A distributed Frank-Wolfe framework for trace norm minimization via the bulk synchronous parallel model

Wenjie Zheng

Sorbonne Université

École doctorale Informatique, Télécommunications et Électronique (Paris)

LIP6/MLIA

A Distributed Frank-Wolfe Framework for Trace Norm Minimization via the Bulk Synchronous Parallel Model

Par Wenjie Zheng

Thèse de doctorat d'Informatique

Dirigée par Patrick Gallinari

Présentée et soutenue publiquement le 13 Juin 2018

Devant le jury composé de :

Taïani François	Professeur	Rapporteur
Amini Massih-Reza	Professeur	Rapporteur
Naacke Hubert	Maître de conférence	Examinateur
Bellet Aurélien	Chargé de recherche 2	Examinateur
Germain Cécile	Professeur	Examinateur
Denoyer Ludovic	Professeur	Examinateur
Gallinari Patrick	Professeur	Examinateur

Preface

Our society is moving into the *Big Data* era, with an incredible amount of data generated daily. For instance, the world's largest and most powerful particle collider, Large Hadron Collider, is expected to produce 50 petabytes of data in 2017. Also in 2017, a total duration of 300 hours of videos appears on YouTube per minute. One year earlier, in 2016, around 6000 tweets are tweeted on Twitter per second. Furthermore, scientists are developing wearable computing device for Body Area Networks, which can monitor human medical conditions (*e.g.*, heart rate) in real time. Besides the generation of the data, the reduction of the storage material cost also contributes to the takeoff of this Big Data era. In March 2015, the price of storage per GB had dropped to $0.02.

These data can have one or more of the following characteristics. (1) Huge amounts of records or samples. For instance, the DNA microarrays of all human beings have more than 7 billion records and will have more and more as time goes on. (2) Huge amounts of fields. Each piece of record is composed of many fields, which can be the name, sex, age, and so forth for a demography database or thousands of probes for a DNA microarray database. (3) The databases are interrelated. For example, the combination of the DNA microarray database and the demography database can reveal the correlation or even the causality between genes and physical expression. (4) Complex data type. The records may contain missing fields because of, among others, data corruption and privacy constraints, or they have a complex structure such as articles, images, and videos.

A database can be represented as a table or, using the mathematical terminology, as a matrix. Each row of this matrix represents a record, and each column a field. In statistics or in machine learning, the row size is conventionally noted as n, and the column size d or p. The column size is often understood as the *dimension* of the data. While a large n had already been substantially studied by the traditional statistics (*i.e.*, by sampling), the problem of large d gave birth to the *high-dimensional statistics* over the past decade. This subject is still in intense research nowadays and produces many interesting and useful concepts such as Sparsity, Compressed Sensing, Lasso, and so forth. A principled approach in this domain is nonparametric statistics, both frequentist and Bayesian. Then, it remains the cases where both n and d are large, which are the case I study in this thesis. Nonparametric statistics tells us that its convergence rates are slower than the standard square root as in parametric statistics. This disadvantage encourages us to use as much data as possible instead of just sampling a subset.

My thesis is one of the many responses to this need. It combines recent cutting-edge technologies from optimization, trace norm minimization, and distributed com-

puting. Of course, it is not a panacea solving all Big Data problems, but it *is* a milestone upon which readers can build their own work. To make this dissertation as valuable as possible, I have tried to express more thoughts than necessary. Although they may be interesting and even open new research directions, do not hesitate to skip some minor points. In particular, for some facts not indispensable to understand this dissertation but good to know, I put them as *remarks*. Readers can consider them as complementary material for extended reading.

Several groups of people who, in my opinion, may be potential readers of this dissertation. Firstly, it can interest those Frank-Wolfe theorists. This dissertation provides an analysis of the cases where the sublinear problem can only be approximately solved in a *nondeterministic* way, which includes the estimation of a matrix's top singular value/vectors.

Secondly, it can interest those practitioners who need to recover low-rank matrices frequently and efficiently. Instead of the popular matrix factorization method, this dissertation provides an alternative way, accompanied by the source code, to solve this same problem. It can be applied to datasets which are large-scale both in samples and in dimensions. It can be either distributed or parallel, just like matrix factorization.

Thirdly, it can interest those Frank-Wolfe developers and Apache SPARK enthusiasts. For Frank-Wolfe developers, it provides a template to develop a comprehensive Frank-Wolfe package. For Apache SPARK enthusiasts, regardless of their adoption of Frank-Wolfe, my code can undoubtedly inspire them to manage Apache SPARK's behaviors better and to use the resources more wisely.

Fourthly, it may interest researchers on distributed learning and deep learning. Although the current most valuable player in these two domains is stochastic gradient descent, I certainly see the potential of Frank-Wolfe to shine there. One day, we may see a new research domain of machine learning.

Finally, if you do not belong to any of the categories above, I still hope you can find something inspiring here. For example, the introduction to multi-task learning is quite exotic, which adopts a point of view of econometrics. Some discussion in the distributed machine learning section is also quite impressive, which is not likely to see in any scientific publications but is more than relevant in the daily usage.

I have made significant efforts to provide the readers with novel and original knowledge, just like my many teachers had done for me during my life. Following their path, I hope my knowledge can be useful to people, to the world, and let the enthusiasm resonate among young hearts and make the world better.

Wenjie ZHENG

Paris, December 1, 2017

Acknowledgments

First and foremost, I would like to thank Patrick Gallinari and Ludovic Denoyer for finding the fund necessary to my thesis – this research is also partially financed by myself – as well as for reviewing my dissertation draft. It has been a great challenge to do research, and I broke through my limits several times along the journey. By overcoming all these difficulties, I significantly improved my mathematical, programming, engineering, research, academical writing, and project management skills.

I would also like to thank Aurélien Bellet not only for the initial research question but also for his constant, albeit not always timely, high-quality feedbacks toward my progress. Whenever I made a great achievement and felt hence self-satisfied, his request pushed me to dive deeper and to challenge even harder research questions. By studying these problems originally beyond my reach, I often made unexpected discoveries of even higher significance. I appreciate this collaboration.

There are other important people who have helped me during this period. I would like to thank François Taïani for being in my midterm jury as well as other people following my research or planning to be in the jury of my thesis defense; the UPMC faculty of the mathematics, with whose help I further developed my mathematical skills; Hubert Naacke, Mohamed-Amine Baazizi, and the technical staff of LIP6 for their help about the cluster; the administrative staff, especially Marguerite, who has unfortunately left us, for their help during my enrollment; and Eloi Zablocki for proofreading the French abstract of this dissertation.

Last but not least, I would like to thank deeply my mother for her love and self-sacrifice.

Abstract

Learning low-rank matrices is a problem of great importance in statistics, machine learning, computer vision, recommender systems, etc. Because of its NP-hard nature, a principled approach is to solve its tightest convex relaxation: trace norm minimization. Among various algorithms capable of solving this optimization is the Frank-Wolfe method, which is particularly suitable for high-dimensional matrices. In preparation for the usage of distributed infrastructures to further accelerate the computation, this study aims at exploring the possibility of executing the Frank-Wolfe algorithm in a star network with the Bulk Synchronous Parallel (BSP) model and investigating its efficiency both theoretically and empirically.

In the theoretical aspect, this study revisits Frank-Wolfe's fundamental deterministic sublinear convergence rate and extends it to nondeterministic cases. In particular, it shows that with the linear subproblem *appropriately* solved, Frank-Wolfe can achieve a sublinear convergence rate both in expectation and with high probability. This contribution lays the theoretical foundation of using power iteration or Lanczos iteration to solve the linear subproblem for trace norm minimization.

In the algorithmic aspect, within the BSP model, this study proposes and analyzes four strategies for the linear subproblem as well as methods for the line search. Moreover, noticing Frank-Wolfe's rank-1 update property, it updates the gradient recursively, with either a *dense* or a *low-rank* representation, instead of repeatedly recalculating it from scratch. All of these designs are generic and apply to *any* distributed infrastructures compatible with the BSP model.

In the empirical aspect, this study tests the proposed algorithmic designs in an Apache SPARK cluster. According to the experiment results, for the linear subproblem, centralizing the gradient or averaging the singular vectors is sufficient in the low-dimensional case, whereas distributed power iteration, with as few as one or two iterations per epoch, excels in the high-dimensional case. The Python package developed for the experiments is modular, extensible and ready to deploy in an industrial context.

This study has achieved its function as *proof of concept*. Following the path it sets up, solvers can be implemented for various infrastructures, among which GPU clusters, to solve practical problems in specific contexts. Besides, its excellent performance in the ImageNet dataset makes it promising for deep learning.

