} = 10^{-2}$, $\alpha_{\text{SQB}} = 1$), *covtype* ($\alpha_{\text{SGD}} = 10^{-4}$, $\alpha_{\text{ASGD}} = 10^{-3}$, $\alpha_{\text{SQB}} = 10^{-1}$), *protein* ($\alpha_{\text{SGD}} = 10^{-3}$, $\alpha_{\text{ASGD}} = 10^{-2}$, $\alpha_{\text{SQB}} = 1$) and *quantum* ($\alpha_{\text{SGD}} = 10^{-4}$, $\alpha_{\text{ASGD}} = 10^{-2}$, $\alpha_{\text{SQB}} = 10^{-1}$) datasets. This figure is best viewed in color. 72

4.1 Blue: upper-bound on the balancing factor, red: lower-bound on the balancing factor, green: interval where the balancing factor lies for different values of the objective function $J(h)$. 83

4.2 Red: the upper-bound on the purity factor as a function of $J(h)$ when the balancing factor is fixed to $\frac{1}{2}$. 87

5.1 Clustering analogs to learning curves: k-means cost versus t for Cloud dataset. 116

5.2 Evolution of weights over experts for Fixed-Share algorithms using different values of the α parameter; Forest Fires dataset; 6-experts. 117

5.3 Evolution of weights maintained by Learn-α, over its α-experts, in the 6-expert Forest Fires experiment. Lowest values of α receive highest weight. 118

5.4 k-means cost on the entire sequence versus k for Cloud, Spambase, Intrusion, Forest Fires, Robot and 25 Gaussians datasets. 119

7.1 Junction tree of depth a) 2 and b) 3. 141

7.2 Illustration of the time spans over which the loss is being computed in Equation 7.6. T = 10, W = 3. Colors red, blue and black correspond to different partitionings, with respect to j, of time span T illustrated in Figure 7.2. 162

7.3 Different partitioning of time span T = 10 into time windows, W = 3, with respect to different values of j. 163

Acknowledgments

I would like to express my gratitude to Tony Jebara, my thesis advisor, for familiarizing me with the bound majorization method, for being significantly involved in my research and for guiding me. He was very supportive and helpful throughout these years. I would like to thank him for sharing with me his expertise, and in particular for encouraging me to explore the bound technique as well as for supporting all my other research endeavors. I am particularly grateful to him for helping me to understand that theory and application in machine learning are complementary worlds that cannot exist without each other. I feel extremely lucky to be his student. He was also the one who exposed me to wonderful collaborations within academia and with industry which I deeply appreciate and highly value. I would also like to express special thanks to John Langford. His extraordinary patience, his persistence and endless support helped me to endure difficult moments in our research where we could not find the way to solve the multi class classification problem under our constraints. The algorithm we finally obtained is the effect of

our long meetings and conversations in Microsoft Research New York. His ability to extract the essence of existing machine learning algorithms and create a completely new learning algorithm of much more general scope was very inspiring to me. And in particular working together on proofs and theoretical justification for our approach was an adventurous and exciting journey I will not forget as well as many hours he spent teaching me and helping me to analyze my code under Vowpal Wabbit environment. Beyond all I thank him for believing in me. I would like to thank Shih-Fu Chang for helping me to develop my machine learning interests during my early years of doctoral studies and supporting my choices. I would also like to thank Claire Monteleoni for working with me on my very first theoretical machine learning paper and carefully listening to my proofs, Dimitri Kanevsky for sharing his expertise in bound optimization techniques with me, Aleksandr Aravkin for the most beautiful collaboration full of joy of discovering and for being such an amazing friend, Alekh Agarwal for exciting research discussions and for supporting my endeavors, and Leon Bottou, Robert Schapire, Dean Foster and Matus Telgarsky for creating the most wonderful research team guiding me during my internship in MSR. I would also like to thank Antoni Grzanka, Ryszard and Czeslawa Grygorczyk and Zygmunt and Ignacy Gryczynscy for making me realize that biology and medicine are very important disciplines of science which can be significantly advanced using machine learning tools. I would like to thank Richard Doty for making me feel very welcomed during my internship in his laboratory, for

supporting me when I decided to do PhD at Columbia University, and for treating me like an equal partner in the scientific discussion since the first day we met, and David Yun who was throughout these years an inspiration to me sharing with me his wisdom and experience. I would also like to thank my big brother, Krzysztof for being a role model to me and a great support when I needed it the most. I cherish all our conversations, both private and research-oriented. I would also like to thank my loving husband, Mariusz for all the love and support he gave me. Thank you both for making me feel so valuable and unique. Finally I would like to thank my Mum and Dad for encouraging me to study mathematics when I was a child and for their wonderful support throughout all these years. With you by my side, my sorrows hurt less and my joys were greater.

1 Introduction

Machine Learning is the field of computer science originated from artificial intelligence (AI) which in recent years became one of the most popular academic and industrial tools for processing and analyzing data. The recent widespread development of web search engines, sensors as well as data-storage and data-acquisition devices has helped make datasets common place. This, however, poses several serious computational challenges for the traditional algorithms which often do not scale well to these datasets and as a result well motivates the study of new machine learning optimization techniques which can better handle underlying learning problems. This thesis provides several fundamental theoretical and algorithmic results that address three such computational challenges. In the supervised setting,

when all data instances are labeled (a label indicates the class to which an example belongs to), we address two important problems. One is the need to design new optimization tools, both batch (updating parameter vector after passing through the entire datasets) and stochastic (updating parameter vector after passing through single data example or mini-batch of examples), which better adapt to the learning problem and thus can accelerate the learning process and, in particular for certain non-convex problems, potentially recover better quality solution to the learning problem. The only constraint we introduce is on the number of classes which is assumed to be finite and enumerable (in practice we assume it is not too large).

Having an access to such efficient optimization tools, a common way to handle the multi class classification problem, where the number of classes is large, is to reduce this more complex problem to a set of smaller sub-problems with much fewer classes, or even binary problems, on which those tools can be applied. The multi class classification problem is the second learning problem that this thesis considers. We introduce very stringent constraints where train and test running times of the algorithm has to be logarithmic in the label complexity and the algorithm works in the online learning setting (the data come in a stream). This, however, is a realistic scenario in many applications, such as classifying online news stories, weather forecasting or real-time decision making.

Another important problem that the online algorithm analyzing streams of data has

to face is that most data sources produce raw data (e.g. speech signal, or images on the web), that is not yet labeled for any classification task, which motivates the study of unsupervised learning and clustering. Clustering typically refers to a broad class of unsupervised learning tasks aimed at partitioning the data into clusters that are appropriate to the specific application and is widely used in practice, in order to summarize the data (e.g. aggregating similar online news stories). And in particular online clustering of data streams minimizing the sum of squared distances of the data points to their closest cluster centers is the third major learning problem that this thesis addresses. We again approach this problem by reducing it to a set of simpler sub-problems that can be solved using existing optimization tools.

Next in this chapter, we will introduce the various frameworks studied that were motivated above, and we will conclude it with the outline of our contributions which also explains the organization of the thesis.

1.1 Selected machine learning optimization methods for log-linear models

The problem of optimizing a cost function expressed as the sum of a loss term over each sample in an input dataset is pervasive in machine learning and will be of central focus of this thesis. One example of a cost function of this type is the partition function. Partition function is a central quantity in many different

learning tasks such as the estimation of probability density functions over sets of random variables. Estimation often requires minimizing the partition function as is the case in conditional random fields (CRFs) and log-linear models [3, 4]. We will focus entirely on this optimization problem in Chapter 2 and 3 of this thesis.

1.1.1 Partition function

For simplicity consider a log-linear density model over discrete $y \in \Omega$

$$p_j(y|\boldsymbol{\theta}) = \frac{1}{Z_j(\boldsymbol{\theta})} h_j(y) \exp(\boldsymbol{\theta}^\top \boldsymbol{f}_j(y)), \quad \text{where} \quad Z_j(\boldsymbol{\theta}) = \sum_y h_j(y) \exp(\boldsymbol{\theta}^\top \boldsymbol{f}_j(y))$$

which is parametrized by a vector $\boldsymbol{\theta} \in \mathbb{R}^d$ of dimensionality $d \in \mathbb{N}$. Here, j indexes each sample in an input dataset, $\mathbf{f}_j : \Omega \mapsto \mathbb{R}^d$ is any vector-valued function mapping an input y to some arbitrary vector. The prior $h_j : \Omega \mapsto \mathbb{R}^+$ is a fixed non-negative measure. The partition function $Z_j(\boldsymbol{\theta})$ is a scalar that ensures that $p_j(y|\boldsymbol{\theta})$ normalizes. Assume that the number of configurations of y is $|\Omega| = n$ and is finite and enumerable (y can be thought of as a class indicator and therefore n is the total number of classes). The partition function is log-convex in $\boldsymbol{\theta}$ and has a lower-bound given via Jensen's inequality. A plethora of machine learning and statistical frameworks involve linear combinations of soft maximum functions of the form $\sum_{j=1}^t \log p_j(y|\boldsymbol{\theta})$ (e.g. multi-class logistic regression, CRFs [3], hidden variable problems [5], maximum entropy problems [6], and many others).

1.1.2 Bound majorization methods

Many decomposition methods for CRFs and structured prediction have sought for more manageable ways to render the learning and prediction problems. An important family of such techniques are iterative scaling [6] and bound majorization methods [7, 8, 9, 10]. They achieve monotonic convergence and, traditionally, were used in learning log-linear models, logistic regression, maximum entropy models and CRFs [11, 7, 6, 3]. Bound majorization techniques decompose an optimization of complicated functions with no closed-form solution, i.e. $\min_{\boldsymbol{\theta}} f(\boldsymbol{\theta})$, through the iterative solution of simpler sub-problems [12, 13]. They use a surrogate function ρ with closed form to monotonically improve from the initial value of the parameter vector by interleaving the following two steps until convergence (Figure 1.1 depicts them graphically):

(a) Find bound $\rho(\boldsymbol{\theta}, \boldsymbol{\theta}_i) \geq f(\boldsymbol{\theta})$, where $\rho(\boldsymbol{\theta}_i, \boldsymbol{\theta}_i) = f(\boldsymbol{\theta}_i)$ and $\boldsymbol{\theta}_i$ is the current value of the parameter vector

(b) Update $\boldsymbol{\theta}_{i+1} = \arg\min_{\boldsymbol{\theta}} \rho(\boldsymbol{\theta}, \boldsymbol{\theta}_i)$

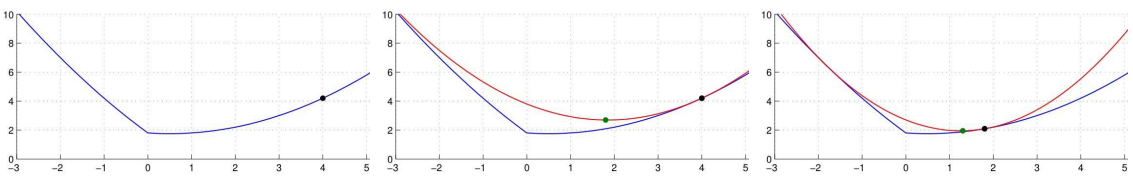


Figure 1.1: Bound majorization scheme.

Similar iterative learning schemes can be found in number of machine learning techniques like Expectation-Maximization (EM) [13], variational methods or variational Bayes [14], the Convex-Concave Procedure (CCCP) [15], and many others.

1.1.3 First- and second-order generic optimization methods

Iterative scaling and bound majorization approaches, typically used to train CRFs and log-linear models, were later surpassed by faster first-order methods [16, 2, 17], second-order methods like Newton, and quasi-Newton methods like Broyden-Fletcher-Goldfarb-Shanno method (BFGS) or L-BFGS [18, 19] (quasi-Newton methods are first-order methods, thus they access the objective function through the first-order oracle, but they also approximate second-order term (Hessian)). These generic first- and second-order optimization techniques perform additive update on the parameter vector of a general form ([20])

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \eta_k \mathbf{G}_k^{-1} \nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_k),$$

where η_k is the current value of the step size (there are different step size strategies that the methods are using [20]), $\mathbf{G}_k$ is the second-order term which in case of first-order methods becomes just the identity matrix $\mathbf{I}$, and $\nabla_{\boldsymbol{\theta}} f(\boldsymbol{\theta}_k)$ is the gradient of the objective function. Many of these techniques have batch, stochastic and semi-stochastic variants. Standard learning systems based on batch methods such as

Newton [21], quasi-Newton [22], e.g. BFGS (full-rank [23] and memory-limited (L-BFGS [24])), steepest descent (see e.g. [20]) or conjugate gradient [25] need to make a full pass through an entire dataset before updating the parameter vector. Even though these methods can converge quickly (e.g. gradient method has linear convergence rate, Newton method has quadratic convergence rate and quasi-Newton methods have superlinear convergence rate), as datasets grow in size, this learning strategy becomes increasingly inefficient. To facilitate learning on massive datasets, the community increasingly turns to stochastic methods.

Stochastic optimization methods interleave the update of parameters after only processing a small mini-batch of examples (potentially as small as a single data-point). They typically have low computational complexity per iteration ([26, 27, 28]). Standard stochastic gradient methods [29, 30] (they are first-order methods) typically have sublinear convergence rate [31, 32]), but in practice they often converge in significantly less iterations than their batch counterparts ([33, 34]) which results in additional significant computational savings. Due to its simplicity and low computational cost, the most popular contemporary stochastic learning technique is stochastic gradient descent (SGD) [29, 26, 30]. SGD updates the parameter vector using the gradient of the objective function as evaluated on a single example (or, alternatively, a small mini-batch of examples). This algorithm admits multiple extensions, including (i) stochastic average gradient method (SAG) that averages

the most recently computed gradients for each training example [31], (ii) methods that compute the (weighted) average of all previous gradients [35, 36], (iii) averaged stochastic gradient descent method (ASGD) that computes a running average of parameters obtained by SGD [37], (iv) stochastic dual coordinate ascent, that optimizes the dual objective with respect to a single dual vector or a mini-batch of dual vectors chosen uniformly at random [38, 39], (v) variance reduction techniques [40, 41, 31, 42] (some do not require storage of gradients, c.f. [40]), (vi) majorization-minimization techniques that minimize a majoring surrogate of an objective function [43, 44] and (vii) gain adaptation techniques [45, 46]. Among those techniques, incremental schemes [40, 38, 31, 43] became particularly popular recently due to their linear convergence rate and superior performance over many state-of-the-art methods, like steepest descent of L-BFGS. Furthermore, incorporating second-order information (i.e. Hessian) into the optimization problem ([45, 33, 47, 48, 49, 50]) was shown to often improve the performance of traditional SGD methods which typically provide fast improvement initially, but are slow to converge near the optimum (see e.g. [51]), require step-size tuning and are difficult to parallelize [52].

Semi-stochastic methods can be viewed as an interpolation between the expensive reliable updates used by full batch methods, and inexpensive noisy updates used by stochastic methods. They inherit the best of both worlds by approaching the

solution more quickly when close to the optimum (like a full batch method) while simultaneously reducing the computational complexity per iteration (though less aggressively than stochastic methods). The semi-stochastic methods begin with iterations that resemble stochastic approach and use relatively few measurements to approximate the gradient (and the second-order term if used), and as the iterations proceed, they gradually increase the number of measurements [51]. Several semi-stochastic extensions have been explored in previous works [51, 53, 54].

Majorization techniques, though slower than generic first- and second-order optimization methods due to often using loose and complicated bounds, exhibit a strong advantage over the latter of adapting not only locally but also globally to the optimization problem. In Chapter 2 and 3 of this thesis we will revisit majorization and show the way to repair its slow convergence. Chapter 2 addresses batch setting and Chapter 3 addresses semi-stochastic setting.

The optimization methods that we have discussed so far are commonly used in machine learning to optimize cost functions of a given model. They often require the computation of the value of the objective, its gradient (first-order term) and sometimes the second-order term (e.g. Hessian or its approximation). The complexity of these computations heavily depends on the number of data examples, data dimensionality and, in the supervised learning setting, the number of classes. Stochastic and semi-stochastic approaches solve the problem of the dependence of

the computations on the number of data examples, when it is large, by using a randomly chosen subset of the dataset to perform a single update of parameters (for stochastic methods the size of this subset is small, for semi-stochastic methods it is initially small and gradually increases). Computational slow-down due to large dimensionality can be solved in various ways, e.g. using low-rank approximations (this solution is also discussed in Chapter 2 of this thesis). The dependence of computational complexity of previously described optimization methods popularly used in numerous machine learning algorithms on the number of classes, when it is large, is a serious drawback of these algorithms. We next provide a gentle introduction to this problem and further discuss it in Chapter 4 of this thesis.

1.2 Online multi class classification problem

The central problem of Chapter 4 of this thesis is the computational complexity in a setting where the number of classes n is very large. In particular we focus on the multi class classification problem. Almost all machine learning algorithms, except for decision trees, have running times for multi class classification which are $O(n)$ with a canonical example being one-against-all classifiers ([55]). For large n this approach becomes intractable in practice which motivates the study of algorithms that has a running time of $\mathcal{O}(\log n)$ (or possibly better, though under uniform distribution assumption one cannot do any better [56]) for both training

and testing, and use online learning algorithms to minimize passes over the data.

1.2.1 From multi class to binary classification setting

A number of prior works have addressed the multi class classification problem. A common approach is to reduce it to a set of binary classification problems [57, 58, 59, 60, 61]. Trees are naturally structured to allow logarithmic time prediction and thus often resulting binary classification problems are organized in tree structures. One example of such approach is Filter Tree ([61]). Filter Tree addresses consistent (and robust) multi class classification, showing that it is possible in the statistical limit. A critical problem not addressed there is the choice of partition (each node in the tree contains a partition that splits the labels assigned to it into two subsets) and thus can only succeed when the chosen partition are easy.

The problem of learning the tree structure by finding good partitions, which result in small multi class classification error, is an important challenge for tree-based approaches. The partition finding problem is addressed in the conditional probability tree ([62]), but that paper addresses conditional probability estimation. Conditional probability estimation can be converted into multi class prediction, but doing so is not a logarithmic time operation. Further work by [63] addresses the partitioning problem by recursively applying spectral clustering on a confusion graph. This approach is at least $O(n)$ at training time and thus is intractable in

practice. Empirically, this approach has been found to sometimes lead to badly imbalanced splits ([64]). Another work by [65] make use of the k-means hierarchical clustering (or other distortion-minimizing clustering) to recover the label sets for a given partition though this work primarily addresses the problem of ranking a very large set of labels rather than the multi class classification problem.

Decision trees are naturally structured to allow logarithmic time prediction. Traditional decision trees often have difficulties with a large number of classes because their splitting criteria are not well-suited to the large class setting (some newer approaches ([66]) have addressed this effectively) and in particular it is unclear how to optimize them online. There exist two important factors that need to be taken into consideration while designing good splitting criterion enabling logarithmic time prediction, and resulting in small classification error. They will be discussed next.

1.2.2 Optimization with statistical and computational constraints

The computational constraint of logarithmic training and testing time requires that the splitting criterion (the objective function) that is optimized in every node of the tree has to induce balanced split of labels. Satisfying this constraint does not guarantee small classification error as illustrated on an example in Figure 1.2. Therefore, additionally to this constraint one most also consider a statistical constraint requiring the splits to induce small classification error.

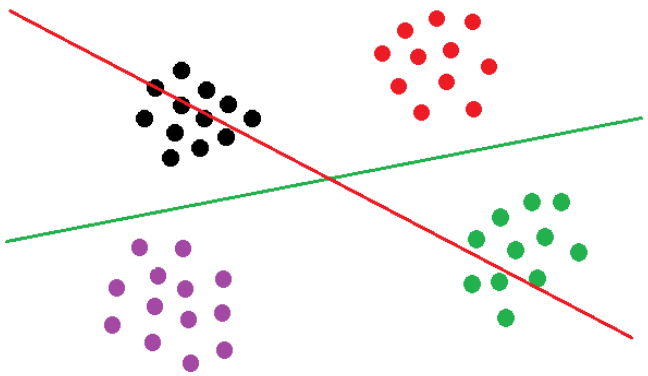


Figure 1.2: Two balanced linear splits: red and green. Red split induces larger classification error than the green one.

Some examples of the objective functions satisfying both criteria are entropy or Gini criterion ([67]), though it is unclear how to optimize them in an online setting. In the setting where the data come in a stream, it is often desirable to be able to online optimize the tree node objective function and in particular use any stochastic optimization method discussed before to perform this optimization (e.g. conditional probability tree uses stochastic gradient descent [62] to train node regressors). We will focus entirely on the problem of designing an objective function permitting such easy optimization in Chapter 4 of this thesis.

1.3 Online *k*-means clustering problem

Chapter 4 of this thesis considers the reduction of the multi class classification problem to a set of binary sub-problems organized in a tree structure, where the

emerging sub-problems can be solved using standard optimization techniques, like in example those discussed before which explore additive update rule for the parameter vector. In Chapter 5 we consider another setting where, instead of having large number of classes, we have no information about the identity of the class for any given data point at all. We therefore focus on the online unsupervised (clustering) learning setting (as opposed to the supervised learning settings we focused on so far), where we aim to learn the representation of the unlabeled data minimizing the sum of squared distances of the data points to their closest cluster centers (a sum of this form is called the k-means clustering objective). To be more specific, we explore the family of algorithms learning with expert advice, which in the literature are traditionally used in the context of supervised learning, in unsupervised (clustering) setting, where the experts solve simple clustering sub-problems (and therefore our approach can also be thought of as a reduction scheme) and, as opposed to the previously described optimization techniques, the parameter vector is updated multiplicatively. Some examples of the members of this family of algorithms include variants of Winnow algorithm [27], the weighted majority algorithm [68], Static-Expert, Fixed-Share(α) and Learn-α algorithms [69, 70] tracking the best expert or a set of experts, and many others [71]. We next explain learning with expert advice setting, where we focus on Static-Expert, Fixed-Share(α) and Learn-α algorithms, then we review existing techniques approximating the k-means clustering objective, and finally in Chapter 5 we focus entirely on the problem of approximat-

ing this objective in the online setting by extending the algorithms learning with expert advice from the supervised to the unsupervised learning setting.

1.3.1 Learning with expert advice

In realistic online settings it is often the case that the observations dynamically change in time, like in case of media streams or weather measurements. Thus the ability to online respond to those changes is an important adaptivity mechanism that the learning algorithm should have. The algorithms that track the best expert or a set of experts ([69, 70]) is a very important group of such adaptive techniques that we focus on. They form their predictions on the basis of a set of n "expert" predictors, i, subject to a probability distribution over experts, $p_t(i)$, which is updated dynamically with time, t (This distribution therefore correspond to a parameter vector that is being updated as was the case for optimization techniques we discussed before. The only difference is that now the update will be

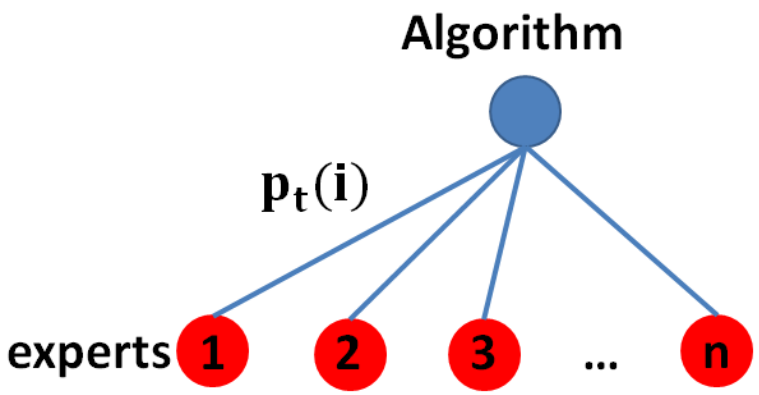


Figure 1.3: The learner maintains the distribution over n experts.

done in multiplicative, not additive way.). Experts can be anything. They can be thought of as black-box procedures that can vary with time and depend on one another; they can be sub-predictors that need not be good or simply the features. Learner maintains the distribution over the experts as shown in Figure 1.3 ([70]) and updates it to reflect how well the experts performed recently.

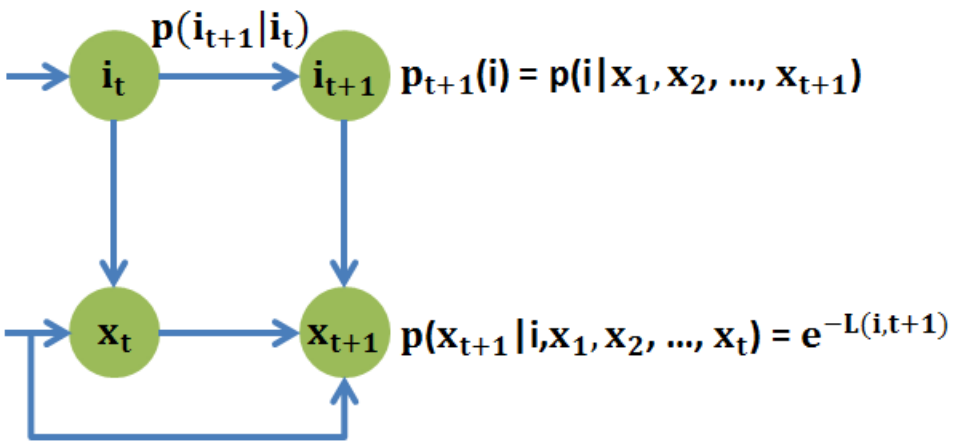


Figure 1.4: A generalized Hidden Markov Model (HMM) of probability of the next observation, given past observations, and the current best expert.

The tracking algorithms we discuss can be derived as Bayesian updates of an appropriately defined generalized Hidden Markov Model (HMM) ([70]) as illustrated in Figure 1.4 ([70]). The identity of the best performing expert at a given time is the hidden variable. Different update rules for $p_t(i)$ correspond to different transition dynamics, modeling different levels of non-stationarity in the data. The multiplicative update on the parameter vector is of a general form ([69, 70])

$$p_{t+1}(i) = \sum_{h=1}^{n} p_t(h) e^{-\frac{1}{2}L(h,t)} P(i|h;\alpha),$$

where $L(h,t)$ is the loss of h^{th} expert at time t, $P(i|h;\alpha)$ is the transition dynamics, parametrized by a scalar α $(0 \leq \alpha \leq 1)$, modeling how the identity of the best expert may change with time, and $P(i|h;\alpha) = \begin{cases} 1-\alpha & \text{if } i = h; \\ \frac{\alpha}{n-1} & \text{o.w.} \end{cases}$

The algorithm using the above multiplicative update rule is commonly known as Fixed-Share(α) and is well-suited to model non-stationary sequences. Its special case is the Static-Expert algorithm, for which $\alpha = 0$ and thus the identity of the best expert cannot change with time, that is used to model stationary sequences. The hybrid of those two is called Learn-α setting, shown in Figure 1.5 ([**70**]), and is used to learn the parameter α online. It uses Static-Expert's updates over a set of Fixed-Share(α) algorithms, each with a different value of the α parameter.

Static-Expert, Fixed-Share(α) and Learn-α algorithms were not explored in the unsupervised learning setting (like clustering) in the literature before. In Chapter 5 of this thesis we will discuss how to use these algorithms in the clustering setting

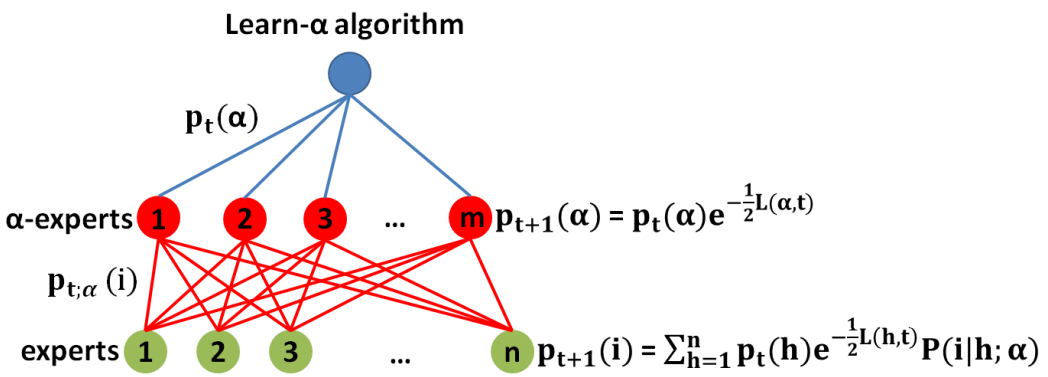


Figure 1.5: Learn-α setting.

to optimize the k-means clustering objective. The definition of this objective and a brief review of the existing techniques optimizing it is provided next.

1.3.2 *k*-means clustering objective

Notice that every chapter of this thesis is dedicated to a different objective function of major significance in machine learning and selected methods well-suited to efficiently optimize it. Chapter 2 and 3 focus on optimizing the partition function and Chapter 4 considers optimizing the tree node splitting criterion inducing both balanced and small-error splits. Both objective functions are considered in the context of supervised learning setting. In Chapter 5 of this thesis we consider the third objective function frequently used in unsupervised learning setting, the k-means clustering objective. It is one of the most widely-cited clustering objectives for data in Euclidean space and is also very simple and intuitively more robust to outliers that many others, like in example k-center clustering objective, but is very hard to optimize. The k-means clustering objective is defined as

$$\Phi_X(C) = \sum_{x \in S} \min_{c \in C} \|x - c\|^2,$$

where S is a finite set of n points in $\mathbb{R}^d$, k is a fixed positive integer denoting the number of clusters (we assume it is given) and $C \in \mathbb{R}^d$ is a set of cluster centers. We often refer to this objective as the "k-means cost" of C on X. This objective

formalizes an intuitive measure of goodness for a clustering of points in Euclidean space. Optimizing the k-means objective is known to be NP-hard, even for $k = 2$ [72] and there only exist few algorithms provably approximating it (Definitions 1 and 2 explain the notion of approximation), even in the batch setting.

Definition 1. *A b-**approximate** k-means clustering algorithm, for a fixed constant b, on any input data set X, returns a clustering C such that: $\Phi_X(C) \leq b \cdot OPT_X$, where OPT_X is the optimum of the k-means objective on data set X.*

Definition 2. *An (a, b)-**approximate** k-means clustering algorithm, is an algorithm that is b-approximate and returns at most $a \cdot k$ centers.*

1.3.3 Optimization methods for k-means clustering objective

We will now briefly review existing approaches attempting to approximate the k-means clustering objective. The widely used "k-means algorithm," also known as Lloyd's algorithm [73], is a batch clustering algorithm that can be viewed as a hard-assignment variant of Expectation-Maximization (EM). While it typically converges quickly, its solution has not been shown to approximate the k-means objective. There exist some batch clustering algorithms with approximation guarantees with respect to the k-means objective. Some examples include k-means++, an algorithm that approximates the k-means objective, by a factor of $O(\log k)$ [74]. Constant approximations have been shown for a local search technique [75], the k-means#

algorithm [76], which outputs $O(k \log k)$ centers, and the work of [77], which outputs $O(k)$ centers. There has also been some work on ensemble methods for clustering, in a batch setting, e.g. [78, 79]. Several works have studied clustering finite data streams [80, 76, 81]. Work by [76] does so with respect to the k-means objective, and extends a streaming clustering algorithm ([80]) for the k-medoid objective (also known as k-median), which is to minimize the sum of distances, in a general metric space, of the points to their closest centers, using a subset of the input points.

A number of techniques for online clustering have enjoyed success in practice, such as [82], and variants of EM (e.g. [83]), and some have been analyzed under stochastic assumptions on the data, *e.g.* [84]. Several online clustering algorithms have approximation guarantees with respect to clustering objectives other than k-means. A doubling algorithm due to [85], and the cover tree algorithm of [86], both provide a constant approximation to the k-center objective, minimizing the maximum distance from an input point to its closest cluster center, in a general metric space.

We are not aware of any existing before online approach provably approximating the k-means clustering objective, other than the randomized PCA method [87], which could be viewed as clustering for the case $k = 1$. It is an open problem posed by Dasgupta who suggested to use an evaluation framework analogous to regret [88] to evaluate the performance of the online algorithm approximating k-

means clustering objective. The regret framework, for the analysis of supervised online learning algorithms like previously discussed Static-Expert, Fixed-Share(α) or Learn-α algorithms, evaluates algorithms with respect to their additional prediction loss relative to a hindsight-optimal comparator method. With the goal of analyzing online clustering algorithms, Dasgupta proposed bounding the difference between the cumulative clustering loss of the algorithm since the first observation:

$$L_T(\texttt{alg}) = \sum_{t \leq T} \min_{c \in C_t} \|x_t - c\|^2$$

where the algorithm outputs a clustering, C_t, before observing the current point, x_t, and the optimal k-means cost on the points seen so far. We will focus entirely on the optimization technique of learning with expert advice and the evaluation framework proposed by Dasgupta to provably approximate the k-means clustering objective online in Chapter 5 of this thesis.

1.4 Outline of contributions

The organization of our contributions is as follows.

- Chapter 2 and 3

  Chapter 2 and 3 of this thesis focus on the optimization techniques performing the additive update on the parameter vector. Both chapters consider

supervised learning setting. The objective function of central interest in this chapter is the partition function. In Chapter 2 of this thesis we revisit majorization and repair its slow convergence by proposing a tighter quadratic upper-bound on the log-partition function. We first consider the batch setting and prove the linear convergence rate of our bound optimization method. We show various applications of the bound to ℓ_2-regularized logistic regression, maximum entropy problems, training CRFs and graphical models (with small tree-width) and maximum latent likelihood settings. We provide low-rank versions of the bound for large-scale problems. In Chapter 3 we consider semi-stochastic variant of the quadratic bound method, which we call SQB. We prove both global convergence (to a stationary point) of SQB under very weak assumptions, and linear convergence rate under stronger assumptions on the objective. The experiments we present in both chapters show advantages of the new bound majorization technique over state-of-the-art first- and second-order generic optimization methods on convex and non-convex learning problems.

- Chapter 4

  In Chapter 4 we address the multi class classification problem where the number of classes is large which is the setting that we deliberately omit in Chapter 2 and 3 (thus in this chapter we still consider supervised learning setting).

We show a reduction of this problem to a set of binary regression problems organized in a tree structure and we introduce a simple top-down criterion for purification of labels. This splitting criterion is the objective function of central interest in this chapter. Depending on the regression model used in tree nodes, this new splitting criterion allows efficient online optimization using any standard optimization technique, like gradient descent style optimization or bound optimization. We prove that maximizing the proposed objective function (splitting criterion) leads simultaneously to pure and balanced splits. The latter guarantees train and test running times that are logarithmic in the label complexity. We show an upper-bound on the number of splits required to reduce the entropy of the tree leafs, a standard measure of the quality of decision trees, below threshold ϵ. We present the experimental evaluation showing that our algorithm leads to significantly smaller error than random trees.

- Chapter 5

  Chapter 5 focuses on the optimization techniques performing the multiplicative update on the parameter vector. This chapter considers unsupervised learning setting (clustering). The objective function of central interest in this chapter is the k-means clustering objective. We present new generic clustering algorithms, as opposed to application-specific ones, optimizing the

k-means clustering objective online. Our new clustering algorithms, which we refer to as OCE, extend algorithms for online supervised learning, with access to expert predictors, to the unsupervised learning setting. When the experts are batch clustering algorithms with b-approximation guarantees with respect to the k-means objective (for example, the k-means++ or k-means# algorithms), applied to a sliding window of the data stream, our algorithms obtain approximation guarantees with respect to the k-means objective on the entire data stream seen so far. The form of these online clustering approximation guarantees is novel, and extends an evaluation framework proposed by Dasgupta as an analog to regret. We empirically show that OCE exploits the performance advantages of the clustering algorithms used as experts, and has comparable or better performance over state-of-the-art clustering methods.

- Chapter 6

  In Chapter 6 we summarize various contributions of the thesis and provide future extensions.

- Chapter 7

  In the Appendix we present additional proofs.

2

Quadratic bound majorization for partition function optimization

This chapter is based on joint work with Tony Jebara that originally appeared in [1]. *All codes are released and are publicly available at* www.columbia.edu/∼aec2163/NonFlash/Papers/Papers.html.

In this chapter we study the maximum likelihood inference based on partition function optimization. The existing first- and second-order optimization techniques were shown to significantly outperform iterative scaling and bound majorization techniques in this setting. Slow performance of traditional majorization methods

is due to loose and complicated bounds that they are using. Intuitively, developing tighter and simpler bounds should accelerate bound majorization methods and further enhance their advantage over generic first- and second-order optimization techniques of adapting simultaneously locally and globally to the objective function.

The objective of this chapter is to revisit majorization and repair its slow convergence by deriving a tighter bound on the log-partition function. In this chapter we focus on the batch framework where the learner makes a full pass through the entire dataset before updating the parameter vector. We consider the supervised learning setting where the number of classes is assumed to be finite and enumerable (in practice we assume it is not too large). We want to emphasize, however, that the bound technique could in general be as well used in the unsupervised learning framework. This chapter is organized as follows. Section 2.1 presents the partition function bound, Section 2.2 shows how to apply this new bound to different convex and non-convex learning problems: Section 2.2.1 shows that the bound can naturally be applied in maximum entropy estimation or minimum relative entropy estimation, Section 2.2.2 uses it for majorization in CRFs, Section 2.2.3 shows extensions to latent likelihood, and Section 2.2.4 shows extensions to graphical models. Extending the bound to high dimensional problems is done in Section 2.3. Section 2.4 provides experiments and Section 2.5 concludes with a brief discussion.

2.1 Partition Function Bound

Consider a log-linear density model over discrete $y \in \Omega$

$$p(y|\boldsymbol{\theta}) \quad = \quad \frac{1}{Z(\boldsymbol{\theta})} h(y) \exp\left(\boldsymbol{\theta}^\top \mathbf{f}(y)\right)$$

which is parameterized by a vector $\boldsymbol{\theta} \in \mathbb{R}^d$ of dimensionality $d \in \mathbb{N}$. Here, $\mathbf{f} : \Omega \mapsto \mathbb{R}^d$ is any vector-valued function mapping an input y to some arbitrary vector. The prior $h : \Omega \mapsto \mathbb{R}^+$ is a fixed non-negative measure. The partition function $Z(\boldsymbol{\theta})$ is a scalar that ensures that $p(y|\boldsymbol{\theta})$ normalizes, i.e. $Z(\boldsymbol{\theta}) = \sum_y h(y) \exp(\boldsymbol{\theta}^\top \mathbf{f}(y))$. Assume that the number of configurations of y is $|\Omega| = n$ and is finite and enumerable. The partition function, as was already discussed before, is log-convex in $\boldsymbol{\theta}$ and has lower-bound given via Jensen's inequality. We will now show an analogous quadratic upper-bound on the log-partition function that is parameterized by three parameters: additive scalar z, first-order term $\boldsymbol{\mu}$ which is just the gradient of the log-partition function and second-order term $\boldsymbol{\Sigma}$. The bound is given in Theorem 1 and Algorithm 1 computes[1] the bound's parameters.

Theorem 1. *Algorithm 1[2] finds* $z, \boldsymbol{\mu}, \boldsymbol{\Sigma}$ *such that* $z \exp(\frac{1}{2}(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^\top \boldsymbol{\Sigma} (\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}) + (\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^\top \boldsymbol{\mu})$ *upper-bounds* $Z(\boldsymbol{\theta}) = \sum_y h(y) \exp(\boldsymbol{\theta}^\top \mathbf{f}(y))$ *for any* $\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}, \mathbf{f}(y) \in \mathbb{R}^d$ *and* $h(y) \in \mathbb{R}^+$ *for all* $y \in \Omega$.

[1] By continuity, take $\tanh(\frac{1}{2}\log(1))/(2\log(1)) = \frac{1}{4}$ and $\lim_{z \to 0^+} \tanh(\frac{1}{2}\log(\alpha/z))/(2\log(\alpha/z)) = 0$.

[2] The figure inside Algorithm 1 depicts the bound on $\log Z(\boldsymbol{\theta})$ for various choices of $\tilde{\boldsymbol{\theta}}$.

Algorithm 1 ComputeBound

Input Parameters $\tilde{\boldsymbol{\theta}}, \mathbf{f}(y), h(y)\ \forall y \in \Omega$

Init $z \to 0^+, \boldsymbol{\mu} = \mathbf{0}, \boldsymbol{\Sigma} = z\mathbf{I}$

For each $y \in \Omega$ {

$\alpha \quad = h(y)\exp(\tilde{\boldsymbol{\theta}}^\top \mathbf{f}(y))$

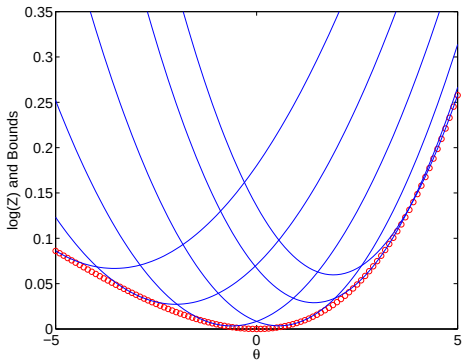
$\mathbf{l} \quad = \mathbf{f}(y) - \boldsymbol{\mu}$

$\boldsymbol{\Sigma} += \frac{\tanh(\frac{1}{2}\log(\alpha/z))}{2\log(\alpha/z)}\mathbf{l}\mathbf{l}^\top$

$\boldsymbol{\mu} += \frac{\alpha}{z+\alpha}\mathbf{l}$

$z \ += \alpha \quad \}$

Output $z, \boldsymbol{\mu}, \boldsymbol{\Sigma}$



Notice that the second-order term $\boldsymbol{\Sigma}$ is different than the Hessian of the log-partition function. In particular, changing the update on $\boldsymbol{\Sigma}$ in Algorithm 1 and choosing $\boldsymbol{\Sigma} = \sum_y h(y)\exp(\tilde{\boldsymbol{\theta}}^\top \mathbf{f}(y))(\mathbf{f}(y) - \boldsymbol{\mu})(\mathbf{f}(y) - \boldsymbol{\mu})^\top$ (with the update on $\boldsymbol{\mu}$ and z unchanged) would yield the second-order Taylor approximation (the Hessian) of the log-partition function.

Proof of the bound. Without loss of generality assume y in Algorithm 1 loops from 1 to n and for the ease of notation $h(y)$ and $\mathbf{f}(y)$ will be h_y and $\mathbf{f}_y$ respectively. Next, write $Z(\boldsymbol{\theta}) = \sum_{j=1}^n \alpha_j \exp(\boldsymbol{\lambda}^\top \mathbf{f}_j)$ by introducing $\boldsymbol{\lambda} = \boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}$ and $\alpha_j = h_j \exp(\tilde{\boldsymbol{\theta}}^\top \mathbf{f}_j)$. Define the partition function over only the first i components as $Z_i(\boldsymbol{\theta}) = \sum_{j=1}^i \alpha_j \exp(\boldsymbol{\lambda}^\top \mathbf{f}_j)$. When $i = 0$, a trivial quadratic upper-bound holds

$$Z_0(\boldsymbol{\theta}) \quad \leq \quad z_0 \exp\left(\tfrac{1}{2}\boldsymbol{\lambda}^\top \boldsymbol{\Sigma}_0 \boldsymbol{\lambda} + \boldsymbol{\lambda}^\top \boldsymbol{\mu}_0\right)$$

with the parameters $z_0 \to 0^+, \boldsymbol{\mu}_0 = \mathbf{0}$, and $\boldsymbol{\Sigma}_0 = z_0\mathbf{I}$. Next, add one term to the

current partition function $Z_1(\boldsymbol{\theta}) = Z_0(\boldsymbol{\theta}) + \alpha_1 \exp(\boldsymbol{\lambda}^\top \mathbf{f}_1)$. Apply the current bound $Z_0(\boldsymbol{\theta})$ to obtain

$$Z_1(\boldsymbol{\theta}) \;\leq\; z_0 \exp(\tfrac{1}{2}\boldsymbol{\lambda}^\top \boldsymbol{\Sigma}_0 \boldsymbol{\lambda} + \boldsymbol{\lambda}^\top \boldsymbol{\mu}_0) + \alpha_1 \exp(\boldsymbol{\lambda}^\top \mathbf{f}_1).$$

Consider the following change of variables

$$\begin{aligned} \mathbf{u} &= \boldsymbol{\Sigma}_0^{1/2}\boldsymbol{\lambda} - \boldsymbol{\Sigma}_0^{-1/2}(\mathbf{f}_1 - \boldsymbol{\mu}_0)) \\ \gamma &= \tfrac{\alpha_1}{z_0} \exp(\tfrac{1}{2}(\mathbf{f}_1 - \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}_0^{-1}(\mathbf{f}_1 - \boldsymbol{\mu}_0)) \end{aligned}$$

The logarithm of the bound becomes

$$\log Z_1(\boldsymbol{\theta}) \;\leq\; \log z_0 - \tfrac{1}{2}(\mathbf{f}_1 - \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}_0^{-1}(\mathbf{f}_1 - \boldsymbol{\mu}_0) + \boldsymbol{\lambda}^\top \mathbf{f}_1 + \log\left(\exp(\tfrac{1}{2}\|\mathbf{u}\|^2) + \gamma\right).$$

Apply Lemma 12 (see Appendix A) to the last term to get

$$\begin{aligned} \log Z_1(\boldsymbol{\theta}) \leq \log z_0 - \tfrac{1}{2}(\mathbf{f}_1 - \boldsymbol{\mu}_0)^\top \boldsymbol{\Sigma}_0^{-1}(\mathbf{f}_1 - \boldsymbol{\mu}_0) + \boldsymbol{\lambda}^\top \mathbf{f}_1 + \log\left(\exp\left(\tfrac{1}{2}\|\mathbf{v}\|^2\right) + \gamma\right) \\ + \frac{\mathbf{v}^\top(\mathbf{u}-\mathbf{v})}{1+\gamma\exp(-\frac{1}{2}\|\mathbf{v}\|^2)} + \frac{1}{2}(\mathbf{u}-\mathbf{v})^\top\left(I + \Gamma \mathbf{v}\mathbf{v}^\top\right)(\mathbf{u}-\mathbf{v}) \end{aligned}$$

where $\Gamma = \frac{\tanh(\frac{1}{2}\log(\gamma\exp(-\frac{1}{2}\|\mathbf{v}\|^2)))}{2\log(\gamma\exp(-\frac{1}{2}\|\mathbf{v}\|^2))}$. The bound in [89] is tight when $\mathbf{u} = \mathbf{v}$ and thus to achieve tightness when $\boldsymbol{\theta} = \tilde{\boldsymbol{\theta}}$ or, equivalently, $\boldsymbol{\lambda} = \mathbf{0}$, we choose $\mathbf{v} =$

$\boldsymbol{\Sigma}_0^{-1/2}(\boldsymbol{\mu}_0 - \mathbf{f}_1)$ yielding

$$Z_1(\boldsymbol{\theta}) \leq z_1 \exp\left(\tfrac{1}{2}\boldsymbol{\lambda}^\top \boldsymbol{\Sigma}_1 \boldsymbol{\lambda} + \boldsymbol{\lambda}^\top \boldsymbol{\mu}_1\right)$$

where we have

$$\begin{aligned} z_1 &= z_0 + \alpha_1 \\ \boldsymbol{\mu}_1 &= \frac{z_0}{z_0 + \alpha_1}\boldsymbol{\mu}_0 + \frac{\alpha_1}{z_0 + \alpha_1}\mathbf{f}_1 \\ \boldsymbol{\Sigma}_1 &= \boldsymbol{\Sigma}_0 + \frac{\tanh(\frac{1}{2}\log(\alpha_1/z_0))}{2\log(\alpha_1/z_0)}(\boldsymbol{\mu}_0 - \mathbf{f}_1)(\boldsymbol{\mu}_0 - \mathbf{f}_1)^\top. \end{aligned}$$

The process is iterated n times (replacing 1 with i and 0 with $i-1$) to produce an overall bound on all terms. □

The bound recovered by Algorithm 1 improves the inequalities previously existing in the literature, e.g. it tightens [7, 8] by eliminating curvature tests and bounding the objective function rather than its Hessian, it generalizes [9] which only holds for $n = 2$ and $h(y)$ constant, and it generalizes [10] which only handles one-dimensional case. Also, unlike a general bounding scheme such as CCCP ([90]) which breaks down a function into convex and concave components, the proposed bound is easy and efficient to manipulate/optimize due to its quadratic form. The bound is computed using Algorithm 1 by iterating over the y variables ("for each $y \in \Omega$") according to an arbitrary ordering. The order in which we enumerate over

Ω varies the $\boldsymbol{\Sigma}$ in the bound (but not the $\boldsymbol{\mu}$ and z) when $|\Omega| > 2$. However, we empirically investigated the influence of various orderings on bound performance (in all the experiments presented in Section 2.4) and noticed no significant effect across ordering schemes.

If there are no constraints on the parameters (i.e. any $\boldsymbol{\theta} \in \mathbb{R}^d$ is admissible), the bound technique give rise to a simple closed-form iterative update rule for the parameter vector: $\tilde{\boldsymbol{\theta}} \leftarrow \tilde{\boldsymbol{\theta}} - \boldsymbol{\Sigma}^{-1}\boldsymbol{\mu}$. Alternatively, if $\boldsymbol{\theta}$ must satisfy linear (convex) constraints it is straightforward to compute an update by solving a quadratic (convex) program. This update rule is interleaved with the bound computation.

2.2 Bound in convex and non-convex learning problems

Majorization based on the new bound can be applied to many different convex and non-convex learning problems involving the minimization of the partition function. In this section we show selected partition function-based optimization problems and discuss how to apply the bound to them.

2.2.1 Maximum entropy problem

We show here that partition functions arise naturally in maximum entropy estimation or minimum relative entropy $\mathcal{RE}(p\|h) = \sum_y p(y) \log \frac{p(y)}{h(y)}$ estimation and thus in these problems quadratic bound majorization method can be applied. Consider

the following problem:

$$\min_p \mathcal{RE}(p\|h) \text{ s.t.} \sum_y p(y)\mathbf{f}(y) = \mathbf{0}, \sum_y p(y)\mathbf{g}(y) \geq \mathbf{0}.$$

Here, assume that $\mathbf{f} : \Omega \mapsto \mathbb{R}^d$ and $\mathbf{g} : \Omega \mapsto \mathbb{R}^{d'}$ are arbitrary (non-constant) vector-valued functions over the sample space. The solution distribution $p(y) = h(y) \exp\left(\boldsymbol{\theta}^\top \mathbf{f}(y) + \boldsymbol{\vartheta}^\top \mathbf{g}(y)\right)/Z(\boldsymbol{\theta}, \boldsymbol{\vartheta})$ is recovered by the dual optimization

$$\boldsymbol{\theta}, \boldsymbol{\vartheta} = \operatorname*{arg\,max}_{\boldsymbol{\vartheta} \geq \mathbf{0}, \boldsymbol{\theta}} - \log \sum_y h(y) \exp\left(\boldsymbol{\theta}^\top \mathbf{f}(y) + \boldsymbol{\vartheta}^\top \mathbf{g}(y)\right)$$

where $\boldsymbol{\theta} \in \mathbb{R}^d$ and $\boldsymbol{\vartheta} \in \mathbb{R}^{d'}$. These are obtained by minimizing $Z(\boldsymbol{\theta}, \boldsymbol{\vartheta})$ or equivalently by maximizing its negative logarithm. Algorithm 1 permits variational maximization of the dual via the quadratic program

$$\min_{\boldsymbol{\vartheta} \geq \mathbf{0}, \boldsymbol{\theta}} \tfrac{1}{2}(\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}})^\top \boldsymbol{\Sigma}(\boldsymbol{\beta} - \tilde{\boldsymbol{\beta}}) + \boldsymbol{\beta}^\top \boldsymbol{\mu}$$

where $\boldsymbol{\beta}^\top = [\boldsymbol{\theta}^\top \boldsymbol{\vartheta}^\top]$.

2.2.2 Conditional Random Fields and Log-Linear Models

The partition function arises naturally in maximum entropy estimation or minimum relative entropy estimation as well as in conditional extensions of the maximum entropy paradigm, like conditional random fields (CRFs), where the model is

conditioned on the observations. CRFs are often used in structured prediction problems [3, 91]. Let $\{(x_1, y_1), \ldots, (x_t, y_t)\}$ be the dataset of independent identically-distributed (*iid*) input-output pairs where y_j is the observed sample in a (discrete) space Ω_j conditioned on the observed input x_j. A CRF defines a distribution over all $y \in \Omega_j$ (of which y_j is a single element) as the log-linear model

$$p(y|x_j, \boldsymbol{\theta}) \;=\; \frac{1}{Z_{x_j}(\boldsymbol{\theta})} h_{x_j}(y) \exp(\boldsymbol{\theta}^\top \mathbf{f}_{x_j}(y))$$

where $Z_{x_j}(\boldsymbol{\theta}) \;=\; \sum_{y \in \Omega_j} h_{x_j}(y) \exp(\boldsymbol{\theta}^\top \mathbf{f}_{x_j}(y))$. For the j'th training pair, we are given a non-negative function $h_{x_j}(y) \in \mathbb{R}^+$ and a vector-valued function $\mathbf{f}_{x_j}(y) \in \mathbb{R}^d$ defined over the domain $y \in \Omega_j$ (for simplicity, assume $n = \max_{j=1}^t |\Omega_{y_j}|$). Each partition function $Z_{x_j}(\boldsymbol{\theta})$ is a function of $\boldsymbol{\theta}$. The parameter $\boldsymbol{\theta}$ for CRFs is estimated by maximizing the regularized conditional log-likelihood[3] or log-posterior:

$$J(\boldsymbol{\theta}) = \sum_{j=1}^{t} \log p(y_j|x_j, \boldsymbol{\theta}) - \tfrac{t\lambda}{2}\|\boldsymbol{\theta}\|^2 = \sum_{j=1}^{t} \log \frac{h_{x_j}(y_j)}{Z_{x_j}(\boldsymbol{\theta})} + \boldsymbol{\theta}^\top \mathbf{f}_{x_j}(y_j) - \tfrac{t\lambda}{2}\|\boldsymbol{\theta}\|^2, \quad (2.1)$$

where $\lambda \in \mathbb{R}^+$ is a regularizer. Algorithm 2 proposes a method for maximizing the regularized conditional likelihood $J(\boldsymbol{\theta})$ or, equivalently minimizing the partition function $Z(\boldsymbol{\theta})$. It solves the problem in Equation 2.1 subject to convex constraints

[3] Alternatively, variational Bayesian approaches can be used instead of maximum likelihood via expectation propagation (EP) or power EP [92]. These, however, assume Gaussian posterior distributions over parameters, require approximations, are computationally expensive and are not necessarily more efficient than BFGS.

by interleaving the quadratic bound with a quadratic programming procedure. Theorem 2 establishes the convergence of the algorithm.

Algorithm 2 ConstrainedMaximization

0: Input x_j, y_j and functions $h_{x_j}, \mathbf{f}_{x_j}$ for $j=1,\ldots,t$, regularizer $\lambda \in \mathbb{R}^+$ and convex hull $\mathbf{\Lambda} \subseteq \mathbb{R}^d$

1: Initialize $\boldsymbol{\theta}_0$ anywhere inside $\mathbf{\Lambda}$ and set $\tilde{\boldsymbol{\theta}} = \boldsymbol{\theta}_0$
While not converged

2: For $j = 1,\ldots,t$
Get $\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j$ from $h_{x_j}, \mathbf{f}_{x_j}, \tilde{\boldsymbol{\theta}}$ via Algorithm 1

3: Set $\tilde{\boldsymbol{\theta}} = \arg\min_{\boldsymbol{\theta}\in\mathbf{\Lambda}} \sum_j \frac{1}{2}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}})^\top(\boldsymbol{\Sigma}_j+\lambda\mathbf{I})(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}}) + \sum_j \boldsymbol{\theta}^\top(\boldsymbol{\mu}_j - \mathbf{f}_{x_j}(y_j) + \lambda\tilde{\boldsymbol{\theta}})$

4: Output $\hat{\boldsymbol{\theta}} = \tilde{\boldsymbol{\theta}}$

Theorem 2. *For any $\boldsymbol{\theta}_0 \in \mathbf{\Lambda}$, all $\|\mathbf{f}_{x_j}(y)\| \leq r$ and all $|\Omega_j| \leq n$, Algorithm 2 outputs a $\hat{\boldsymbol{\theta}}$ such that $J(\hat{\boldsymbol{\theta}}) - J(\boldsymbol{\theta}_0) \geq (1-\epsilon)\max_{\boldsymbol{\theta}\in\mathbf{\Lambda}}(J(\boldsymbol{\theta}) - J(\boldsymbol{\theta}_0))$ in no more than $\left\lceil \log\left(\frac{1}{\epsilon}\right) / \log\left(1 + \frac{\lambda}{2r^2}\left(\sum_{i=1}^{n-1} \frac{\tanh(\log(i)/2)}{\log(i)}\right)^{-1}\right)\right\rceil$ iterations.*

Proof. First, we will upper-bound (in the Loewner ordering sense) the matrices $\boldsymbol{\Sigma}_j$ in Algorithm 2. Since $\|\mathbf{f}_{x_j}(y)\|^2 \leq r$ for all $y \in \Omega_j$ and since $\boldsymbol{\mu}_j$ in Algorithm 1 is a convex combination of $\mathbf{f}_{x_j}(y)$, the outer-product terms in the update for $\boldsymbol{\Sigma}_j$ satisfy

$$(\mathbf{f}_{x_j}(y) - \boldsymbol{\mu})(\mathbf{f}_{x_j}(y) - \boldsymbol{\mu})^\top \preceq 4r^2\mathbf{I}.$$

Thus

$$\boldsymbol{\Sigma}_j \preceq 4r^2 \sum_{i=2}^{n} \frac{\tanh(\frac{1}{2}\log(\frac{\alpha_i}{\sum_{k=1}^{i-1}\alpha_k}))}{2\log(\frac{\alpha_i}{\sum_{k=1}^{i-1}\alpha_k})}\mathbf{I}$$

using the definition of $\alpha_1, \ldots, \alpha_n$ in the proof of Theorem 1. Furthermore one can show that

$$\boldsymbol{\Sigma}_j \preceq \left(2r^2 \sum_{i=1}^{n-1} \frac{\tanh(\log(i)/2)}{\log(i)}\right) \mathbf{I} = \omega \mathbf{I}.$$

After apply this bound to each $\boldsymbol{\Sigma}_j$, we obtain the lower-bound on $J(\boldsymbol{\theta})$. There also exists an upper-bound coming from Jensen's inequality. Both are provided below.

$$J(\boldsymbol{\theta}) \geq J(\tilde{\boldsymbol{\theta}}) - \tfrac{t\omega + t\lambda}{2} \|\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}\|^2 - \sum_j (\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^\top (\boldsymbol{\mu}_j - \mathbf{f}_{x_j}(y_j))$$
$$J(\boldsymbol{\theta}) \leq J(\tilde{\boldsymbol{\theta}}) - \tfrac{t\lambda}{2} \|\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}\|^2 - \sum_j (\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^\top (\boldsymbol{\mu}_j - \mathbf{f}_{x_j}(y_j))$$

which follows from Jensen's inequality. Define the current $\tilde{\boldsymbol{\theta}}$ at time τ as $\boldsymbol{\theta}_\tau$ and denote by $L_\tau(\boldsymbol{\theta})$ the above lower-bound and by $U_\tau(\boldsymbol{\theta})$ the above upper-bound at time τ. Clearly, $L_\tau(\boldsymbol{\theta}) \leq J(\boldsymbol{\theta}) \leq U_\tau(\boldsymbol{\theta})$ with equality when $\boldsymbol{\theta} = \boldsymbol{\theta}_\tau$. Algorithm 2 maximizes $J(\boldsymbol{\theta})$ after initializing at $\boldsymbol{\theta}_0$ and performing an update by maximizing a lower-bound based on $\boldsymbol{\Sigma}_j$. Since $L_\tau(\boldsymbol{\theta})$ replaces the definition of $\boldsymbol{\Sigma}_j$ with $\omega \mathbf{I} \succeq \boldsymbol{\Sigma}_j$, $L_\tau(\boldsymbol{\theta})$ is a looser bound than the one used by Algorithm 2. Thus, performing $\boldsymbol{\theta}_{\tau+1} = \arg\max_{\boldsymbol{\theta} \in \boldsymbol{\Lambda}} L_\tau(\boldsymbol{\theta})$ makes less progress than a step of Algorithm 1. Consider computing the slower update at each iteration τ and returning $\boldsymbol{\theta}_{\tau+1} = \arg\max_{\boldsymbol{\theta} \in \boldsymbol{\Lambda}} L_\tau(\boldsymbol{\theta})$. Setting $\boldsymbol{\Phi} = (t\omega + t\lambda)\mathbf{I}$, $\boldsymbol{\Psi} = t\lambda \mathbf{I}$ and $\kappa = \frac{\omega + \lambda}{\lambda}$ allows us

to apply Lemma 13 (see Appendix A) as follows

$$\sup_{\boldsymbol{\theta}\in\boldsymbol{\Lambda}} L_\tau(\boldsymbol{\theta}) - L_\tau(\boldsymbol{\theta}_\tau) \;\geq\; \frac{1}{\kappa}\sup_{\boldsymbol{\theta}\in\boldsymbol{\Lambda}} U_\tau(\boldsymbol{\theta}) - U_\tau(\boldsymbol{\theta}_\tau).$$

Since $L_\tau(\boldsymbol{\theta}_\tau) = J(\boldsymbol{\theta}_\tau) = U_\tau(\boldsymbol{\theta}_\tau)$, $J(\boldsymbol{\theta}_{\tau+1}) \geq \sup_{\boldsymbol{\theta}\in\boldsymbol{\Lambda}} L_\tau(\boldsymbol{\theta})$ and $\sup_{\boldsymbol{\theta}\in\boldsymbol{\Lambda}} U_\tau(\boldsymbol{\theta}) \geq J(\boldsymbol{\theta}^*)$, we obtain

$$J(\boldsymbol{\theta}_{\tau+1}) - J(\boldsymbol{\theta}^*) \;\geq\; \left(1 - \frac{1}{\kappa}\right)\left(J(\boldsymbol{\theta}_\tau) - J(\boldsymbol{\theta}^*)\right).$$

Iterate the above inequality starting at $t = 0$ to obtain

$$J(\boldsymbol{\theta}_\tau) - J(\boldsymbol{\theta}^*) \;\geq\; \left(1 - \frac{1}{\kappa}\right)^{\tau}\left(J(\boldsymbol{\theta}_0) - J(\boldsymbol{\theta}^*)\right).$$

A solution within a multiplicative factor of ϵ implies that $\epsilon = \left(1 - \frac{1}{\kappa}\right)^{\tau}$ or $\log(1/\epsilon) = \tau \log \frac{\kappa}{\kappa-1}$. Inserting the definition for κ shows that the number of iterations τ is at most $\left\lceil \frac{\log(1/\epsilon)}{\log \frac{\kappa}{\kappa-1}} \right\rceil$ or $\left\lceil \frac{\log(1/\epsilon)}{\log(1+\lambda/\omega)} \right\rceil$. Inserting the definition for ω gives the bound. □

The series $\sum_{i=1}^{n-1} \frac{\tanh(\log(i)/2)}{\log(i)} = \sum_{i=1}^{n-1} \frac{i-1}{(i+1)\log(i)}$ is the logarithmic integral which is $\mathrm{O}\left(\frac{n}{\log n}\right)$ asymptotically [**93**]. Note that the convergence rate we obtained for Algorithm 2 is linear. Also, as an additional remark, we would like to note that one may consider more aggressive update rules than the one in Algorithm 2. Instead of computing an update to $\tilde{\boldsymbol{\theta}}$ via step 3 of Algorithm 2, one can update it by

scaling $(\boldsymbol{\Sigma}_j + \lambda \boldsymbol{I})$ by $1/\zeta$ in step 3, where $\zeta \geq 1$. Setting ζ to any value from the interval $(0, 2)$ still ensures monotonic improvement [94]. However, in practice in this chapter we always use ζ set to $\zeta = 1$. The next sections show how to handle hidden variables in the learning problem, exploit graphical modeling, and further accelerate the underlying algorithms.

2.2.3 Latent Conditional Likelihood

Section 2.2.2 showed how the partition function is useful for maximum conditional likelihood problems involving CRFs. In this section, maximum conditional likelihood is extended to the setting where some variables are latent. Latent models may provide more flexibility than fully observable models [5, 95, 96]. For instance, hidden conditional random fields were shown to outperform generative hidden-state and discriminative fully-observable models [5].

Let $\mathcal{D} = \{(x_1, y_1), \ldots, (x_t, y_t)\}$ be the dataset, where $x_1, \ldots, x_t$ are sampled from some unknown distribution $\bar{p}(x)$ and t corresponding samples $y_1, \ldots, y_t$ drawn from identical conditional distributions $\bar{p}(y|x_1), \ldots, \bar{p}(y|x_t)$ respectively. Assume that the true generating distributions $\bar{p}(x)$ and $\bar{p}(y|x)$ are unknown. We aim to estimate a conditional distribution $p(y|x)$ from some set of hypotheses that achieves high conditional likelihood given the dataset. We will consider the latent setting, where we select this conditional distribution by assuming it emerges from a con-

ditioned joint distribution over x and y as well as a hidden variable m which is being marginalized as $p(y|x,\Theta) = \frac{\sum_m p(x,y,m|\Theta)}{\sum_{y,m} p(x,y,m|\Theta)}$. Here $m \in \Omega_m$ represents a discrete hidden variable, $x \in \Omega_x$ is an input and $y \in \Omega_y$ is a discrete output variable. The parameter Θ contains all parameters that explore the function class of such conditional distributions. The latent likelihood of the data $L(\Theta) = p(\mathcal{D}|\Theta)$ is the objective function that needs to be maximized and is expressed as

$$L(\Theta) = \prod_{j=1}^{t} p(y_j|x_j,\Theta) = \prod_{j=1}^{t} \frac{\sum_m p(x_j,y_j,m|\Theta)}{\sum_{y,m} p(x_j,y,m|\Theta)}. \tag{2.2}$$

We will assume that each $p(x|y,m,\Theta)$ is an exponential family distribution

$$p(x|y,m,\Theta) = h(x)\exp\big(\boldsymbol{\theta}_{y,m}^\top \boldsymbol{\phi}(x) - a(\boldsymbol{\theta}_{y,m})\big)$$

where each conditional is specified by a function $h : \Omega_x \mapsto \mathbb{R}^+$ and a feature mapping $\boldsymbol{\phi} : \Omega_x \mapsto \mathbb{R}^d$ which are then used to derive the normalizer $a : \mathbb{R}^d \mapsto \mathbb{R}^+$. A parameter $\boldsymbol{\theta}_{y,m} \in \mathbb{R}^d$ selects a specific distribution. Multiply each exponential family term by an unknown marginal distribution called the *mixing proportions* $p(y,m|\pi) = \frac{\pi_{y,m}}{\sum_{y,m}\pi_{y,m}}$. This is parameterized by an unknown parameter $\pi = \{\pi_{y,m}\}\ \forall y,m$ where $\pi_{y,m} \in [0,\infty)$. Finally, the collection of all parameters is $\Theta = \{\boldsymbol{\theta}_{y,m}, \pi_{y,m}\}\ \forall y,m$. Thus, we have the complete likelihood $p(x,y,m|\Theta) = \frac{\pi_{y,m}h(x)}{\sum_{y,m}\pi_{y,m}} \exp\big(\boldsymbol{\theta}_{y,m}^\top \boldsymbol{\phi}(x) - a(\boldsymbol{\theta}_{y,m})\big)$. Insert this expression into Equation 2.2 and re-

move constant factors that appear in both denominator and numerator. Apply the change of variables $\exp(\nu_{y,m}) = \pi_{y,m}\exp(-a(\boldsymbol{\theta}_{y,m}))$ and rewrite the objective as a function[4] of a vector $\boldsymbol{\theta}$:

$$L(\Theta) \;=\; \prod_{j=1}^{t} \frac{\sum_m \exp\left(\boldsymbol{\theta}_{y_j,m}^\top \boldsymbol{\phi}(x_j) + \nu_{y_j,m}\right)}{\sum_{y,m}\exp\left(\boldsymbol{\theta}_{y,m}^\top \boldsymbol{\phi}(x_j) + \nu_{y,m}\right)} = \prod_{j=1}^{t} \frac{\sum_m \exp\left(\boldsymbol{\theta}^\top \mathbf{f}_{j,y_j,m}\right)}{\sum_{y,m}\exp\left(\boldsymbol{\theta}^\top \mathbf{f}_{j,y,m}\right)}.$$

The last equality emerges by rearranging all Θ parameters as a vector $\boldsymbol{\theta}$ equal to $[\boldsymbol{\theta}_{1,1}^\top \nu_{1,1}\, \boldsymbol{\theta}_{1,2}^\top \nu_{1,2} \;\cdots\; \boldsymbol{\theta}_{|\Omega_y|,|\Omega_m|}^\top \nu_{|\Omega_y|,|\Omega_m|}]^\top$ (note that $\boldsymbol{\theta} \in \mathbb{R}^{|\Omega_y||\Omega_m|(d+1)}$) and introducing $\mathbf{f}_{j,\hat{y},\hat{m}} \in \mathbb{R}^{|\Omega_y||\Omega_m|(d+1)}$ defined as $\left[\left[\boldsymbol{\phi}(x_j)^\top 1\right]\delta[(\hat{y},\hat{m})=(1,1)] \;\cdots\; \left[\boldsymbol{\phi}(x_j)^\top 1\right]\delta[(\hat{y},\hat{m})=(|\Omega_y|,|\Omega_m|)]\right]^\top$ (thus the feature vector $[\boldsymbol{\phi}(x_j)^\top 1]^\top$ is positioned appropriately in the longer $\mathbf{f}_{j,\hat{y},\hat{m}}$ vector which is elsewhere zero). We will now show a variational lower-bound on $L(\boldsymbol{\theta}) \geq Q(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}})$ which is tight when $\boldsymbol{\theta} = \tilde{\boldsymbol{\theta}}$ such that $L(\tilde{\boldsymbol{\theta}}) = Q(\tilde{\boldsymbol{\theta}}, \tilde{\boldsymbol{\theta}})$. The lower-bound can be obtained by bounding each numerator and each denominator in the product over $j = 1, \ldots, t$. Each numerator term can be bounded using Jensen's inequality as follows

$$\sum_m \exp\left(\boldsymbol{\theta}^\top \mathbf{f}_{j,y_j,m}\right) \geq e^{\boldsymbol{\theta}^\top \sum_m \eta_{j,m}\mathbf{f}_{j,y_j,m} - \sum_m \eta_{j,m}\log\eta_{j,m}}$$

where $\eta_{j,m} = (e^{\tilde{\boldsymbol{\theta}}^\top \mathbf{f}_{j,y_j,m}})/(\sum_{m'} e^{\tilde{\boldsymbol{\theta}}^\top \mathbf{f}_{j,y_j,m'}})$. Algorithm 1 then bounds the denomina-

[4] It is now easy to regularize $L(\boldsymbol{\theta})$ by adding $-\frac{t\lambda}{2}\|\boldsymbol{\theta}\|^2$.

tor

$$\sum_{y,m} \exp\left(\boldsymbol{\theta}^\top \mathbf{f}_{j,y,m}\right) \;\leq\; z_j e^{\frac{1}{2}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}})^\top \boldsymbol{\Sigma}_j (\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}}) + (\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}})^\top \boldsymbol{\mu}_j}.$$

The overall lower-bound on the likelihood is then

$$Q(\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}) = L(\tilde{\boldsymbol{\theta}}) e^{-\frac{1}{2}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}})^\top \tilde{\boldsymbol{\Sigma}} (\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}}) - (\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}})^\top \tilde{\boldsymbol{\mu}}}$$

where $\tilde{\boldsymbol{\Sigma}} = \sum_{j=1}^{t} \boldsymbol{\Sigma}_j$ and $\tilde{\boldsymbol{\mu}} = \sum_{j=1}^{t} (\boldsymbol{\mu}_j - \sum_m \eta_{j,m} \mathbf{f}_{j,y_j,m})$. The right hand side is simply an exponentiated quadratic function in $\boldsymbol{\theta}$ which is easy to maximize. This yields an iterative scheme similar to Algorithm 2 for monotonically maximizing latent conditional likelihood.

2.2.4 Graphical Models for Large n

The bounds in the previous sections are straightforward to compute when Ω is small. However, for graphical models, enumerating over Ω can be computationally too expensive. This section provides faster algorithms that recover the bound efficiently for graphical models of bounded tree-width. A graphical model represents the factorization of a probability density function. A factor graph, which represents a graphical model, is a bipartite graph $\mathcal{G} = (V, W, E)$ with variable vertices $V = \{1, \ldots, k\}$, factor vertices $W = \{1, \ldots, m\}$ and a set of edges E be-

tween V and W. In addition, define a set of random variables $Y = \{y_1, \ldots, y_k\}$ each associated with the elements of V and a set of non-negative scalar functions $\Psi = \{\psi_1, \ldots, \psi_m\}$ each associated with the elements of W. $p(Y)$ factorizes as $p(y_1, \ldots, y_k) = \frac{1}{Z} \prod_{c \in W} \psi_c(Y_c)$ where Z is a normalizing partition function (the dependence on parameters is suppressed here) and Y_c is a subset of the random variables that are associated with the neighbors of node c. In other words, $Y_c = \{y_i | i \in \mathsf{Ne}(c)\}$ where $\mathsf{Ne}(c)$ is the set of vertices that are neighbors of c. Inference in graphical models requires the evaluation and the optimization of Z. These computations can be NP-hard in general yet are efficient when $\mathcal{G}$ satisfies certain properties (low tree-width). Consider a log-linear model (a function class) indexed by a parameter $\boldsymbol{\theta} \in \boldsymbol{\Lambda}$ in a convex hull $\boldsymbol{\Lambda} \subseteq \mathbb{R}^d$ as follows

$$p(Y|\boldsymbol{\theta}) \;=\; \frac{1}{Z(\boldsymbol{\theta})} \prod_{c \in W} h_c(Y_c) \exp\left(\boldsymbol{\theta}^\top \mathbf{f}_c(Y_c)\right)$$

where $Z(\boldsymbol{\theta}) \;=\; \sum_Y \prod_{c \in W} h_c(Y_c) \exp\left(\boldsymbol{\theta}^\top \mathbf{f}_c(Y_c)\right)$. The model is defined by a set of vector-valued functions $\mathbf{f}_c(Y_c) \in \mathbb{R}^d$ and scalar-valued functions $h_c(Y_c) \in \mathbb{R}^+$. Algorithm 1 may be inapplicable in this setting due to the large number of configurations in Y. Instead, consider a more efficient surrogate algorithm which computes the same bound parameters by efficiently exploiting the factorization of the graphical model. Begin by assuming that the graphical model in question is a *junction tree* and satisfies the running intersection property [97]. It is known that any graphical

model can be converted into such a form at the expense of growing the cardinality of the factors $c \in W$. Starting with a general graph $\mathcal{G}$, convert it into a junction tree $\mathcal{T}$ as follows. First, triangulate it by adding as few edges as possible (this does not change $p(Y|\boldsymbol{\theta})$). Once triangulated, form a junction tree $\mathcal{T}$ by connecting overlapping cliques to maximize the sum of the cardinalities of clique intersections. Then, using the junction tree $\mathcal{T}$, compute the bound parameters via Algorithm 3.