Résumé

L'apprentissage des matrices de rang faible est un problème de grande importance dans les statistiques, l'apprentissage automatique, la vision par ordinateur et les systèmes de recommandation. En raison de sa nature NP-difficile, une des approches principale consiste à résoudre sa relaxation convexe la plus étroite : la minimisation de la norme de trace. Parmi les différents algorithmes capables de résoudre cette optimisation, on peut citer la méthode de Frank-Wolfe, particulièrement adaptée aux matrices de grande dimension.

En préparation à l'utilisation d'infrastructures distribuées pour accélérer le calcul, cette étude vise à explorer la possibilité d'exécuter l'algorithme de Frank-Wolfe dans un réseau en étoile avec le modèle BSP (Bulk Synchronous Parallel) et à étudier son efficacité théorique et empirique.

Concernant l'aspect théorique, cette étude revisite le taux de convergence déterministe de Frank-Wolfe et l'étend à des cas non déterministes. En particulier, il montre qu'avec le sous-problème linéaire résolu de manière *appropriée*, Frank-Wolfe peut atteindre un taux de convergence sous-linéaire à la fois en espérance et avec une probabilité élevée. Cette contribution pose la fondation théorique de l'utilisation de la méthode de la puissance itérée ou de l'algorithme de Lanczos pour résoudre le sous-problème linéaire de Frank-Wolfe associé à la minimisation de la norme de trace.

Concernant l'aspect algorithmique, dans le cadre de BSP, cette étude propose et analyse quatre stratégies pour le sous-problème linéaire ainsi que des méthodes pour la recherche linéaire. En outre, remarquant la propriété de mise à jour de rang-1 de Frank-Wolfe, il met à jour le gradient de manière récursive, avec une représentation dense ou de rang faible, au lieu de le recalculer de manière répétée à partir de zéro. Toutes ces conceptions sont génériques et s'appliquent à toutes les infrastructures distribuées compatibles avec le modèle BSP.

Concernant l'aspect empirique, cette étude teste les conceptions algorithmiques proposées dans un cluster Apache SPARK. Selon les résultats des expériences, pour le sous-problème linéaire, la centralisation des gradients ou la moyenne des vecteurs singuliers est suffisante dans le cas de faible dimension, alors que la méthode de la puissance itérée distribuée, avec aussi peu qu'une ou deux itérations par époque, excelle dans le cas de grande dimension. La librairie Python développée pour les expériences est modulaire, extensible et prêt à être déployée dans un contexte industriel.

Contents

List of Tables

List of Figures

List of Algorithms

Notations

$\mathbf{conv}(S)$	convex hull of S
$A \subset B$	A is a subset of B
$\mathbb{R}^n$	n-dimensional real Euclidean space
$\mathbf{dom} f$	domain of function f
A^T	transpose of A
$a^T b$	inner product of vector a and vector b
$\langle \cdot, \cdot \rangle$	inner product
∇f	gradient of f
$\nabla^2 f$	Hessian of f
$A \succeq B, B \preceq A$	$A - B$ is positive semidefinite
$A \succ B, B \prec A$	$A - B$ is positive definite
$\|x\|$	Euclidean norm of the vector x
$\|x\|_p$	p-norm of the vector x
$\|A\|_{\text{sp}}$	spectral norm of the matrix A
$\|A\|_{\text{F}}$	Frobenius norm of the matrix A
$\|A\|_{\text{tr}}$	trace/nuclear norm of the matrix A
$f^{(p)}$	p-th derivative of function f
$a \wedge b$	minimum of a and b
$a \vee b$	maximum of a and b
$\sigma_i(X)$	i-th largest singular value of matrix X
$:=$	defined by
$\leftarrow$	assigned by
$[n]$	$\{1, 2, 3, \ldots, n\}$
$S_1 \times S_2$	Cartesian product of set S_1 and set S_2
2^S	power set of S
S^c	complementary of S
$\mathbf{Tr}(A)$	trace of matrix A
$\ell_{p,q}(A)$	$\left\{\sum_i (\sum_j a_{ij}^q)^{\frac{p}{q}}\right\}^{\frac{1}{p}}$
s.t.	subject to
ln	natural logarithm
Pr	probability measure
$\mathcal{N}(\mu, \sigma^2)$	Gaussian distribution with mean μ and variance σ^2
$\text{diam}_{\|\cdot\|}$	$\|\cdot\|$-diameter
a.s.	almost surely
sgn	sign function
$\mathbb{R}^n_+$	the positive orthant of $\mathbb{R}^n$
I	Identity matrix, the dimension is according to the context
E	a matrix with all entries equal to 1

Chapter 1

Introduction

IT companies nowadays appear more and more intelligent by not only providing us with products and information but also *predicting* our preferences about these items. Smartphone application stores, such as App Store (iOS)[1] and Google Play[2], can recommend the most suitable applications for you. Video distributing or sharing companies, such as Netflix[3] and Youtube[4], can recommend videos the most attractive to you. This impressive service is called *recommender system*, and it revolutionizes our modern life.

Beneath these achievements lies the idea of learning low-rank matrices. This concept of great importance can be explained by simple algebra that any first-year undergraduate with science major understands. In all these cases, there are two groups of entities – *users* and *items*. These two entities are, in the application store scenario, the smartphone users and the apps; and, in the video sharing scenario, the viewers and the videos. The users and the items form a matrix, with each entry representing the corresponding user's *rating* on the corresponding item. These ratings can be either collected by studying the user behavior or given explicitly by users (Figure 1.1). The enormous number of users and items makes the matrix high-dimensional. Meanwhile, this matrix is also sparse – users cannot learn about every item in the world let alone rate it. Here comes the question: how can the recommender system know the user preference on items that the users themselves have never even heard about? Mathematically, this task is equivalent to filling the missing values in the sparse rating matrix, which is generally impossible unless we impose some extra structure on the rating matrix. One of the widely adopted hypotheses is the low-rank property of the matrix. We suppose that this matrix is low-rank and hence can be represented by the multiplication of two low-dimensional matrices (Figure 1.2).[5] These low-dimensional matrices are often called user profile and item profile (or user embedding and item embedding according to the domains). The intuition behind is that each user or item can be represented by a low-dimensional

[1] https://itunes.apple.com/us/genre/ios/id36?mt=8

[2] https://play.google.com/store/apps

[3] https://www.netflix.com/

[4] https://www.youtube.com/

[5] There exist other hypotheses; for example, the rating matrix can be represented by the multiplication of two *nonnegative* matrices.

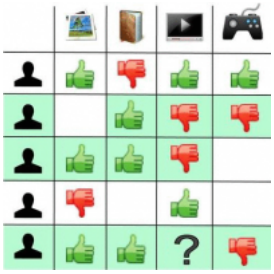


Figure 1.1 – Rating matrix, created by Wikimedia user *Moshanin*.

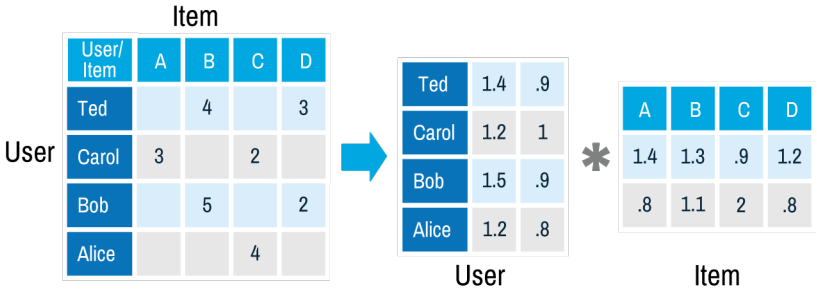


Figure 1.2 – Low-rank rating matrix. Source: Apache Github repository

vector characterizing the nature of the user or the item. The rating is nothing else than the inner product of the vector of the user and the one of the item. A high inner product implies a high preference, and a low value suggests the contrary. Since these profile matrices are low-dimensional, we now have much fewer parameters than the observed ratings and hence are more likely to recover the full rating matrix.

Learning low-rank matrices is not only a central problem in recommender systems (Koren et al. 2009) but also an important one in statistics, machine learning, and computer vision. It has led to many successful applications, among which multi-task learning (Argyriou et al. 2008, Pong et al. 2010), multi-class and multi-label classification (Goldberg et al. 2010, Cabral et al. 2011, Harchaoui et al. 2012), robust PCA (Cabral et al. 2013), phase retrieval (Candès et al. 2013) and video denoising (Ji et al. 2010). These tasks consist in learning a low-rank matrix fit for some specific purpose. Let $\ell(W)$ be the problem-specific convex loss associated with the matrix $W \in \mathbb{R}^{d\times m}$. We intend to solve

$$\begin{aligned} \min_{W\in\mathbb{R}^{d\times m}} \quad & \operatorname{rank}(W) \\ \text{s.t.} \quad & \ell(W) \leq \mu \end{aligned} \tag{1.1}$$

for some predetermined constant μ characterizing the tolerance level.