Algorithm 3 JunctionTreeBound

Input Reverse-topological tree $\mathcal{T}$ with $c = 1, \ldots, m$ factors $h_c(Y_c)\exp(\tilde{\boldsymbol{\theta}}^\top \mathbf{f}_c(Y_c))$ and $\tilde{\boldsymbol{\theta}} \in \mathbb{R}^d$

For $c = 1, \ldots, m$

If $(c < m)$ $\{Y_{both} = Y_c \cap Y_{pa(c)},\ Y_{solo} = Y_c \setminus Y_{pa(c)}\}$

Else $\{Y_{both} = \{\},\ Y_{solo} = Y_c\}$

For each $u \in Y_{both}$

{

Initialize $z_{c|x} \leftarrow 0^+,\ \boldsymbol{\mu}_{c|x} = \mathbf{0},\ \boldsymbol{\Sigma}_{c|x} = z_{c|x}\mathbf{I}$

For each $v \in Y_{solo}$

{

$w = u \otimes v$

$\alpha_w = h_c(w)e^{\tilde{\boldsymbol{\theta}}^\top \mathbf{f}_c(w)} \prod_{b \in ch(c)} z_{b|w}$

$\mathbf{l}_w = \mathbf{f}_c(w) - \boldsymbol{\mu}_{c|u} + \sum_{b \in ch(c)} \boldsymbol{\mu}_{b|w}$

$\boldsymbol{\Sigma}_{c|u} = \sum_{b \in ch(c)} \boldsymbol{\Sigma}_{b|w} + \frac{\tanh(\frac{1}{2}\log(\frac{\alpha_w}{z_{c|u}}))}{2\log(\frac{\alpha_w}{z_{c|u}})} \mathbf{l}_w \mathbf{l}_w^\top$

$\boldsymbol{\mu}_{c|u} = \frac{\alpha_w}{z_{c|u} + \alpha_w} \mathbf{l}_w$

$z_{c|u} = \alpha_w$

}

}

Output Bound as $z = z_m,\ \boldsymbol{\mu} = \boldsymbol{\mu}_m,\ \boldsymbol{\Sigma} = \boldsymbol{\Sigma}_m$

This algorithm only requires enumerating over all configurations of each clique in the junction tree which is clearly more efficient than enumerating over all configurations of Y's. This shows that the computation involved is $\mathrm{O}(\sum_c |Y_c|)$ rather

than O($|\Omega|$) as in Algorithm 1. In Algorithm 3 (Appendix A provides a proof of its correctness), take $ch(c)$ to be the set of children-cliques of clique c and $pa(c)$ to be the parent of c. Note that the algorithm enumerates over $u \in Y_{pa(c)} \cap Y_c$ and $v \in Y_c \setminus Y_{pa(c)}$. The algorithm stores a quadratic bound for each configuration of u (where u is the set of variables in common across both clique c and its parent). It then forms the bound by summing over $v \in Y_c \setminus Y_{pa(c)}$, each configuration of each variable a clique c has that is *not* shared with its parent clique. The algorithm also collects precomputed bounds from children of c. Also define $w = u \otimes v \in Y_c$ as the conjunction of both indexing variables u and v. Thus, the two inner for loops enumerate over all configurations $w \in Y_c$ of each clique. Note that w is used to query the children $b \in ch(c)$ of a clique c to report their bound parameters $z_{b|w}, \boldsymbol{\mu}_{b|w}, \boldsymbol{\Sigma}_{b|w}$. This is done for each configuration w of the clique c. Note, however, that not every variable in clique c is present in each child b so only the variables in w that intersect Y_b are relevant in indexing the parameters $z_{b|w}, \boldsymbol{\mu}_{b|w}, \boldsymbol{\Sigma}_{b|w}$ and the remaining variables do not change the values of $z_{b|w}, \boldsymbol{\mu}_{b|w}, \boldsymbol{\Sigma}_{b|w}$.

Additionally, notice that Algorithm 3 is a simple extension of the known recursions that are used to compute the partition function and its gradient vector. Thus, in addition to the $\boldsymbol{\Sigma}$ matrix which represents the curvature of the bound, Algorithm 3 is recovering the partition function value z and the gradient since $\boldsymbol{\mu} = \frac{\partial \log Z(\boldsymbol{\theta})}{\partial \boldsymbol{\theta}}\Big|_{\boldsymbol{\theta}=\tilde{\boldsymbol{\theta}}}$.

2.3 Low-Rank Bounds for Large d

In many realistic situations, the dimensionality d is large and this prevents the storage and inversion of the matrix $\boldsymbol{\Sigma}$. We next present a low-rank extension that can be applied to any of the algorithms presented so far. As an example, we will consider a low-rank incarnation of Algorithm 2. Each iteration of Algorithm 2 requires $\mathrm{O}(tnd^2 + d^3)$ time since step 2 computes several $\boldsymbol{\Sigma}_j \in \mathbb{R}^{d\times d}$ matrices and 3 performs inversion. Instead, the new algorithm provides a low-rank version of the bound which still majorizes the log-partition function but requires only $\tilde{\mathrm{O}}(tnd)$ complexity (putting it on par with L-BFGS).

First, note that step 3 in Algorithm 2 can be written as $\tilde{\boldsymbol{\theta}} = \tilde{\boldsymbol{\theta}} - \boldsymbol{\Sigma}^{-1}\mathbf{u}$ where $\mathbf{u} = t\lambda\tilde{\boldsymbol{\theta}} + \sum_{j=1}^{t} \boldsymbol{\mu}_j - \mathbf{f}_{x_j}(y_j)$. Clearly, Algorithm 1 can recover $\mathbf{u}$ by *only* computing $\boldsymbol{\mu}_j$ for $j = 1, \ldots, t$ and skipping all steps involving matrices. This merely requires $\mathrm{O}(tnd)$ work. Second, we store $\boldsymbol{\Sigma}$ using a low-rank representation $\mathbf{V}^\top\mathbf{S}\mathbf{V} + \mathbf{D}$ where $\mathbf{V} \in \mathbb{R}^{k\times d}$ is orthonormal, $\mathbf{S} \in \mathbb{R}^{k\times k}$ is positive semi-definite, and $\mathbf{D} \in \mathbb{R}^{d\times d}$ is non-negative diagonal. Rather than increment the matrix by a rank one update of the form $\boldsymbol{\Sigma}_i = \boldsymbol{\Sigma}_{i-1} + \mathbf{r}_i\mathbf{r}_i^\top$ where $\mathbf{r}_i = \sqrt{\frac{\tanh(\frac{1}{2}\log(\alpha/z))}{2\log(\alpha/z)}}(\mathbf{f}_i - \boldsymbol{\mu}_i)$, simply project $\mathbf{r}_i$ onto each eigenvector in $\mathbf{V}$ and update matrix $\mathbf{S}$ and $\mathbf{V}$ via a singular value decomposition ($\mathrm{O}(k^3)$ work). After removing k such projections, the remaining residual from $\mathbf{r}_i$ forms a new eigenvector $\mathbf{e}_{k+1}$ and its magnitude forms a new singular value. The resulting rank $(k+1)$ system is orthonormal with $(k+1)$ sin-

Algorithm 4 LowRankBound

Input Parameter $\tilde{\boldsymbol{\theta}}$, regularizer $\lambda \in \mathbb{R}^+$, model $\mathbf{f}_t(y) \in \mathbb{R}^d$ and $h_t(y) \in \mathbb{R}^+$ and rank $k \in \mathbb{N}$

Initialize $\mathbf{S} = \mathbf{0} \in \mathbb{R}^{k \times k}, \mathbf{V} = \mathsf{orthonormal} \in \mathbb{R}^{k \times d}, \mathbf{D} = t\lambda\mathbf{I} \in \mathsf{diag}(\mathbb{R}^{d \times d})$

For each t {

Set $z \to 0^+; \boldsymbol{\mu} = \mathbf{0}$;

For each y {

$\alpha = h_t(y) e^{\tilde{\boldsymbol{\theta}}^\top \mathbf{f}_t(y)}$

$\mathbf{r} = \frac{\sqrt{\tanh(\frac{1}{2}\log(\frac{\alpha}{z}))}}{\sqrt{2\log(\frac{\alpha}{z})}}(\mathbf{f}_t(y) - \boldsymbol{\mu})$

For $i = 1, \ldots, k$: $\mathbf{p}(i) = \mathbf{r}^\top \mathbf{V}(i, \cdot); \mathbf{r} = \mathbf{r} - \mathbf{p}(i)\mathbf{V}(i, \cdot);$

For $i = 1, \ldots, k$: For $j = 1, \ldots, k$: $\mathbf{S}(i, j) = \mathbf{S}(i, j) + \mathbf{p}(i)\mathbf{p}(j);$

$\mathbf{Q}^\top \mathbf{A} \mathbf{Q} = \mathrm{svd}(\mathbf{S}); \quad \mathbf{S} \leftarrow \mathbf{A}; \quad \mathbf{V} \leftarrow \mathbf{Q}\mathbf{V};$

$\mathbf{s} = [\mathbf{S}(1,1), \ldots, \mathbf{S}(k,k), \|\mathbf{r}\|^2]^\top$

$\tilde{k} = \arg\min_{i=1,\ldots,k+1} \mathbf{s}(i)$

if $(\tilde{k} \leq k)$

{ $\mathbf{D} = \mathbf{D} + \mathbf{S}(\tilde{k}, \tilde{k}) \mathbf{1}^\top |\mathbf{V}(j, \cdot)| \; \mathsf{diag}(|\mathbf{V}(k, \cdot)|)$

$\mathbf{S}(\tilde{k}, \tilde{k}) = \|\mathbf{r}\|^2$

$\mathbf{r} = \|\mathbf{r}\|^{-1} \mathbf{r}$

$\mathbf{V}(k, \cdot) = \mathbf{r}$ }

else

{

$\mathbf{D} = \mathbf{D} + \mathbf{1}^\top |\mathbf{r}| \mathsf{diag}(|\mathbf{r}|)$ }

$\boldsymbol{\mu} \mathrel{+}= \frac{\alpha}{z+\alpha}(\mathbf{f}_t(y) - \boldsymbol{\mu})$

$z \mathrel{+}= \alpha;$ } }

Output $\mathbf{S} \in \mathsf{diag}(\mathbb{R}^{k \times k}), \mathbf{V} \in \mathbb{R}^{k \times d}, \mathbf{D} \in \mathsf{diag}(\mathbb{R}^{d \times d})$

gular values. We discard its smallest singular value and corresponding eigenvector to revert back to an order k eigensystem. However, instead of merely *discarding* we can *absorb* the smallest singular value and eigenvector into the $\mathbf{D}$ component by bounding the remaining outer-product with a diagonal term. This provides a guaranteed overall upper-bound in $\tilde{\mathrm{O}}(tnd)$ (k is assumed to be logarithmic with dimension d). Algorithm 4 provides a low-rank incarnation of Algorithm 2.

Finally, to invert $\mathbf{\Sigma}$, we apply the Woodbury formula: $\mathbf{\Sigma}^{-1} = \mathbf{D}^{-1} + \mathbf{D}^{-1}\mathbf{V}^\top(\mathbf{S}^{-1} + \mathbf{V}\mathbf{D}^{-1}\mathbf{V}^\top)^{-1}\mathbf{V}\mathbf{D}^{-1}$ which only requires $\mathrm{O}(k^3)$ work. A proof of correctness of Algorithm 4 can be found in the Appendix A.

2.4 Batch bound method versus generic optimization methods: empirical evaluation

In this section we provide the empirical evaluation of the performance of the bound-majorization method based on the new quadratic bound on the log-partition function, and compare it with the performance of first- and second-order generic optimization methods. We consider three learning tasks: ℓ_2-regularized logistic regression, training CRFs with linear Markov chain structure and Maximum latent conditional likelihood problems.

2.4.1 ℓ_2-regularized logistic regression

We first focus on the ℓ_2-regularized logistic regression task for small-scale experiments. Small-scale problems may be interesting in real-time learning settings, for example, when a website has to learn from a user's uploaded labeled data in a split second to perform real-time retrieval. We compared Algorithm 2 with steepest descent (SD), conjugate gradient (CG), BFGS and Newton-Raphson on five UCI datasets, downloaded from http://archive.ics.uci.edu/ml/, where missing values were

handled via mean-imputation. A range of regularization settings $\lambda \in \{10^0, 10^2, 10^4\}$ was explored. Results are averaged over 10 random initializations (all algorithms were initialized from the same start-point). Table 2.1 shows the average number of seconds each algorithm needed to achieve the same solution that BFGS converged to (all algorithms achieve the same solution due to the concavity of the problem).

data$\|\lambda$	$a\|10^0$	$a\|10^2$	$a\|10^4$	$b\|10^0$	$b\|10^2$	$b\|10^4$	$c\|10^0$	$c\|10^2$	$c\|10^4$	$d\|10^0$	$d\|10^2$	$d\|10^4$	$e\|10^0$	$e\|10^2$	$e\|10^4$
BFGS	1.90	0.89	2.45	3.14	2.00	1.60	4.09	1.03	1.90	5.62	2.88	3.28	2.63	2.01	1.49
SD	1.74	0.92	1.60	2.18	6.17	5.83	1.92	0.64	0.56	12.04	1.27	1.94	2.68	2.49	1.54
CG	0.78	0.83	0.85	0.70	0.67	0.83	0.65	0.64	0.72	1.36	1.21	1.23	0.48	0.55	0.43
Newton	0.31	0.25	0.22	0.43	0.37	0.35	0.39	0.34	0.32	0.92	0.63	0.60	0.35	0.26	0.20
Bound	**0.01**	**0.01**	**0.01**	**0.07**	**0.04**	**0.04**	**0.07**	**0.02**	**0.02**	**0.16**	**0.09**	**0.07**	**0.03**	**0.03**	**0.03**

Table 2.1: Convergence time in seconds under various regularization levels for a) Bupa b) Wine c) Heart d) Ion and e) Hepatitis datasets.

We next focus on large-scale experiments. We compare the performance of the bound (using the low-rank Algorithm 2) with first-order and second order methods such as L-BFGS, conjugate gradient (CG) and steepest descent (SD). We use 4 benchmark datasets: the *SRBCT* and *Tumors* datasets from [**98**] as well as the *Text* and *SecStr* datasets from http://olivier.chapelle.cc/ssl-book/benchmarks.html. For all experiments here and till the end of this section, the setup is as follows. Each dataset is split into training (90%) and testing (10%) parts. The termination criterion for all algorithms is a change in estimated parameter or function values smaller than 10^{-6} (with a ceiling on the number of passes through the data of 10^6). Results are averaged over 10 random initializations close to $\mathbf{0}$. The regularization parameter λ, when used, was chosen through crossvalidation (and set to 10^1 for all

the experiments except the experiment with *Text* dataset where it was set to 10^2). In Table 2.2 we report times in seconds and the number of effective passes through the data[5] for each algorithm to achieve the L-BFGS termination solution modulo a small constant ϵ (set to 10^{-4}; in practice all algorithms achieve the same quality solution due to the concavity of the problem).

Dataset	SRBCT		Tumors		Text		SecStr	
Algorithm	time	pass	time	pass	time	pass	time	pass
L-BFGS	6.10	42	3246.83	8	15.54	7	881.31	47
SD	7.27	43	18749.15	53	153.10	69	1490.51	79
CG	40.61	100	14840.66	42	57.30	23	667.67	36
Bound	**3.67**	**8**	**1639.93**	**4**	**6.18**	**3**	**27.97**	**9**

Table 2.2: Time in seconds and number of effective passes through the data required to obtain within ϵ of the L-BFGS solution (where $\epsilon = 10^{-4}$) for logistic regression problems on SRBCT, Tumors, Text and SecStr datasets.

On both small- and large-scale experiments experiments, the bound remained the fastest as indicated in bold.

2.4.2 Markov CRFs

Structured prediction problems are explored using two popular datasets. The first one contains Spanish news wire articles from the session of the *CoNLL* 2002 conference. This corpus involves a named entity recognition problem and consists of sentences where each word is annotated with one of $m = 9$ possible labels. The second task is from the *PennTree* Bank. This corpus involves a tagging problem

[5]By effective passes we mean the number of outer loop passes of the methods plus the number of line search passes of those methods that perform line search.

Dataset	CoNLL		PennTree	
Algorithm	time	pass	time	pass
L-BFGS	25661.54	17	62848.08	7
CG	88973.93	23	76332.39	18
Bound	**16445.93**	**4**	**27073.42**	**2**

Table 2.3: Time in seconds and number of effective passes through the data required to obtain within ϵ of the L-BFGS solution (where $\epsilon = 10^{-4}$) for Markov CRF problems on CoNLL and PennTree datasets. λ was set to $\lambda = 10^1$.

and consists of sentences where each word is labeled with one of $m = 45$ possible parts-of-speech. A conditional random field is estimated with a Markov chain structure to give word labels a sequential dependence. The features describing the words are constructed as in [99]. Table 2.3 provides results for this experiment (in practice all algorithms achieve the same quality solution due to the concavity of the problem). We used the low-rank version of Algorithm 3. In both experiments, the bound always remained fastest as indicated in bold.

Additionally, in Table 2.4 we provide the comparison of the results we obtained on *CoNLL* dataset with the results of a similar experiment on the same dataset obtained from [2], where generic methods where shown to significantly outperform majorization methods, such as iterative scaling.

our results ([1])			results from [2]		
Algorithm	time	pass	Algorithm	time	pass
L-BFGS	t	17	L-BFGS	$t^{'}$	22
CG	$3.47t$	23	CG	$5.94t^{'}$	27
Bound	$\mathbf{0.64}t$	**4**	IIS	$\geq 6.35t^{'}$	≥ 150

Table 2.4: Comparison of the results obtained by us ([1]) with the results from [2] for Markov CRF problem on CoNLL dataset.

2.4.3 Maximum latent conditional likelihood problems

We next performed experiments with maximum latent conditional likelihood problems. We denote by m the number of hidden variables. Due to the non-concavity of this objective, we are most interested in finding good local maxima. We compared the algorithms (the bound, Newton-Raphson, BFGS, CG and SD) on five benchmark datasets. The datasets included four UCI datasets (*ion*, *bupa*, *hepatitis* and *wine*) and the previously used *SRBCT* dataset. The feature mapping used was $\boldsymbol{\phi}(x) = \mathbf{x} \in \mathbb{R}^d$. We used a value of $\lambda = 0$ throughout the latent experiments. We explored setting $m \in \{1, 2, 3, 4\}$. Table 2.5 shows the testing latent log-likelihood at convergence. In bold, we show the algorithm that obtained the highest log-likelihood. Furthermore, Figure 2.1 depicts the convergence of testing latent log-likelihood. The bound is the best performer overall and finds better

Dataset	ion				bupa				hepatitis			
Algorithm	m = 1	m = 2	**m = 3**	m = 4	m = 1	**m = 2**	m = 3	m = 4	m = 1	**m = 2**	m = 3	m = 4
BFGS	-4.96	-5.55	-5.88	-5.79	-22.07	-21.78	-21.92	-21.87	-4.42	-5.28	-4.95	-4.93
SD	-11.80	-9.92	-5.56	-8.59	-21.76	-21.74	-21.73	-21.83	-4.93	-5.14	-5.01	-5.20
CG	-5.47	-5.81	-5.57	-5.22	-21.81	-21.81	-21.81	-21.81	-4.84	-4.84	-4.84	-4.84
Newton	-5.95	-5.95	-5.95	-5.95	-21.85	-21.85	-21.85	-21.85	-5.50	-5.50	-5.50	-4.50
Bound	-6.08	-4.84	**-4.18**	-5.17	-21.85	**-19.95**	-20.01	-19.97	-5.47	**-4.40**	-4.75	-4.92

Dataset	wine				SRBCT			
Algorithm	m = 1	m = 2	**m = 3**	m = 4	m = 1	m = 2	m = 3	**m = 4**
BFGS	-0.90	-0.91	-1.79	-1.35	-5.99	-6.17	-6.09	-6.06
SD	-1.61	-1.60	-1.37	-1.63	-5.61	-5.62	-5.62	-5.61
CG	-0.51	-0.78	-0.95	-0.51	-5.62	-5.49	-5.36	-5.76
Newton	-0.71	-0.71	-0.71	-0.71	-5.54	-5.54	-5.54	-5.54
Bound	-0.51	-0.51	**-0.48**	-0.51	-5.31	-5.31	-4.90	**-0.11**

Table 2.5: Test latent log-likelihood at convergence for different values of $m \in \{1, 2, 3, 4\}$ on ion, bupa, hepatitis, wine and SRBCT data-sets.

solutions in less time.

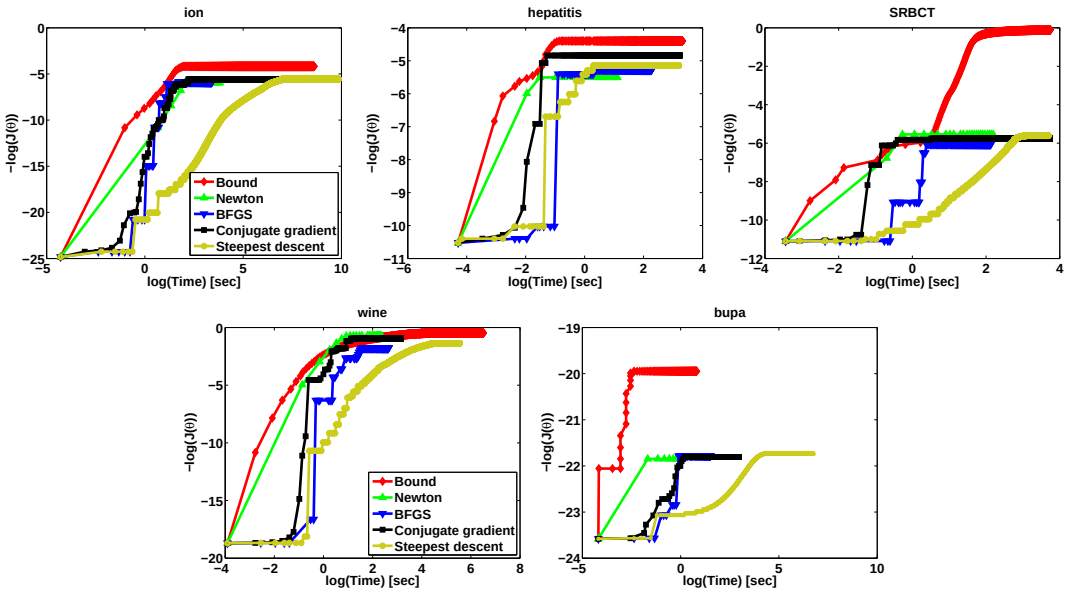


Figure 2.1: Convergence of test latent log-likelihood on ion, hepatitis, SRBCT, wine and bupa datasets.

2.5 Conclusion

We showed a simple quadratic upper-bound for the partition function of log-linear models. The bound is efficiently recoverable for graphical models and admits low-rank variants for high-dimensional data. It allows fast and monotonically convergent majorization in CRF learning and maximum latent conditional likelihood problems (where it also finds better local maxima). The bound majorization method based on a new bound makes majorization approaches competitive with state-of-the-art first- and second-order optimization methods. So far we focused on the batch setting where the algorithm makes a full pass through the entire dataset be-

fore updating the parameter vector. As datasets grow in size, this learning strategy becomes increasingly inefficient. In the next chapter we will focus on applying the bound in the semi-stochastic settings.

3

Semi-stochastic partition function bound majorization

This chapter is based on joint work with Aleksandr Aravkin, Tony Jebara and Dimitri Kanevsky that originally appeared in [100]. *All codes are released and are publicly available at* www.columbia.edu/∼aec2163/NonFlash/Papers/Papers.html.

Standard learning systems based on batch methods, including the partition function bound majorization method described in the previous chapter, need to make a full pass through an entire dataset before updating the parameter vector. For massive datasets, this update strategy becomes expensive and computing it hurt system's ef-

ficiency. In this setting, using semi-stochastic or fully stochastic update rule, where the parameter is updated after seeing mini-batch of data points (stochastic methods can potentially use as small mini-batch as a single data-point; semi-stochastic methods typically start from small mini-batch, like stochastic methods, but then gradually increase it), becomes computationally much cheaper and furthermore was empirically shown to lead to convergence in significantly less iterations than in a batch case. In particular, semi-stochastic methods exhibit an important advantage of exploring simultaneously the benefits of both settings, batch and stochastic, by approaching the solution more quickly when close to the optimum (like a full batch method and unlike stochastic methods, which typically provide fast improvement initially, but are slow to converge near the optimum) while simultaneously reducing the computational complexity per iteration (though less aggressively than stochastic methods).

The objective of this chapter is to propose a semi-stochastic variant of the partition function bound majorization method discussed in the previous chapter. In this chapter we furthermore show that the fast convergence property of the batch method is also preserved in the semi-stochastic setting. This chapter is organized as follows. Section 3.1 discusses stochastic and semi-stochastic extensions of the bound. Section 3.2 presents convergence theory for the proposed methods. In particular, we discuss very general stationary convergence theory under very weak

assumptions, and also present a much stronger theory, including convergence rate analysis, for logistic regression. Section 3.3 discusses implementation details and shows numerical experiments illustrating the use of the proposed methods for l_2-regularized logistic regression problems. Conclusions end the chapter.

3.1 Quadratic bound methods

In this chapter, we will consider the log-linear density model of the following form

$$p(y|x_j, \boldsymbol{\theta}) = \frac{1}{Z_{x_j}(\boldsymbol{\theta})} h_{x_j}(y) \exp\left(\boldsymbol{\theta}^\top \mathbf{f}_{x_j}(y)\right),$$

where $Z_{x_j}(\boldsymbol{\theta})$ is the *partition function* (the notation remains the same as in the previous chapter). In Chapter 2 we proposed a fast method to find a tight quadratic bound for $Z_{x_j}(\boldsymbol{\theta})$, shown in the subroutine Bound Computation in Algorithm 5, which finds $z, \mathbf{r}, \mathbf{S}$ so that

$$Z_{x_j}(\boldsymbol{\theta}) \leq z \exp(\tfrac{1}{2}(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^\top \mathbf{S}(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}) + (\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^\top \mathbf{r}) \tag{3.1}$$

for any $\boldsymbol{\theta}, \tilde{\boldsymbol{\theta}}, \mathbf{f}_{x_j}(y) \in \mathbb{R}^d$ and $h_{x_j}(y) \in \mathbb{R}^+$ for all $y \in \Omega$.

Algorithm 5 Semi-stochastic Quadratic Bound (SQB)

Input Parameters $\tilde{\boldsymbol{\theta}}, \mathbf{f}_{x_j}(y) \in \mathbb{R}^d$ and $h_{x_j}(y) \in \mathbb{R}^+$ for $y \in \Omega$, $j \in \mathcal{T}$

Initialize $\boldsymbol{\mu}_{\mathcal{T}} = \mathbf{0}, \boldsymbol{\Sigma}_{\mathcal{T}} = \mathbf{0}(d, d)$

For each $j \in \mathcal{T}$

  Subroutine **Bound Computation:**

  $z \to 0^+, \mathbf{r} = \mathbf{0}, \mathbf{S} = z\mathbf{I}$

  For each $y \in \Omega$

    $\alpha = h_{x_j}(y) \exp(\tilde{\boldsymbol{\theta}}^\top \mathbf{f}_{x_j}(y))$

    $\mathbf{S} += \frac{\tanh(\frac{1}{2}\log(\alpha/z))}{2\log(\alpha/z)} (\mathbf{f}_{x_j}(y) - \mathbf{r})(\mathbf{f}_{x_j}(y) - \mathbf{r})^\top$

    $\mathbf{r} = \frac{z}{z+\alpha}\mathbf{r} + \frac{\alpha}{z+\alpha}\mathbf{f}_{x_j}(y)$

    $z += \alpha$

  Subroutine output $z, \mathbf{r}, \mathbf{S}$

  $\boldsymbol{\mu}_{\mathcal{T}} += \mathbf{r} - \mathbf{f}_{x_j}(y)$

  $\boldsymbol{\Sigma}_{\mathcal{T}} += \mathbf{S}$

$\boldsymbol{\mu}_{\mathcal{T}} /= |\mathcal{T}|$

$\boldsymbol{\Sigma}_{\mathcal{T}} /= |\mathcal{T}|$

Output $\boldsymbol{\mu}_{\mathcal{T}}, \boldsymbol{\Sigma}_{\mathcal{T}}$

The (regularized) maximum likelihood inference problem is equivalent to

$$\begin{aligned} \min_{\boldsymbol{\theta}} \Big\{ \mathcal{L}_\eta(\boldsymbol{\theta}) &:= -\frac{1}{T}\sum_{j=1}^{T} \log(p(y_j|x_j, \boldsymbol{\theta})) + \frac{\eta}{2}\|\boldsymbol{\theta}\|^2 \\ &\approx \frac{1}{T}\sum_{j=1}^{T} \left(\log(Z_{x_j}(\boldsymbol{\theta})) - \boldsymbol{\theta}^\top \mathbf{f}_{x_j}(y_j)\right) + \frac{\eta}{2}\|\boldsymbol{\theta}\|^2 \Big\}, \end{aligned} \tag{3.2}$$

where $\approx$ means up to an additive constant. The bound (3.1) suggests the iterative minimization scheme

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \alpha_k (\boldsymbol{\Sigma}_{\mathcal{T}}^k + \eta \boldsymbol{I})^{-1} (\boldsymbol{\mu}_{\mathcal{T}}^k + \eta \boldsymbol{\theta}^k), \tag{3.3}$$

where k is the iteration index, $\boldsymbol{\Sigma}_{\mathcal{T}}^{k}$ and $\boldsymbol{\mu}_{\mathcal{T}}^{k}$ are computed using Algorithm 5, η is the regularization term and α_k is the step size at iteration k. $\mathcal{T}$ is the mini-batch of data points and thus in the batch setting it is simply the the entire dataset denoted as T. When T is large, this strategy can be expensive. In fact, computing the bound has complexity $O(Tnd^2)$, since Tn outer products must be summed to obtain $\boldsymbol{\Sigma}$, and each other product has complexity $O(d^2)$. When the dimension d is large, considerable speedups can be gained by using low-rank version of $\boldsymbol{\Sigma}$ as was discussed in Chapter 2, or obtaining a factored form of $\boldsymbol{\Sigma}$, as described in Section 3.3.1 of this chapter. Nonetheless, in either strategy, the size of T is a serious issue.

A natural approach is to subsample a smaller selection $\mathcal{T}$ from dataset Ω, so that at each iteration, we run Algorithm 5 over $\mathcal{T}$ rather than over the full data to get $\boldsymbol{\mu}_{\mathcal{T}}, \boldsymbol{\Sigma}_{\mathcal{T}}$. When $|\mathcal{T}|$ is fixed (and smaller than T), we refer to the resulting method as a *stochastic* extension. If instead $|\mathcal{T}|$ is allowed to grow as iterations proceed, we call this method *semi-stochastic*; these methods are analyzed in [**51**]. One can also decouple the computation of gradient and curvature approximations, using different data selections (which we call $\mathcal{T}$ and $\mathcal{S}$). We will show in the next sections that this development is theoretically justifiable and practically very useful.

The appeal of the stochastic approach is that when $|\mathcal{T}|, |\mathcal{S}| << T$, the complexity of Algorithm 5 to compute $\boldsymbol{\mu}_{\mathcal{T}}, \boldsymbol{\Sigma}_{\mathcal{S}}$ is much lower than in the batch framework;

and then we can still implement a (modified) iteration (3.3). Also semi-stochastic approach results in computational savings as it interpolates between stochastic and batch settings. For the stochastic and semi-stochastic methods discussed in this section, the quadratic bound property (3.1) does not hold for $Z(\boldsymbol{\theta})$, so the convergence analysis from the previous chapter does not immediately apply. Nonetheless, it is possible to analyze the algorithm in terms of sampling strategies for $\mathcal{T}$. We will now present the theoretical analysis of Algorithm 5. Till the end of this chapter we will focus on the semi-stochastic approach alone, however the first theoretical result that we show provide the guarantee of global convergence to a stationary point that also holds in the fully stochastic setting.

3.2 Theoretical analysis of the semi-stochastic quadratic bound majorization method

We first prove that under very weak assumption, in particular using only the Lipschitz property, but not requiring convexity of the problem, the proposed algorithm converges to a stationary point. The proof technique easily carries over to other objectives, such as the ones discussed in the previous chapter (e.g. the objective used in maximum latent conditional likelihood problems), since it relies mainly only on the sampling method used to obtain $\mathcal{T}$. Then we focus on problem (3.2), which is convex, and strictly convex under appropriate assumptions on the data. We use

the structure of (3.2) to prove much stronger results, and in particular analyze the rate of convergence of Algorithm 5.

3.2.1 General Convergence Theory

We present a general global convergence theory that relies on the Lipschitz property of the objective and on the sampling strategy in the context of Algorithm 5. The end result we show here is that any limit point of the iterates is *stationary*. We begin with two simple preliminary results.

Lemma 1. *If every $i \in [1, \ldots, T]$ is equally likely to appear in $\mathcal{T}$, then $E[\boldsymbol{\mu}_{\mathcal{T}}] = \boldsymbol{\mu}$.*

Proof. Algorithm 5 returns $\boldsymbol{\mu}_{\mathcal{T}} = \frac{1}{|\mathcal{T}|} \sum_{j \in \mathcal{T}} \psi_j(\boldsymbol{\theta})$, where $\psi_j(\boldsymbol{\theta}) = -\nabla_{\boldsymbol{\theta}} \log(p(y_j|x_j, \boldsymbol{\theta}))$. If each j has an equal chance to appear in $\mathcal{T}$, then

$$E\left[\frac{1}{|\mathcal{T}|}\sum_{j\in\mathcal{T}}\psi_j(\boldsymbol{\theta})\right] = \frac{1}{|\mathcal{T}|}\sum_{j\in\mathcal{T}} E[\psi_j(\boldsymbol{\theta})] = \frac{1}{|\mathcal{T}|}\sum_{j\in\mathcal{T}}\boldsymbol{\mu} = \boldsymbol{\mu}\,.$$

□

Note that the hypothesis here is very weak: there is no stipulation that the batch size be of a certain size, grow with iterations, etc. This lemma therefore applies to a wide class of randomized bound methods.

Lemma 2. *Denote by $\lambda_{\min}$ the infimum over all possible eigenvalues of $\boldsymbol{\Sigma}_{\mathcal{S}}$ over*

all choices of batches ($\lambda_{\min}$ may be 0). Then $E[(\boldsymbol{\Sigma}_{\mathcal{S}} + \eta \boldsymbol{I})^{-1}]$ satisfies

$$\frac{1}{\eta + \lambda_{\max}} \boldsymbol{I} \leq E[(\boldsymbol{\Sigma}_{\mathcal{S}} + \eta \boldsymbol{I})^{-1}] \leq \frac{1}{\eta + \lambda_{\min}} \boldsymbol{I} \ .$$

Proof. For any vector $\boldsymbol{x}$ and any realization of $\boldsymbol{\Sigma}_{\mathcal{S}}$, we have

$$\frac{1}{\eta + \lambda_{\max}} \|\boldsymbol{x}\|^2 \leq \boldsymbol{x}^T (\boldsymbol{\Sigma}_{\mathcal{S}} + \eta \boldsymbol{I})^{-1} \boldsymbol{x} \leq \frac{1}{\eta + \lambda_{\min}} \|\boldsymbol{x}\|^2 \ ,$$

where $\lambda_{\max}$ depends on the data. Taking the expectation over $\mathcal{T}$ of be above inequality gives the result. □

Theorem 3. *For any problem of form (3.2), apply iteration (3.3), where at each iteration $\boldsymbol{\mu}_{\mathcal{T}}, \boldsymbol{\Sigma}_{\mathcal{S}}$ are obtained by Algorithm 5 for two independently drawn batches subsets $\mathcal{T}, \mathcal{S} \subset [1, \ldots, T]$ selected to satisfy the assumptions of Lemma 1. Finally, suppose also that the step sizes α_k are square summable but not summable. Then $\mathcal{L}_\eta(\boldsymbol{\theta}^k)$ converges to a finite value, and $\nabla \mathcal{L}_\eta(\boldsymbol{\theta}^k) \to 0$. Furthermore, every limit point of $\boldsymbol{\theta}^k$ is a stationary point of $\mathcal{L}_\eta$.*

Theorem 3 states the conclusions of Proposition 3 from [**101**], and so to prove it we need only check that the hypotheses of this proposition are satisfied.

Proof. [101] consider algorithms of the form

$$\boldsymbol{\theta}^{k+1} = \boldsymbol{\theta}^k - \alpha_k(\boldsymbol{s}^k + \boldsymbol{w}^k) \ .$$

In the context of iteration (3.3), at each iteration we have

$$\boldsymbol{s}^k + \boldsymbol{w}^k = (\boldsymbol{\Sigma}_{\mathcal{S}}^k + \lambda \boldsymbol{I})^{-1} \boldsymbol{g}_{\mathcal{T}}^k,$$

where $\boldsymbol{g}_{\mathcal{T}}^k = \boldsymbol{\mu}_{\mathcal{T}}^k + \eta \boldsymbol{\theta}^k$, and $\boldsymbol{g}^k$ is the full gradient of the regularized problem (3.2). We choose

$$\boldsymbol{s}^k \ = E[(\boldsymbol{\Sigma}_{\mathcal{S}}^k + \eta \boldsymbol{I})^{-1}] \boldsymbol{g}^k, \quad \boldsymbol{w}^k = (\boldsymbol{\Sigma}_{\mathcal{S}}^k + \eta \boldsymbol{I})^{-1} \boldsymbol{g}_{\mathcal{T}}^k - \boldsymbol{s}^k.$$

We now have the following results:

1. Unbiased error:

$$E[\boldsymbol{w}^k] = E[(\boldsymbol{\Sigma}_{\mathcal{S}}^k + \eta \boldsymbol{I})^{-1} \boldsymbol{g}_{\mathcal{T}}^k - \boldsymbol{s}^k] = E[(\boldsymbol{\Sigma}_{\mathcal{S}}^k + \eta \boldsymbol{I})^{-1}] E[\boldsymbol{g}_{\mathcal{T}}^k] - \boldsymbol{s}^k = 0 \ , \tag{3.4}$$

where the second equality is obtained by independence of the batches $\mathcal{T}$ and $\mathcal{S}$, and the last equality uses Lemma 1.

2. Gradient related condition:

$$(\boldsymbol{g}^k)^T \boldsymbol{s}^k = (g^k)^T E[(\boldsymbol{\Sigma}_{\mathcal{S}}^k + \eta \boldsymbol{I})^{-1}] \boldsymbol{g}^k \quad \geq \frac{\|\boldsymbol{g}^k\|^2}{\eta + \lambda_{\max}}. \tag{3.5}$$

3. Bounded direction:

$$\|\boldsymbol{s}^k\| \leq \frac{\|\boldsymbol{g}^k\|}{\eta + \lambda_{\min}}. \tag{3.6}$$

4. Bounded second moment: By part 1, we have

$$\begin{aligned} E[\|\boldsymbol{w}^k\|^2] &\leq E[\|(\boldsymbol{\Sigma}_{\mathcal{S}}^k + \eta \boldsymbol{I})^{-1} \boldsymbol{g}_{\mathcal{T}}^k\|^2 \\ &\leq \frac{E[\|\boldsymbol{g}_{\mathcal{T}}^k\|^2]}{(\eta + \lambda_{\min})^2} = \frac{\operatorname{tr}(\operatorname{cov}[\boldsymbol{g}_{\mathcal{T}}^k]) + \|\boldsymbol{g}^k\|^2}{(\eta + \lambda_{\min})^2}. \end{aligned} \tag{3.7}$$

The covariance matrix of $\boldsymbol{g}_{\mathcal{T}}^k$ is proportional to the covariance matrix of the set of individual (data-point based) gradient contributions, and for problems of form (3.2) these contributions lie in the convex hull of the data, so in particular the trace of the covariance must be finite. Taken together, these results show all hypotheses of Proposition 3 from [**101**] are satisfied, and the result follows. □

Theorem 3 applies to any stochastic and semi-stochastic variant of the method. Note that two independent data samples $\mathcal{T}$ and $\mathcal{S}$ are required to prove (3.4). Computational complexity motivates different strategies for selecting choose $\mathcal{T}$ and $\mathcal{S}$. In particular, it is natural to use larger mini-batches to estimate the gradient

and smaller mini-batch sizes for the estimation of the second-order curvature term. Algorithms of this kind have been explored in the context of stochastic Hessian methods [50]. We describe our implementation details in Section 3.3.

3.2.2 Rates of Convergence for Logistic Regression

The structure of objective (3.2) allows for a much stronger convergence theory. We first present a lemma characterizing strong convexity and Lipschitz constant for (3.2). Both of these properties are crucial to the convergence theory.

Lemma 3. *The objective $\mathcal{L}_\eta$ in (3.2) has a gradient that is uniformly norm bounded, and Lipschitz continuous.*

Proof. The function $\mathcal{L}_\eta$ has a Lipschitz continuous gradient if there exists an L such that

$$\|\nabla \mathcal{L}_\eta(\boldsymbol{\theta}^1) - \nabla \mathcal{L}_\eta(\boldsymbol{\theta}^0)\| \leq L\|\boldsymbol{\theta}^1 - \boldsymbol{\theta}^0\|$$

holds for all $(\boldsymbol{\theta}^1, \boldsymbol{\theta}^0)$. Any uniform bound for $\text{trace}(\nabla^2 \mathcal{L}_\eta)$ is a Lipschitz bound for $\nabla \mathcal{L}_\eta$. Define

$$a_{y,j} := h_{x_j}(y) \exp(\boldsymbol{\theta}^\top \mathbf{f}_{x_j}(y)) \ ,$$

and note $a_{y,j} \geq 0$. Let $\boldsymbol{p}_j$ be the empirical density where the probability of observing

y is given by $\frac{a_{y,j}}{\sum_y a_{y,j}}$. The gradient of (3.2) is given by

$$\frac{1}{T}\sum_{j=1}^{T}\left(\left(\sum_y \frac{a_{y,j}\mathbf{f}_{x_j}(y)}{\sum_y a_{y,j}}\right) - \mathbf{f}_{x_j}(y_j)\right) + \eta\boldsymbol{\theta} = \frac{1}{T}\sum_{j=1}^{T}\left(E_{\boldsymbol{p}_j}[\mathbf{f}_{x_j}(\cdot)] - \mathbf{f}_{x_j}(y_j)\right) + \eta\boldsymbol{\theta}$$

It is straightforward to check that the Hessian is given by

$$\nabla^2\mathcal{L}_\eta = \frac{1}{T}\sum_{j=1}^{T} cov_{\boldsymbol{p}_j}[\mathbf{f}_{x_j}(\cdot)] + \eta\boldsymbol{I}$$

where $cov_{\boldsymbol{p}_j}[\cdot]$ denotes the covariance matrix with respect to the empirical density function $\boldsymbol{p}_j$. Therefore a global bound for the Lipschitz constant L is given by $\max_{y,j}\|\mathbf{f}_{x_j}(y)\|^2 + \eta\boldsymbol{I}$, which completes the proof. □

Corollary 1. *The function $\mathcal{L}_0$ is strongly convex exactly when $\sum_{j,y}\mathbf{f}_{x_j}(y)\mathbf{f}_{x_j}(y)^T$ is positive definite, and $\mathcal{L}_\eta$ is strongly convex for any positive η.*

We now present a convergence rate result, using results of Theorem 2.2 from [51].

Theorem 4. *Suppose that the selection strategy of $\mathcal{S}$ satisfies Lemma 2. Then there exist $\mu, L > 0$ such that*

$$\begin{aligned}\|\nabla\mathcal{L}_\eta(\boldsymbol{\theta}_1) - \nabla\mathcal{L}_\eta(\boldsymbol{\theta}_2)\|_{**} &\leq L\|\boldsymbol{\theta}_2 - \boldsymbol{\theta}_1\|_* \\ \mathcal{L}_\eta(\boldsymbol{\theta}_2) \geq \mathcal{L}_\eta(\boldsymbol{\theta}_1) + (\boldsymbol{\theta}_2 - \boldsymbol{\theta}_1)^T\nabla\mathcal{L}_\eta(\boldsymbol{\theta}_1) &+ \frac{1}{2}\rho\|\boldsymbol{\theta}_2 - \boldsymbol{\theta}_1\|_*\end{aligned} \tag{3.8}$$

where $\|\boldsymbol{\theta}\|_* = \sqrt{\boldsymbol{\theta}^T(\boldsymbol{\Sigma}_{\mathcal{S}}^k + \eta\boldsymbol{I})\boldsymbol{\theta}}$ and $\|\boldsymbol{\theta}\|_{**}$ is the corresponding dual norm

$\sqrt{\boldsymbol{\theta}^T(\boldsymbol{\Sigma}_S^k + \eta \boldsymbol{I})^{-1}\boldsymbol{\theta}}$. *Furthermore, take* $\alpha_k = \frac{1}{L}$ *in* (3.3)*, and define* $B_k = \|\nabla \mathcal{L}_\eta^k - \boldsymbol{g}_T^k\|^2$*, the square error incurred in the gradient at iteration* k*. Provided a batch growth schedule with* $\lim_{k\to\infty} \frac{B_{k+1}}{B_k} \leq 1$*, for each iteration* (3.3) *we have (for any* $\epsilon > 0$*)*

$$\mathcal{L}_\eta(\boldsymbol{\theta}^k) - \mathcal{L}_\eta(\boldsymbol{\theta}^*) \leq \left(1 - \frac{\rho}{L}\right)^k [\mathcal{L}_\eta(\boldsymbol{\theta}^0) - \mathcal{L}_\eta(\boldsymbol{\theta}^*)] + \mathcal{O}(C_k) \,, \tag{3.9}$$

with $C_k = \max\{B_k, (1 - \frac{\rho}{L} + \epsilon)^k\}$.

Proof. Let $\tilde{L}$ denote the bound on the Lipschitz constant of g is provided in (3.2.2). By the conclusions of Lemma 2, we can take $L = \frac{1}{\sqrt{\eta+\lambda_{\min}}}\tilde{L}$. Let $\tilde{\rho}$ denote the minimum eigenvalue of (3.2.2) (note that $\tilde{\rho} \geq \eta$). Then take $\rho = \frac{1}{\sqrt{\eta+\lambda_{\max}}}\tilde{\rho}$. The convergence rate result follows immediately by Theorem 2.2 from [**51**]. □

3.3 Semi-stochastic bound method versus generic optimization methods: empirical evaluation

In this section, we provide the empirical evaluation of the performance of the semi-stochastic bound majorization method and compare it with the performance of generic optimization methods. We first briefly discuss important implementation details as well as describe the comparator methods we use for our algorithm.

3.3.1 Efficient inexact solvers

The linear system we have to invert in iteration (3.3) has very special structure. The matrix $\boldsymbol{\Sigma}$ returned by Algorithm 5 may be written as $\boldsymbol{\Sigma} = \boldsymbol{S}\boldsymbol{S}^T$, where each column of $\boldsymbol{S}$ is proportional to one of the vectors $(\mathbf{f}_{x_j}(y) - \mathbf{r})$ computed by the bound. When the dimensions of $\boldsymbol{\theta}$ are large, it is not practical to compute the $\boldsymbol{\Sigma}$ explicitly. Instead, to compute the update in iteration (3.3), we take advantage of the fact that

$$\boldsymbol{\Sigma}\boldsymbol{x} = \boldsymbol{S}(\boldsymbol{S}^T\boldsymbol{x}),$$

and use $\boldsymbol{S}$ (computed with a simple modification to the bound method) to implement the action of $\boldsymbol{\Sigma}$. When $\boldsymbol{S} \in \mathbb{R}^{d\times k}$ (k is a mini-batch size), the action of the transpose on a vector can be computed in $O(dk)$, which is very efficient for small k. The action of the regularized curvature approximation $\boldsymbol{\Sigma} + \eta\boldsymbol{I}$ follows immediately. Therefore, it is efficient to use iterative minimization schemes, such as lsqr, conjugate gradient, or others to compute the updates. Moreover, using only a few iterations of these methods further regularizes the sub-problems [**102**].

It is interesting to note that even when $\eta = 0$, and $\boldsymbol{\Sigma}_{\mathcal{T}}$ is not invertible, it makes sense to consider inexact updates. To justify this approach, we first present a range lemma.

Lemma 4. *For any $\mathcal{T}$, we have $\boldsymbol{\mu}_{\mathcal{T}} \in \mathcal{R}(\boldsymbol{\Sigma}_{\mathcal{T}})$.*

Proof. The matrix $\boldsymbol{\Sigma}_{\mathcal{T}}$ is formed by a sum of weighted outer products $(\mathbf{f}_{x_j}(y) - \boldsymbol{r})(\mathbf{f}_{x_j}(y) - \boldsymbol{r})^\top$. We can therefore write

$$\boldsymbol{\Sigma}_{\mathcal{T}} = \boldsymbol{L}\boldsymbol{D}\boldsymbol{L}^T$$

where $\boldsymbol{L} = [\boldsymbol{l}_1, \ldots, \boldsymbol{l}_{|\Omega|\cdot|\mathcal{T}|}]$, $\boldsymbol{l}_k = \mathbf{f}_{x_j}(y_k) - \boldsymbol{r}^k$ (k is the current iteration of the bound computation), and $\boldsymbol{D}$ is a diagonal matrix with weights $\boldsymbol{D}_{kk} = \frac{1}{|\mathcal{T}|}\frac{\tanh(\frac{1}{2}\log(\alpha_k/z_k))}{2\log(\alpha_k/z_k)}$, where the quantities α_k, z_k correspond to iterations in Algorithm (5). Since $\boldsymbol{\mu}$ is in the range of $\boldsymbol{L}$ by construction, it must also be the range of $\boldsymbol{\Sigma}_{\mathcal{T}}$. □

Lemma 4 tells us that there is always a solution to the linear system $\boldsymbol{\Sigma}_{\mathcal{T}}\boldsymbol{\Delta}\theta = \boldsymbol{\mu}_{\mathcal{T}}$, even if $\boldsymbol{\Sigma}_{\mathcal{T}}$ is singular. In particular, a minimum norm solution can be found using the Moore-Penrose pseudoinverse, or by simply applying lsqr or cg, which is useful in practice when the dimension d is large. For many problems, using a small number of cg iterations both speeds up the algorithm and serves as additional regularization at the earlier iterations, since the (highly variable) initially small problems are not fully solved.

3.3.2 Mini-batches selection scheme

In our experiments, we use a simple linear interpolation scheme to grow the batch sizes for both the gradient and curvature term approximations. In particular, each batch size (as a function of iteration k) is given by

$$b^k = \min(b^{\text{cap}}, b^1 + \mathsf{round}((k-1)\gamma)),$$

where b^{cap} represents the cap on the maximum allowed size, b^1 is the initial batch size, and γ gives the rate of increase. In order to specify the selections chosen, we will simply give values for each of $(b^1_{\boldsymbol{\mu}}, b^1_{\boldsymbol{\Sigma}}, \gamma_{\boldsymbol{\mu}}, \gamma_{\boldsymbol{\Sigma}})$. For all experiments, the cap $b^{\text{cap}}_{\boldsymbol{\mu}}$ on the gradient computation was the full training set, the cap $b^{\text{cap}}_{\boldsymbol{\Sigma}}$ for the curvature term was taken to be 200, initial $b^1_{\boldsymbol{\mu}}$ and $b^1_{\boldsymbol{\Sigma}}$ were both set to 5. At each iteration of SQB, the parameter vector is updated as follows:

$$\boldsymbol{\theta}_{k+1} = \boldsymbol{\theta}_k - \boldsymbol{\xi}^k,$$

where $\boldsymbol{\xi}^k = \alpha(\boldsymbol{\Sigma}^k_{\mathcal{S}} + \eta \boldsymbol{I})^{-1}(\boldsymbol{\mu}^k_{\mathcal{T}} + \eta \boldsymbol{\theta}^k)$ (α is the step size; we use constant step size for SQB in our experiments). Notice that $\boldsymbol{\xi}^k$ is the solution to the linear system $(\boldsymbol{\Sigma}^k_{\mathcal{S}} + \eta \boldsymbol{I})\boldsymbol{\xi}^k = \boldsymbol{\mu}^k_{\mathcal{T}} + \eta \boldsymbol{\theta}^k$ and can be efficiently computed using the lsqr solver (or any other iterative solver). For all experiments, we ran a small number (l) iterations of lsqr, where l was chosen from the set $\{5, 10, 20\}$, before updating the

parameter vector (this technique may be viewed as performing conjugate gradient on the bound), and chose l with the best performance (the fastest and most stable convergence).

3.3.3 Step size

One of the most significant disadvantages of standard stochastic gradient methods [29, 30] is the choice of the step size. Stochastic gradient algorithms can achieve dramatic convergence rate if the step size is badly tuned [103, 31]. An advantage of computing updates using approximated curvature terms is that the inversion also establishes a scale for the problem, and requires minimal tuning. This is well known (phenomenologically) in inverse problems. In all experiments below, we used a constant step size; for well-conditioned examples we used step size of 1, and otherwise 0.1.

3.3.4 Comparator methods

We compared SQB method with the variety of competitive state-of-the-art methods which we list below:

- **L-BFGS**: limited-memory BFGS method (quasi-Newton method) tuned for log-linear models which uses both first- and second-order information about the objective function (for L-BFGS this is gradient and approximation to the

Hessian); we use the competitive implementation obtained from http://www.di.ens.fr/∼mschmidt/Software/minFunc.html

- **SGD**: stochastic gradient descent method with constant step size; we use the competitive implementation obtained from http://www.di.ens.fr/∼mschmidt/Software/SAG.html which is analogous to L. Bottou implementation but with pre-specified step size

- **ASGD**: averaged stochastic gradient descent method with constant step size; we use the competitive implementation obtained from http://www.di.ens.fr/∼mschmidt/Software/SAG.html which is analogous to L. Bottou implementation but with pre-specified step size

- **SAG**: stochastic average gradient method using the estimate of Lipschitz constant L_k at iteration k set constant to the global Lipschitz constant; we use the competitive implementation of [31] obtained from http://www.di.ens.fr/∼mschmidt/Software/SAG.html

- **SAGls**: stochastic average gradient method with line search, we use the competitive implementation of [31] obtained from http://www.di.ens.fr/∼mschmidt/Software/SAG.html; the algorithm adaptively estimates Lipschitz constant L with respect to the logistic loss function using line-search

Since our method uses the constant step size we chose to use the same scheme

for the competitor methods like SGD, ASGD and SAG. For those methods we tuned the step size to achieve the best performance (the fastest and most stable convergence). Remaining comparators (L-BFGS and SAGls) use line-search.

3.3.5 Experiments

We performed experiments with l_2-regularized logistic regression on binary classification task with regularization parameter $\eta = \frac{1}{T}$. We report the results for six datasets: *protein* and *quantum* downloaded from the KDD Cup 2004 website http://osmot.cs.cornell.edu/kddcup, *sido* downloaded from the Causality Workbench website http://www.causality.inf.ethz.ch/home.php, *rcv1* and *covtype* downloaded from the LIBSVM Data website http://www.csie.ntu.edu.tw/ cjlin/libsvmtools/datasets and *adult*, a UCI dataset downloaded from http://archive.ics.uci.edu/ml/. Three datasets are sparse: *rcv1* ($T = 20242$, $d = 47236$; SQB parameters: $l = 5$, $\gamma_{\boldsymbol{\mu}} = 0.005$, $\gamma_{\boldsymbol{\Sigma}} = 0.0003$), *adult* ($T = 32561$, $d = 123$; SQB parameters: $l = 5$, $\gamma_{\boldsymbol{\mu}} = 0.05$, $\gamma_{\boldsymbol{\Sigma}} = 0.001$) and *sido* ($T = 12678$, $d = 4932$; SQB parameters: $l = 5$, $\gamma_{\boldsymbol{\mu}} = 0.01$, $\gamma_{\boldsymbol{\Sigma}} = 0.0008$). The remaining datasets are dense: *covtype* ($T = 581012$, $d = 54$; SQB parameters: $l = 10$, $\gamma_{\boldsymbol{\mu}} = 0.0005$, $\gamma_{\boldsymbol{\Sigma}} = 0.0003$), *protein* ($T = 145751$, $d = 74$; SQB parameters: $l = 20$, $\gamma_{\boldsymbol{\mu}} = 0.005$, $\gamma_{\boldsymbol{\Sigma}} = 0.001$) and *quantum* ($T = 50000$, $d = 78$; SQB parameters: $l = 5$, $\gamma_{\boldsymbol{\mu}} = 0.001$, $\gamma_{\boldsymbol{\Sigma}} = 0.0008$). Each dataset was split to training and testing datasets such that 90% of the original

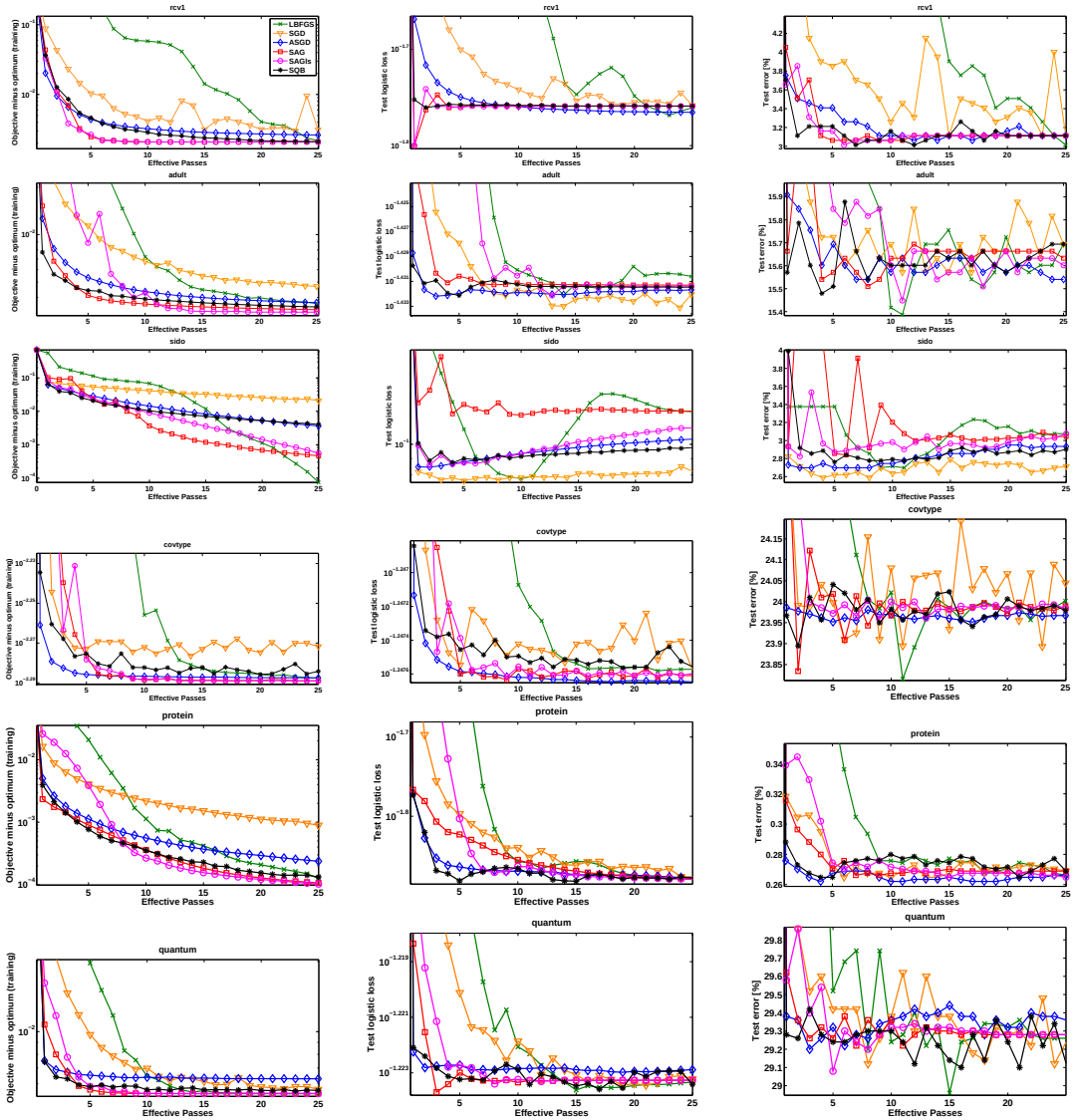


Figure 3.1: Comparison of optimization strategies for l_2-regularized logistic regression. From left to right: training excess cost, testing cost and testing error. From top to bottom: *rcv1* ($\alpha_{\mathrm{SGD}} = 10^{-1}$, $\alpha_{\mathrm{ASGD}} = 1$, $\alpha_{\mathrm{SQB}} = 10^{-1}$), *adult* ($\alpha_{\mathrm{SGD}} = 10^{-3}$, $\alpha_{\mathrm{ASGD}} = 10^{-2}$, $\alpha_{\mathrm{SQB}} = 1$), *sido* ($\alpha_{\mathrm{SGD}} = 10^{-3}$, $\alpha_{\mathrm{ASGD}} = 10^{-2}$, $\alpha_{\mathrm{SQB}} = 1$), *covtype* ($\alpha_{\mathrm{SGD}} = 10^{-4}$, $\alpha_{\mathrm{ASGD}} = 10^{-3}$, $\alpha_{\mathrm{SQB}} = 10^{-1}$), *protein* ($\alpha_{\mathrm{SGD}} = 10^{-3}$, $\alpha_{\mathrm{ASGD}} = 10^{-2}$, $\alpha_{\mathrm{SQB}} = 1$) and *quantum* ($\alpha_{\mathrm{SGD}} = 10^{-4}$, $\alpha_{\mathrm{ASGD}} = 10^{-2}$, $\alpha_{\mathrm{SQB}} = 10^{-1}$) datasets. This figure is best viewed in color.

datasets was used for training and the remaining part for testing. Only *sido* and *protein* were split in half to training and testing datasets due to large disproportion

of the number of data points belonging to each class. The experimental results we obtained are shown in Figure 3.1. We report the training excess cost and the testing cost as well as the testing error as a function of the number of effective passes through the data and thus the results do not rely on the implementation details. We would like to emphasize however that under current implementation the average running time for the bound method across the datasets is comparable to that of the competitor methods. The experimental evaluation shows that the semi-stochastic quadratic bound method is competitive with state-of-the-art generic methods.

3.4 Conclusion

We showed a semi-stochastic quadratic bound method, which is a semi-stochastic variant of the partition function bound majorization technique proposed in the previous chapter of this thesis. The convergence theory we presented is divided into two parts. First, we proved the global convergence to a stationary point under weak hypotheses (in particular, convexity is not required). Second, for the logistic regression problem, we provided a stronger convergence theory, including a rate of convergence analysis. We therefore developed and analyzed a flexible framework that allows sample-based approximations of the bound derived in the previous chapter of this thesis which are appropriate in the large-scale setting, computationally efficient, and competitive with state-of-the-art methods.

In Chapter 2 and 3 of this thesis we did not address the problem of computational complexity in a setting where the number of classes for multi class prediction is very large. We will address this problem in the next chapter where we will be reducing the multi class classification problem to a set of simpler (binary) sub-problems which can further be optimized using any optimization technique discussed before (e.g. gradient descent style optimization, bound optimization etc.).

4

Online multi class partition trees for logarithmic time predictions

This chapter is based on joint work with Alekh Agarwal and John Langford and is currently in preparation for submission [104]. *All codes are released and are publicly available at* https://github.com/AnnaChoromanska/vowpal_wabbit.

In this chapter we study the multi class classification problem in the setting where the data comes in a stream and the number of classes is large. The existing approaches to this problem, including the bound majorization method discussed before, are either intractable (their running time is often $\mathcal{O}(k)$, where k is the number

of classes, whereas for large k it is desirable to achieve train and test running times which are $\mathcal{O}(\log k)$ or possibly better), or do not adapt well to the data. In this chapter we consider a common approach to this problem which reduces it to a set of binary regression problems organized in a tree structure that naturally allows logarithmic time prediction.

The objective of this chapter is to introduce a new splitting criterion (objective function) which could be easily optimized online using standard optimization tools like gradient descent style optimization or bound majorization method, and simultaneously gives balanced (logarithmic depth) trees and small multi class classification error. This chapter is organized as follows. Section 4.1 proposes a splitting criterion and shows its basic properties, Section 4.2 introduces a probabilistic interpretation of the proposed splitting criterion and continues analyzing its properties, and Section 4.3 presents a boosting statement addressing the quality of the obtained multi class partition trees. Finally, Section 4.4 shows the resulting algorithm for online, logarithmic time multi class classification with partition trees and Section 4.5 provides the empirical evaluation of the quality of splits obtained by optimizing the proposed splitting criterion. We conclude this chapter with a brief discussion.

4.1 Splitting criterion

In this section we will introduce the splitting criterion that we use in every node of the tree to decide whether the data point coming to this node should be sent to the left or right child node, and next we will show several desirable properties of this splitting criterion (objective function). In particular, we will show that maximizing this objective function induces simultaneously balanced and pure splits and furthermore this objective function can be online optimized using easy gradient descent-style optimization. The first property can potentially be satisfied by other existing objective functions, such as entropy-based objectives (the in-depth discussion of these objectives and the conditions where they can lead to balanced and pure splits is done for example in [67]). However, to the best of our knowledge, the objective function we developed is the only existing one up-till-now used in the context of decision trees which can be easily optimized online (as opposed to other objectives, like the previously mentioned entropy-based objectives).

4.1.1 Formulation of the objective function

For notation simplicity consider the split done in the root. Let $\mathcal{X}$ denotes the input dataset. The objective function that we aim to maximize is given as follows

$$J(n_r, k_r) = \sum_{x \in \mathcal{X}} \left[\frac{n_l}{k_t(x)} \left| \frac{k_t(x)}{n_t} - \frac{k_l(x)}{n_l} \right| + \frac{n_r}{k_t(x)} \left| \frac{k_t(x)}{n_t} - \frac{k_r(x)}{n_r} \right| \right]$$

where n_t is the total number of examples, n_r (resp. n_l) is the number of examples going to the right (resp. left) child. Notice that $n_t = n_r + n_l$. $k_t(x)$ is the total number of examples labeled in the same way as x, $k_r(x)$ (resp. $k_l(x)$) is the number of examples labeled in the same way as x that are going to the right (resp. left) child. Notice that $\forall_{x \in \mathcal{X}} k_t(x) = k_r(x) + k_l(x)$. Thus the objective function can be rewritten as

$$J(n_r, k_r) = 2 \sum_{x \in \mathcal{X}} \frac{n_r}{k_t(x)} \left| \frac{k_t(x)}{n_t} - \frac{k_r(x)}{n_r} \right|$$

By the maximally balanced split we will understand the one for which $n_r = n_t - n_r$ and thus the same number of examples were directed to the left and right child nodes. By the maximally pure split we will understand the one for which $k_r(x) = 0$ or $k_r(x) = k_t(x)$ and thus there exists no two distinct data points with the same label that are in different child nodes. The proposed objective function has certain desirable properties which are captured in Lemma 5 and Lemma 6 and in the entire Section 4.2.

Lemma 5. *If a maximally pure and balanced split exists, this split maximizes the objective.*

Proof. Let $\mathcal{X}_R$ be the set of data points in the right child node such that no data point in the left child node has the same label as any data point in $\mathcal{X}_R$ ($\mathcal{X}_L$ is

defined in analogy). Let $\mathcal{X}_B$ be the set of data points such that their labels appear in both child nodes.

$$J(n_r, k_r) = 2 \sum_{x \in \mathcal{X}} \frac{n_r}{k_t(x)} \left| \frac{k_t(x)}{n_t} - \frac{k_r(x)}{n_r} \right| = 2 \sum_{x \in \mathcal{X}_R} \frac{n_r}{k_t(x)} \left| \frac{k_t(x)}{n_t} - \frac{k_r(x)}{n_r} \right|$$

$$+2 \sum_{x \in \mathcal{X}_L} \frac{n_r}{k_t(x)} \left| \frac{k_t(x)}{n_t} - \frac{k_r(x)}{n_r} \right| + 2 \sum_{x \in \mathcal{X}_B} \frac{n_r}{k_t(x)} \left| \frac{k_t(x)}{n_t} - \frac{k_r(x)}{n_r} \right|$$

$$= 2 \sum_{x \in \mathcal{X}_R} \frac{n_t - n_r}{n_t} + 2 \sum_{x \in \mathcal{X}_L} \frac{n_r}{n_t}$$

The last step comes from the fact that we consider maximally pure split thus $\forall_{x \in \mathcal{X}_R} k_r(x) = k_t(x)$, $\forall_{x \in \mathcal{X}_L} k_r(x) = 0$ and thus furthermore $\mathcal{X}_B$ must be an empty set (that eliminates the third term). We can further simplify as follows

$$2\frac{n_r(n_t - n_r)}{n_t} + 2\frac{n_r(n_t - n_r)}{n_t} = 4\frac{n_r(n_t - n_r)}{n_t}$$

Since we are maximizing the objective, we set n_r to $n_r = n_l = \frac{1}{2} n_t$. That shows the split is also maximally balanced. □

Lemma 6. *For any class the optimal value of $k_r(x)$ in isolation (fix other classes) is either 0 or 1 (thus in isolation, maximizing the objective prevents from splitting the class between two children (leading to a higher purity split)).*

Proof. Objective function is

$$J(n_r, k_r) = 2\sum_{x\in\mathcal{X}} \frac{n_r}{k_t(x)} \left| \frac{k_t(x)}{n_t} - \frac{k_r(x)}{n_r} \right| = 2\sum_{x\in\mathcal{X}} \frac{1}{k_t(x)} \left| \frac{n_r k_t(x) - n_t k_r(x)}{n_t} \right|$$

$$= 2\sum_{x\in\mathcal{X}} \frac{1}{k_t(x)} \left| \frac{n_r k_t(x)}{n_t} - k_r(x) \right| = 2\sum_{x\in\mathcal{X}} \frac{1}{k_t(x)} \left| \frac{(k_r(x)+\mathcal{C})k_t(x)}{n_t} - k_r(x) \right|$$

where $\mathcal{C}$ is a fixed constant. In order to optimize this expression for $k_r(x)$ (notice $\forall_{x\in\mathcal{X}} k_r(x) \in< 0, k_t(x) >$) one has to set $k_r(x)$ to either $k_r(x) = 0$ or $k_r(x) = k_t(x)$. □

The properties of the proposed objective function that we showed so far are promising thought still insufficient to be convinced that it is a reasonable objective to optimize in order to obtain highly balanced and pure splits (or ideally maximally balanced and pure splits). We will now show two more very important properties which indicate that increasing the value of the objective functions leads to more balanced splits and simultaneously more pure splits.

4.2 Purity and balancing factors

In order to show some more interesting properties of the objective function we need to introduce more formal notation. Let k be the number of labels. Let $\mathcal{H}$ be the

hypothesis class. Let π_i be the probability that randomly chosen data point from the dataset has label i. Consider hypothesis $h \in \mathcal{H}$ and denote $Pr(h(x) > 0|i)$ to be the probability that $h(x) > 0$ given that x has label i. We can then define pure and balanced splits as follows

Definition 3. *The hypothesis $h \in \mathcal{H}$ induces a pure split if*

$$\sum_{i=1}^{k} \pi_i \min(Pr(h(x) > 0|i), Pr(h(x) < 0|i)) \leq \delta.$$

Definition 4. *The hypothesis $h \in \mathcal{H}$ induces a balanced split if*

$$\exists_{c_1 \in (0,1)} c \leq Pr(h(x) > 0) \leq 1 - c.$$

We will refer to $\epsilon = \sum_{i=1}^{k} \pi_i \min(Pr(h(x) > 0|i), Pr(h(x) < 0|i))$ as the purity factor as it determines how pure the split is and we will refer to $p = Pr(h(x) > 0)$ as the balancing factor as it determines how balanced the split is. One can express the objective function in the equivalent form given below (notice that n_r and k_r depend on h)

$$J(h) = 2 \sum_{i=1}^{k} \pi_i \left[|P(h(x) > 0) - P(h(x) > 0|i)| \right]$$

We will now show that increasing the value of objective function leads to recovering

the hypothesis that induces more balanced splits and also more pure splits.