Nonetheless, (1.1) is in general NP-hard (see the introduction in Fazel et al. 2004). To make it tractable, there exist mainly two approaches. The first consists in explicitly factorizing the matrix W as in Koren et al. (2009). Mathematically, it considers $W = U^T V$, where $U \in \mathbb{R}^{r\times d}$ and $V \in \mathbb{R}^{r\times m}$, for some small constant $r \ll \min\{d, m\}$ representing the estimated rank, and solves

$$\min_{U\in\mathbb{R}^{r\times d}, V\in\mathbb{R}^{r\times m}} \quad \ell(U^T V) + \text{regularization}. \tag{1.2}$$

The constant r can be chosen by cross-validation (Kanagal and Sindhwani 2010). This optimization is solved by alternating minimization with regard to U and V. Since the function is convex with regard to U and V *separately*, there is no difficulty in conducting the computation. However, this function is *nonconvex* when considering U and V as a whole, which may trap the solver at some local minimum. For a long time, this approach was justified only by its excellent empirical results and easy implementation upon parallel and distributed infrastructures. Only over the few past years, it was proved for a few specific problems (*e.g.*, phase retrieval/synchronization, orthogonal tensor decomposition, dictionary learning, matrix sensing, matrix completion) that every local minimum must also be global (Ge et al. 2015, Sun et al. 2015, Bandeira et al. 2016, Ge et al. 2016, Bhojanapalli et al. 2016).

The other approach consists in using (1.1)'s tightest convex relaxation, trace norm minimization (Fazel et al. 2001)

$$\begin{aligned} \min_{W \in \mathbb{R}^{d \times m}} \quad & \|W\|_* \\ \text{s.t.} \quad & \ell(W) \leq \mu, \end{aligned} \tag{1.3}$$

where $\|\cdot\|_*$ is the trace norm, defined by the sum of all singular values of the matrix in question. The convexity of (1.3) implies that (1) there is no local minimum and (2) it can be tractably solved by, say, the interior-point method. Besides the interior-point method, which only applies to low-dimensional W, modern approaches such as proximal algorithms and Frank-Wolfe algorithms are proposed for high-dimensional cases. In particular, Frank-Wolfe algorithms are extremely suitable for high-dimensional cases because of its low (quadratic) computation complexity per iteration, contrary to proximal methods, which rely on singular value decomposition entailing high (cubic) computation complexity.

It is well known that the matrix factorization approach (1.2) can be easily and has been implemented on parallel and distributed infrastructures, whereas there is limited literature about solving (1.3) under such conditions. Therefore, there is scientific interest to investigate the possibility of solving (1.3) in a parallel or distributed way. Besides the scientific relevance, this study has practical usages as well. Let us consider the kind of ℓ which can be written as a sum of functions: $\ell(\cdot) := \sum_{i=1}^{n} f_i(\cdot)$, which is a typical scenario in machine learning and statistics. In a machine learning context, f_i can be the loss incurred by the i-th data point. In a statistics context, it can be the negative log-density of the i-th sample. When n is large, very common in our big data era, we have to spend a lot of time on the evaluation of the gradient. The time spent on the gradient is especially a problem for Frank-Wolfe algorithms. The low computation complexity of Frank-Wolfe algorithms makes the evaluation of the gradient the most time-consuming task. It is obvious that we can conduct the gradient evaluation in a parallel or distributed manner. If we can further conduct Frank-Wolfe algorithms in the same way, we will be able to improve the efficiency greatly and to develop solvers as efficient as the state-of-the-art ones for the matrix factorization. Moreover, because of the existence of the regularization path adaption for Frank-Wolfe which alleviates the computation cost associated with the hyperparameter choice (Jaggi 2011, Chapter 6), we will be eventually able to outperform the matrix factorization approach, where the regularization path possibility is not evident. Therefore, there are both scientific and practical motivations to carry out this research.

In response to this motivation, this study aims at exploring the possibility of executing the Frank-Wolfe algorithm in a star network with the Bulk Synchronous Parallel (BSP) model. I chose Frank-Wolfe, instead of other solvers, because it is the algorithm the most friendly to high-dimensional dataset and also because it can benefit the most from parallelization and distribution. The difference between parallelization and distribution lies in the hypothesis upon the *memory*. The former assumes the availability of a shared memory among all parties, whereas the latter does not. Since algorithms working in the distributed case also work in the parallel case, I chose therefore to study the more complex one, the distributed case. The BSP model in the star network is the most straightforward computation model. Its behavior is the closest to a serial computation model. In addition, the BSP model is common across various frameworks, which makes it easy to be adopted by the industry. In short, this study is a preparation for high-performance computing (HPC) in trace norm minimization involving datasets both high-dimensional and huge in size. Its achievement will have an immediate impact on the industry.

Regarding trace norm minimization, there are two other equivalent formulations with regard to (1.3). Instead of representing it as a constrained optimization problem, we can use the trace norm as a regularization like

$$\min_{W \in \mathbb{R}^{d \times m}} \ell(W) + \lambda \|W\|_* , \tag{1.4}$$

where λ is some predefined parameter. This formulation is called trace norm regularization and is the standard form used by proximal methods. The other one switches the objective function and the constraint in (1.3) and has this form:

$$\begin{aligned} \min_{W \in \mathbb{R}^{d \times m}} \quad & \ell(W) \\ \text{s.t.} \quad & \|W\|_* \leq \theta, \end{aligned} \tag{1.5}$$

where θ is some predefined parameter. This formulation is called optimization with trace norm constraint, which is an instance of the generalized matching pursuit (Locatello et al. 2017), and is the standard form used by Frank-Wolfe methods. All of these three are mathematically equivalent. In my dissertation, I refer to them all as trace norm minimization without further distinction, but I will stick to the formulation (1.5) throughout my thesis for its synergy with Frank-Wolfe.

I consider the kind of objective functions that can be (naturally) decomposed as a sum of functions: $\ell(\cdot) := \sum_i f_i$. Each party i (except for the center) in the star network holds a unique copy of f_i. They communicate with the center to solve (1.5) by Frank-Wolfe algorithms. The main difficulty here is to solve the linear subproblem in a distributed context. This research was conducted first to design various strategies to solve the linear subproblem and then to investigate their performance both experimentally and theoretically.

The outline of my dissertation is as follows. Chapter 2 serves as the preliminary, which presents the information necessary for a better understanding of the remaining part of my dissertation. The majority of concepts introduced there are well-established ones, though some extension and innovation are specially made for the reference in the following chapters. Chapter 3 lays the theoretical foundation of using power iteration or Lanczos iteration to solve the Frank-Wolfe linear subproblem. It starts with a general formulation which applies to any type of subproblem

and then shows its relevance with trace norm minimization. This chapter does not discuss any distributed computing, but its conclusion also applies in my BSP model. Chapter 4 discusses four *generic* strategies to conduct the distributed computation of the Frank-Wolfe linear subproblem. Besides, the method for the line search as well as the recursive updates are described for several *specific* objective functions. The experiment results on Apache SPARK follow immediately.

Contributions

The main contributions of this study are three-fold. In the theoretical aspect, it generalizes the deterministic sublinear convergence rate and extends it to nondeterministic cases. This generalization is essential for trace norm minimization, for its linear subproblem cannot be solved with a deterministic precision. Jaggi (2013) has made the pioneering work in pointing out the necessary Power/Lanczos iterations needed in each Frank-Wolfe iteration. However, it is extremely concise and uses the deterministic language, which is inappropriate in a nondeterministic context. This study repairs this imperfection and makes it rigorous.

In the algorithmic aspect, this study proposes and analyzes four strategies to solve the linear subproblem as well as methods for the line search under the BSP model. Moreover, noticing Frank-Wolfe's rank-1 update property, it updates the gradient recursively, with either a dense or low-rank representation, instead of repeatedly recalculating it from scratch. All of these designs are generic and apply to *any* distributed infrastructures compatible with the BSP model. In both the theoretical and the algorithmic aspect, several types of matrices (positive semi-definite, symmetric and asymmetric) are investigated.