4.2.1 Balancing factor

We want to show the relation between the balancing factor and the value of the objective function. In order to do that we will start from deriving an upper-bound on $J(h)$, where $h \in \mathcal{H}$ is some hypothesis in the hypothesis class. For the ease of notation let $P_i = Pr(h(x) > 0|i)$. Thus

$$J(h) = 2\sum_{i=1}^{k} \pi_i \left|P(h(x) > 0|i) - P(h(x) > 0)\right| = 2\sum_{i=1}^{k} \pi_i \left|P_i - \sum_{j=1}^{k} \pi_j P_j\right|,$$

where $\forall_{i=\{1,2,\ldots,k\}} 0 \leq P_i \leq 1$. The objective $J(h)$ is definitely maximized on the extremes of the $[0, 1]$ interval. The upper-bound on $J(h)$ can be thus obtained by setting some of the P_i's to 1's and remaining ones to 0's. To be more precise, let $L_1 = \{i : i \in \{1, 2, \ldots, k\}, P_i = 1\}$ and $L_2 = \{i : i \in \{1, 2, \ldots, k\}, P_i = 0\}$. We can then write that

$$J(h) \leq 2\left[\sum_{i\in L_1} \pi_i(1 - \sum_{j\in L_1} \pi_j) + \sum_{i\in L_2} \pi_i \sum_{j\in L_1} \pi_j\right]$$

$$= 2\left[\sum_{i\in L_1} \pi_i - (\sum_{i\in L_1} \pi_i)^2 + (1 - \sum_{i\in L_1} \pi_i)\sum_{i\in L_1} \pi_i\right] = 4\left[\sum_{i\in L_1} \pi_i - (\sum_{i\in L_1} \pi_i)^2\right]$$

Recall the balancing factor $p = Pr(h(x) > 0)$. Notice that $p = \sum_{i \in L_1} \pi_i$ thus

$$J(h) \leq 4p(1-p) \Leftrightarrow 4p^2 - 4p + J(h) \leq 0$$

Thus

$$p \in \left[\frac{1 - \sqrt{1 - J(h)}}{2}, \frac{1 + \sqrt{1 - J(h)}}{2} \right].$$

Thus the balancing factor p belongs to the interval $[c, 1-c]$, where $c = \frac{1-\sqrt{1-J(h)}}{2}$, and this interval is symmetric around $\frac{1}{2}$. Maximizing $J(h)$ leads to narrowing

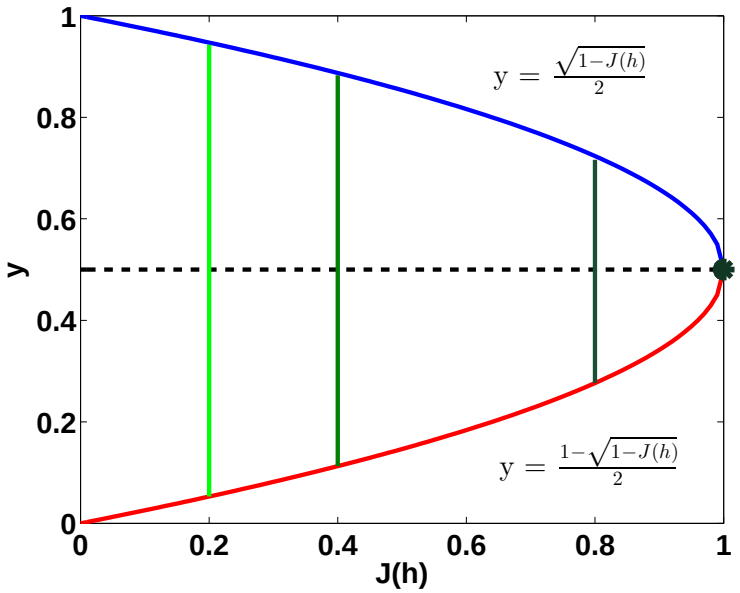


Figure 4.1: Blue: upper-bound on the balancing factor, red: lower-bound on the balancing factor, green: interval where the balancing factor lies for different values of the objective function $J(h)$.

the $[c, 1-c]$ interval around value $\frac{1}{2}$ which is the value that corresponds to the maximally balanced split. In particular for the objective-maximizing hypothesis h^* (then $J(h^*) = 1$) we obtain that $c = \frac{1}{2}$ and the split is maximally balanced then. Figure 4.1 illustrates the dependence of the balancing factor on the value of the objective function.

4.2.2 Purity factor

In analogy to what was shown before now we want to find the relation between the purity factor and the value of the objective function. As before in order to do that we will start from deriving an upper-bound on $J(h)$, where $h \in \mathcal{H}$. Again for the ease of notation let $P_i = Pr(h(x) > 0|i)$. Thus

$$J(h) = 2\sum_{i=1}^{k} \pi_i \left| P(h(x) > 0|i) - P(h(x) > 0) \right| = 2\sum_{i=1}^{k} \pi_i \left| P_i - \sum_{j=1}^{k} \pi_j P_j \right|,$$

where $\forall_{i=\{1,2,\ldots,k\}} 0 \leq P_i \leq 1$. Let $\epsilon_i = \min(P_i, 1 - P_i)$ and recall the purity factor $\epsilon = \sum_{i=1}^{k} \pi_i \epsilon_i$ and the balancing factor $p = P(h(x) > 0)$. Without loss of generality, let $p \leq \frac{1}{2}$. Let $L_1 = \{i : i \in \{1, 2, \ldots, k\}, P_i \geq \frac{1}{2}\}$, $L_2 = \{i : i \in \{1, 2, \ldots, k\}, P_i \in [p, \frac{1}{2})\}$ and $L_3 = \{i : i \in \{1, 2, \ldots, k\}, P_i < p\}$. First notice that:

$$p = \sum_{i=1}^{k} \pi_i P_i = \sum_{i \in L_1} \pi_i (1 - \epsilon_i) + \sum_{i \in L_2 \cup L_3} \pi_i \epsilon_i = \sum_{i \in L_1} \pi_i - 2\sum_{i \in L_1} \pi_i \epsilon_i + \epsilon$$

We can then write that

$$\frac{J(h)}{2} = \sum_{i=1}^{k} \pi_i \left| P_i - p \right| = \sum_{i \in L_1} \pi_i (1 - \epsilon_i - p) + \sum_{i \in L_2} \pi_i (\epsilon_i - p) + \sum_{i \in L_3} \pi_i (p - \epsilon_i)$$

$$= \sum_{i \in L_1} \pi_i (1 - p) - \sum_{i \in L_1} \pi_i \epsilon_i + \sum_{i \in L_2} \pi_i \epsilon_i - \sum_{i \in L_2} \pi_i p + \sum_{i \in L_3} \pi_i p - \sum_{i \in L_3} \pi_i \epsilon_i$$

$$= \sum_{i \in L_1} \pi_i (1 - p) - \sum_{i \in L_1} \pi_i \epsilon_i + \sum_{i \in L_2} \pi_i \epsilon_i - \sum_{i \in L_2} \pi_i p + p(1 - \sum_{i \in L_1} \pi_i - \sum_{i \in L_2} \pi_i) - \sum_{i \in L_3} \pi_i \epsilon_i$$

$$= \sum_{i \in L_1} \pi_i (1 - 2p) - \sum_{i \in L_1} \pi_i \epsilon_i + \sum_{i \in L_2} \pi_i \epsilon_i + p(1 - 2 \sum_{i \in L_2} \pi_i) - \sum_{i \in L_3} \pi_i \epsilon_i$$

$$= \sum_{i \in L_1} \pi_i (1 - 2p) + p(1 - 2 \sum_{i \in L_2} \pi_i) - \epsilon + 2 \sum_{i \in L_2} \pi_i \epsilon_i$$

$$= (1 - 2p)(p + 2 \sum_{i \in L_1} \pi_i \epsilon_i - \epsilon) + p(1 - 2 \sum_{i \in L_2} \pi_i) - \epsilon + 2 \sum_{i \in L_2} \pi_i \epsilon_i$$

$$= 2(1 - p)(p - \epsilon) + 2(1 - 2p) \sum_{i \in L_1} \pi_i \epsilon_i - 2p \sum_{i \in L_2} \pi_i + 2 \sum_{i \in L_2} \pi_i \epsilon_i$$

$$= 2(1-p)(p-\epsilon) + 2(1-2p)\sum_{i\in L_1}\pi_i\epsilon_i + 2\sum_{i\in L_2}\pi_i(\epsilon_i - p)$$

$$\leq 2(1-p)(p-\epsilon) + 2(1-2p)\sum_{i\in L_1}\pi_i\epsilon_i + 2\sum_{i\in L_2}\pi_i(\frac{1}{2} - p)$$

$$\leq 2(1-p)(p-\epsilon) + 2(1-2p)\epsilon + 1 - 2p$$

$$= 2p(1-p) - 2\epsilon(1-p) + 2\epsilon(1-2p) + 1 - 2p$$

$$= 1 - 2p^2 - 2p\epsilon$$

Thus:

$$\epsilon \leq \frac{2 - J(h)}{4p} - p$$

Thus the upper-bound on the purity factor is δ , where $\delta = \frac{2-J(h)}{4p} - p$. We already know that maximizing $J(h)$ leads to narrowing the $[c, 1-c]$ interval around value $\frac{1}{2}$ which is the value that corresponds to the maximally balanced split. Thus narrowing this interval results in pushing p closer to value $\frac{1}{2}$ and that will result in

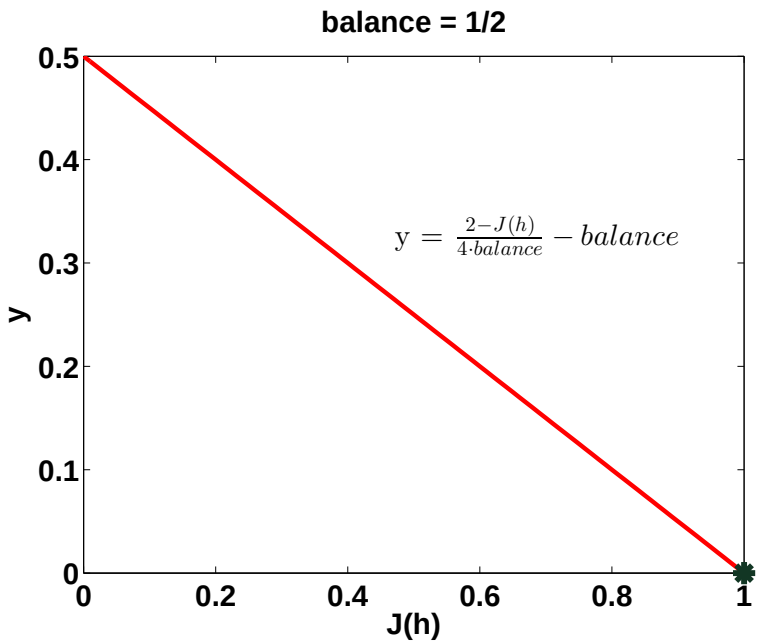


Figure 4.2: Red: the upper-bound on the purity factor as a function of $J(h)$ when the balancing factor is fixed to $\frac{1}{2}$.

the decrease of δ. In particular for the objective-maximizing hypothesis h^* (then $J(h^*) = 1$) we obtain the maximally balanced split ($p = \frac{1}{2}$) and simultaneously we obtain that $\delta = 0$ and thus this split is also maximally pure then. Figure 4.2 illustrates the dependence of the balancing factor on the value of the objective function.

4.3 Boosting statement

We will now use the entropy of the tree leafs, a standard measure used in decision trees, to measure the quality of obtained tree and show the upper-bound on the number of splits required to reduce this measure below threshold ϵ. We borrow from

the theoretical analysis of decision tree algorithms in [67] originally developed to show the boosting properties of the decision trees for binary classification problems. Our analysis generalizes the analysis there to the multi class classification setting. Consider the tree T, where every node except for leafs (we will refer to the set of the tree leafs as $\mathcal{L}$) is 'characterized' by the splitting hypothesis $h \in \mathcal{H}$ recovered by maximizing the objective function introduced before. We will consider the entropy function G as the measure of the quality of tree T:

$$G(T) = \sum_{n \in \mathcal{L}} w(n) \sum_{i=1}^{k} -\pi_{ni} \ln(\pi_{ni})$$

where π_{ni}'s are the probabilities that randomly chosen x drawn from the underlying target distribution $\mathcal{P}$ has label i given that x reaches node n and $w(n)$ is the weight of leaf n defined as the probability of randomly chosen x drawn from $\mathcal{P}$ to reach leaf n (note that $\sum_{n \in \mathcal{L}} w(n) = 1$).

Fix a leaf node n. For the ease of notation let $w = w_n$. We will consider splitting the leaf to two children n_0 and n_1. For the ease of notation let $w_0 = w_{n_0}$ and $w_1 = w_{n_1}$. Also for the ease of notation let $p = P(h_n(x) > 0)$ and $P_i = P(h_n(x) > 0|i)$. Let π_i be the probability that randomly chosen x drawn from $\mathcal{P}$ has label i given that x reaches node n. Recall that $p = \sum_{i=1}^{k} \pi_i P_i$ and $\sum_{i=1}^{k} \pi_i = 1$. Also notice that $w_0 = w(1-p)$ and $w_1 = wp$. Let $\boldsymbol{\pi}$ be the k-element vector with i^{th} entrance equal to π_i. Furthermore let $G(\boldsymbol{\pi}) = \sum_{i=1}^{k} -\pi_i \ln(\pi_i)$. Before the

split the contribution of node n to the total loss of the tree T, that we will refer to as G_t (t is the index of the current iteration), was $wG(\pi_1, \pi_2, \ldots, \pi_k)$. Let $\pi_i(n_0) = \frac{\pi_i(1-P_i)}{1-p}$ and $\pi_i(n_1) = \frac{\pi_i P_i}{p}$ be the probabilities that randomly chosen x drawn from $\mathcal{P}$ has label i given that x reaches node respectively n_0 and n_1. Furthermore let $\boldsymbol{\pi}(n_0)$ be the k-element vector with i^{th} entrance equal to $\pi_i(n_0)$ and let $\boldsymbol{\pi}(n_1)$ be the k-element vector with i^{th} entrance equal to $\pi_i(n_1)$. Notice that $\boldsymbol{\pi} = (1-p)\boldsymbol{\pi}(n_0) + p\boldsymbol{\pi}(n_1)$. After the split the contribution of the same, now internal, node n changes to $w((1-p)G(\pi(n_0)) + pG(\pi(n_1))$. We will denote the difference between them as Δ_t and thus

$$\Delta_t = w\left[G(\boldsymbol{\pi}_1) - (1-p)G(\boldsymbol{\pi}(n_0)) - pG(\boldsymbol{\pi}(n_1)\right]$$

We aim to lower-bound Δ_t. First, without loss of generality assume that $P_1 \leq P_2 \leq \cdots \leq P_k$. For the ease of notation let $J = J(h_n)$. Recall that

$$\frac{J}{2} = \sum_{i=1}^{k} \pi_i |P_i - p|$$

From strong concavity we know that

$$\Delta_t \geq wp(1-p)\|\boldsymbol{\pi}(n_0) - \boldsymbol{\pi}(n_1)\|_1^2 = wp(1-p)\left(\sum_{i=1}^{k} |\pi_i(n_0) - \pi_i(n_1)|\right)^2$$

$$= wp(1-p)\left(\sum_{i=1}^{k}\pi_i\left|\frac{P_i}{p}-\frac{1-P_i}{1-p}\right|\right)^2 = wp(1-p)\left(\sum_{i=1}^{k}\pi_i\left|\frac{P_i-p}{p(1-p)}\right|\right)^2$$

$$= \frac{w}{p(1-p)}\left(\sum_{i=1}^{k}|\pi_i(P_i-p)|\right)^2 = \frac{wJ^2}{4p(1-p)}$$

Furthermore notice that at round t there must be a leaf n such that $w(n) \geq \frac{G_t}{2t\ln k}$ (we assume we selected this leaf to the currently considered split), where $G_t = \sum_{n\in\mathcal{L}} w(n)\sum_{i=1}^{k} -\pi_{ni}\ln(\pi_{ni})$. That is because
$G_t = \sum_{n\in\mathcal{L}} w(n)\sum_{i=1}^{k} -\pi_{ni}\ln(\pi_{ni}) \leq \sum_{n\in\mathcal{L}} w(n)\ln k \leq 2tw_{max}\ln k$ where $w_{max} = \max_n w(n)$. Thus $w_{max} \geq \frac{G_t}{2t\ln k}$. Thus

$$\Delta_t \geq \frac{J^2 G_t}{8p(1-p)t\ln k}$$

Definition 5 (Weak Hypothesis Assumption)**.** *Let $\gamma \in (0, \min(p_n, 1-p_n)]$. Let for any distribution $\mathcal{P}$ over $\mathcal{X}$ at each non-leaf node n of the tree T there exists a hypothesis $h \in \mathcal{H}$ such that $\forall_{i\in\{1,2,\ldots,k\}}|P_{ni} - p_n| \geq \gamma$.*

Note that the Weak Hypothesis Assumption in fact requires that each non-leaf node of the tree T have a hypothesis h in its hypothesis class $\mathcal{H}$ which guarantees certain 'weak' purity of the split on any distribution $\mathcal{P}$ over $\mathcal{X}$. Also note that the condition $\gamma \in (0, \min(p_n, 1-p_n)]$ implies that $\gamma \leq \frac{1}{2}$. From the Weak Hypothesis Assumption

it follows that for any n, p_n cannot be too near 0 or 1 since $1 - \gamma \geq p_n \geq \gamma$. We will now proceed to further lower-bounding Δ_t. Note that

$$\frac{J}{2} = \sum_{i=1}^{k} \pi_i(|p - P_i|) \geq \sum_{i=1}^{k} \pi_1 \gamma = \gamma$$

Thus $J \geq 2\gamma$ and finally

$$\Delta_t \geq \frac{\gamma^2 G_t}{2(1-\gamma)^2 t \ln k}.$$

Let $\eta = \sqrt{\frac{8}{(1-\gamma)^2 \ln k}}\gamma$. Then

$$\Delta_t > \frac{\eta^2 G_t}{16t}$$

Thus we obtain the recurrence inequality

$$G_{t+1} \leq G_t - \Delta_t < G_t - \frac{\eta^2 G_t}{16t} = G_t \left[1 - \frac{\eta^2}{16t}\right]$$

We can now compute the minimum number of splits required to reduce G_t below ϵ, where $\epsilon \in [0, 1]$. We use the result from [67] (see the proof of Theorem 10) and obtain the following theorem.

Theorem 5. *Under the Weak Hypothesis Assumption, for any $\epsilon \in [0, 1]$, to obtain*

$G_t \leq \epsilon$ it suffices to make

$$t \geq \left(\frac{1}{\epsilon}\right)^{\frac{4(1-\gamma)^2 \ln k}{\gamma^2}}$$

splits.

The tree depth is the logarithm of the number of splits and thus under favorable conditions Theorem 5 guarantees logarithmic depth tree which implies logarithmic train and test running times (note that the Weak Hypothesis Assumption may be violated for some datasets, like in example xor-type dataset for which simultaneously balanced and pure split does not exist).

Alongside the theoretical results we have already shown, we also obtained one more interesting theoretical result for the special case when having $k = 2$ classes. We can show that the worst case value of the balancing factor p w.r.t. the change in the potential is a balanced p, in particular we can prove the lemma which states that in the worst case setting when Δ_t is minimized, the value of p has to lie in the interval $[0.4, 0.6]$. We defer this result to the Appendix B.

4.4 Algorithm

The objective function that we showed in Section 4.2 and consider throughout the paper has one more convenient equivalent form which will yield a simple online

algorithm for tree construction and training. Notice that

$$J(h) = 2\mathbb{E}_{x,i}[|P(h(x) > 0) - P(h(x) > 0|i)|] = 2\mathbb{E}_{x,i}[|P(h(x) > 0) - P(h(x) > 0|i)|]$$

$$= 2\mathbb{E}_{x,i}\left[\left|\frac{\sum_{j=1}^{n}\mathbf{1}(h(x_j) > 0)}{n_t} - \frac{\sum_{j=1}^{n_i}\mathbf{1}(h(x_j) > 0)}{n_i}\right|\right],$$

where n_i is the number of data points in class i. This is a discrete optimization problem that we relax to the following form

$$J(h) = 2\mathbb{E}_{x,i}\left[\left|\mathbb{E}_x[h(x) > 0] - \mathbb{E}_{x|i}[h(x) > 0]\right|\right],$$

where $E_{x|i}[h(x) > 0]$ denotes expected margin of class i ($h(x)$ is always clipped to the interval $[-1, 1]$). We can store the empirical estimate of the expectations and very easily update them online. The sign of the difference decides whether the currently seen example should be send to the left or right child node.

Algorithm 6 shows how the tree is simultaneously constructed and trained as we pass over the data. In this chapter we assume the simplest possible model and thus we consider linear regressors in the tree nodes. For the binary regression problems at the tree nodes, we used Vowpal Wabbit [105], which is a simple linear regressor trained by stochastic gradient descent. However, we would like to strongly

Algorithm 6 Online tree training (regression algorithm R)

Subroutine **Initialize node** (v)

$\boldsymbol{m}_c^v = \text{zeros}(k,1), \quad m_t^v = 0$ (sum of the margins per class and in total)
$\boldsymbol{n}_c^v = \text{zeros}(k,1), \quad n_t^v = 0$ (number of data points per class and in total)
$Y_R^v = \emptyset, \quad Y_L^v = \emptyset$ (label sets in the right and left child nodes of node v)

---

create the root node r: run **Subroutine Initialize node** (r)
foreach example $s(x,y)$ **do**
  Set $j = r$
  while j is not a leaf **do**
    update
      $\boldsymbol{m}_c^j(y) \mathrel{+}= h^j(s(x,y)); \quad \boldsymbol{n}_c^j(y)\mathrel{+}\mathrel{+}; \quad \boldsymbol{e}_c^j(y) = \boldsymbol{m}_c^j(y)/\boldsymbol{n}_c^j(y)$
      $m_t^j \mathrel{+}= h^j(s(x,y)); \quad n_t^j\mathrel{+}\mathrel{+}; \quad e_t^j = m_t^j/n_t^j$
    if $(\boldsymbol{e}_c^j(y) > e_t^j)$
      $Y_R^j = Y_R^j \cup y, \quad c = 1$
    else
      $Y_L^j = Y_L^j \cup y, \quad c = 0$
    Train h^j with example $(s(x,y), c)$ (use absolute value loss and take step in the subgradient direction)
    Set j to the child of j corresponding to c (if the child does not exist, but has to be created:[1] run subroutine **Initialize node** (j))

emphasize that, depending on the model used at the tree nodes, for the binary problems one could also use other generic optimization techniques discussed before as well as the bound majorization method.

During training, the algorithm assigns a unique label to each node of the tree which is currently a leaf. This label is the most frequent label that occurred in that leaf

[1] Each node of the tree creates simultaneously two children. The child of node j is created only if the regressor at this node wants to send the current data point $s(x,y)$ to the opposite direction to where the previous points reaching j were sent to by this regressor.

(the number of data points reaching this leaf that had this label was larger than the number of data points reaching this leaf with any other label).[2] When the tree is already constructed and trained, the testing is done by pushing a test example down the tree along the path starting from the root and ending at the leaf, where in each node of the path a corresponding regressor is directing this example either to the left or right child node. The label assigned to the test example is then the label assigned to the leaf that the example descended to.

4.5 Empirical evaluation of the splitting criterion

In this section we show the empirical evaluation of the proposed splitting criterion. Each dataset in our experiments was split into training and testing set. We report the test error obtained by our online multi class partition tree and the test error obtained by random tree, where both trees have the same number of nodes. Both methods were implemented in the open source system Vowpal Wabbit ([**105**]). We first use artificial datasets. We generated mixtures of well-separated spherical Gaussians with the means placed on the vertices of k-hypercube (on these type of datasets one-against-all algorithm achieves zero error but is intractable for large k). In Table 4.1 we compare the test error obtained by online multi class partition

[2]Note that the order of coming examples, while training the tree, can potentially affect the algorithm's performance though we haven't observed any strong sensitivity to data ordering while performing experiments where the data was subject to random permutations.

trees with the test error obtained by random trees.

k	Algorithm	depth	Testing error [%]
64	Partition tree	7	**1.6**
	Random tree	6	43.8
128	Partition tree	9	**0.8**
	Random tree	7	40.3
1024	Partition tree	13	**1.2**
	Random tree	10	65.2
8192	Partition tree	16	**1.4**
	Random tree	13	58.7

Table 4.1: Testing error for online multi class partition trees and random trees on artificial datasets.

Secondly, we performed similar experiments on some real datasets, MNIST and RCV1. The results are provided in Table 4.2. Clearly, in both cases (experiments with artificial and real datasets) our algorithm recovers balanced trees with significantly smaller error than random trees.

Dataset	k	Algorithm	depth	Testing error [%]
MNIST	10	Partition tree	5	**9.1**
		Random tree	4	34.8
RCV1	99	Partition tree	14	**37.5**
		Random tree	7	46.0

Table 4.2: Testing error for online multi class partition trees and random trees on MNIST and RCV1 datasets.

Finally, we performed preliminary experiments[3] on challenging ImageNet dataset with 22000 classes. The depth of the partition tree recovered by our algorithm was 29. The depth of the random tree was 15. The test error obtained with partition tree was 91.6%, while the random tree obtained 98.6%. We then examined top 5

[3]Experiments were run on a single machine.

test error (we count an error if the data point is not among top 5 classes predicted by the tree) and top $\log(k)$ test error. In the first case the test error obtained with partition tree was 81.9%, while the test error obtained with random tree improved only slightly to 96.5%. In the second case the test error for partition tree and random tree was respectively 74.8% and 94.5%. Clearly, also this experiment shows that partition trees significantly outperform random trees and better adapt to the data structure.

4.6 Conclusion

We showed an algorithm for online multi class classification allowing logarithmic train and test running times. It is a decision tree algorithm which differs from the traditional decision tree approaches in the objective optimized, and in how that optimization is done. The different objective guarantees simultaneously pure and balanced splits and moreover can be easily optimized online using efficient standard optimization tools, where the latter property significantly distinguishes this objective from other existing objectives used in the context of decision trees. The boosting statement that we proved provides an upper-bound on the number of splits required to reduce the entropy of the leafs (a standard measure of the quality of decision trees) of the multi class partition trees obtained by our algorithm below threshold ϵ and, to the best of our knowledge, is the first existing boosting statement

with logarithmic dependence on the label complexity.

In this chapter we considered a reduction of the multi class classification problem to a set of simpler sub-problems (binary classification problems) that we solve using standard optimization techniques, where the parameter vector is updated additively. In the real-life online settings however, it is also often the case that the data is not yet labeled to any classification task but we still want to find its meaningful representation. The next chapter will focus on this case and address the online k-means clustering problem by reducing it to a set of simpler sub-problems that we solve using standard optimization techniques (that we call experts, where each expert is a clustering subroutine run on a data mini-batch), where the parameter vector is updated multiplicatively.

5

Online k-means clustering with expert advice

This chapter is based on joint work with Claire Monteleoni that originally appeared in [106, 107]. *All codes are released and are publicly available at* www.columbia.edu/∼aec2163/NonFlash/Papers/Papers.html.

In the previous chapter we focused on the supervised learning setting and the multi class classification problem, where the number of data classes is large. However, labeling the data for any classification task is often very expensive in the first place which motivates the study of unsupervised learning setting and clustering. In this

chapter we study how to extend selected algorithms for online supervised learning to the unsupervised learning setting. We focus on the family of online supervised learning algorithms with access to expert predictors. It is an important family of learning algorithms which differs from the optimization tools we have considered so far. One important difference lies in the update rule which is multiplicative, not additive as was the case before.

The objective of this chapter is to introduce a family of online clustering algorithms obtaining approximation guarantees with respect to the k-means clustering objective, a common measure of the quality of clustering, while using an evaluation framework proposed by Dasgupta as an analog to regret. We show that there exists an extension of the supervised learning algorithms, with access to expert predictors, to the unsupervised learning setting which allows obtaining such approximation guarantees and that the obtained algorithms track the performance of the best expert in its expert set. This chapter is organized as follows. Section 5.1 shows the extension of supervised learning algorithms, with access to expert predictors, to the unsupervised learning setting and presents the family of online clustering algorithms we obtained, Section 5.2 provides the performance guarantees and Section 5.3 shows the empirical evaluation of our algorithms. Conclusions end this chapter.

5.1 From supervised to unsupervised learning setting

Here we extend algorithms from [69] and [70] to the unsupervised learning setting. These supervised online learning algorithms form their predictions on the basis of a set of n "expert" predictors, i, subject to a probability distribution over experts, $p_t(i)$, which is updated dynamically with time, t. Different update rules for $p_t(i)$ correspond to different transition dynamics, modeling different levels of non-stationarity in the data.

In our setting, the experts output clusterings, and instead of computing prediction errors in order to re-weight the experts, we compute an approximation to the current value of the k-means objective obtained by each expert. We define "loss" and "clustering" functions, unsupervised analogs to "prediction loss," and "prediction". In the clustering setting, the algorithm is not "predicting" the current observation x_t; x_t is in fact used in the clusterings of each of the experts, that inform the current clustering. This is a real-time analog to the standard clustering task, in which the action of the algorithm is not to predict, but to assign x_t to a (dynamic) cluster center. We show in Theorem 6 that our choice of loss and clustering functions satisfies (c, η)-*realizability*, a condition that allows us to extend regret analyses from [69, 70], for a family of algorithms.

5.1.1 Preliminaries

Here we define notation. Let k be the desired number of clusters. We index time by t, and let x_t be the most recent observation in the stream. There are n experts, which we index by i. Let C_t^i denote the set of centers output by the i^{th} expert at time t. We denote the center in C_t^i closest to the data point x_t as $c_t^i = \arg\min_{c \in C_t^i} \|x_t - c\|^2$. We use the notation $D(\cdot\|\cdot)$ for the Kullback-Leibler divergence, and $H(\cdot)$ for the entropy.

Definition 6. *The* **loss** *of a center, c_t, output by a clustering algorithm, at time t, is $L(x_t, c_t) = \left\| \frac{x_t - c_t}{2R} \right\|^2$. The normalization factor takes some $R \geq \|x_t\|$ for all t.*

The bound, $\|x_t\| \leq R$, is justified by the fact that any algorithm that can store points, x_t, in memory, has a physical constraint on the size of a word in memory. We assume that R also upper-bounds $\|c_t^i\|$ for i and all t. Given the bound on $\|x\|$, this would certainly hold for any reasonable centers. L_t with a single argument computes loss of any clustering algorithm or set of centers, and evaluates to our loss function computed on the closest center to x_t. We refer to cumulative loss over time (from the first observation) of any clustering algorithm as: $L_T = \sum_{t=1}^{T} L_t$. For the loss on a sequence indexed from time s to t, we use the notation: $L_{<s,t>} = \sum_{t'=s}^{t} L_{t'}$, and for the loss on a sequence indexed from time $s+1$ to t, we use $L_{(s,t>}$.

The family of algorithms that we introduce differ in their update rules for the

distribution over experts, $p_t(i)$. With respect to that distribution, we define the output of our algorithms as follows.

Definition 7. *The* **clustering** *of our algorithm at time t, with respect to its current distribution over experts, $p_t(i)$, is the weighted sum of the closest centers to x_t, per expert:* $\texttt{clust}(t) = \sum_{i=1}^{n} p_t(i) c_t^i$.

In our setting, the experts output a set of cluster centers at each iteration, and filter them to just return the center closest to x_t to the master algorithm. Note that, according to Definition 7, the master algorithm takes a convex combination of these centers and output a single center. For any usage in which k (or ak) centers must be output at every time-step, it is equivalent, with respect to all our analyses, for the algorithm to output the center above, in addition to the $k-1$ (or $ak-1$) centers, $C_t^{i^*} - \{c_t^{i^*}\}$, where $i^* = \arg\max_{i \in \{1,\ldots,n\}} p_t(i)$, *i.e.* the remaining centers output by the clustering algorithm with the current highest weight. In the experiments, we use this version to evaluate the k-means cost of the algorithm on the entire stream seen so far, however since this does not affect our analysis, we will simply refer to the clustering as defined above.

5.1.2 Algorithms

Algorithm 7 summarizes our Online Clustering with Experts (OCE) algorithms. We present 3 variants of OCE in Algorithm 7: the Static-Expert algorithm (from [69],

Algorithm 7 `OCE: Online Clustering with Experts` (3 variants)

INPUTS: Clustering algorithms $\{a_1, a_2, \ldots, a_n\}$, R: large constant,
`Fixed-Share` only: $\alpha \in [0,1]$, `Learn-`α only: $\{\alpha_1, \alpha_2, \ldots, \alpha_m\} \in [0,1]^m$
INITIALIZATION: $t = 1$, $p_1(i) = 1/n$, $\forall i \in \{1, \ldots, n\}$,
`Learn-`α only: $p_1(j) = 1/m$, $p_{1,j}(i) = 1/n$, $\forall j \in \{1, \ldots, m\}$, $\forall i \in \{1, \ldots, n\}$
AT t-TH ITERATION:
Receive vectors $\{c^1, c^2, \ldots, c^n\}$, where c^i is the output of algorithm a_i at time t.
`clust`$(t) = \sum_{i=1}^{n} p_t(i) c^i$ // `Learn-`α: `clust`$(t) = \sum_{j=1}^{m} p_t(j) \sum_{i=1}^{n} p_{t,j}(i) c^i$
OUTPUT: `clust`(t) // Optional: additionally output $C^{i^*} - \{c^{i^*}\}$,
where $i^* = \arg\max_{i \in \{1,\ldots,n\}} p_t(i)$
View x_t in the stream.
For each $i \in \{1, \ldots, n\}$:
$L(i,t) = \|\frac{x_t - c^i}{2R}\|^2$

$p_{t+1}(i) = p_t(i) e^{-\frac{1}{2}L(i,t)}$ — 1. `Static-Expert`

For each $i \in \{1, \ldots, n\}$: — 2. `Fixed-Share`
$p_{t+1}(i) = \sum_{h=1}^{n} p_t(h) e^{-\frac{1}{2}L(h,t)} P(i \mid h; \alpha)$ // $P(i|h;\alpha) = \begin{cases} 1-\alpha & \text{if } i = h; \\ \frac{\alpha}{n-1} & \text{o.w.} \end{cases}$

For each $j \in \{1, \ldots, m\}$: — 3. `Learn-`α
`lossPerAlpha[j]` $= -\log \sum_{i=1}^{n} p_{t,j}(i) e^{-\frac{1}{2}L(i,t)}$
$p_{t+1}(j) = p_t(j) e^{-\texttt{lossPerAlpha[j]}}$
For each $i \in \{1, \ldots, n\}$:
$p_{t+1,j}(i) = \sum_{h=1}^{n} p_{t,j}(h) e^{-\frac{1}{2}L(h,t)} P(i \mid h; \alpha_j)$
Normalize $p_{t+1,j}$.

Normalize p_{t+1}.
t=t+1

---

with a prior history in the literature), and Fixed-share, introduced by Herbster and Warmuth [69] as a simple way to model time-varying data in the experts setting. We also provide an OCE version of Learn-α, an algorithm introduced by Monteleoni and Jaakkola [70], in the supervised setting, to address the question of how to run

Fixed-Share algorithms without the knowledge of the hindsight optimal value of the parameter α. Learn-α learns the parameter online using Static-Expert updates over a set of Fixed-share algorithms, each with a different value of the α parameter; we use the discretization procedure from [70].[1]

We state the algorithms in their most general form, in which the experts are arbitrary black boxes that output a set of cluster centers at each iteration, and filter them to just return the center closest to x_t, the current observation. The OCE algorithm views only this set of n centers, $\{c^1, c^2, \ldots, c^n\}$, before producing its output. Then, to compute loss and perform the weight updates, it views x_t, the current observation in the stream. Our regret analyses hold in this setting. When the experts are instantiated as batch clustering algorithms, we denote by W_t a sliding window, of size W, of the stream through and including x_t, on which the experts compute the current clustering. This aligns with approximation guarantees for clustering algorithms in the literature, *i.e.* algorithms cluster an input dataset and are then evaluated with respect to the optimum of some objective function, computed on the same dataset. Our approximation guarantees hold in this setting, and we show that no b-approximate clustering algorithm can trivially optimize our loss function by outputting x_t. The algorithm also permits the use of (a, b)-approximation algorithms as experts.

[1] We also have analyses for a generalization of Fixed-Share, with arbitrary transition dynamics, studied in [70].

5.2 Performance guarantees

We will give two types of performance guarantees for our family of online clustering algorithms. First we will provide similar bounds to those in [69, 70] for the setting of supervised online learning with experts. Then we will instantiate the experts as batch clustering algorithms, with approximation guarantees, to yield online variants of approximation guarantees with respect to the k-means objective. Omitted proofs appear in the Appendix C.

5.2.1 Regret bounds

In order to make use of analysis tools from the literature, we first need to show that for our clustering and loss functions, a certain property holds. Here we reformulate the definition of (c, η)-**realizability** presented in [69], and due to [108, 109], and demonstrate that it holds in our setting.

Theorem 6. *The loss function L defined in Definition 6 and the clustering function defined in Definition 7 are $(2, \frac{1}{2})$-***realizable***, i.e.:*

$$L(x_t, \texttt{clust}(t)) \leq -2 \log \sum_{i=1}^{n} p(i) e^{-\frac{1}{2} L(x_t, c_t^i)}$$

for all $n \in \mathbb{N}$ and all x_t and c_t^i such that $\|x_t\| \leq R$, $\|c_t^i\| \leq R$, and all stochastic vectors $p \in [0\,1]^n$.

Using this, we can now provide regret bounds for a family of online clustering algorithms. For our OCE algorithm's Static-Expert variant, shown in Algorithm 7, we have the following bound.

Theorem 7. *Given any sequence of length T, let a_i* be the best expert in hindsight (with respect to cumulative loss on the sequence). Then the cumulative loss of the algorithm obeys the following bound, with respect to that of the best expert:*

$$L_T(\texttt{alg}) \leq L_T(a_i*) + 2\log n$$

The proof follows by an almost direct application of an analysis technique of [70]. We now provide regret bounds for our OCE Fixed-Share variant. First, we follow the analysis framework of [69] to bound the regret of the Fixed-Share algorithm (parameterized by α) with respect to the best s-partition of the sequence.[2]

Theorem 8. *For any sequence of length T, and for any $s < T$, consider the best partition, computed in hindsight, of the sequence into s segments, mapping each segment to its best expert. Then, letting $\alpha' = s/(T-1)$: $L_T(\texttt{alg}) \leq$*

$L_T(\texttt{best s-partition}) + 2[\log n + s\log(n-1) + (T-1)(H(\alpha') + D(\alpha'\|\alpha))]$.

We also provide bounds on the regret with respect to the Fixed-Share algorithm running with the hindsight optimal value of α. We extend analyses in [70, 110]

[2] To evaluate this bound, one must specify s.

to the clustering setting, which includes providing a more general bound in which the weight update over experts is parameterized by an arbitrary transition dynamics; Fixed-Share follows as a special case. Following [110, 70], we will express regret bounds for these algorithms in terms of the following log-loss: $L_t^{log} = -\log \sum_{i=1}^{n} p_t(i) e^{-\frac{1}{2}L(x_t, c_t^i)}$. This log-loss relates our clustering loss (Definition 6) as follows.