In the empirical aspect, this study tests the proposed algorithmic designs in an Apache SPARK cluster. According to the experiment results, for the linear subproblem, centralizing the gradient or averaging the singular vectors is sufficient in the low-dimensional case, whereas distributed power iteration, with as few as one or two iterations per epoch, excels in the high-dimensional case. The Python package developed for the experiments is modular, extensible, and ready to deploy in an industrial context. The code involves only asymmetric matrices, but the conclusions drawn from the experiments are general for all types of matrices.

Related work

There have been some efforts to make Frank-Wolfe algorithm distributed. Most of them study parameters in vector spaces as opposed to matrix spaces as I do. Bellet et al. (2014) proposed a generic Frank-Wolfe framework for ℓ_1-norm minimization, whereas I target the trace norm minimization problem, where I cannot use the same technique to solve the linear subproblem as in ℓ_1-norm minimization. Wang et al. (2016) proposed a parallel and distributed version of the Block-Coordinate Frank-Wolfe algorithm (Lacoste-Julien et al. 2012). Their work is about a specific Frank-Wolfe used for a specific problem, whereas I work on the standard Frank-Wolfe, and

my framework is generic. Moharrer and Ioannidis (2017) are particularly interested in the map-reduce type framework. They identified two properties that can make the map-reduce applicable. My work has two differences from theirs. On the one hand, my framework is not specific to a particular implementation and can be implemented by either map-reduce or message passing. On the other hand, I am particularly interested in the matrix parameter instead of the vector parameter as in their work. Concerning the implementation, we share a lot in common. We both use Apache SPARK (Zaharia et al. 2012a) and map-reduce (Dean and Ghemawat 2008) and make use of SPARK's particular properties to accelerate the calculation. For instance, the core concept, called *common information*, in their paper also enjoys a recursive update. Wai et al. (2017a) proposed a decentralized Frank-Wolfe framework for general problems, which is best when the parameter in each node is sparse, whereas I do not make this hypothesis. Besides, their network is decentralized, whereas I apply BSP model on a star network.

Based on the work of Wai et al. (2017a), for the trace norm minimization problem, the same authors further incorporated a decentralized power method to solve the FW linear subproblem with the aim of reducing the communication cost (Wai et al. 2017b). Although we all study trace norm minimization and use the power method, my work has at least four differences from theirs. First, their algorithm is based on the Gossip protocol and works on any network topology, whereas I assume the availability of a master-slaves star network and can hence have much less communication overhead. Second, they proved the convergence in probability for their Gossip-based algorithm, whereas I prove the convergence in both probability and in expectation for the master-slave scenario. Third, in their paper, they provided some primary experiment results with a single-thread MATLAB environment, whereas I demonstrate my algorithms on a physical cluster and with larger-scale datasets. Last but not least, they used more than 6 power iterations per FW iteration, whereas I use as few as 1 or 2 power iterations.

Chapter 2

Background

This chapter prepares the necessary knowledge that will be used later in other chapters. It first reminds the reader of the core concepts in convex optimization. This knowledge is essential to Frank-Wolfe in that the latter is more built around the convexity than any other algorithms. Meanwhile, this knowledge is also standard, and anyone having worked on convex optimization has at least studied it once. Although I have made some extensions, it does not prevent experts from skipping these sections and referring to them later when needed. After two sections of core concepts, it gives a detailed presentation on trace norm minimization as well as the various solvers associated. Then, it introduces multi-task learning, which is the main problem I use as proof of concept in my experiments. To bring refreshing ideas to the multi-task learning community, I deliberately adopt the econometrician's viewpoint. Last but not least, it presents the general information about distributed machine learning, including the Bulk Synchronous Parallel and the star network, whose importance in my dissertation is self-evident. Besides, it includes some of my personal thoughts about large-scale machine learning.

2.1 Convexity and smoothness

This section describes the convexity and the smoothness, which are essential for convex optimization. The material is mainly from two classic books – Boyd and Vandenberghe (2004) and Nesterov (2013). The most important content in this subsection is the extension of some classical results in Nesterov (2013). In particular, Corollary 2.1.7 and Theorem 2.1.8 will be the building bricks of the next chapter.

2.1.1 Convex set

A set C is *convex* if the line segment between any two points in C lies in C, *i.e.*, if for any $x_1, x_2 \in C$ and any θ with $0 \leq \theta \leq 1$, we have

$$\theta x_1 + (1 - \theta) x_2 \in C.$$

We call a point of the form $\theta_1 x_1 + \cdots + \theta_k x_k$ a *convex combination* of points $x_1, \ldots, x_k$, where $\theta_1 + \cdots + \theta_k = 1$ and $\theta_i \geq 0$, $i = 1, \ldots, k$. Obviously, a set is convex if and only if it contains every combination of its points.

The *convex hull* of a set S, denoted $\mathbf{conv}(S)$, is the set of all convex combinations of points in S:

$$\mathbf{conv}(S) = \{\theta_1 x_1 + \cdots + \theta_k x_k | x_i \in S, \theta_i \geq 0, \theta_1 + \cdots + \theta_k = 1\}.$$

Obviously, a convex hull is convex. Furthermore, it is also the smallest convex set that contains S. That is, if C is convex and $S \subset C$, then $\mathbf{conv}(S) \subset C$.

Notable convex sets include polytopes and simplexes. A *polytope* is a bounded solution set of a finite number of linear equalities and inequalities:

$$\mathcal{P} = \{x | a_i^T x \leq b_i, i = 1, \ldots, m, c_j^T x = d_j, j = 1, \ldots, p\}.$$

A *simplex* is a kind of polytope. Let $v_0, \ldots, v_k \in \mathbb{R}^n$ be $k+1$ points affine independent, which means that $v_1 - v_0, \ldots, v_k - v_0$ are linearly independent. The simplex determined by them is given by

$$C = \mathbf{conv}\{v_0, \ldots, v_k\},$$

i.e., the set of all convex combinations of the $k+1$ points $v_0, \ldots, v_k$. The dimension of this simplex is defined as k. We sometimes refer to it as a k-dimensional simplex in $\mathbb{R}^n$.

2.1.2 Convex function

A function $f : \mathbb{R}^n \to \mathbb{R}$ is *convex* if its domain $\mathbf{dom} f$ is a convex set and if for all $x, y \in \mathbf{dom} f$ and θ with $0 \leq \theta \leq 1$, we have

$$f(\theta x + (1-\theta)y) \leq \theta f(x) + (1-\theta) f(y). \tag{2.1}$$

Geometrically, the chord of any two points on the graph lies above the graph of f. A function f is *strictly convex* if strict inequality holds whenever $x \neq y$ and $0 < \theta < 1$. We say f is *concave* if $-f$ is convex, and *strictly concave* if $-f$ is strictly convex.

A convex function is a continuous function (Rudin and others 1964, Chapter 4 Problem 23 in the 3rd edition). However, it is not necessarily differentiable. In fact, the absolute value function is convex, but it is not differentiable at the origin. In the case where f *is* differentiable, we have the following first order condition of convexity (Boyd and Vandenberghe 2004, Section 2.1.3).

Proposition 2.1.1 (**First order condition**). *A differentiable function f is convex if and only if*

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) \tag{2.2}$$

holds for all $x, y \in \mathbf{dom} f$, where ∇f is the gradient of f.

If the function f is twice differentiable, we can express the convexity even more concisely as follows (Boyd and Vandenberghe 2004, Section 2.1.4).

Proposition 2.1.2 (**Second order condition**). *A twice differentiable function f is convex if and only if its Hessian is positive semidefinite:*

$$\nabla^2 f(x) \succeq 0 \tag{2.3}$$

for all $x \in \mathbf{dom} f$.

Remark. Actually, the twice differentiability is already implied in the convexity. Indeed, if U is an open subset of $\mathbb{R}^n$ and $f : U \to \mathbb{R}^m$ is a convex function, then f has a second derivative almost everywhere. This is by Alexandrov Theorem (Busemann and Feller 1936, Aleksandorov 1939).

Similarly, f is concave if and only if $\nabla^2 f(x) \preceq 0$ for all $x \in \mathbf{dom} f$. Strict convexity can be partially expressed by the second order condition. If $\nabla^2 f(x) \succ 0$ for all $x \in \mathbf{dom} f$, then f is strictly convex. The converse, however, is not true: the function $f(x) = x^4$ is strictly convex but has zero second derivative at $x = 0$.