Lemma 7. $L_t \leq 2L_t^{log}$

Proof. By Theorem 6: $L_t = L(x_t, \texttt{clust}(t)) \leq -2\log \sum_{i=1}^{n} e^{-\frac{1}{2}L(x_t, c_t^i)} = 2L_t^{log}$ □

Theorem 9. *The cumulative log-loss of a generalized OCE algorithm that performs Bayesian updates with arbitrary transition dynamics, Θ, a matrix where rows, Θ_i, are stochastic vectors specifying an expert's distribution over transitioning to the other experts at the next time step, obeys*[3]*:*

$$L_T^{\log}(\Theta) \leq L_T^{\log}(\Theta^*) + (T-1)\sum_{i=1}^{n} \rho_i^* D(\Theta_i^* \| \Theta_i)$$

where Θ^* is the hindsight optimal (minimizing log-loss) setting of the transition matrix, for the observed sequence, and ρ_i^* is the marginal probability of being in state i, of the hindsight optimal algorithm.

[3] As in [110, 70], we express cumulative loss with respect to the algorithms' transition dynamics parameters.

Corollary 2. *For our OCE algorithm's Fixed-share(α) variant:*

$$L_T^{\log}(\alpha) \leq L_T^{\log}(\alpha^*) + (T-1)D(\alpha^* \| \alpha)$$

where α^* is the hindsight optimal setting of the parameter α for the observed sequence.

In the supervised setting, the analogous regret for Fixed-share has a sequence-dependent lower bound which can be linear in T [70]. Now we address the setting in which α^* is not known beforehand, and provide a regret bound for our OCE algorithm's Learn-α variant.

Theorem 10. *For our OCE algorithm's Learn-α variant, using a discretization of the α parameter, $\{\alpha_j\}$ of size m, where α^* is the hindsight optimal α:*

$$L_T^{\log}(\texttt{alg}) \leq L_T^{\log}(\alpha^*) + (T-1) \min_{\{\alpha_j\}} D(\alpha^* \| \alpha_j) + \log m$$

The proof uses Corollary 2 and a variant of regret bound for Static-Expert, which is used by Learn-α to update the weights over Fixed-share algorithms. By choice of discretization, $\{\alpha_j\}$, we can control the third term. For example, allowing the discretization to depend on T, [70] optimized their analogous regret bound. It is future work to discretize α optimally with respect to our new bounds.

5.2.2 Approximation guarantees

When the experts are k-means approximation algorithms, we can extend our regret bounds to provide online variants of approximation guarantees for OCE.

Lemma 8. *If an algorithm is b-approximate with respect to the k-means objective, there exist sequences ending in x_t for which it cannot output x_t as a center.*

We continue by providing the following lemmas, which may also be of general interest.

Lemma 9. *Let OPT_{W_1} be the optimum value of the k-means objective for the dataset seen in window W_1, OPT_{W_2} be the optimum value of the k-means objective for the dataset seen in window W_2, and $OPT_{W_1 \cup W_2}$ be the optimum value of the k-means objective for the dataset seen in window $W_1 \cup W_2$. Then: $OPT_{W_1} + OPT_{W_2} \leq OPT_{W_1 \cup W_2}$.*

Proof. Let C_1 be the clustering minimizing the k-means objective on the window W_1 and C_2 be the clustering minimizing the k-means objective on the window W_2 and let the C_3 be the clustering minimizing the k-means objective on the window $W_1 \cup W_2$. Then: $OPT_{W_1 \cup W_2} = \sum_{x \in W_1 \cup W_2} \min_{z \in C_3} \|x - z\|^2 = \sum_{x \in W_1} \min_{z \in C_3} \|x - z\|^2 + \sum_{x \in W_2} \min_{z \in C_3} \|x - z\|^2 \geq \sum_{x \in W_1} \min_{z \in C_1} \|x - z\|^2 + \sum_{x \in W_2} \min_{z \in C_2} \|x - z\|^2 = OPT_{W_1} + OPT_{W_2}$. □

Lemma 10. *Given any b-approximate k-means clustering algorithm, the sum over a sequence of length T of its k-means costs when run on sliding windows W_t of size $\leq W$, obeys: $\sum_{t=1}^{T} \Phi_{W_t} \leq b \cdot W \cdot OPT_{<1,T>}$.*

Lemma 11. *Given any b-approximate k-means clustering algorithm, a, its cumulative loss when run on a sliding window of size $\leq W$ on a stream of length T, obeys: $\sum_{t=1}^{T} L_t(a) \leq \frac{b \cdot W}{4R^2} OPT_{<1,T>}$.*

In all results below, we denote by OPT_T the optimum value of the k-means objective for the entire sequence of length T. Now we state the bound for the OCE Static-Expert algorithm.

Theorem 11. *Given any sequence of length T, let a_i* be the best expert in hindsight (with respect to cumulative loss on the sequence). When a_i* is a b-approximate batch clustering algorithm run on sliding windows W_t of size $\leq W$: $L_T(\texttt{alg}) \leq \frac{b \cdot W}{4R^2} OPT_T + 2 \log n$.*

Proof. We will expand the result from Theorem 7, using our instantiation of the experts as b_i-approximate clustering algorithms, trained on sliding windows of the data. For ease of notation let us denote by b, the approximation factor for a_i^*, the best expert with respect to minimizing $L_T(a_i)$ on the observed sequence of length T. $L_T(\texttt{alg}) \leq L_T(a_i^*) + 2 \log n = \sum_{t=1}^{T} L_t(a_i^*) + 2 \log n \leq \frac{b \cdot W}{4R^2} OPT_{<1,T>} + 2 \log n$. where the last inequality follows from Lemma 11. □

Using similar arguments, along with Lemma 7, we can extend our regret bounds for the other algorithms to provide online variants of approximation guarantees. We provide two such bounds for our OCE variant of Fixed-Share, corresponding to Theorems 8 and 2; Appendix C contains our bound for the general version.

Theorem 12. *For any sequence of length T, and for any $s < T$, consider the best partition, computed in hindsight, of the sequence into s segments, mapping each segment to its best expert. Let each of the n experts be b-approximate w.r.t. k-means, and run on sliding windows W_t of size $\leq W$. Then, letting $\alpha' = s/(T-1)$:* $L_T(\texttt{alg}) \leq \frac{bW}{4R^2} OPT_T + 2[\log n + s\log(n-1) + (T-1)(H(\alpha') + D(\alpha'\|\alpha))]$.

Theorem 13. *For our OCE algorithm's Fixed-share(α) variant, where α^* is the hindsight-optimal α:*

$$L_T(\alpha) \leq \frac{bW}{4R^2} OPT_T + 2(T-1)D(\alpha^*\|\alpha)$$

In our bound for the OCE Learn-α variant, the choice of discretization, $\{\alpha_j\}$, governs the third term.

Corollary 3. *For our OCE algorithm's Learn-α variant, using a discretization of the α parameter, $\{\alpha_j\}$ of size m, where α^* is the hindsight optimal α:*

$$L_T(\texttt{alg}) \leq \frac{bW}{4R^2} OPT_T + 2(T-1)\min_{\{\alpha_j\}} D(\alpha^*\|\alpha_j) + \log m$$

The proof combines Theorem 10 with the bound on $L_T^{\log}(\alpha^*)$ that was derived in proving Theorem 13.

5.3 Experiments

In this section we provide the empirical evaluation of the performance of our OCE algorithms. We ran OCE using other clustering algorithms from the literature as experts. To demonstrate the practical advantages of our approach, the experiments reported were run predictively in that the sliding windows did not include the current observation, x_t; we also ran experiments in the non-predictive setting that we analyzed, and observed comparable or better empirical performance. We used real and simulated datasets, some of which had been used in past works on batch clustering [74], and clustering (finite) data streams [76]. The simulated data consists of samples from a mixture of 25 well-separated Gaussians in $\mathbb{R}^{15}$; the "true" k is therefore 25. We also experimented on 5 UCI datasets, in which the "true" k is unknown: *Cloud*, *Spambase*, *Intrusion* (KDD cup 99), *Forest Fires*, and *Wall-following robot navigation* [111]. We instantiated the experts with 6 clustering algorithms: 1. Lloyd's algorithm.[4] 2. k-means++. 3. An heuristic called "Online k-means."[5] 4.-6. 3 variants of k-means# [76], in particular they were: 4. k-means# that outputs $3 \cdot k \cdot \log k$ centers. 5. k-means# that outputs $2.25 \cdot k \cdot \log k$ centers.

[4] Centers were initialized randomly, per sliding window.

[5] This algorithm has been used in practice for a while; pseudocode is stated in [88].

6. k-means# that outputs $1.5 \cdot k \cdot \log k$ centers. These are all batch clustering algorithms, except Online k-means, which we also restricted to operate only on a sliding window. Only the k-means++ and k-means# variants have approximation guarantees with respect to the k-means objective. Our Theorem 7 holds regardless, but Theorem 11 requires at least the best performing expert (in hind-sight) to be b-approximate with respect to the k-means objective.

The window size W was set arbitrarily. While we set $W = 200$, we also experimented over window size, and observed, as expected, decreasing loss with increasing window size for very stationary datasets, yet no trend for non-stationary datasets, which is consistent with our analysis (where the upper-bound on loss increases with window size).

Table 5.1 reports mean and standard deviation, over the sequence, of the the k-means cost on all points seen so far. The experts are denoted e_i; the OCE methods are: se=Static-Expert (equivalent to Fixed-Share with $\alpha = 0$), f_{1-3} are Fixed-Share algorithms using the smallest 3 values of α in the discretization, f_4 uses the middle value, and f_5 uses the highest value, and la=Learn-α. In the 3-expert experiments, we also compared with several online clustering algorithms from the literature, run on the whole sequence: ol=Online k-means (which, run on windows, was also used as expert 3.), and the "Doubling algorithm" (da).[6] Neither of these have

[6]These algorithms output k centers so running them with experts 4-6, which can output more than k centers, would not be a fair comparison, since OCE can output as many centers as its experts.

	25 Gaussians	Cloud $\times 10^7$	Spam $\times 10^8$	Intrus. $\times 10^{10}$	For. fire $\times 10^6$	Robot $\times 10^4$
e_1	0.6193±0.3195×10^8	1.3180±1.9395	0.2706±0.2793	0.1988±0.2104	0.7766±0.6413	1.8362±1.2172
e_2	**0.0036**±0.0290×10^7	**0.8837**±1.3834	**0.1042**±0.1463	**0.0743**±0.1041	**0.6616**±0.4832	**1.8199**±1.2102
e_3	2.0859±0.9204×10^8	4.6601±7.8013	1.6291±1.3292	0.7145±0.5376	7.1172±7.6576	2.3590±1.4070
da	0.0179±0.0723×10^8	**0.5285**±0.2959	0.1971±0.0826	**0.0050**±0.0529	1.4496±0.6484	2.5514±1.4239
ol	1.7714±0.6888×10^8	4.2322±2.4965	0.8222±0.7619	1.3518±0.3827	2.9617±1.3006	1.9806±1.0160
se	**0.0014**±0.0143×10^8	**0.8855**±1.3824	**0.1059**±0.1469	**0.0778**±0.1094	0.6620±0.4831	1.8139±1.2032
f_1	**0.0014**±0.0143×10^8	**0.8855**±1.3823	**0.1059**±0.1469	0.0779±0.1100	**0.6614**±0.4819	1.8137±1.2032
f_2	**0.0014**±0.0143×10^8	0.9114±1.4381	**0.1059**±0.1470	**0.0778**±0.1099	0.7008±0.5382	**1.8134**±1.2031
f_3	**0.0014**±0.0143×10^8	1.0715±1.6511	**0.1059**±0.1470	0.0779±0.1099	0.6996±0.5361	1.8145±1.2031
f_4	0.0124±0.0193×10^8	1.4806±2.6257	0.3723±0.7351	0.1803±0.2358	1.0489±1.4817	1.8334±1.2212
f_5	1.3811±1.0881×10^8	3.0837±6.3553	0.8212±1.1583	0.4126±0.5040	4.4481±6.2816	2.2576±1.3849
la	**0.0012**±0.0136×10^8	0.8862±1.3920	**0.1076**±0.1483	0.0785±0.1108	**0.6616**±0.4805	**1.8130**±1.2026
e_4	**7.3703**±4.2635×10^3	**0.6742**±1.2301	**0.0687**±0.1355	**0.0704**±0.1042	**0.2316**±0.2573	**1.3667**±1.0176
e_5	8.2289±4.4386×10^3	0.6833±1.2278	0.0692±0.1356	**0.0704**±0.1042	0.2625±0.2685	1.4385±1.0495
e_6	9.8080±4.7863×10^3	0.7079±1.2364	0.0710±0.1360	0.0705±0.1042	0.3256±0.2889	1.5713±1.1011
se	**0.1360**±1.4323×10^6	**0.6743**±1.2300	**0.0687**±0.1355	**0.0705**±0.1045	0.2322±0.2571	1.3642±1.0138
f_1	**0.1360**±1.4323×10^6	**0.6743**±1.2300	**0.0687**±0.1355	**0.0705**±0.1045	0.2322±0.2571	1.3640±1.0135
f_2	0.1361±1.4322×10^6	0.6746±1.2298	**0.0687**±0.1355	**0.0705**±0.1045	0.2322±0.2572	1.3636±1.0130
f_3	0.1364±1.4322×10^6	**0.6743**±1.2300	**0.0687**±0.1355	0.0711±0.1055	**0.2321** ±0.2570	**1.3634**±1.0127
f_4	0.0027±0.0144×10^8	0.7207±1.3025	**0.0707**±0.1357	0.0773±0.1203	0.2776±0.4917	1.3963±1.0339
f_5	1.4039±1.0790×10^8	3.0786±6.4109	0.7155±1.0650	0.4227±0.5179	4.6103±6.3019	2.3142±1.4127
la	**0.0012**±0.0134×10^8	**0.6742**±1.2300	**0.0687**±0.1355	**0.0708**±0.1046	**0.2318**±0.2573	**1.3632**±1.0128

Table 5.1: Mean and standard deviation, over the sequence, of k-means cost on points seen so far. $k = 25$ for Gaussians, $k = 15$ otherwise. The best expert and the best 2 scores of the algorithms, per experiment, are bold. Below the triple lines, 3 more experts are added to the ensemble.

known approximation guarantees with respect to the k-means objective, however the Doubling algorithm approximates the k-center objective [85]. This evaluation is a k-means cost variant of progressive validation analysis, a standard evaluation of online learning algorithms in the supervised setting, analyzed in [112] with respect to k-fold cross-validation error and standard batch holdout error. In two of the experiments, the Doubling algorithm achieves the lowest mean k-means cost over the sequence; in the other experiments, at least one of our OCE methods does. Both the 3-expert and the 6-expert experiments demonstrate that, without prior knowledge of the sequence, Learn-α is the most applicable, as its performance

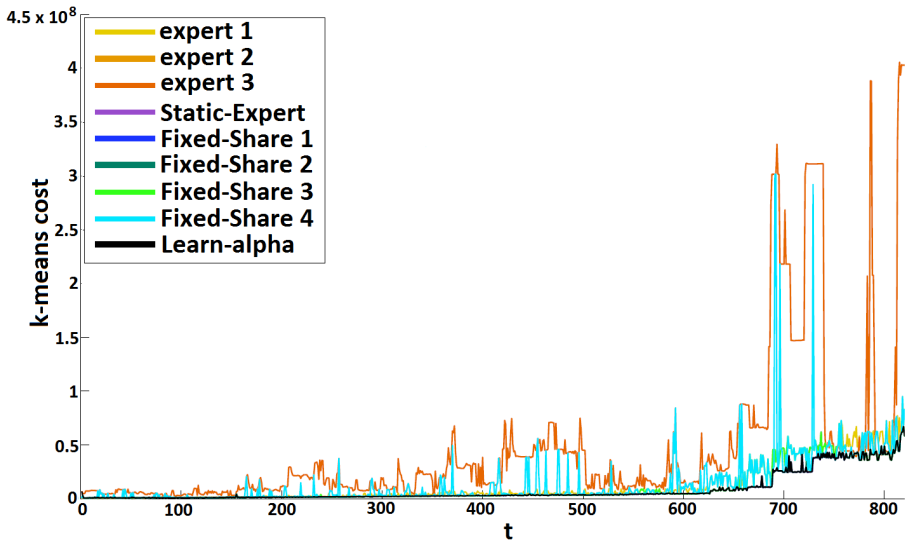


Figure 5.1: Clustering analogs to learning curves: k-means cost versus t for Cloud dataset.

tracks that of the best performing Fixed-Share(α). Interestingly, we observe that OCE (and in particular Learn-α) tracks the best expert much more effectively on all the real datasets than on the 25 Gaussian experiment. The means reported for the OCE algorithms were for this datasets hurt by the algorithms' uniform priors over experts. That is, in a few early iterations, OCE incurred costs from clusterings giving non-trivial weights to all the experts' predictions, so in those iterations costs could be orders of magnitude higher than in examples those of experts 4.-6. Moreover, as mentioned above, our regret bounds instead upper-bound loss.

In Figure 5.1 we show an example of clustering analogs to learning curves from our experiment on *Cloud* dataset, in the predictive setting, which generated the

statistics for Table 5.1. We plot the batch k-means cost of each expert, and the OCE algorithms, on all the data seen so far, versus t. While Fixed-Share algorithms with high values of α suffer large oscillations in cost, Learn-α's performance tracks, and often surpasses that of the Fixed-Share algorithms.

Figure 5.2: Evolution of weights over experts for Fixed-Share algorithms using different values of the α parameter; Forest Fires dataset; 6-experts.

We also demonstrate the evolution of the weights maintained by the Fixed-Share and Learn-α OCE algorithms. In Figure 5.2 we show the exemplary evolution over time of the weights maintained by the OCE Fixed-Share algorithm over the experts (clustering algorithms). For smaller values of α we observe an inversion in the

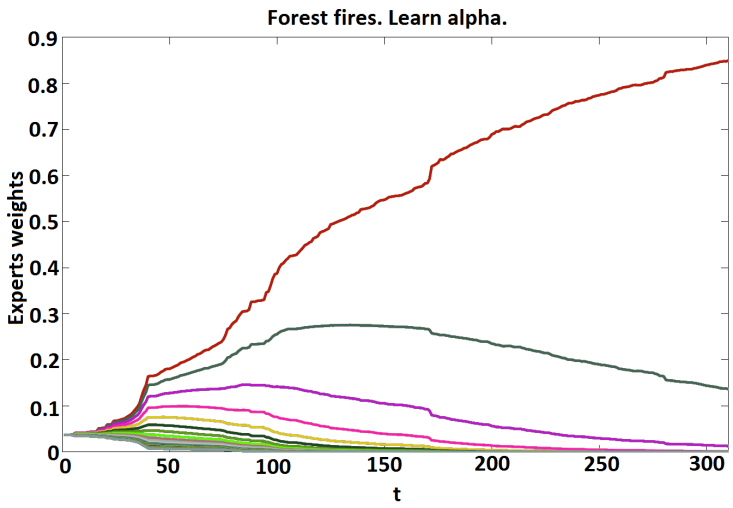


Figure 5.3: Evolution of weights maintained by Learn-α, over its α-experts, in the 6-expert Forest Fires experiment. Lowest values of α receive highest weight.

weight ordering between experts 4 and 5 at around iteration 48 for this particular experiment. For α values closer to 1/2 and 1 there is a higher amount of shifting of weights among experts. In Figure 5.3 we show the exemplary evolution over time of the weights maintained by the OCE Learn-α algorithm over α-experts (Fixed-Share algorithms run with a different setting of the α parameter). The experiment was performed with 45 α-experts (Fixed-Share algorithms with different values of the α parameter). Lower α values received higher weights. One value of α (the lowest) receives an increasing share of the weight, which is consistent with the fact that the Static-Expert algorithm is used to update weights over α-experts.

Finally, in Figure 5.4 we vary k from 5 to 25 in multiples of 5 as in [**76**], and state the final k-means cost achieved by each method in the 3-experts experiments. The

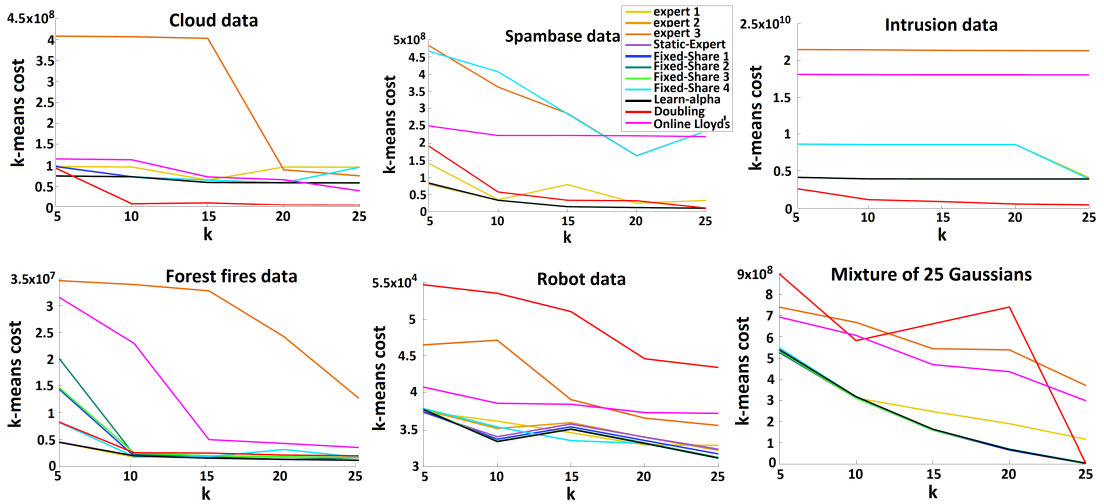


Figure 5.4: k-means cost on the entire sequence versus k for Cloud, Spambase, Intrusion, Forest Fires, Robot and 25 Gaussians datasets.

OCE methods often achieve lower final k-means cost than the other algorithms. In particular, Learn-α suffers lower k-means cost than Online k-means in all experiments except Cloud: $k = 25$. While for most values of k, the Doubling algorithm suffers lower cost on 2 datasets, and comparable cost on 2 datasets, for Robot data, which is non-stationary, the performance of Doubling is significantly worse than that of Learn-α for all k tested. This is evidence that OCE has (at least) comparable performance over a variety of datasets to these existing methods that have not been analyzed with respect to the k-means objective; moreover, it exploits the performance advantages of the clustering algorithms used as experts.

5.4 Conclusion

We showed a family of online clustering algorithms, with regret bounds, and approximation guarantees with respect to the k-means objective, of a novel form for the online clustering setting. Notably, our approximation bounds are with respect to the optimal k-means cost on the *entire* data stream seen so far, even though the algorithm is online. Our algorithms were obtained by extending algorithms from the supervised learning setting, with access to access to expert predictors, to the unsupervised learning setting. We introduced a flexible framework in which our algorithms take a set of candidate clustering algorithms, as experts, and track the performance of the "best" expert, or best sequence of experts, for the data. Instead of computing prediction errors in order to re-weight the experts, the algorithms compute an approximation to the current value of the k-means objective obtained by each expert. Our approach lends itself to settings in which the user is unsure of which clustering algorithm to use for a given data stream, and exploits the performance advantages of many batch clustering algorithms used as experts. Our algorithms vary in their models of the time-varying nature of the data and achieve encouraging performance on a variety of datasets.

6

Conclusion

In this thesis we showed several optimization techniques for convex and non-convex learning problems of central focus in machine learning. We focused on the learning approach based on reductions, where the original problem is 'reduced' to a set of simpler sub-problems that are much easier to solve. We first presented a new optimization technique based on quadratic bound majorization (this is the first machine learning reduction scheme addressed in this thesis) for optimizing the objective functions involving the partition function. The bound technique is explored in the supervised learning framework however it can be extended to unsupervised learning settings as well. We proved the linear convergence rate of the batch and semi-stochastic variants of our method and showed it can be easily recovered in

various learning problems where the only constraint we introduce is on the number of classes which is assumed to be finite and enumerable (in practice we assume it is not too large; we address the case of large number of classes separately in this thesis). The new technique differs from previously existing majorization approaches in that rather than aiming for exclusively tight bounds, it also produces a more convenient (and easier to optimize) bound. The results we obtained run counter to the conventional wisdom: machine learning problems are best handled via generic optimization algorithms.

There are still many open questions connected with our bound majorization technique that are worth exploring. Extending the bound technique to a general class of loss functions (differentiable and non-differentiable), like squared loss or hinge loss, is one example of an important future research direction. For instance, one can construct 'the widest' quadratic upper-bound on the hinge loss (similarly as it is done in our original approach for partition function-based objectives) and then use it as a surrogate function for the majorization technique. Furthermore, it turns out that using the bound one can turn the log-linear modeling problem into an iterative least squares majorization problem. Thus an interesting future research direction is to parallelize the bound method such that it can run on a distributed CPU and/or GPU-based framework by using distributed least-squares solvers. Simultaneously we are interested in extending the bound method to other models

such as deep belief networks (we have some results in this direction [113]), and furthermore applying it to sparse and group-sparse models, as well as applying it in variational Bayesian settings (e.g. integral approximations in maximum entropy discrimination and evidence approximation) and to intractable log-linear models, where the partition function in general can be NP-hard to compute exactly however fast incremental improvements on the model parameters can potentially be obtained using our bound method. Finally, from the theoretical perspective it seems important to view the bound method as a bridge between first- and second-order generic optimization methods. Alike first-order methods, the bound method access the objective function through the 1^{st} order oracle and simultaneously, alike second-order methods, it approximates the global curvature of the objective function that is however quite different than the Hessian. It seems therefore that it should be possible to prove better than linear convergence rate for the batch bound method (at least superlinear like in case of quasi-Newton methods). Similarly we are interested in further exploring the theoretical guarantees for the stochastic bound method (we have some results in this direction [114]). We are particularly interested in developing a common framework that would enable the comparison of the convergence rate of the bound method with the state-of-the-art existing approaches.

Second in this thesis, we addressed the online mutli class classification problem. We developed a practical tractable logarithmic-time online multi class classification

algorithm based on decision trees for handling problems involving learning with large number of classes, where the original multi class classification problem is reduced to a set of binary classification problems organized in a tree structure. This work proposed the first simultaneously tractable and data-adaptive algorithm for handling these types of problems, which allows easy optimization via standard optimization tools. Our work makes decision trees applicable to the multi class classification settings by proposing a new objective function to optimize in the tree nodes inducing pure and balanced splits.

There are still many open questions connected with the online logarithmic depth multi class partition trees we developed that are worth to work on. One example is to consider a different construction of the tree than the one we proposed, i.e. the setting where the tree construction is started from the leafs rather than the root which guarantees consistency (optimal binary classifiers yield optimal multi class classifier). In this sense we would like to follow the Filter Tree approach and improve it by adding the data-adaptivity step to the tree construction where the tree topology is learned rather than imposed. One way to do it is to push the most difficult classification problems to the leaf level of the tree and the easier ones to the tree root. The intuition behind this approach is that similar labels should not be split into two different nodes high in the tree (close to the root) since it may potentially incur large classification error. Another possible way to learn the tree

structure is to iteratively improve the topology of the initial tree using predefined dictionary of plausible operations on the tree (i.e. merging or splitting the nodes) which can provably increase the tree quality. Apart from these approaches we believe it is also interesting to consider the reduction of the multi class classification problem to a set of simpler sub-problems organized in other than tree structures. Finally, our work could also be extended to develop new hierarchical clustering algorithms.

Finally, in the third part of this thesis we addressed the online k-means clustering problem. We proposed the first online clustering algorithm with approximation guarantees with respect to the k-means clustering objective simultaneously solving an open problem posed in the literature by Dasgupta. Our approach explores learning with expert advice setting where experts are clustering subroutines solving clustering problems on a sliding window of the data stream (enclosing small data mini-batches).

There are still many open questions connected with the online clustering with experts algorithms we developed that are worth to work on. The natural extension of our work would be to develop an online clustering algorithm with approximation guarantees with respect to the k-means clustering objective where the experts instead of being clustering algorithms, are more 'primitive' entities, possibly satisfying no specific constraints. We are also interested in considering other clustering

settings, i.e. online spectral clustering (we have some results in this direction [115]). One way to develop a good online spectral clustering algorithm would be to explore the duality phenomenon between the spectral clustering objective and the weighted kernel k-means objective. To be more specific, the weighted kernel k-means algorithm may be used to directly optimize the graph partitioning objectives, including spectral clustering objective and conversely, spectral methods may be used to optimize the weighted kernel k-means objective. Additionally to this result, it turns out that there also exists a strong link between the (kernel) k-means and the (kernel) PCA and in particular the principal components are the continuous solutions to the discrete cluster membership indicators for the k-means clustering and furthermore the solution for the kernel k-means is given by the kernel PCA. It might be that the strong link between those methods could be adapted to the online setting. In particular the techniques developed for the online PCA algorithm and the online kernel PCA algorithm might be used to develop the online k-means clustering algorithms with approximation guarantees with respect to the k-means clustering objective as well as the online spectral clustering algorithms with approximation guarantees with respect to the spectral clustering objective.

"Simplicity is the highest goal, achievable when you have overcome all difficulties. After one has played a vast quantity of notes and more notes, it is simplicity that emerges as the crowning reward of art."

Fryderyk Chopin

Bibliography

[1] T. Jebara and A. Choromanska. Majorization for CRFs and latent likelihoods. In *NIPS*, 2012.

[2] H. Wallach. Efficient training of conditional random fields. *Master's thesis, University of Edinburgh*, 2002.

[3] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[4] A. Globerson, T. Koo, X. Carreras, and M. Collins. Exponentiated gradient algorithms for log-linear structured prediction. In *ICML*, 2007.

[5] A. Quattoni, S. Wang, L. P. Morency, M. Collins, and T. Darrell. Hidden conditional random fields. *IEEE PAMI*, 29(10):1848–1852, October 2007.

[6] A. Berger. The improved iterative scaling algorithm: A gentle introduction. *Technical report, Carnegie Mellon University*, 1997.

[7] D. Bohning and B. Lindsay. Monotonicity of quadratic approximation algorithms. *Ann. Inst. Statist. Math.*, 40:641–663, 1988.

[8] T. Jebara and A. Pentland. On reversing Jensen's inequality. In *NIPS*, 2000.

[9] T. Jaakkola and M. Jordan. Bayesian parameter estimation via variational methods. *Statistics and Computing*, 10:25–37, 2000.

[10] G. Bouchard. Efficient bounds for the softmax and applications to approximate inference in hybrid models. In *NIPS AIHM Workshop*, 2007.

[11] J. Darroch and D. Ratcliff. Generalized iterative scaling for log-linear models. *Annals of Math. Stat.*, 43:1470–1480, 1972.

[12] J. De Leeuw and W. J. Heiser. Convergence of correction matrix algorithms for multidimensional scaling. *chapter Geometric representations of relational data, Mathesis Press*, pages 735–752, 1977.

[13] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[14] H. Attias. A variational bayesian framework for graphical models. In *NIPS*, 2000.

[15] A. L. Yuille and Anand Rangarajan. The concave-convex procedure. *Neural Comput.*, 15(4):915–936, 2003.

[16] R. Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *CoNLL*, 2002.

[17] F. Sha and F. Pereira. Shallow parsing with conditional random fields. In *NAACL*, 2003.

[18] G. Andrew and J. Gao. Scalable training of ℓ_1-regularized log-linear models. In *ICML*, 2007.

[19] S. Benson and J. More. A limited memory variable metric method for bound constrained optimization. *Technical report, Argonne National Laboratory*, 2001.

[20] Y. Nesterov. *Introductory lectures on convex optimization : a basic course*. Applied optimization. Kluwer Academic Publ., Boston, Dordrecht, London, 2004.

[21] T. J. Ypma. Historical development of the newton-raphson method. *SIAM Rev.*, 37(4):531–551, 1995.

[22] J. Nocedal and S. J. Wright. *Numerical Optimization*. Springer Series in Operations Research, 1999.

[23] C. G. Broyden. The Convergence of a Class of Double-rank Minimization Algorithms: 2. The New Algorithm. *IMA J Appl Math*, 6(3):222–231, 1970.

[24] C. Zhu, R. Byrd, P Lu, and J. Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound-constrained optimization. *TOMS*, 23(4), 1997.

[25] M.R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of Research of the National Bureau of Standards*, 49(6):409–436, 1952.

[26] L. Bottou. Online algorithms and stochastic approximations. In David Saad, editor, *Online Learning and Neural Networks*. Cambridge University Press, Cambridge, UK, 1998.

[27] N. Littlestone. Learning quickly when irrelevant attributes abound: A new linear-threshold algorithm. *Mach. Learn.*, 2(4):285–318, April 1988.

[28] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[29] H. Robbins and S. Munro. A stochastic approximation method. *Ann. Math. Stat.*, 22(400-407), 1951.

[30] L. Bottou and Y. LeCun. Large scale online learning. In *NIPS*, 2003.

[31] N. Le Roux, M. W. Schmidt, and F. Bach. A stochastic gradient method with an exponential convergence rate for finite training sets. In *NIPS*, 2012.

[32] A. Agarwal, P. L. Bartlett, P. D. Ravikumar, and M. J. Wainwright. Information-theoretic lower bounds on the oracle complexity of stochastic

convex optimization. *IEEE Transactions on Information Theory*, (5):3235–3249.

[33] N. N. Schraudolph, J. Yu, and S. Günter. A stochastic quasi-newton method for online convex optimization. In *AISTATS*, 2007.

[34] SVN. Vishwanathan, N. Schraudolph, M. Schmidt, and K. Murphy. Accelerated training of conditional random fields with stochastic gradient methods. In *ICML*, 2006.

[35] Y. Nesterov. Primal-dual subgradient methods for convex problems. *Math. Program.*, 120(1):221–259, 2009.

[36] P. Tseng. An incremental gradient(-projection) method with momentum term and adaptive stepsize rule. *SIAM J. on Optimization*, 8(2):506–531, February 1998.

[37] B. T. Polyak and A. B. Juditsky. Acceleration of stochastic approximation by averaging. *SIAM J. Control Optim.*, 30(4):838–855, July 1992.

[38] S. Shalev-Shwartz and T. Zhang. Proximal stochastic dual coordinate ascent. *CoRR*, abs/1211.2717, 2012.

[39] S. Shalev-Shwartz and T. Zhang. Accelerated mini-batch stochastic dual coordinate ascent. *CoRR*, abs/1305.2581, 2013.

[40] R. Johnson and T. Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *NIPS*. 2013.

[41] C. Wang, X. Chen, A. Smola, and E. Xing. Variance reduction for stochastic gradient optimization. In *NIPS*. 2013.

[42] S. Shalev-Shwartz and T. Zhang. Stochastic dual coordinate ascent methods for regularized loss minimization. *CoRR*, abs/1209.1873, 2012.

[43] J. Mairal. Optimization with first-order surrogate functions. In *ICML*, 2013.

[44] J. Mairal. Stochastic majorization-minimization algorithms for large-scale optimization. In *NIPS*. 2013.

[45] N. N. Schraudolph. Local gain adaptation in stochastic gradient descent. In *ICANN*, 1999.

[46] N. N. Schraudolph. Fast curvature matrix-vector products for second-order gradient descent. *Neural Computation*, 14:2002, 2002.

[47] Y. Le Cun, L. Bottou, G. B. Orr, and K.-R. Müller. Efficient backprop. In *Neural Networks, Tricks of the Trade*, Lecture Notes in Computer Science LNCS 1524. Springer Verlag, 1998.

[48] A. Bordes, L. Bottou, and P. Gallinari. Sgd-qn: Careful quasi-newton stochastic gradient descent. *J. Mach. Learn. Res.*, 10:1737–1754, December 2009.

[49] J. Martens. Deep learning via hessian-free optimization. In *ICML*, 2010.

[50] R. H. Byrd, G. M. Chin, W. Neveitt, and J. Nocedal. On the use of stochastic hessian information in optimization methods for machine learning. *SIAM Journal on Optimization*, 21(3):977–995, 2011.

[51] M. P. Friedlander and M. Schmidt. Hybrid deterministic-stochastic methods for data fitting. *SIAM J. Scientific Computing*, 34(3), 2012.

[52] Q. V. Le, J. Ngiam, A. Coates, A. Lahiri, B. Prochnow, and A. Y. Ng. On optimization methods for deep learning. In *ICML*, 2011.

[53] R. H. Byrd, G. M. Chin, J. Nocedal, and Y. Wu. Sample size selection in optimization methods for machine learning. *Math. Program.*, 134(1):127–155, August 2012.

[54] A. Aravkin, M. P. Friedlander, F. Herrmann, and T. van Leeuwen. Robust inversion, dimensionality reduction, and randomized sampling. *Mathematical Programming*, 134(1):101–125, 2012.

[55] R. Rifkin and A. Klautau. In defense of one-vs-all classification. *J. Mach. Learn. Res.*, 5:101–141, 2004.

[56] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley-Interscience, 2006.

[57] E. L. Allwein, R. E. Schapire, and Y. Singer. Reducing multiclass to binary: A unifying approach for margin classifiers. *J. Mach. Learn. Res.*, 1:113–141, 2001.

[58] T. G. Dietterich and G. Bakiri. Solving multiclass learning problems via error-correcting output codes. *J. Artif. Int. Res.*, 2(1):263–286, 1995.

[59] V. Guruswami and A. Sahai. Multiclass learning, boosting, and error-correcting codes. In *COLT*, 1999.

[60] J. Langford and A. Beygelzimer. Sensitive error correcting output codes. In *COLT*, 2005.

[61] A. Beygelzimer, J. Langford, and P. D. Ravikumar. Error-correcting tournaments. In *ALT*, 2009.

[62] A. Beygelzimer, J. Langford, Y. Lifshits, G. B. Sorkin, and A. L. Strehl. Conditional probability tree estimation analysis and algorithms. In *UAI*, 2009.

[63] S. Bengio, J. Weston, and D. Grangier. Label embedding trees for large multi-class tasks. In *NIPS*, 2010.

[64] J. Deng, S. Satheesh, A. C. Berg, and F.-F. Li. Fast and balanced: Efficient label tree learning for large scale object recognition. In *NIPS*, 2011.

[65] J. Weston, A. Makadia, and H. Yee. Label partitioning for sublinear ranking. In *ICML*, 2013.

[66] R. Agarwal, A. Gupta, Y. Prabhu, and M. Varma. Multi-label learning with millions of labels: Recommending advertiser bid phrases for web pages. In *WWW*, 2013.

[67] M. Kearns and Y. Mansour. On the boosting ability of top-down decision tree learning algorithms. In *In Proceedings of the Twenty-Eighth Annual ACM Symposium on the Theory of Computing*, pages 459–468. ACM Press, 1995.

[68] N. Littlestone and M. K. Warmuth. The weighted majority algorithm. In *Proc. IEEE Symposium on Foundations of Computer Science*, pages 256–261, 1989.

[69] M. Herbster and M. K. Warmuth. Tracking the best expert. *Machine Learning*, 32:151–178, 1998.

[70] C. Monteleoni and T. Jaakkola. Online learning of non-stationary sequences. In *NIPS*, 2003.

[71] N. Cesa-Bianchi and G. Lugosi. *Prediction, learning, and games*. Cambridge University Press, 2006.

[72] D. Aloise, A. Deshpande, P. Hansen, and P. Popat. Np-hardness of euclidean sum-of-squares clustering. *Mach. Learn.*, 75:245–248, 2009.

[73] S. P. Lloyd. Least square quantization in pcm. *Bell Telephone Laboratories Paper*, 1957.

[74] D. Arthur and S. Vassilvitskii. k-means++: the advantages of careful seeding. In *SODA*, 2007.

[75] T. Kanungo, D. M. Mount, N. Netanyahu, C. Piatko, R. Silverman, and A. Y. Wu. A local search approximation algorithm for k-means clustering. *Computational Geometry: Theory and Applications*, 28:89–112, 2004.

[76] N. Ailon, R. Jaiswal, and C. Monteleoni. Streaming k-means approximation. In *NIPS*, 2009.

[77] A. Aggarwal, A. Deshpande, and R. Kannan. Adaptive sampling for k-means clustering. In *APPROX*, 2009.

[78] X. Z. Fern and C. E. Brodley. Solving cluster ensemble problems by bipartite graph partitioning. In *ICML*, 2004.

[79] V. Singh, L. Mukherjee, J. Peng, and J. Xu. Ensemble clustering using semidefinite programming with applications. *Mach. Learn.*, 79:177–200, 2010.

[80] S. Guha, A. Meyerson, N. Mishra, R. Motwani, and L. O'Callaghan. Clustering data streams: Theory and practice. *IEEE Transactions on Knowledge and Data Engineering*, 15(3):515–528, 2003.

[81] M. R. Ackermann, C. Lammersen, M. Märtens, C. Raupach, C. Sohler, and K. Swierkot. Streamkm++: A clustering algorithms for data streams. In *ALENEX*, 2010.

[82] P. Kranen, I. Assent, C. Baldauf, and T. Seidl. The clustree: Indexing micro-clusters for anytime stream mining. In *KAIS*, 2010.

[83] P. Liang and D. Klein. Online EM for unsupervised models. In *NAACL*, 2009.

[84] O. Cappé and E. Moulines. Online EM algorithm for latent data models. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 71:593–613, 2009.

[85] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. *SIAM J. Comput.*, 33(6):1417–1440, 2004.

[86] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *ICML*, 2006.

[87] M. K. Warmuth and D. Kuzmin. Randomized pca algorithms with regret bounds that are logarithmic in the dimension. In *NIPS*, 2007.

[88] S. Dasgupta. Course notes, CSE 291: Topics in unsupervised learning. Lecture 6: Clustering in an online/streaming setting. Section 6.2.3. In *http://www-cse.ucsd.edu/∼dasgupta/291/lec6.pdf, University of California, San Diego, Spring Quarter*, 2008.

[89] T. Jebara. Multitask sparsity via maximum entropy discrimination. *JMLR*, 12:75–110, 2011.

[90] A. Yuille and A. Rangarajan. The concave-convex procedure (cccp). In *NIPS*, 2002.

[91] B. Taskar, C. Guestrin, and D. Koller. Max margin Markov networks. In *NIPS*, 2004.

[92] Y. Qi, M. Szummer, and T. P. Minka. Bayesian conditional random fields. In *AISTATS*, 2005.

[93] T. Bromwich and T. MacRobert. *An Introduction to the Theory of Infinite Series*. Chelsea, 1991.

[94] R. Salakhutdinov and S.T. Roweis. Adaptive overrelaxed bound optimization methods. In *ICML*, 2003.

[95] S. B. Wang, A. Quattoni, L.-P. Morency, and D. Demirdjian. Hidden conditional random fields for gesture recognition. In *CVPR*, 2006.

[96] Y. Wang and G. Mori. Max-margin hidden conditional random fields for human action recognition. In *CVPR*, pages 872–879. IEEE, 2009.

[97] M. Wainwright and M Jordan. Graphical models, exponential families and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[98] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. *Optimization for Machine Learning*, chapter Convex optimization with sparsity-inducing norms. MIT Press, 2011.

[99] Y. Altun, I. Tsochantaridis, and T. Hofmann. Hidden Markov support vector machines. In *ICML*, 2003.

[100] A. Y. Aravkin, A. Choromanska, T. Jebara, and D. Kanevsky. Semistochastic quadratic bound methods. In *ICLR Workshop*, 2014.

[101] D. P. Bertsekas and J. N. Tsitsiklis. Gradient convergence in gradient methods with errors. *SIAM J. on Optimization*, 10(3):627–642, July 1999.

[102] D.P. Bertsekas and J.N. Tsitsiklis. *Neuro-dynamic programming*. Athena Scientific, 1996.

[103] A. Nemirovski, A. Juditsky, G. Lan, and A. Shapiro. Robust stochastic approximation approach to stochastic programming. *SIAM J. on Optimization*, 19(4):1574–1609, January 2009.

[104] J. Langford A. Choromanska, A. Agarwal. Extreme multi class classification. In *NIPS Workshop: eXtreme Classification, submitted*, 2013.

[105] J. Langford, L. Li, and A. Srehl. Vowpal wabbit program, http://hunch.net/ vw. 2007.

[106] A. Choromanska and C. Monteleoni. Online clustering with experts. In *ICML 2011 Workshop on Online Trading of Exploration and Exploitation 2, Journal of Machine Learning Research (JMLR) Workshop and Conference Proceedings*, 2011.

[107] A. Choromanska and C. Monteleoni. Online clustering with experts. In *AISTATS*, 2012.

[108] David Haussler, Jyrki Kivinen, and Manfred K. Warmuth. Sequential prediction of individual sequences under general loss functions. *IEEE Trans. on Information Theory*, 44(5):1906–1925, 1998.

[109] V. G. Vovk. A game of prediction with expert advice. *J. Comput. Syst. Sci.*, 56:153–173, April 1998.

[110] Claire E. Monteleoni. Online learning of non-stationary sequences. SM Thesis. In *MIT Artificial Intelligence Technical Report 2003-011*, 2003.

[111] A. Asuncion and D.J. Newman. UCI machine learning repository, University of California, Irvine, School of Information and Computer Sciences, 2007.

[112] A. Blum, A. Kalai, and J. Langford. Beating the hold-out: Bounds for k-fold and progressive cross-validation. In *COLT*, 1999.

[113] A. Choromanska, D. Kanevsky, and T. Jebara. Majorization for deep belief networks. In *NIPS Workshop: Log-linear models*, 2012.

[114] A. Choromanska and T. Jebara. Stochastic bound majorization. In *Arxiv*, 2013.

[115] A. Choromanska, T. Jebara, H. Kim, M. Mohan, and C. Monteleoni. Fast spectral clustering via the nyström method. In *ALT*, 2013.

7 Appendix

7.1 Appendix A

Lemma 12. *(See [89] p. 100)*

For all $\mathbf{u} \in \mathbb{R}^d$*, any* $\mathbf{v} \in \mathbb{R}^d$ *and any* $\gamma \geq 0$*, the bound* $\log\left(\exp\left(\frac{1}{2}\|\mathbf{u}\|^2\right) + \gamma\right) \leq$

$$\log\left(\exp\left(\tfrac{1}{2}\|\mathbf{v}\|^2\right) + \gamma\right) + \frac{\mathbf{v}^\top(\mathbf{u}-\mathbf{v})}{1+\gamma\exp(-\frac{1}{2}\|\mathbf{v}\|^2)} + \frac{1}{2}(\mathbf{u}-\mathbf{v})^\top\left(I + \Gamma\mathbf{v}\mathbf{v}^\top\right)(\mathbf{u}-\mathbf{v})$$

holds when the scalar term $\Gamma = \frac{\tanh(\frac{1}{2}\log(\gamma\exp(-\|\mathbf{v}\|^2/2)))}{2\log(\gamma\exp(-\|\mathbf{v}\|^2/2))}$. *Equality is achieved when* $\mathbf{u} = \mathbf{v}$.

Proof. The proof is provided in [89]. □

Lemma 13. *If* $\kappa\boldsymbol{\Psi} \succeq \boldsymbol{\Phi} \succ \mathbf{0}$ *for* $\boldsymbol{\Phi}, \boldsymbol{\Psi} \in \mathbb{R}^{d\times d}$, *then*

$$L(\boldsymbol{\theta}) = -\tfrac{1}{2}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}})^\top\boldsymbol{\Phi}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}}) - (\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}})^\top\boldsymbol{\mu}$$
$$U(\boldsymbol{\theta}) = -\tfrac{1}{2}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}})^\top\boldsymbol{\Psi}(\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}}) - (\boldsymbol{\theta}-\tilde{\boldsymbol{\theta}})^\top\boldsymbol{\mu}$$

satisfy $\sup_{\boldsymbol{\theta}\in\boldsymbol{\Lambda}} L(\boldsymbol{\theta}) \geq \frac{1}{\kappa}\sup_{\boldsymbol{\theta}\in\boldsymbol{\Lambda}} U(\boldsymbol{\theta})$ *for any convex* $\boldsymbol{\Lambda} \subseteq \mathbb{R}^d$, $\tilde{\boldsymbol{\theta}} \in \boldsymbol{\Lambda}$, $\boldsymbol{\mu} \in \mathbb{R}^d$ *and* $\kappa \in \mathbb{R}^+$.

Proof of Lemma 13 Define the primal problems of interest as $\mathbf{P}_L = \sup_{\boldsymbol{\theta}\in\boldsymbol{\Lambda}} L(\boldsymbol{\theta})$ and $\mathbf{P}_U = \sup_{\boldsymbol{\theta}\in\boldsymbol{\Lambda}} U(\boldsymbol{\theta})$. The constraints $\boldsymbol{\theta} \in \boldsymbol{\Lambda}$ can be summarized by a set of linear inequalities $\mathbf{A}\boldsymbol{\theta} \leq \mathbf{b}$ where $\mathbf{A} \in \mathbb{R}^{k\times d}$ and $\mathbf{b} \in \mathbb{R}^k$ for some (possibly infinite) $k \in \mathbb{Z}$. Apply the change of variables $\mathbf{z} = \boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}$. The constraint $\mathbf{A}(\mathbf{z}+\tilde{\boldsymbol{\theta}}) \leq \mathbf{b}$ simplifies into $\mathbf{A}\mathbf{z} \leq \tilde{\mathbf{b}}$ where $\tilde{\mathbf{b}} = \mathbf{b} - \mathbf{A}\tilde{\boldsymbol{\theta}}$. Since $\tilde{\boldsymbol{\theta}} \in \boldsymbol{\Lambda}$, it is easy to show that $\tilde{\mathbf{b}} \geq \mathbf{0}$. We obtain the equivalent primal problems $\mathbf{P}_L = \sup_{\mathbf{A}\mathbf{z}\leq\tilde{\mathbf{b}}} -\frac{1}{2}\mathbf{z}^\top\boldsymbol{\Phi}\mathbf{z} - \mathbf{z}^\top\boldsymbol{\mu}$ and $\mathbf{P}_U = \sup_{\mathbf{A}\mathbf{z}\leq\tilde{\mathbf{b}}} -\frac{1}{2}\mathbf{z}^\top\boldsymbol{\Psi}\mathbf{z} - \mathbf{z}^\top\boldsymbol{\mu}$. The corresponding dual problems are

$$\mathbf{D}_L = \inf_{\mathbf{y}\geq\mathbf{0}} \frac{\mathbf{y}^\top\mathbf{A}\boldsymbol{\Phi}^{-1}\mathbf{A}^\top\mathbf{y}}{2} + \mathbf{y}^\top\mathbf{A}\boldsymbol{\Phi}^{-1}\boldsymbol{\mu} + \mathbf{y}^\top\tilde{\mathbf{b}} + \frac{\boldsymbol{\mu}^\top\boldsymbol{\Phi}^{-1}\boldsymbol{\mu}}{2}$$
$$\mathbf{D}_U = \inf_{\mathbf{y}\geq\mathbf{0}} \frac{\mathbf{y}^\top\mathbf{A}\boldsymbol{\Psi}^{-1}\mathbf{A}^\top\mathbf{y}}{2} + \mathbf{y}^\top\mathbf{A}\boldsymbol{\Psi}^{-1}\boldsymbol{\mu} + \mathbf{y}^\top\tilde{\mathbf{b}} + \frac{\boldsymbol{\mu}^\top\boldsymbol{\Psi}^{-1}\boldsymbol{\mu}}{2}.$$

Due to strong duality, $\mathbf{P}_L = \mathbf{D}_L$ and $\mathbf{P}_U = \mathbf{D}_U$. Apply the inequalities $\boldsymbol{\Phi} \preceq \kappa\boldsymbol{\Psi}$ and $\mathbf{y}^\top\tilde{\mathbf{b}} > 0$ as

$$\begin{aligned}\mathbf{P}_L \geq & \sup_{\mathbf{Az}\leq\tilde{\mathbf{b}}} -\frac{\kappa}{2}\mathbf{z}^\top\boldsymbol{\Psi}\mathbf{z} - \mathbf{z}^\top\boldsymbol{\mu} = \inf_{\mathbf{y}\geq\mathbf{0}} \frac{\mathbf{y}^\top\mathbf{A}\boldsymbol{\Psi}^{-1}\mathbf{A}^\top\mathbf{y}}{2\kappa} + \frac{\mathbf{y}^\top\mathbf{A}\boldsymbol{\Psi}^{-1}\boldsymbol{\mu}}{\kappa} + \mathbf{y}^\top\tilde{\mathbf{b}} + \frac{\boldsymbol{\mu}^\top\boldsymbol{\Psi}^{-1}\boldsymbol{\mu}}{2\kappa} \\ = & \frac{1}{\kappa}\mathbf{D}_U = \frac{1}{\kappa}\mathbf{P}_U.\end{aligned}$$

This proves that $\mathbf{P}_L \geq \frac{1}{\kappa}\mathbf{P}_U$.

Proof of correctness of Algorithm 3

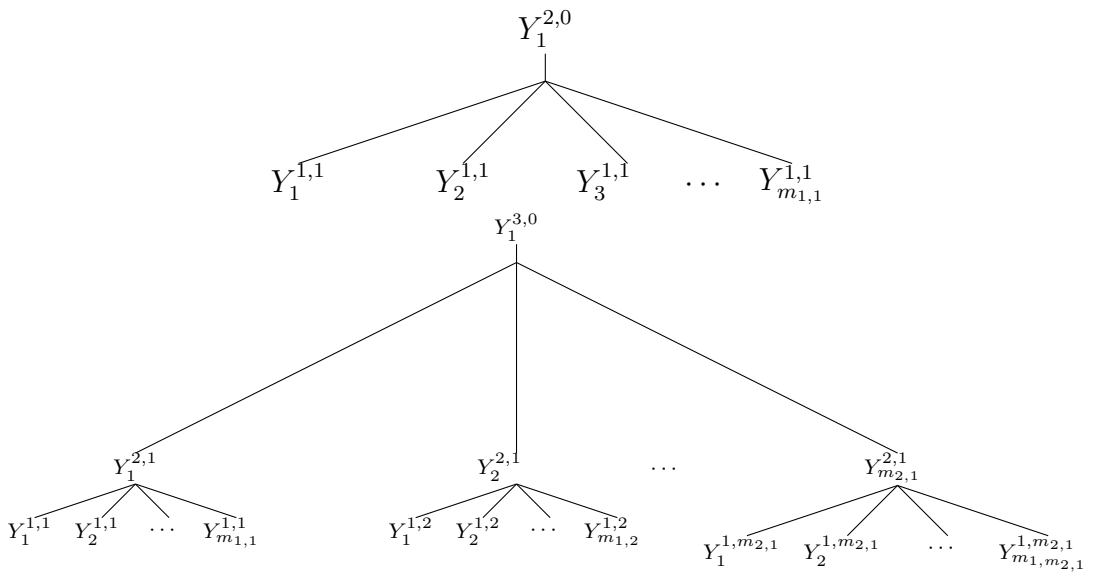


Figure 7.1: Junction tree of depth a) 2 and b) 3.

Consider a simple junction tree of depth 2 shown on Figure 7.1a. The notation $Y_c^{a,b}$ refers to the c^{th} tree node located at tree level a (first level is considered as the one with tree leaves) whose parent is the b^{th} from the higher tree level (the root has no parent so $b = 0$). Let $\sum_{Y_{c_1}^{a_1,b_1}}$ refer to the sum over all configurations of variables in

$Y_{c_1}^{a_1,b_1}$ and $\sum_{Y_{c_1}^{a_1,b_1} \setminus Y_{c_2}^{a_2,b_2}}$ refers to the sum over all configurations of variables that are in $Y_{c_1}^{a_1,b_1}$ but not in $Y_{c_2}^{a_2,b_2}$. Let $m_{a,b}$ denote the number of children of the b^{th} node located at tree level $a+1$. For short-hand, use $\psi(Y) = h(Y)\exp(\boldsymbol{\theta}^\top f(Y))$. The partition function can be expressed as:

$$
\begin{aligned}
Z(\boldsymbol{\theta}) &= \sum_{u \in Y_1^{2,0}} \left[\psi(u) \prod_{i=1}^{m_{1,1}} \left(\sum_{v \in Y_i^{1,1} \setminus Y_1^{2,0}} \psi(v) \right) \right] \\
&\leq \sum_{u \in Y_1^{2,0}} \left[\psi(u) \prod_{i=1}^{m_{1,1}} z_i \exp(\frac{1}{2} \left(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}} \right)^\top \boldsymbol{\Sigma}_i (\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}) + (\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^\top \boldsymbol{\mu}_i \Big) \right] \\
&= \sum_{u \in Y_1^{2,0}} \left[h(u) \exp(\boldsymbol{\theta}^\top f(u)) \prod_{i=1}^{m_{1,1}} z_i \exp\left(\frac{1}{2} (\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^\top \boldsymbol{\Sigma}_i (\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}) + (\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^\top \boldsymbol{\mu}_i \right) \right],
\end{aligned}
$$

where the upper-bound is obtained by applying Theorem 1 to each of the terms $\sum_{v \in Y_i^{1,1} \setminus Y_1^{2,0}} \psi(v)$. By simply rearranging terms we get:

$$
\begin{aligned}
Z(\boldsymbol{\theta}) \leq \sum_{u \in Y_1^{2,0}} \Bigg[h(u) \left(\prod_{i=1}^{m_{1,1}} z_i \exp(-\tilde{\boldsymbol{\theta}}^\top \boldsymbol{\mu}_i) \right) \exp\left(\boldsymbol{\theta}^\top \left(f(u) + \sum_{i=1}^{m_{1,1}} \boldsymbol{\mu}_i \right) \right) \\
\cdot \exp\left(\frac{1}{2} (\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^\top \left(\sum_{i=1}^{m_{1,1}} \boldsymbol{\Sigma}_i \right) (\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}) \right) \Bigg].
\end{aligned}
$$

One can prove that this expression can be upper-bounded by an expression of the form $z \exp\left(\frac{1}{2} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \boldsymbol{\Sigma} (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) + (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \boldsymbol{\mu} \right)$ where z, $\boldsymbol{\Sigma}$ and $\boldsymbol{\mu}$ can be computed using Algorithm 8 (a simplification of Algorithm 3). We will call this result Lemma L3. The proof is similar to the proof of Theorem 1 so is not repeated here.

Algorithm 8 SmallJunctionTree

Input Parameters $\tilde{\boldsymbol{\theta}}$ and $h(u), f(u)\,\forall u \in Y_1^{2,0}$ and $z_i, \boldsymbol{\Sigma}_i, \boldsymbol{\mu}_i\,\forall i = 1, \ldots, m_{1,1}$

Initialize $z \to 0^+, \boldsymbol{\mu} = \mathbf{0}, \boldsymbol{\Sigma} = z\mathbf{I}$

For each configuration $u \in Y_1^{2,0}$ {

$\alpha \quad = h(u)(\prod_{i=1}^{m_{1,1}} z_i \exp(-\tilde{\boldsymbol{\theta}}^\top \boldsymbol{\mu}_i)) \exp(\tilde{\boldsymbol{\theta}}^\top (f(u) + \sum_{i=1}^{m_{1,1}} \boldsymbol{\mu}_i))$

$\quad\;\; = h(u) \exp(\tilde{\boldsymbol{\theta}}^\top f(u)) \prod_{i=1}^{m_{1,1}} z_i$

$\mathbf{l} \quad = f(u) + \sum_{i=1}^{m_{1,1}} \boldsymbol{\mu}_i - \boldsymbol{\mu}$

$\boldsymbol{\Sigma} += \sum_{i=1}^{m_{1,1}} \boldsymbol{\Sigma}_i + \frac{\tanh(\frac{1}{2}\log(\alpha/z))}{2\log(\alpha/z)} \mathbf{l}\mathbf{l}^\top$

$\boldsymbol{\mu} += \frac{\alpha}{z+\alpha}\mathbf{l}$

$z \;\; += \alpha \quad \}$

Output $z, \boldsymbol{\mu}, \boldsymbol{\Sigma}$

Consider enlarging the tree to a depth 3 as shown on Figure 7.1b. The partition function is now

$$Z(\boldsymbol{\theta}) = \sum_{u \in Y_1^{3,0}} \left[\psi(u) \prod_{i=1}^{m_{2,1}} \left(\sum_{v \in Y_i^{2,1} \setminus Y_1^{3,0}} \left(\psi(v) \prod_{j=1}^{m_{1,i}} \left(\sum_{w \in Y_j^{1,i} \setminus Y_i^{2,1}} \psi(w) \right) \right) \right) \right].$$

We can upper-bound each $\sum_{v \in Y_i^{2,1} \setminus Y_1^{3,0}} \left(\psi(v) \prod_{j=1}^{m_{1,i}} \left(\sum_{w \in Y_j^{1,i} \setminus Y_i^{2,1}} \psi(w) \right) \right)$ term by the expression $z_i \exp\left(\frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \boldsymbol{\Sigma}_i (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) + (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \boldsymbol{\mu}_i \right)$ using Lemma L3. This yields

$$Z(\boldsymbol{\theta}) \leq \sum_{u \in Y_1^{3,0}} \left[\psi(u) \prod_{i=1}^{m_{2,1}} z_i \exp\left(\frac{1}{2}(\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^\top \boldsymbol{\Sigma}_i (\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}}) + (\boldsymbol{\theta} - \tilde{\boldsymbol{\theta}})^\top \boldsymbol{\mu}_i \right) \right].$$

This process can be viewed as collapsing the sub-trees $S_1^{2,1}$, $S_2^{2,1}$, ..., $S_{m_{2,1}}^{2,1}$ to super-nodes that are represented by bound parameters, z_i, $\boldsymbol{\Sigma}_i$ and $\boldsymbol{\mu}_i, i = \{1, 2, \cdots, m_{2,1}\}$,

where the sub-trees are defined as:

$$\begin{aligned} S_1^{2,1} &= \{Y_1^{2,1}, Y_1^{1,1}, Y_2^{1,1}, Y_3^{1,1}, \ldots, Y_{m_{1,1}}^{1,1}\} \\ S_2^{2,1} &= \{Y_2^{2,1}, Y_1^{1,2}, Y_2^{1,2}, Y_3^{1,2}, \ldots, Y_{m_{1,2}}^{1,2}\} \\ &\vdots \\ S_{m_{2,1}}^{2,1} &= \{Y_{m_{2,1}}^{2,1}, Y_1^{1,m_{2,1}}, Y_2^{1,m_{2,1}}, Y_3^{1,m_{2,1}}, \ldots, Y_{m_{1,m_{2,1}}}^{1,m_{2,1}}\}. \end{aligned}$$

Notice that the obtained expression can be further upper-bounded using again Lemma L3 by an expression of the form: $z \exp\left(\frac{1}{2}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \boldsymbol{\Sigma}(\boldsymbol{\theta} - \hat{\boldsymbol{\theta}}) + (\boldsymbol{\theta} - \hat{\boldsymbol{\theta}})^\top \boldsymbol{\mu}\right)$.

Finally, for a general tree, follow the same steps described above, starting from leaves and collapsing nodes to super-nodes, each represented by bound parameters. This procedure effectively yields Algorithm 3.

Proof of correctness of Algorithm 4

We begin by proving a lemma that will be useful later.

Lemma 14. *For all* $\mathbf{x} \in \mathbb{R}^d$ *and for all* $\mathbf{l} \in \mathbb{R}^d$,

$$\sum_{i=1}^{d} \mathbf{x}(i)^2 \mathbf{l}(i)^2 \geq \left(\sum_{i=1}^{d} \mathbf{x}(i) \frac{\mathbf{l}(i)^2}{\sqrt{\sum_{j=1}^{d} \mathbf{l}(j)^2}} \right)^2.$$

Proof of Lemma 14 By Jensen's inequality,

$$\sum_{i=1}^{d} \mathbf{x}(i)^2 \frac{\mathbf{l}(i)^2}{\sum_{j=1}^{d} \mathbf{l}(j)^2} \geq \left(\sum_{i=1}^{d} \frac{\mathbf{x}(i)\mathbf{l}(i)^2}{\sum_{j=1}^{d} \mathbf{l}(j)^2} \right)^2 \Longleftrightarrow \sum_{i=1}^{d} \mathbf{x}(i)^2 l(i)^2 \geq \left(\sum_{i=1}^{d} \frac{\mathbf{x}(i)\mathbf{l}(i)^2}{\sqrt{\sum_{j=1}^{d} \mathbf{l}(j)^2}} \right)^2 .$$

Now we prove the correctness of Algorithm 4. At the i^{th} iteration, the algorithm stores $\mathbf{\Sigma}_i$ using a low-rank representation $\mathbf{V}_i^\top \mathbf{S}_i \mathbf{V}_i + \mathbf{D}_i$ where $\mathbf{V}_i \in \mathbb{R}^{k \times d}$ is orthonormal, $\mathbf{S}_i \in \mathbb{R}^{k \times k}$ positive semi-definite and $\mathbf{D}_i \in \mathbb{R}^{d \times d}$ is non-negative diagonal. The diagonal terms $\mathbf{D}$ are initialized to $t\lambda\mathbf{I}$ where λ is the regularization term. To mimic Algorithm 1 we must increment the $\mathbf{\Sigma}$ matrix by a rank one update of the form $\mathbf{\Sigma}_i = \mathbf{\Sigma}_{i-1} + \mathbf{r}_i \mathbf{r}_i^\top$. By projecting $\mathbf{r}_i$ onto each eigenvector in $\mathbf{V}$, we can decompose it as $\mathbf{r}_i = \sum_{j=1}^{k} \mathbf{V}_{i-1}(j, \cdot)\mathbf{r}_i \mathbf{V}_{i-1}(j, \cdot)^\top + \mathbf{g} = \mathbf{V}_{i-1}^\top \mathbf{V}_{i-1}\mathbf{r}_i + \mathbf{g}$ where $\mathbf{g}$ is the remaining residue. Thus the update rule can be rewritten as:

$$\mathbf{\Sigma}_i = \mathbf{\Sigma}_{i-1} + \mathbf{r}_i \mathbf{r}_i^\top = \mathbf{V}_{i-1}^\top \mathbf{S}_{i-1} \mathbf{V}_{i-1} + \mathbf{D}_{i-1} + (\mathbf{V}_{i-1}^\top \mathbf{V}_{i-1}\mathbf{r}_i + \mathbf{g})(\mathbf{V}_{i-1}^\top \mathbf{V}_{i-1}\mathbf{r}_i + \mathbf{g})^\top$$

$$= \mathbf{V}_{i-1}^\top (\mathbf{S}_{i-1} + \mathbf{V}_{i-1}\mathbf{r}_i \mathbf{r}_i^\top \mathbf{V}_{i-1}^\top)\mathbf{V}_{i-1} + \mathbf{D}_{i-1} + \mathbf{g}\mathbf{g}^\top = \mathbf{V}_{i-1}'^\top \mathbf{S}_{i-1}' \mathbf{V}_{i-1}' + \mathbf{g}\mathbf{g}^\top + \mathbf{D}_{i-1}$$

where we define $\mathbf{V}_{i-1}' = \mathbf{Q}_{i-1}\mathbf{V}_{i-1}$ and defined $\mathbf{Q}_{i-1}$ in terms of the singular value decomposition, $\mathbf{Q}_{i-1}^\top \mathbf{S}_{i-1}' \mathbf{Q}_{i-1} = \text{svd}(\mathbf{S}_{i-1} + \mathbf{V}_{i-1}\mathbf{r}_i \mathbf{r}_i^\top \mathbf{V}_{i-1}^\top)$. Note that $\mathbf{S}_{i-1}'$ is diagonal and nonnegative by construction. The current formula for $\mathbf{\Sigma}_i$ shows that we have a rank $(k+1)$ system (plus diagonal term) which needs to be converted back to a rank k system (plus diagonal term) which we denote by $\mathbf{\Sigma}_i'$. We have two

options as follows.

Case 1) Remove $\mathbf{g}$ from $\mathbf{\Sigma}_i$ to obtain

$$\mathbf{\Sigma}_i^{'} = \mathbf{V}_{i-1}^{'\top}\mathbf{S}_{i-1}^{'}\mathbf{V}_{i-1}^{'} + \mathbf{D}_{i-1} = \mathbf{\Sigma}_i - \mathbf{g}\mathbf{g}^\top = \mathbf{\Sigma}_i - c\mathbf{v}\mathbf{v}^\top$$

where $c = \|\mathbf{g}\|^2$ and $\mathbf{v} = \frac{1}{\|\mathbf{g}\|}\mathbf{g}$.

Case 2) Remove the m^{th} (smallest) eigenvalue in $\mathbf{S}_{i-1}^{'}$ and its corresponding eigenvector:

$$\mathbf{\Sigma}_i^{'} \;=\; \mathbf{V}_{i-1}^{'\top}\mathbf{S}_{i-1}^{'}\mathbf{V}_{i-1}^{'} + \mathbf{D}_{i-1} + \mathbf{g}\mathbf{g}^\top - \mathbf{S}^{'}(m,m)\mathbf{V}^{'}(m,\cdot)^\top\mathbf{V}^{'}(m,\cdot) = \mathbf{\Sigma}_i - c\mathbf{v}\mathbf{v}^\top$$

where $c = \mathbf{S}^{'}(m,m)$ and $\mathbf{v} = \mathbf{V}(m,\cdot)^{'}$.

Clearly, both cases can be written as an update of the form $\mathbf{\Sigma}_i^{'} = \mathbf{\Sigma}_i + c\mathbf{v}\mathbf{v}^\top$ where $c \geq 0$ and $\mathbf{v}^\top\mathbf{v} = 1$. We choose the case with smaller c value to minimize the change as we drop from a system of order $(k+1)$ to order k. Discarding the smallest singular value and its corresponding eigenvector would violate the bound. Instead, consider absorbing this term into the diagonal component to preserve the bound. Formally, we look for a diagonal matrix $\mathbf{F}$ such that $\mathbf{\Sigma}_i^{''} = \mathbf{\Sigma}_i^{'} + \mathbf{F}$ which also maintains $\mathbf{x}^\top\mathbf{\Sigma}_i^{''}\mathbf{x} \geq \mathbf{x}^\top\mathbf{\Sigma}_i\mathbf{x}$ for all $\mathbf{x} \in \mathbb{R}^d$. Thus, we want to satisfy:

$$\mathbf{x}^\top\mathbf{\Sigma}_i^{''}\mathbf{x} \geq \mathbf{x}^\top\mathbf{\Sigma}_i\mathbf{x} \iff \mathbf{x}^\top c\mathbf{v}\mathbf{v}^\top\mathbf{x} \leq \mathbf{x}^\top\mathbf{F}\mathbf{x} \iff c\left(\sum_{i=1}^{d}\mathbf{x}(i)\mathbf{v}(i)\right)^2 \leq \sum_{i=1}^{d}\mathbf{x}(i)^2\mathbf{F}(i)$$

where, for ease of notation, we take $\mathbf{F}(i) = \mathbf{F}(i, i)$.

Define $\mathbf{v}^{'} = \frac{1}{w}\mathbf{v}$ where $w = \mathbf{v}^{\top}\mathbf{1}$. Consider the case where $\mathbf{v} \geq \mathbf{0}$ though we will soon get rid of this assumption. We need an $\mathbf{F}$ such that $\sum_{i=1}^{d} \mathbf{x}(i)^2\mathbf{F}(i) \geq c\left(\sum_{i=1}^{d} \mathbf{x}(i)\mathbf{v}(i)\right)^2$. Equivalently, we need $\sum_{i=1}^{d} \mathbf{x}(i)^2\frac{\mathbf{F}(i)}{cw^2} \geq \left(\sum_{i=1}^{d} \mathbf{x}(i)\mathbf{v}(i)^{'}\right)^2$. Define $\mathbf{F}(i)^{'} = \frac{\mathbf{F}(i)}{cw^2}$ for all $i = 1, \ldots, d$. So, we need an $\mathbf{F}^{'}$ such that $\sum_{i=1}^{d} \mathbf{x}(i)^2\mathbf{F}(i)^{'} \geq \left(\sum_{i=1}^{d} \mathbf{x}(i)\mathbf{v}(i)^{'}\right)^2$. Using Lemma 14 it is easy to show that we may choose $\mathbf{F}^{'}(i) = \mathbf{v}(i)^{'}$. Thus, we obtain $\mathbf{F}(i) = cw^2\mathbf{F}(i)^{'} = cw\mathbf{v}(i)$. Therefore, for all $\mathbf{x} \in \mathbb{R}^d$, all $\mathbf{v} \geq \mathbf{0}$, and for $\mathbf{F}(i) = c\mathbf{v}(i)\sum_{j=1}^{d} \mathbf{v}(j)$ we have

$$\sum_{i=1}^{d} \mathbf{x}(i)^2\mathbf{F}(i) \geq c\left(\sum_{i=1}^{d} \mathbf{x}(i)\mathbf{v}(i)\right)^2. \tag{7.1}$$

To generalize the inequality to hold for all vectors $\mathbf{v} \in \mathbb{R}^d$ with potentially negative entries, it is sufficient to set $\mathbf{F}(i) = c|\mathbf{v}(i)|\sum_{j=1}^{d} |\mathbf{v}(j)|$. To verify this, consider flipping the sign of any $\mathbf{v}(i)$. The left side of the Inequality 7.1 does not change. For the right side of this inequality, flipping the sign of $\mathbf{v}(i)$ is equivalent to flipping the sign of $\mathbf{x}(i)$ and not changing the sign of $\mathbf{v}(i)$. However, in this case the inequality holds as shown before (it holds for any $\mathbf{x} \in \mathbb{R}^d$). Thus for all $\mathbf{x}, \mathbf{v} \in \mathbb{R}^d$ and for $\mathbf{F}(i) = c|\mathbf{v}(i)|\sum_{j=1}^{d} |\mathbf{v}(j)|$, Inequality 7.1 holds.

7.2 Appendix B

We consider the the special case when $k = 2$. We will show that the worst case value of the balancing factor p w.r.t. the change in the potential is a balanced p, in particular we will prove the lemma which states that in the worst case setting when Δ_t is minimized, the value of p has to lie in the interval $[0.4, 0.6]$.