2.1.3 Jensen's inequality

Inequality (2.1) is sometimes called *Jensen's inequality*. It can be easily extended to convex combinations of more than two points: for a convex function f, k points $x_1, \ldots, x_k \in \mathbf{dom} f$, and $\theta_1, \ldots, \theta_k \geq 0$ with $\theta_1 + \cdots + \theta_k = 1$, we have

$$f(\theta_1 x_1 + \cdots + \theta_k x_k) \leq \theta_1 f(x_1) + \cdots + \theta_k f(x_k).$$

Moreover, the inequality extends to infinite sums, integrals, and expected values. For instance, given a convex function f, for $p(x) \geq 0$ on $S \subset \mathbf{dom} f$ with $\int_S p(x)\,\mathrm{d}x = 1$, we have

$$f\left(\int_S p(x) x\,\mathrm{d}x\right) \leq \int_S f(x) p(x)\,\mathrm{d}x$$

provided that the above integrals exist.

This idea generalizes naturally to any probability measure with support in $\mathbf{dom} f$. Given a convex function f, if x is a random variable such that $x \in \mathbf{dom} f$ with probability one, then we have

$$f(\mathbb{E}x) \leq \mathbb{E} f(x),$$

provided that the above expectations exist.

All of these inequalities are called *Jensen's inequality* today and are widely used in many domains,. This fact reflects the importance of convex functions.

2.1.4 Strong convexity

Strong convexity is a condition stronger than the convexity alone and hence entails better results in convex optimization. We say that a twice differentiable function f is *strongly convex* on its domain if there exists an $m > 0$ such that

$$\nabla^2 f(x) \succeq mI \tag{2.4}$$

for all $x \in \mathbf{dom} f$, where I is the identity matrix. (For matrices A and B, we write $A \succeq B$ if $A - B$ is positive semidefinite.)

Strong convexity can also be alternatively expressed in the form of the first order condition (Boyd and Vandenberghe 2004, Section 5.1.2) as follows.

Proposition 2.1.3. *A m-strongly convex function f satisfies*

$$f(y) \geq f(x) + \nabla f(x)^T (y - x) + \frac{m}{2} \|y - x\|^2 \tag{2.5}$$

for all x and y in $\mathbf{dom} f$.

It is worth noting that (2.5) does not require that the Hessian of f exist. Therefore, some people use this inequality instead as the definition of strong convexity. Furthermore, it gives the geometric interpretation of strong convexity. As a special case, when $m = 0$, we recover the first order condition (2.2). If $m > 0$, the inequality (2.5) is tighter than (2.2).

2.1.5 Lipschitz continuity

Besides the convexity, another important concept in optimization literature is *Lipschitz continuity*. It describes the smoothness of a function. For a subset $Q \subset \mathbb{R}^n$, a function f is *L-Lipschitz continuous* on Q (with regard to Euclidean norm $\|\cdot\|$) if there exists a constant L such that

$$\|f(x) - f(y)\| \leq L \|x - y\| \tag{2.6}$$

for all $x, y \in Q$.

With this concept, we shall introduce several classes of functions. We denote by $C_L^{p,k}(Q)$ the class of functions with the following properties.

- Any $f \in C_L^{p,k}(Q)$ is p times continuously differentiable on Q.
- Its kth derivative is Lipschitz continuous on Q with the constant L:

  $$\left\| f^{(k)}(x) - f^{(k)}(y) \right\| \leq L \|x - y\| \tag{2.7}$$

  for all $x, y \in Q$.

Obviously, we always have $p \geq k$, and we have $C_L^{q,k}(Q) \subset C_L^{p,k}(Q)$ for any constant $q \geq p$. In addition, we denote by $C^p(Q)$ the class of functions with p times continuous derivative on Q.

One of the most important classes in the optimization community is $C_L^{1,1}(Q)$. However, it is technically painful to verify via (2.7) whether a function belongs to this class. Therefore, researchers work with its subset $C_L^{2,1}(Q)$ instead since it is easier to verify whether a function belongs to this subset by using the following proposition (Nesterov 2013, Lemma 1.2.2).

Proposition 2.1.4. *Let $\|\cdot\|_{sp}$ be the spectral norm. A function $f \in C^2(\mathbb{R}^n)$ belongs to $C_L^{2,1}(\mathbb{R}^n)$ if and only if*

$$\left\|f''(x)\right\|_{sp} \leq L, \quad \forall x \in \mathbb{R}^n. \tag{2.8}$$

Remark. The above proposition requires f to be twice continuously differentiable. Actually, a similar result holds for functions with *absolutely continuous*[1] derivatives. Let us consider, say, the one-dimensional case and regard the function g below as the derivative of f. An absolutely continuous function is differentiable almost everywhere (cf. Rademacher's theorem in Federer 2014, Theorem 3.1.6; and in Heinonen 2005, Theorem 3.1). If an absolutely continuous function g is further Lipschitz continuous with the constant L, then its derivative g' is *essentially bounded*[2] in magnitude by the Lipschitz constant L. Conversely, if g is absolutely continuous and satisfies $|g'(x)| \leq L$ for almost all x in the domain, then g is Lipschitz continuous with Lipschitz constant at most L (Gariepy and Ziemer 1995).

The following statement is important, for it gives the geometric interpretation of functions from $C_L^{1,1}(\mathbb{R}^n)$ (Nesterov 2013, Lemma 1.2.3).

Proposition 2.1.5. *Let $f \in C_L^{1,1}(\mathbb{R}^n)$. Then, for any x, y from $\mathbb{R}^n$, we have*

$$\left|f(y) - f(x) - \nabla f(x)^T(y - x)\right| \leq \tfrac{L}{2}\|y - x\|^2. \tag{2.9}$$

The inequality (2.9) gives exactly the opposite direction of the inequality of (2.5). By combining these two, we can prove many important results in convex optimization as are shown in the next section.

2.1.6 Lipschitz continuity wrt. an arbitrary norm

This subsection discusses Lipschitz continuity with regard to an arbitrary norm (instead of the Euclidean norm). To simplify the notation, only in this section, I denote $\|\cdot\|$ as an arbitrary norm in this subsection, instead of the Euclidean norm.

Before the main result, I have to introduce the concept of *dual norm*. Let $\|\cdot\|$ be an arbitrary norm on a Hilbert space. Its *dual norm* $\|\cdot\|_*$ (on the same space) is defined by

$$\|u\|_* := \sup\{\langle u, x\rangle \mid \|x\| \leq 1\}.$$

For example, the dual norm of $\|\cdot\|_p$ is $\|\cdot\|_q$, where $\frac{1}{p} + \frac{1}{q} = 1$ and $p, q \geq 1$. Given a matrix $X \in \mathbb{R}^{m\times n}$, its *spectral norm* is defined by

$$\|X\|_{\text{sp}} = \max_{i\in\{1,\ldots,m\wedge n\}} \sigma_i(X),$$

[1] Let J be an interval in the real line $\mathbb{R}$. A function $f : J \to \mathbb{R}$ is *absolutely continuous* on J if for every positive number ε, there is a positive number δ such that whenever a finite sequence of pairwise disjoint sub-interval (x_k, y_k) of J with $x_k, y_k \in J$ satisfies $\sum_k (y_k - x_k) < \delta$ then $\sum_k |f(y_k) - f(x_k)| < \varepsilon$.

[2] (X, Σ, μ) is a measure space and assume that the function f is measurable. A number a is called an *essential upper bound* of f if the measurable set $f^{-1}(a, \infty)$ is a set of measure zero, *i.e.*, if $f(x) \leq a$ for almost all x in X.

where $m \wedge n$ means the minimum of m and n, and its *trace norm* (*a.k.a. nuclear norm*) is defined by

$$\|X\|_{\mathrm{tr}} = \sum_{i=1}^{m \wedge n} \sigma_i(X),$$

where $\sigma_i(X)$ is the ith largest singular value of matrix X. The trace norm and the spectral norm are mutually dual norm of each other (Fazel et al. 2001).

The following definition generalizes the Lipschitz continuity with regard to the Euclidean norm. It will be used as a condition in the next chapter.

Definition 2.1.1. A function $f : \mathbb{R}^n \to \mathbb{R}$ is called L-Lipschitz continuous with regard to an arbitrary norm $\|\cdot\|$ if there exists a constant L such that

$$|f(x) - f(y)| \leq L \|x - y\|$$

for all $x, y \in \mathbb{R}^n$.

The following theorem gives an equivalent formulation of the above definition when the function in question is continuously differentiable. It plays an important role in the demonstration of the sublinear convergence rate for trace norm minimization.