Recall that $\pi(n_0)(1-p) + \pi(n_1)p = \pi_1$ and thus $\pi(n_0) = \frac{\pi_1 - \pi(n_1)p}{1-p}$. Therefore we can write that:

$$\pi(n_0) - \pi(n_1) = \left[\frac{\pi_1 - \pi(n_1)p}{1-p} - \pi(n_1)\right] = \frac{\pi_1}{1-p} - \frac{\pi(n_1)}{1-p} = \frac{\pi_1}{1-p} - \frac{\pi_1 P_1}{p(1-p)},$$

where the last equality comes from the fact that $\pi(n_1) = \frac{\pi_1 P_1}{p}$. Let $\delta = \pi(n_0) - \pi(n_1)$ and thus

$$p(1-p)\delta = p\pi_1 - P_1\pi_1 = \pi_1(p - P_1) \geq \gamma\pi_1 \geq \gamma\pi_1(1-\pi_1),$$

where the first inequality comes from the Weak Learning Assumption. Thus we obtained that $p(1-p)\delta \geq \gamma\pi_1(1-\pi_1)$. One can compare this result with Lemma 2 from [67] and observe that as expected we obtained similar condition. Now recall that

$$\Delta_t = G(\pi_1) - (1-p)G(\pi(n_0)) - pG(\pi(n_1)),$$

where without loss of generality weight w was assumed to be $w = 1$. Notice that $\pi(n_0) = \pi_1 - p\delta$ and $\pi(n_1) = \pi_1 + (1-p)\delta$ (it can be easily verified by substituting $\pi_1 = \pi(n_0)(1-p) + \pi(n_1)p$). We can further express Δ_t as a function of π_1, p, δ as follows

$$\Delta_t(\pi_1, p, \delta) = G(\pi_1) - (1-p)G(\pi_1 - p\delta) - pG(\pi_1 + (1-p)\delta).$$

For any fixed values $\pi_1, p \in [0,1]$, $\Delta_t(\pi_1, p, \delta)$ is minimized by choosing δ as small as possible. Thus let us set $\delta = \frac{\gamma\pi_1(1-\pi_1)}{p(1-p)}$ and define

$$\Delta_t(\pi_1, p) = \Delta_t(\pi_1, p, \frac{\gamma\pi_1(1-\pi_1)}{p(1-p)})$$

$$= G(\pi_1) - (1-p)G\left(\pi_1 - \frac{\gamma\pi_1(1-\pi_1)}{(1-p)}\right) - pG\left(\pi_1 + \frac{\gamma\pi_1(1-\pi_1)}{p}\right). \tag{7.2}$$

The next Lemma is a direct application of Lemma 4 from [67].

Lemma 15. *Let $\gamma \in [0,1]$ be any fixed value, and let $\Delta_t(\pi_1, p)$ be as defined in Equation 7.2. Then for any fixed $\pi_1 \in [0,1]$, $\Delta_t(\pi_1, p)$ is minimized by a value of p falling in the interval $[0.4, 0.6]$.*

That implies that in the worst case setting when Δ_t is minimized the split has to be close to balanced.

7.3 Appendix C

First we provide some lemmas that will be used in subsequent proofs. These follow the approach in [110]. We use the short-hand notation $L(i,t)$ for $L(x_t, c_t^i)$, which is valid since our loss is symmetric with respect to its arguments.

Lemma 16.

$$-\sum_{t=1}^{T} \log \sum_{i=1}^{n} p_t(i) e^{-\frac{1}{2}L(i,t)}$$

$$= -\log[\sum_{i_1,...,i_T} p_1(i_1) e^{-\frac{1}{2}L(i_1,1)} \prod_{t=2}^{T} e^{-\frac{1}{2}L(i_t,t)} P(i_t|i_{t-1},\Theta)]$$

Proof. First note that following [70], we design the HMM such that we equate our loss function, $L(i,t)$, with the negative log-likelihood of the observation given that expert i is the current value of the hidden variable. In the unsupervised setting, the observation is x_t. Thus $L(i,t) = -\log P(x_t|a_i, x_1, ..., x_{t-1})$. Therefore, we can expand the left hand side of the claim as follows.

$$-\sum_{t=1}^{T} \log \sum_{i=1}^{n} p_t(i) e^{-\frac{1}{2}L(i,t)} = -\sum_{t=1}^{T} \log \sum_{i=1}^{n} p_t(i) P(x_t|a_i, x_1, ..., x_{t-1})$$

$$= -\sum_{t=1}^{T} \log P(x_t|x_1, ..., x_{t-1}) = -\log p_1(x_1) \prod_{t=2}^{T} P(x_t|x_1, ..., x_{t-1})$$

$$= -\log P(x_1, ..., x_T) = -\log[\sum_{i_1,...,i_T} P(x_1, ..., x_T|i_1, ..., i_T) P(i_1, ..., i_T|\Theta)]$$

$$= -\log[\sum_{i_1,...,i_T} p_1(i_1) P(x_1|i_1, ..., i_T) \prod_{t=2}^{T} P(x_t|i_1, ..., i_T, x_1, ..., x_{t-1}) P(i_t|i_1, ..., i_{t-1}, \Theta)]$$

$$= -\log[\sum_{i_1,...,i_T} p_1(i_1)P(x_1|i_1)\prod_{t=2}^{T} P(x_t|i_t, x_1, ..., x_{t-1})P(i_t|i_{t-1}, \Theta)]$$
$$= -\log[\sum_{i_1,...,i_T} p_1(i_1)e^{-\frac{1}{2}L(i_1,1)}\prod_{t=2}^{T} e^{-\frac{1}{2}L(i_t,t)}P(i_t|i_{t-1}, \Theta)]$$

□

Lemma 17.

$$\sum_{i=1}^{n} \rho_i^* D(\Theta_i^* \| \Theta_i) = D(\alpha^* \| \alpha)$$

when $\theta_{ij} = 1 - \alpha$ *for* $i = j$ *and* $\theta_{ij} = \frac{\alpha}{n-1}$ *for* $i \neq j$, $\alpha \in [0, 1]$, $\sum_{i=1}^{n} \rho_i^* = 1$.

Proof.

$$\sum_{i=1}^{n} \rho_i^* D(\Theta_i^* \| \Theta_i) = \sum_{i=1}^{n} \rho_i^* \sum_{j=1}^{n} \theta_{ij}^* \log \frac{\theta_{ij}^*}{\theta_{ij}} = \sum_{i=1}^{n} \rho_i^* [\theta_{ii}^* \log \frac{\theta_{ii}^*}{\theta_{ii}} + \sum_{j \neq i}^{n} \theta_{ij}^* \log \frac{\theta_{ij}^*}{\theta_{ij}}]$$
$$= \sum_{i=1}^{n} \rho_i^* [(1 - \alpha^*) \log \frac{1 - \alpha^*}{1 - \alpha} + \sum_{j \neq i}^{n} \frac{\alpha^*}{n-1} \log \frac{\frac{\alpha^*}{n-1}}{\frac{\alpha}{n-1}}]$$
$$= \sum_{i=1}^{n} \rho_i^* [(1 - \alpha^*) \log \frac{1 - \alpha^*}{1 - \alpha} + \alpha^* \log \frac{\alpha^*}{\alpha}]$$
$$= \sum_{i=1}^{n} \rho_i^* D(\alpha^* \| \alpha) = D(\alpha^* \| \alpha) \sum_{i=1}^{n} \rho_i^* = D(\alpha^* \| \alpha)$$

□

Proof of Theorem 6

Proof. Using Definitions 6 and 7, we can express the loss of the algorithm on a

point x_t as

$$L(x_t, \texttt{clust}(t)) = \frac{\| \sum_{i=1}^{n} p(i)(x_t - c_t^i)\|^2}{4R^2}$$

Then the following chains of inequalities are equivalent to each other.

$$\begin{aligned}
L(x_t, \texttt{clust}(t)) &\leq -2 \log \sum_{i=1}^{n} p(i) e^{-\frac{1}{2} L(x_t, c_t^i)} \\
\frac{\| \sum_{i=1}^{n} p(i)(x_t - c_t^i)\|^2}{4R^2} &\leq -2 \log \sum_{i=1}^{n} p(i) e^{-\frac{1}{2} \frac{\|x_t - c_t^i\|^2}{4R^2}} \\
e^{\frac{\| \sum_{i=1}^{n} p(i)(x_t - c_t^i)\|^2}{4R^2}} &\leq \left(\sum_{i=1}^{n} p(i) e^{-\frac{1}{2} \frac{\|x_t - c_t^i\|^2}{4R^2}} \right)^{-2} \\
\sum_{i=1}^{n} p(i) e^{-\frac{1}{2} \frac{\|x_t - c_t^i\|^2}{4R^2}} &\leq e^{-\frac{1}{2} \frac{\| \sum_{i=1}^{n} p(i)(x_t - c_t^i)\|^2}{4R^2}}
\end{aligned} \tag{7.3}$$

Let $v_t^i = \frac{x_t - c_t^i}{2R}$. Since $\|x_t\| \leq R$ and $\|c_t^i\| \leq R$ then $v_t^i \in [-1 \ 1]^d$. Equation (7.3) is equivalent to

$$\sum_{i=1}^{n} p(i) e^{-\frac{1}{2}\|v_t^i\|^2} \leq e^{\frac{-1}{2}\| \sum_{i=1}^{n} p(i) v_t^i\|^2}$$

This inequality holds by Jensen's Theorem since the function $f(v_t^i) = e^{-\frac{1}{2}\|v_t^i\|^2}$ is concave when $v_t^i \in [-1 \ 1]^d$. □

Proof of Theorem 7

Proof. We can proceed by applying Theorem 6 to bound the cumulative loss of the algorithm and then use Lemma 16. As we proceed we follow the approach in the

proof of Theorem 2.1.1 in [110].

$$
\begin{aligned}
& L_T(\texttt{alg}) = \sum_{t=1}^{T} L(x_t, \texttt{clust}(t)) \\
\leq\ & -\sum_{t=1}^{T} 2 \log \sum_{i=1}^{n} p_t(i) e^{-\frac{1}{2} L(i,t)} \\
=\ & -2 \log P(x_1, ..., x_T) \\
=\ & -2 \log \sum_{i=1}^{n} P(x_1, ..., x_T | a_{i,1}, ..., a_{i,T}) P(a_{i,1}, ..., a_{i,T}) \\
=\ & -2 \log \sum_{i=1}^{n} p_1(i) P(x_1 | a_{i,1}) \prod_{t=2}^{T} P(x_t | a_i, x_1, ..., x_{t-1}) \\
=\ & -2 \log \frac{1}{n} \sum_{i=1}^{n} e^{-\frac{1}{2} L(i,1)} \prod_{t=2}^{T} e^{-\frac{1}{2} L(i,t)} \\
=\ & -2 \log \frac{1}{n} \sum_{i=1}^{n} e^{-\frac{1}{2} \sum_{t=1}^{T} L(i,t)} \\
=\ & -2 \log \frac{1}{n} - 2 \log \sum_{i=1}^{n} e^{-\frac{1}{2} \sum_{t=1}^{T} L(i,t)} \\
\leq\ & L_T(a_i) + 2 \log n
\end{aligned}
$$

The last inequality holds for any a_i, so in particular for a_i^*. □

Proof of Theorem 8

Proof. By applying first Theorem 6 and then Lemma 16 and following the proof of

Theorem 3 (Main Theorem) in the [110] we obtain:

$$L_T(alg) = \sum_{t=1}^{T} L_t(alg) \leq \sum_{t=1}^{T} -2\log \sum_{i=1}^{n} p_t(i) e^{-\frac{1}{2}L(i,t)}$$
$$= -2\log[\sum_{i_1,...,i_T} p_1(i_1) \prod_{t=2}^{T} P(i_t|i_{t-1},\Theta) \prod_{t=1}^{T} e^{-\frac{1}{2}L(i_t,t)}]$$
$$= -2\log[\sum_{i_1,...,i_T} P(i_1,...,i_T|\Theta) \prod_{t=1}^{T} e^{-\frac{1}{2}L(i_t,t)}]$$

where $P(i_1,...,i_T|\Theta) = p_1(i_1)\prod_{t=2}^{T} P(i_t|i_{t-1},\Theta)$. Notice that also:

$$P(i_1,...,i_T|\Theta) = p_1(i_1) \prod_{i=1}^{n}\prod_{j=1}^{n} (\theta_{ij})^{n_{ij}(i_1,...,i_T)}$$

where $n_{ij}(i_1,...,i_T)$ is the number of transitions from state i to state j, in a sequence $i_1,...,i_T$ and $\sum_j n_{ij}(i_1,...,i_T)$, is the number of times the sequence was in state i, except at the final time-step. Thus $\sum_j n_{ij}(i_1,...,i_T) = (T-1)\hat{\rho}_i(i_1,...,i_T)$, where $\hat{\rho}_i(i_1,...,i_T)$ is the empirical estimate, from the sequence $i_1,...,i_T$, of the marginal probability of being in state i, at any time-step except the final one. It follows that: $n_{ij}(i_1,...,i_T) = (T-1)\hat{\rho}_i(i_1,...,i_T)\hat{\theta}ij(i_1,...,i_T)$ where $\hat{\theta}_{ij}(i_1,...,i_T) = \frac{n_{ij}(i_1,...,i_T)}{\sum_j n_{ij}(i_1,...,i_T)}$ is the empirical estimate of the probability of that particular state transition, on the basis of $i_1,...,i_T$. Thus:

$$P(i_1,...,i_T|\Theta) = p_1(i_1) \prod_{i=1}^{n}\prod_{j=1}^{n} (\theta_{ij})^{(T-1)\hat{\rho}_i(i_1,...,i_T)\hat{\theta}_{ij}(i_1,...,i_T)}$$

$$= p_1(i_1)e^{(T-1)\sum_{i=1}^{n}\sum_{j=1}^{n}\hat{\rho}_i(i_1,...,i_T)\hat{\theta}_{ij}(i_1,...,i_T)\log\theta_{ij}}$$

Thus:

$$L_T(alg) \leq -2\log[\sum_{i_1,...,i_T} P(i_1,...,i_T|\Theta)\prod_{t=1}^{T} e^{-\frac{1}{2}L(i_t,t)}]$$
$$= -2\log[\sum_{i_1,...,i_T} p_1(i_1)e^{(T-1)\sum_{i=1}^{n}\sum_{j=1}^{n}\hat{\rho}_i(i_1,...,i_T)\hat{\theta}_{ij}(i_1,...,i_T)\log\theta_{ij}}\prod_{t=1}^{T} e^{-\frac{1}{2}L(i_t,t)}]$$

Let $i_1^{'},...,i_T^{'}$ correspond to the best segmentation of the sequence into s segments meaning the s-partitioning with minimal cumulative loss. Obviously then the hindsight-optimal (cumulative loss minimizing) setting of switching rate parameter α, given s, is: $\alpha^{'} = \frac{s}{T-1}$ and since we are in the fixed-share setting: $\theta_{ij}^{'} = 1-\alpha^{'}$ for i = j and $\theta_{ij}^{'} = \frac{\alpha^{'}}{n-1}$ for $i \neq j$. We can continue as follows:

$$\leq -2\log[p_1(i_1^{'})e^{(T-1)\sum_{i=1}^{n}\sum_{j=1}^{n}\hat{\rho}_i(i_1^{'},...,i_T^{'})\hat{\theta}_{ij}(i_1^{'},...,i_T^{'})\log\theta_{ij}}\prod_{t=1}^{T} e^{-\frac{1}{2}L(i_t^{'},t)}]$$
$$= -2\log p_1(i_1^{'}) - 2(T-1)\sum_{i=1}^{n}\sum_{j=1}^{n}\hat{\rho}_i(i_1^{'},...,i_T^{'})]\hat{\theta}_{ij}(i_1^{'},...,i_T^{'})\log\theta_{ij} + \sum_{t=1}^{T} L(i_t^{'},t)$$
$$= 2\log n - 2(T-1)\sum_{i=1}^{n}\sum_{j=1}^{n}\hat{\rho}_i(i_1^{'},...,i_T^{'})\hat{\theta}_{ij}(i_1^{'},...,i_T^{'})\log\theta_{ij} + \sum_{t=1}^{T} L(i_t^{'},t)$$
$$= \sum_{t=1}^{T} L(i_t^{'},t) + 2\log n - 2(T-1)\sum_{i=1,j=i}^{n}\hat{\rho}_i(i_1^{'},...,i_T^{'})\hat{\theta}_{ij}(i_1^{'},...,i_T^{'})\log\theta_{ij}$$
$$- 2(T-1)\sum_{i=1}^{n}\sum_{j=1,j\neq i}^{n}\hat{\rho}_i(i_1^{'},...,i_T^{'})\hat{\theta}_{ij}(i_1^{'},...,i_T^{'})\log\theta_{ij}$$

$$= \sum_{t=1}^{T} L(i_t^{'}, t) + 2\log n - 2(T-1)(1-\alpha^{'})\log(1-\alpha) \sum_{i=1,j=i}^{n} \hat{\rho}_i(i_1^{'}, ..., i_T^{'})$$

$$- 2(T-1)\frac{\alpha^{'}}{n-1}\log(\frac{\alpha}{n-1}) \sum_{i=1}^{n} \sum_{j=1,j\neq i}^{n} \hat{\rho}_i(i_1^{'}, ..., i_T^{'})$$

$$= \sum_{t=1}^{T} L(i_t^{'}, t) + 2\log n - 2(T-1)(1-\alpha^{'})\log(1-\alpha) - 2(T-1)\alpha^{'}\log(\frac{\alpha}{n-1})$$

$$= \sum_{t=1}^{T} L(i_t^{'}, t) + 2\log n - 2(T-1)(1-\alpha^{'})\log(1-\alpha) - 2(T-1)\alpha^{'}\log\alpha$$

$$+ 2(T-1)\alpha^{'}\log(n-1)$$

$$= \sum_{t=1}^{T} L(i_t^{'}, t) + 2\log n - 2(T-1)(1-\alpha^{'})\log(1-\alpha) - 2(T-1)\alpha^{'}\log\alpha$$

$$+ 2s\log(n-1)$$

$$= \sum_{t=1}^{T} L(i_t^{'}, t) + 2\log n + 2s\log(n-1) - 2(T-1)((1-\alpha^{'})\log(1-\alpha) + \alpha^{'}\log\alpha)$$

$$= \sum_{t=1}^{T} L(i_t^{'}, t) + 2\log n + 2s\log(n-1) + 2(T-1)(H(\alpha^{'}) + D(\alpha^{'}\|\alpha))$$

□

Proof of Lemma 8

Proof. Given any b, it will suffice to provide a sequence such that any b-approximate algorithm cannot output x_t, the current point in the stream, as one of its centers. We will provide a counter example in the setting where $k = 2$. Given any b, consider a sequence such that the stream before the current point consists entirely of n_1 data points located at some position A, and n_2 data points located at some position B,

where $n_2 > n_1 > b - 1$. Let x_t be the current point in the stream, and let it be located at a position X that lies on the line segment connecting A and B, but closer to A. That is, let $\|A - X\| = a$ and $\|B - X\| = c$ such that $0 < a < \frac{n_2}{n_2+1}c$. This is reflected by the figure, which includes some additional points D and E:

———A———D————X————E————————-B————-

where A,D,X,E,B are lying on the same line. Let $D \in [A, X]$ and $E \in [X, B]$. Let for particular location of D and E, $\|A - D\| = a_1$, $\|D - X\| = a_2$, $\|X - E\| = c_1$ and $\|E - B\| = c_2$, such that $a_1 + a_2 = a$ and $c_1 + c_2 = c$.

We will first reason about the optimal k-means clustering of the stream including x_t. We will consider cases:

1) Case 1: optimal centers lie inside the interval (A, X). Any such set cannot be optimal since by mirror-reflecting the center closest to X with respect to X (such that it now lies in the interval (B, X) and has the same distance to X as before) we can decrease the cost. In particular the cost of points in B will only decrease, leaving the cost of points in A, plus the cost of X, unchanged.

2) Case 2: optimal centers lie inside the interval (B, X). In this case we can alternately mirror-reflect the closest center to X with respect to X and then with respect to A (reducing the cost with each reflection) until it will end up in interval $[A, X)$. The cost of the final solution is smaller than when both centers were lying in (B, X), because while mirror-reflecting with respect to X, the cost of points in

A can only decrease, leaving the cost of points in B and point X unchanged and while mirror-reflecting with respect to A, the cost of point X can only decrease, leaving the cost of points in A and in B unchanged.

Thus the optimal set of centers (let's call them: C_1, C_2) must be such that: $C_1 \in [A, X]$ and $C_2 \in [B, X]$. The figure above reflects this situation and is sufficiently general; thus $D = C_1$ and $E = C_2$. We will now consider all possible locations of such centers (C_1, C_2) and their costs:

1) (A,X): cost $= n_2(c_1 + c_2)^2$

2.1) (A,E) where $E \in (X, B)$: cost $= (a_1 + a_2)^2 + n_2c_2^2$ when $c_1 \geq a$

2.2) (A,E) where $E \in (X, B)$: cost $= c_1^2 + n_2c_2^2$ when $c_1 < a$

3) (A,B): cost $= (a_1 + a_2)^2$

4) (D,X) where $D \in (A, X)$: cost $= n_1a_1^2 + n_2(c_1 + c_2)^2$

5.1) (D,E) where $D \in (A, X)$ and $E \in (X, B)$: cost $= n_1a_1^2 + a_2^2 + n_2c_2^2$ when $a_2 \leq c_1$

5.2) (D,E) where $D \in (A, X)$ and $E \in (X, B)$: cost $= n_1a_1^2 + c_1^2 + n_2c_2^2$ when $a_2 > c_1$

6) (D,B) where $D \in (A, X)$: cost $= n_1a_1^2 + a_2^2$

7) (X,E) where $E \in (X, B)$: cost $= n_1(a_1 + a_2)^2 + n_2c_2^2$

8) (X,B) where $E \in (X, B)$: cost $= n_1(a_1 + a_2)^2$

Notice:

cost(1) > cost(2.2) thus optimal configuration of centers cannot be as in case 1)

cost(7) > cost(8) thus optimal configuration of centers cannot be as in case 7)

cost(4) > cost(5.2) thus optimal configuration of centers cannot be as in case 4)

cost(8) > cost(6) thus optimal configuration of centers cannot be as in case 8)

cost(5.2) > cost(2.2) thus optimal configuration of centers cannot be as in case 5.2)

cost(5.1) > cost(6) thus optimal configuration of centers cannot be as in case 5.1)

cost(2.1) > cost(3) thus optimal configuration of centers cannot be as in case 2.1)

Consider case (2.2): cost $= c_1^2 + n_2 c_2^2 = c_1^2 + n_2(c - c_1)^2$ and $c_1 \in (0, a)$. Keeping in mind that $0 < a < \frac{n_2}{n_2+1}c$, it is easy to show that cost $> a^2 + n_2(c-a)^2 > n_2 a^2 > n_1 a^2 > a^2 = \text{cost}(case(3))$. Thus the optimal configuration of centers cannot be as in case 2.2). Therefore, the only cases we are left to consider are cases 3 and 6. In both cases, one of the optimal centers lies at B. Let this center be C_2. Since $a < c$, the remaining points (in A and X) are assigned to center C_1 whose location can be computed as follows (please see figure below for notation simplicity):

A—C_1———X————————B $= C_2$

Let $\|A - C_1\| = \delta$ and $\|A - X\| = a$ (as defined above). Since we proved that C_2 must be fixed at B, the points at B will contribute 0 to the objective, and we can solve for C_1 by minimizing the cost from points in A, plus the cost from X: $\min_\delta\{n_1\delta^2 + (a-\delta)^2\}$. The solution is $\delta = \frac{a}{n_1+1}$. Thus the total optimal cost is:

$$OPT = n_1\delta^2 + (a-\delta)^2 = \frac{n_1 a^2}{(n_1+1)^2} + \frac{n_1^2 a^2}{(n_1+1)^2}$$

We will now consider any 2-clustering of set A, X, B when one cluster center is x_t, which is therefore located at X. We can lower bound the k-means cost of any two set of centers that contain X, as follows: $\text{cost}(\{X, \hat{c}\}) \geq n_1 a^2$ for any $\hat{c}$; the minimal cost is achieved when the other center is located at B (when one of the centers is located in X, the location of the other center that gives the smallest possible k-means cost can be either A (case 1), D (case 4), E (case 7) or B (case 8), where case 8 has the smallest cost from among them).

Violating the b-approximation assumption occurs when $\text{cost}(\{X, \hat{c}\}) > b * OPT$. Given the above, it would suffice to show $n_1 a^2 > b * OPT$. That is:

$$n_1 a^2 > b \left(\frac{n_1 a^2}{(n_1+1)^2} + \frac{n_1^2 a^2}{(n_1+1)^2} \right) \Leftrightarrow b < (n_1 + 1)$$

This holds, as we chose $n_1 > b - 1$ in the beginning. Therefore the b-approximation assumption is violated. □

Proof of Lemma 10

Proof. For ease of notation, we denote by $\Phi_{(t-W,t>}$ the k-means cost of algorithm a on the data seen in the window $(t - W, t >$ (omitting the argument, which is the set of centers output by algorithm a at time t). Since a is b-approximate then:

$$\forall_W \quad \Phi_{(t-W,t>} \leq b \cdot OPT_{(t-W,t>} \tag{7.4}$$

For any W such that $W < T$, we can decompose T such that $T = mW + r$ where $m \in \mathbb{N}$ and $r \leq W$. Notice then that for any $j \in \{0, ..., W-1\}$ the following chain of inequalities holds as the direct consequence of Equation 7.4:

$$\begin{aligned}
\Phi_{(T-W+j,T>} \quad &+ \quad \Phi_{(T-2W+j,T-W+j>} \\
&+ \quad \Phi_{(T-3W+j,T-2W+j>} \\
&+ \ldots + \quad \Phi_{(T-mW+j,T-(m-1)W+j>} \\
&+ \quad \Phi_{<1,T-mW+j>} \\
&\leq \quad b \cdot OPT_{(T-W+j,T>} \\
&+ \quad b \cdot OPT_{(T-2W+j,T-W+j>} \\
&+ \quad b \cdot OPT_{(T-3W+j,T-2W+j>} \\
&+ \ldots + \quad b \cdot OPT_{(T-mW+j,T-(m-1)W+j>} \\
&+ \quad b \cdot OPT_{<1,T-mW+j>} \\
&\leq \quad b \cdot OPT_{<1,T>}, \qquad (7.5)
\end{aligned}$$

where the last inequality is the direct consequence of Lemma 9. Notice that different value of j refers to different partitioning of the time span $< 1, T >$. Figure 7.2 illustrates an example when T = 10 and W = 3.

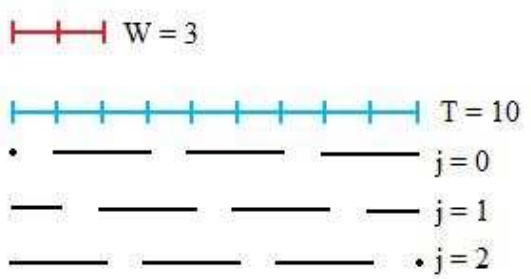


Figure 7.2: Illustration of the time spans over which the loss is being computed in Equation 7.6. T = 10, W = 3. Colors red, blue and black correspond to different partitionings, with respect to j, of time span T illustrated in Figure 7.2.

Let W_t refer to the data chunk seen in time span $< \max(1, t - W + 1), t >$. We finish by showing that,

$$\begin{aligned}
\sum_{t=1}^{T} \Phi_{W_t} \quad &\leq \quad \sum_{j=0}^{W-1} \{\Phi_{(T-W+j,T>} \\
&+ \quad \Phi_{(T-2W+j,T-W+j>} \\
&+ \quad \Phi_{(T-3W+j,T-2W+j>} \\
&+ \ldots + \quad \Phi_{(T-mW+j,T-(m-1)W+j>} \\
&+ \quad \Phi_{<1,T-mW+j>}\} \\
&\leq \quad b \cdot W \cdot OPT_{<1,T>}
\end{aligned} \tag{7.6}$$

The left hand side of the first inequality (7.6) sums the losses over only a subset of all the windows that are induced by partitioning the time span T using all possible values of j. The final inequality follows by applying (7.5).

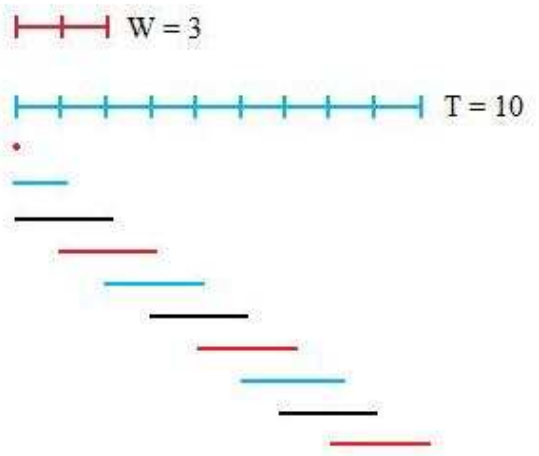


Figure 7.3: Different partitioning of time span T = 10 into time windows, W = 3, with respect to different values of j.

To illustrate some of the ideas used in this proof, Figures 7.2 and 7.3 provide schematics. Figure 7.3 shows the windows over which the loss is computed on the left hand side of inequality (7.6), which is a subset of the set of all windows induced by all possible partitioning of time span T using all possible values of j, which is shown in Figure 7.2. □

Proof of Lemma 11

Proof. The loss of expert i at time t is defined in Definition 6 as the scaled component of the k-means cost of algorithm a's clustering at time t. That is, $\Phi_t(C_t) = \sum_{x'_t \in W_t} \min_{c \in C_t} \|x_{t'} - c\|^2 = \min_{c \in C_t} \|x_t - c\|^2 + \sum_{x'_t \in W_t \setminus x_t} \min_{c \in C_t} \|x_{t'} - c\|^2 = 4R^2 \cdot L_t(C_t) + \sum_{x'_t \in W_t \setminus x_t} \min_{c \in C_t} \|x_{t'} - c\|^2$.

Therefore, since all terms in the sum are positive, $4R^2 \cdot L_t(C_t) \leq \Phi_t(C_t)$, and

$L_t(a) \leq \frac{\Phi_t}{4R^2}$. where in the second inequality we substitute in our simplifying notation, where C_t are the set of clusters generated by algorithm a at time t, and Φ_t is algorithm a's k-means cost on W_t. Now, summing over T iterations, and applying Lemma 10, we obtain $\sum_{t=1}^{T} L_t(a) \leq \frac{b \cdot W}{4R^2} OPT_{<1,T>}$. □

Proof of Theorem 12

Proof. The theorem directly follows from Theorem 8, Lemma 10 and Lemma 11. Notice that both Lemma 10 and Lemma 11 hold in a more general setting when in each time window the identity of the expert (b-approximate algorithm) may change in an arbitrarily way. However we did not provide the generalized proofs of those lemmas since it would only complicate the notation. □

Proof of Theorem 9, Corollary 2, and Theorem 13

Theorem 9, Corollary 2, and Theorem 13 follow directly from the results that we will next show.

Lemma 18.

$$L_T^{log}(\Theta) = \sum_{t=1}^{T} L^{log}(p_t, t) = -\log \left[\sum_{i_1,\ldots,i_T} p_1(i_1) e^{-\frac{1}{2}L(i_1,1)} \prod_{t=2}^{T} e^{-\frac{1}{2}L(i_t,1)} P(i_t|i_{t-1}, \Theta) \right]$$

Proof. It follows directly from Lemma 16. □

Lemma 19.

$$L_T^{log}(\Theta) - L_T^{log}(\Theta^*) = -\log\left[\sum_{\vec{s}} Q(\vec{s}|\theta^*)\exp\left\{T'\sum_{i=1}^{n}\sum_{j=1}^{n}\hat{\rho}_i(\vec{s})\hat{\theta}_{ij}(\vec{s})\log(\frac{\theta_{ij}}{\theta^*_{ij}})\right\}\right]$$

where $\vec{s} = i_1, ..., i_T$ *and* $Q(\vec{s}|\Theta^*)$ *is the posterior probability over the choices of experts along the sequence, induced by hindsight-optimal* Θ^**.*

Proof. It follows directly from applying proof of Lemma A.0.1 in [**110**] with redefined $\phi(\vec{s})$ such that $\phi(\vec{s}) = \prod_{t=1}^{T} e^{-\frac{1}{2}L(i_t,t)}$. □

Lemma 20.

$$L_T^{log}(\Theta) - L_T^{log}(\Theta^*) \leq (T-1)\sum_{i=1}^{n}\rho_i^* D(\Theta_i^*\|\Theta_i) \leq (T-1)\max_i D(\Theta_i^*\|\Theta_i)$$

Proof. It holds by Theorem 3 in [**110**]. □

Lemma 21.

$$L_T^{\log}(\alpha) \leq L_T^{\log}(\alpha^*) + (T-1)D(\alpha^*\|\alpha)$$

Proof. It holds by Lemma 17, Lemma 20. □

Lemma 22.

$$L_T(\alpha) \leq 2L_T^{\log}(\alpha^*) + 2(T-1)D(\alpha^* \| \alpha)$$

Proof. It holds by Lemma 21 and Lemma 7. □

Lemma 23.

$$L_T(\theta) \leq \frac{bW}{2R^2} OPT_T + 2(T-1)\sum_{i=1}^{n} \rho_i^* D(\theta_i^* || \theta_i)$$

Proof.

$$L_T^{log}(\theta^*) = \sum_{t=1}^{T} L^{log}(p_t, t)_{|\theta^*} = (\sum_{t=1}^{T} -\log \sum_{i=1}^{n} p_t(i) e^{-\frac{1}{2}L(i,t)})_{|\theta^*} =$$

$$= (-\sum_{t=1}^{T} \log \sum_{i=1}^{n} p_t(i) e^{-\frac{1}{2}\|\frac{x_t - c_t^i}{2R}\|^2})_{|\theta^*}$$

Define as previously $v_t^i = \frac{x_t - c_t^i}{2R}$ and notice $v_t^i \in [-1; 1]^d$.

Thus we continue:

$$= (-\sum_{t=1}^{T} \log \sum_{i=1}^{n} p_t(i) e^{-\frac{1}{2}\|v_t^i\|^2})_{|\theta^*}$$

By the proof of Lemma 4 $\|v_t^i\|^2 \leq \frac{\phi_t^i}{4R^2}$ where ϕ_t^i is the k-means cost of algorithm i^{th} clustering (i^{th} expert) at time t.

Thus we can continue as follows (we omitt conditioning by θ^* since it holds for any $\vec{p_t}$):

$$\leq -\sum_{t=1}^{T} \log \sum_{i=1}^{n} p_t(i) e^{-\frac{1}{2}\frac{\phi_t^i}{4R^2}} \leq -\sum_{t=1}^{T} \log \sum_{i=1}^{n} p_t(i) e^{-\frac{1}{2}\frac{\max_i \phi_t^i}{4R^2}}$$

$$\leq -\sum_{t=1}^{T} \log e^{-\frac{1}{2}\frac{\max_i \phi_t^i}{4R^2}} = \sum_{t=1}^{T} \frac{1}{2}\frac{\max_i \phi_t^i}{4R^2} = \frac{1}{8R^2}\sum_{t=1}^{T} \max_i \phi_t^i \leq \frac{bW}{8R^2} OPT_T$$

The last inequality follows from the proof of Theorem 7. Thus finally:

$$L_T(\theta) \leq 2L_T^{log}(\theta) \leq 2L_T^{log}(\theta^*) + 2(T-1)\sum_{i=1}^{n} \rho_i^* D(\theta_i^* || \theta_i)$$

$$\leq \frac{bW}{4R^2} OPT_T + 2(T-1)\sum_{i=1}^{n} \rho_i^* D(\theta_i^* || \theta_i)$$

□

Proof of Theorem 10

Proof. The logloss per time step of the top-level algorithm (for the ease of notation

we skip index log), which updates its distribution over α - experts (Fixed-Share (α_j algorithms) is:

$$L^{top}(p_t^{top}, t) = -\log \sum_{j=1}^{m} p_t^{top}(j) e^{-L^{\log}(j,t)}$$

where

$$L^{\log}(j, t) = -\log \sum_{i=1}^{n} p_j(i) e^{-\frac{1}{2}L(i,t)}$$

thus:

$$L^{top}(p_t^{top}, t) = -\log \sum_{j=1}^{m} p_t^{top}(j) \sum_{i=1}^{n} p_j(i) e^{-\frac{1}{2}L(i,t)}$$

The update is done via the Static Expert algorithm. When running Learn-$\alpha(m)$, m possible values α_j are tested.

Following [**70, 110**], the probabilistic "prediction" of the algorithm is defined as:

$$\sum_{j=1}^{m} p_t^{top}(j) \sum_{i=1}^{n} p_j(i) e^{-\frac{1}{2}L(i,t)}$$

Let us first show that this loss-prediction pair are $(1,1)$-realizable, thus they satisfy:

$$L^{top}(p_t^{top}, t) \leq -\log \sum_{j=1}^{m} p_{top}(j) e^{-L^{\log}(j,t)}$$

Thus we have to prove that following holds:

$$-\log \sum_{j=1}^{m} p_t^{top}(j) \sum_{i=1}^{n} p_j(i) e^{-\frac{1}{2}L(i,t)} \leq -\log \sum_{j=1}^{m} p_t^{top}(j) e^{-L^{\log}(j,t)}$$

thus we have to prove:

$$-\log \sum_{j=1}^{m} p_t^{top}(j) \sum_{i=1}^{n} p_j(i) e^{-\frac{1}{2}L(i,t)} \leq -\log \sum_{j=1}^{m} p_t^{top}(j) e^{\log \sum_{i=1}^{n} p_j(i) e^{-\frac{1}{2}L(i,t)}}$$

which is equivalent to

$$-\log \sum_{j=1}^{m} p_t^{top}(j) \sum_{i=1}^{n} p_j(i) e^{-\frac{1}{2}L(i,t)} \leq -\log \sum_{j=1}^{m} p_t^{top}(j) \sum_{i=1}^{n} p_j(i) e^{-\frac{1}{2}L(i,t)}$$

where the last inequality holds. Now, since our logloss-prediction pair are (1,1)-realizable, by Lemma 1 in [**69**] we have:

$$L_T^{top} \leq \min_{\{\alpha_j\}} L_T^{\log}(\alpha_j) + \log m$$

where $\{\alpha_j\}$ is the discretization of the α parameter that the Learn-α algorithm takes as input. Now, by applying Corollary 2 we get:

$$L_T^{\log}(\texttt{alg}) \leq L_T^{\log}(\alpha^*) + (T-1)\min_{\{\alpha_j\}} D(\alpha^* \| \alpha_j) + \log m$$

□