Theorem 2.1.6. *A continuously differentiable function $f : \mathbb{R}^n \to \mathbb{R}$ is L-Lipschitz continuous with regard to an arbitrary norm $\|\cdot\|$ if and only if the dual norm $\|\cdot\|_*$ of its gradient is inferior to L:*

$$\|\nabla f(x)\|_* \leq L$$

Proof. The proof is a direct adaption of the one for Lemma 1.2.2 in Nesterov (2013). The "if" part. For any $x, y \in \mathbb{R}^n$ we have

$$\begin{aligned} f(y) &= f(x) + \int_0^1 \nabla f(x + \tau(y - x))^T (y - x) \, \mathrm{d}\tau \\ &= f(x) + \left(\int_0^1 \nabla f(x + \tau(y - x)) \, \mathrm{d}\tau \right)^T (y - x). \end{aligned}$$

The first integral in the above equation is a line integral of a vector field. Since the vector field is the gradient of a scalar field, the integral is independent to the path and the *fundamental theorem of calculus*[3] holds.

With the upper bound of the dual norm of the gradient, we have

$$\begin{aligned} |f(x) - f(y)| &= \left| \left(\int_0^1 \nabla f(x + \tau(y - x)) \, \mathrm{d}\tau \right)^T (y - x) \right| \\ &\leq \left\| \int_0^1 \nabla f(x + \tau(y - x)) \, \mathrm{d}\tau \right\|_* \|y - x\| \\ &\leq \int_0^1 \|\nabla f(x + \tau(y - x))\|_* \, \mathrm{d}\tau \, \|y - x\| \\ &\leq L \|y - x\| . \end{aligned}$$

[3] The *fundamental theorem of calculus* along curves states that if $f(z)$ has a continuous indefinite integral $F(z)$ in a region R containing a parameterized curve $\gamma : z = z(t)$ for $\alpha \leq t \leq \beta$, then $\int_\gamma f(z) \, \mathrm{d}z = F(z(\beta)) - F(z(\alpha))$.

The first inequality is straightforward from the definition of the dual norm and can be considered as an extension of Cauchy-Schwarz inequality.

The "only if" part. For any $x, s \in \mathbb{R}^n$ and $\alpha > 0$, we have

$$\left| \left(\int_0^\alpha \nabla f(x + \tau s) \, \mathrm{d}\tau \right)^T s \right| = |f(x + \alpha s) - f(x)| \leq \alpha L \|s\| .$$

Dividing this inequality by α and taking the limit as $\alpha \downarrow 0$, by the continuity of the gradient, we get what we want. □

Remark. The continuous differentiability is not necessary. In fact, it holds for all absolutely continuous functions with the modification that the dual norm of its gradient is *essentially bounded* by L. To prove this, we can use the *Lebesgue differentiation theorem* (Lebesgue 1910) at the last step of the "only if" part.

Remark. This result can be extended to vector-valued function. In this case, the absolute value in the definition of Lipschitz continuity should be replaced by the same norm in question, and the dual norm in the theorem should be replaced by the *operator norm*. With this extension, we can get analogous results for higher-order derivatives.

The corollary below is a special case of Theorem 2.1.6 and is the exact form I will use in the next chapter.

Corollary 2.1.7. *A continuous differentiable matrix function $f : \mathbb{R}^{m \times n} \to \mathbb{R}$ is L-Lipschitz continuous with regard to the trace norm $\|\cdot\|_{tr}$ if and only if the spectral norm $\|\cdot\|_{sp}$ of its gradient is inferior to L.*

The following theorem is analogous to Proposition 2.1.5. It extends the result to the Lipschitz continuity. It will be used to upper bound the *global curvature*, defined in the next chapter, of a function.

Theorem 2.1.8. *If ∇f is L-Lipschitz continuous with respect to some arbitrary norm $\|\cdot\|$ in dual pairing*

$$\|\nabla f(x) - \nabla f(y)\|_* \leq L \|x - y\| , \tag{2.10}$$

then

$$\left| f(y) - f(x) - \left\langle f'(x), y - x \right\rangle \right| \leq \tfrac{L}{2} \|y - x\|^2 . \tag{2.11}$$

In particular, since Euclidean norm's dual norm is itself, Proposition 2.1.5 is a special case of this theorem.

Proof. The proof is a direct adaption of the one for Lemma 1.2.3 in Nesterov (2013).

For any x, y in the domain we have

$$\begin{aligned} f(y) &= f(x) + \int_0^1 \left\langle f'(x + \tau(y - x)), y - x \right\rangle \mathrm{d}\tau \\ &= f(x) + \left\langle f'(x), y - x \right\rangle + \int_0^1 \left\langle f'(x + \tau(y - x)) - f'(x), y - x \right\rangle \mathrm{d}\tau. \end{aligned}$$

Therefore

$$\begin{aligned}
&\left|f(y) - f(x) - \left\langle f'(x), y - x\right\rangle\right| \\
=&\left|\int_0^1 \left\langle f'(x + \tau(y - x)) - f'(x), y - x\right\rangle \mathrm{d}\tau\right| \\
\leq&\int_0^1 \left|\left\langle f'(x + \tau(y - x)) - f'(x), y - x\right\rangle\right| \mathrm{d}\tau \\
\leq&\int_0^1 \left\|f'(x + \tau(y - x)) - f'(x)\right\|_* \|y - x\| \,\mathrm{d}\tau \\
\leq&\int_0^1 \tau L \|y - x\|^2 \,\mathrm{d}\tau = \tfrac{L}{2} \|y - x\|^2
\end{aligned}$$

□

2.2 Convex optimization

This section presents some important knowledge about the convex optimization (*e.g.*, descent methods, line search). Also, it explains the reason of the equivalence of the three formulations, described in the Introduction chapter, of trace norm minimization. The material in this section is mainly from Boyd and Vandenberghe (2004). The most important content in this section is the *steepest descent*. I will make link between it and Frank-Wolfe algorithms in the next chapter.

2.2.1 Definition

There are two types of convex optimization problems – *unconstrained* and *constrained*. An unconstrained minimization problem is defined by

$$\min_x f(x)$$

for the convex *objective function* $f : \mathbb{R}^n \to \mathbb{R}$. A constrained minimization problem is defined by

$$\min_{x \in S} f(x),$$

where S is a convex set. This section starts with the unconstrained problems and ends with the constrained problems.

We assume here that the unconstrained problem is solvable, *i.e.*, there exists an optimal point x^* such that $\inf_x f(x) = f(x^*)$. We denote this minimal value as p^* in this section.

Since f is differentiable and convex, a necessary and sufficient condition for a point x^* to be optimal is

$$\nabla f(x^*) = 0$$

(Boyd and Vandenberghe 2004, Section 4.2.3). Except for only a few special cases, where we can solve this equation analytically, we usually resort to an iterative algorithm, which finds a sequence of points $x^{(0)}, x^{(1)}, \ldots \in \mathbf{dom} f$ with $f(x^{(k)}) \to p^*$ as $k \to \infty$.

2.2.2 Useful inequalities

This subsection presents some useful inequalities without proving them. They are results directly derived from (2.5) and (2.9) (Boyd and Vandenberghe 2004, Section 9.1.2).

If the function f is strongly convex with a constant $m > 0$ (or the eigenvalues of its Hessian are lower bounded by m), we have

$$\frac{m}{2} \|x^* - x\|_2^2 \leq f(x) - p^* \leq \frac{1}{2m} \|\nabla f(x)\|_2^2 . \tag{2.12}$$

If the gradient ∇f is M-Lipschitz continuous with regard to the Euclidean norm (or the eigenvalues of the Hessian are upper bounded by M), we have

$$\frac{M}{2} \|x^* - x\|_2^2 \geq f(x) - p^* \geq \frac{1}{2M} \|\nabla f(x)\|_2^2 . \tag{2.13}$$

This M-Lipschitz condition is also known as the M-smoothness. The above inequalities show the relation between three important quantities in optimization – the gradient, the primal gap (*i.e.*, $f(x) - p^*$), and the distance to the optimal point.

For a m-strongly convex, M-smooth function f, we define its *condition number* $\kappa := \frac{m}{M}$. This quantity characterizes the optimizing difficulty of the function f in using first-order algorithms.

2.2.3 Descent methods

A *first-order* algorithm is an algorithm using only the gradient information of the objective function, such as the gradient descent. A *second-order* algorithm is an algorithm using also the Hessian information of the objective function, such as the Newton method.

We are going to use iterative algorithms to solve the problem. They produce a sequence $x^{(k)}, k = 1, 2, \ldots$ by the iterative equation

$$x^{(k+1)} = x^{(k)} + t^{(k)} \Delta x^{(k)},$$

where $t^{(k)} > 0$ (except when $x^{(k)}$ is optimal) is called the *step size* at Iteration k.

We focus on *descent methods*, which means that

$$f(x^{(k+1)}) < f(x^{(k)}),$$

except when $x^{(k)}$ is optimal. Inequality (2.2) implies that this is equivalent to making the search direction $\Delta x^{(k)}$ satisfy

$$\nabla f(x^{(k)})^T \Delta x^{(k)} < 0,$$

i.e., it must make an acute angle with the negative gradient. We name such a direction a *descent direction*.

Algorithm 1 gives the outline of a general descent method, which contains three steps – determining a descent direction Δx, selecting a step size t, and updating. The

Algorithm 1. General descent method

Input: a convex function f, an initial point x.
repeat
Determine a descent direction Δx.
Line search. Choose a step size $t > 0$.
Update. $x \leftarrow x + t\Delta x$.
until stopping criterion is satisfied.

second step is called the line search since the selection of the step size t determines where at the ray $\{x + t\Delta x | t \in \mathbb{R}_+\}$ the next iterate will be.

When people talk about the line search, they usually mean the *exact* line search. That is, t is chosen to minimize f along the ray $\{x + t\Delta x | t \geq 0\}$:

$$t = \arg\min_{s \geq 0} f(x + s\Delta x).$$

This optimization is easier to solve than the original one for its single dimensionality. In some special cases, we have even analytical solutions. Boyd and Vandenberghe (2004, Section 9.2) also describe a backtracking line search, which is an inexact line search and can be used when the exact line search is computationally expensive.

2.2.4 Gradient descent

The most popular descent method may be the *gradient descent*, which uses the negative gradient directly as the descent direction and enjoys the following convergence rate (Boyd and Vandenberghe 2004, (9.18)).

Proposition 2.2.1. *For an m-strongly convex, M-Lipschitz continuous function f, by using the gradient descent with exact line search, we have*

$$f(x^{(k)}) - p^* \leq c^k (f(x^{(0)}) - p^*), \tag{2.14}$$

where $c = 1 - m/M < 1$.

We say that an algorithm *converges linearly*, if there exists a constant $c \in (0, 1)$ such that

$$\lim_{k \to \infty} \frac{\left| x^{(k+1)} - p^* \right|}{\left| x^{(k)} - p^* \right|} = c.$$

This number c is called *rate of convergence*. If $c = 1$, then we call it *sublinear convergence*. Obviously, the proposition above is an example of linear convergence. We shall see an example of sublinear convergence concerning Frank-Wolfe in the next chapter.

2.2.5 Steepest descent

Besides the negative gradient, there are many other choices (*e.g.*, Newton method) for the descent direction. Here, we are going to present a method named *steepest descent*.

Let $\|\cdot\|$ be a norm on $\mathbb{R}^n$. We define a *steepest descent direction* with regard to the norm $\|\cdot\|$ by

$$\Delta x_{\text{sd}} = \arg\min_v \{\nabla f(x)^T v \mid \|v\| = 1\}.$$

This direction points toward the point on the unit ball which gives the largest decrease in the linear approximation of f.

We can use various norms for this purpose and, with different choices, we get different steepest descent directions. In particular, if we choose $\|\cdot\|_2$, then the direction we get is exactly the negative gradient, and it degenerates to gradient descent. Steepest descent covers gradient descent as a special instance. Regarding the convergence rate, steepest descent with line search converges linearly (Boyd and Vandenberghe 2004, Section 9.4.3).

2.2.6 Constrained optimization

This subsection briefly discusses about the *constrained optimization*. A general constrained optimization can take the following form

$$\min_{x \in C} f(x),$$

where C is a convex set and f is a convex function. In most cases, the convex set C can be explicitly specified like

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & g_i(x) \leq 0 \quad \forall i \in [m] \\ & h_j(x) = 0 \quad \forall j \in [n], \end{aligned}$$

where all g_i and h_j are convex functions. The first group of constraints is called *inequality constraints*, while the second group *equality constraints*.

For equality constraints, we can use the *method of Lagrange multipliers* (Lagrange 1811). For inequality constraints, there exist many approaches. The most popular may be Dantzig's *simplex method* for *linear programming* (LP). It has an exponential time complexity in the worst case but is efficient for most of the time.

Interior-point methods are another important approach. It uses a barrier function to encode the constraint set into the objective function. When applied to LP, in contrast to the simplex method, which moves among the vertices of the simplex, the interior-point method moves inside the simplex, hence the name. The interior point method achieves polynomial time complexity.

The interior-point method is a second-order algorithm: it uses Newton direction for its descent direction. There exist also first-order methods, among which the *projected gradient descent*. This method alternates between a gradient descent and projection step

$$x \leftarrow \pi_C(x - \nabla f(x)),$$

where π_C is a projection operator that projects a point to the constraint set C. The most used projection may be

$$\pi_C(z) = \arg\min_{x \in C} \quad \tfrac{1}{2}\|x - z\|_2^2,$$

which finds inside C the closest point to z. Besides the ℓ_2-norm, other norms can also be used to define the projection operator. It is worth mention that this projection step is itself a constrained quadratic optimization, which can be potentially expensive.

Proximal gradient descent is another example. With the *Lagrange function*, we can reformulate the following optimization problem

$$\begin{aligned} \min_x \quad & f(x) \\ \text{s.t.} \quad & g(x) \le t \end{aligned}$$

as

$$\min_x \quad f(x) + \lambda g(x),$$

where g can be potentially non-differentiable. Here λ depends on t and is chosen accordingly to make these two optimization problems have the same optimum. Although this is an indirect way to solve constrained optimization problem by transforming it to an unconstrained problem, it works well when the constraint is "soft", such as in Lasso and trace norm minimization, where we often find ourselves solving the problem with various values of t (or equivalently λ), which further gives birth to various regularization path algorithms. To cope with the non-differentiability of g, people use the *proximal operator*

$$\mathbf{prox}_{\lambda g}(x) := \arg\min_u \left(g(u) + \tfrac{1}{2\lambda} \|u - x\|_2^2 \right).$$

The whole algorithm can be concisely expressed as

$$x \leftarrow \mathbf{prox}_{\lambda g}(x - \lambda \nabla f(x)).$$

Last but not least, *Frank-Wolfe algorithm* (Frank and Wolfe 1956) recently became an important first-order algorithm for constrained optimization problems whose constraint set is bounded. By solving a linear subproblem on the constraint set, it avoids the potentially expensive computation cost of evaluating the projection operator or proximal operator.

2.3 Trace norm minimization

This section gives some examples of trace norm minimization and describes the main methods to solve it.

2.3.1 Lasso vs. trace norm minimization

One of the most popular research topics in recent years is *sparse* methods. This includes Lasso (Tibshirani 1996) and trace norm minimization. Lasso considers optimization problems of the following form

$$\min_{\beta \in \mathbb{R}^d} f(\beta) + \lambda \|\beta\|_1 , \tag{2.15}$$

while trace norm minimization considers

$$\min_{X \in \mathbb{R}^{n \times d}} f(X) + \lambda \|X\|_{\mathrm{tr}}\,, \tag{2.16}$$

where f is a matrix function.

These problems have a common property. They are both the (weighted) sum of a smooth convex loss function f and a non-smooth norm penalty (*e.g.*, $\|\cdot\|_1\,, \|\cdot\|_{\mathrm{tr}}$). It is widely known that (2.15) yields a *sparse* solution (*i.e.*, the optimal point has few non-zero entries), and that (2.16) yields a *low-rank* solution (*i.e.*, the optimal point is a low-rank matrix).

Grave et al. (2011) describe a *trace Lasso* problem, which solves

$$\min_{\beta \in \mathbb{R}^d} \|X\beta - Y\|_2^2 + \lambda \|X\mathbf{diag}(\beta)\|_{\mathrm{tr}}\,.$$

It also uses the trace norm except that its penalty is imposed on $X\mathbf{diag}(\beta)$. In this dissertation, we do not consider it as trace norm minimization.

2.3.2 Low-rank matrix completion

This thesis focuses on the trace norm minimization. Varying the objective function f, we get various specific problems. The first is the low-rank matrix completion problem

$$\min_{X \in \mathbb{R}^{n \times d}} \|\mathcal{P}_\Omega(X - Y)\|_{\mathrm{F}}^2 + \lambda \|X\|_{\mathrm{tr}}\,, \tag{2.17}$$

where $\Omega \subset [n] \times [d]$ is the support[4] of Y, $\mathcal{P}_\Omega$ is a projection operator that puts the coefficients on the complimentary set Ω^c to zero, and $\|\cdot\|_{\mathrm{F}}$ is *Frobenius norm* defined by

$$\|A\|_{\mathrm{F}} := \sqrt{\sum_{i,j} a_{ij}^2}.$$

This optimization problem is commonly seen in the recommender system literature. Indeed, let Y be the user rating matrix, whose rows represent users and whose columns represent items. Since users do not have the time and the energy to try all items, this rating matrix is extremely sparse. Ω represents the support where the ratings exist. One wants to find a matrix X close to Y not only on the support Ω but also on Ω^c. We can prove that, under certain conditions on Y, the solution of (2.17) is exactly Y.

Indeed, along the line of work (Candès and Recht 2012, Candès and Tao 2010, Recht 2011, Gross et al. 2010, Gross 2011, Chen et al. 2014), researchers have proven that, for a $d \times d$ matrix Y with rank r, when the observation is exact (*i.e.*, without noise), under certain conditions, one can recover the exact Y with as few as $O(dr \log^2 d)$ observations. When $r \ll d$ (*e.g.*, r is a small constant), we can recover the whole matrix Y with the observation size growing only linearly with the size of

[4] The set of the non-zero coefficients.

Y. They further show that trace norm minimization has the optimal sample complexity in recovering a low-rank matrix. Candès and Plan (2010) prove that, when the observation is corrupted (*i.e.*, with noise), with the same condition and sample size as above, trace norm minimization can recover the ground truth matrix but with an error proportional to the severity of the noise.

2.3.3 Trace regression

Another example is *trace regression* (Koltchinskii et al. 2011, Bach 2008). Assume that we observe n independent random pairs $(X_i, Y_i), i = 1, \cdots, n$, where X_i are random matrices with dimension $m_1 \times m_2$, and Y_i are generated in the following form

$$Y_i = \mathbf{tr}(X_i^T W) + \xi_i, \qquad i = 1, \cdots, n, \tag{2.18}$$

where $W \in \mathbb{R}^{m_1 \times m_2}$ is an unknown matrix, $\mathbf{tr}(A)$ is the trace of matrix A, and the noise variables ξ_i's are independent and have zero means.

The matrices X_i's are called the *design* of the regression, which can be either fixed or random. If they are i.i.d. uniformly distributed on the set

$$\chi = \{e_j(m_1)e_k^T(m_2), 1 \le j \le m_1, 1 \le k \le m_2\},$$

where $e_k(m)$ are the canonical basis vector in $\mathbb{R}^m$, (2.18) becomes the matrix completion problem mentioned in the last subsection. In other words, matrix completion is a special instance of trace regression.

When the design is nonrandom, people estimate W with the following optimization formulation

$$\min_W \frac{1}{n} \sum_{i=1}^{n} (Y_i - \mathbf{tr}(X_i^T W))^2 + \lambda \|W\|_{\text{tr}},$$

which is also called *matrix Lasso*. Matrix Lasso is another example of trace norm minimization.

2.3.4 Solvers

If the matrix size is not too large, the associated optimization can be solved by *semidefinite programming* (SDP) for some special cases. For example, we use

$$\begin{aligned} \min_{X \in \mathbb{R}^{n \times d}} \quad & \|X\|_{\text{tr}} \\ \text{s.t.} \quad & X_{ij} = Y_{ij} \quad \forall (i,j) \in \Omega \end{aligned}$$

for an exact recovery of the matrix Y. It can be considered as a special case of (2.17). Fazel et al. (2001, Section 3) show that it can be reformulated as

$$\begin{aligned} \min_{X,R,S} \quad & \mathbf{tr}(R) + \mathbf{tr}(S) \\ \text{s.t.} \quad & \begin{bmatrix} R & X \\ X^T & S \end{bmatrix} \succeq 0 \\ & X_{ij} = Y_{ij} \quad \forall (i,j) \in \Omega. \end{aligned}$$

The last equality constraint can be easily expressed as a group of *linear matrix inequalities* and hence make it a SDP problem.

SDP problems can be efficiently solved by the *interior-point method*, and the latter is provided by various numerical packages such as SDPT3 (Toh et al. 1999) and SeDeMi (Sturm 1999). However, the second-order nature of the interior-point method (*i.e.*, requires the computation of the Hessian matrix) makes it not amenable to matrices with large size. In addition, it is not evident how SDP can solve the more general (2.17).

For this purpose, Cai et al. (2010) propose a first-order *singular value thresholding* algorithm. It alternates between a shrinkage step (Ma et al. 2009, Definition 4) and a step similar to the gradient descent:

$$\begin{cases} X \leftarrow \text{shrink}(Z, \tau) \\ Z \leftarrow Z - \delta \nabla f(X), \end{cases}$$

where the shrinkage operator makes the underlying matrix Z's all singular values extracted by τ (it equals 0 if the singular value is already inferior to τ). This method can handle matrices with size 1000×1000 or sparse matrices of high dimension but with a very low rank.

At the same time period, Ma et al. (2009) propose the *fixed point continuation with approximate SVD* algorithm. It has a similar form

$$\begin{cases} X \leftarrow \text{shrink}(Z, \tau) \\ Z \leftarrow X - \delta \nabla f(X) \end{cases}$$

with the exception that it uses X instead of Z as the starting point in the descent step.

This shrinkage idea is closely related to the proximal method. Indeed, it is exactly the proximal operator

$$\mathbf{prox}_\tau(Z) := \arg\min_W \left\{ \tfrac{1}{2} \|W - Z\|_{\text{F}}^2 + \tau \|W\|_{\text{tr}} \right\}$$

(Cai et al. 2010, Theorem 2.1).

Based on the above work, Ji and Ye (2009) and Toh and Yun (2010) independantly propose an accelerated version. It is well-known in the optimization community that, when the objective function is smooth, the first-order algorithm can be accelerated to achieve the optimal convergence rate of $O(\frac{1}{k^2})$, where k is the number of iterations (Nesterov 2013). This method is called *Nesterov acceleration*. Nesterov and others (2007) and Beck and Teboulle (2009) discover that Lasso can be accelerated even though ℓ_1-norm is not smooth. Similarly, Ji and Ye (2009) and Toh and Yun (2010) give the Nesterov acceleration for the trace norm minimization even though the trace norm is not smooth either.

Because of the use of *singular value decomposition* (SVD), which incurs cubic computation complexity, the above algorithms are not amenable to general matrices with dimension higer than 1000. Hazan (2008), Jaggi et al. (2010) and the more

recent Garber (2016a) propose to use Frank-Wolfe algorithm to solve trace norm minimization. It is a first-order algorithm and hence does not need SVD. Its most expensive operation is the matrix-vector multiplication, which has square complexity as opposed to the cubic complexity of SVD. Therefore, this algorithm naturally applies in higher dimension. In addition, Giesen et al. (2012) provide for it a regularization path algorithm. With the same flavor, Dudik et al. (2012) apply the same algorithms to the regularization formulation instead of constraint formulation. Lastly, see Mu et al. (2016) for a combined use of both the proximal method and the Frank-Wolfe method.

There exist also numerous solvers that transform the trace norm minimization problem into the matrix factorization problem via the variational characterization of trace norm. Although this kind of solvers is amenable to large-scale problems and achieves good accuracy, their theoretical convergence has long been under doubt because of the non-convexity of its objective function. Only very recently, it is proved for a few specific objectives (*e.g.*, phase retrieval/synchronization, orthogonal tensor decomposition, dictionary learning, matrix sensing, matrix completion) that every local minimum must also be global (Ge et al. 2015, Sun et al. 2015, Bandeira et al. 2016, Ge et al. 2016, Bhojanapalli et al. 2016), but it remains as an open question for the general problem.

Another approach consists in searching for a low-rank matrix solution directly at the *Riemannian manifold* (Meyer et al. 2011, Vandereycken 2013) without using the convex relaxation. Lastly, Pong et al. (2010) derive a reduced dual formulation for this problem and solves it with gradient projection.

2.4 Multi-task learning

This section describes *multi-task learning* (MTL) (Caruana 1998), which will be the problem in my experiment session (namely multi-task least square and multinomial logistic regression). For a comprehensive exposition, I refer to the SDM 2012 tutorial (Zhou et al. 2012). They have also developed a package MALSAR, which contains various multi-task learning algorithms (Zhou et al. 2011a).

In contrast to the majority of multi-task learning literature, this thesis gives the introduction from an econometrician's point of view. For those who are not familiar with the econometrics, I recommend to skip the first subsection and to use Zhou et al. (2012) as an alternative.

2.4.1 Motivation and approaches

Multi-task learning is different from *single task learning* or *independent task learning*. While the latter learn one task at a time, the former learns all tasks at the same time – hence the name. If the various tasks are uncorrelated, there will be no interest to learn them all together. Multi-task learning is aimed at capturing the relatedness among tasks in order to improve the generalization capacity of every single task.

This approach is especially useful when there are insufficient samples for each single task. Although we can use sparse methods, the deficit of data is still embarrassing, for the convergence rate in high dimensional statistics is not always as fast as in low-dimensional statistics. Multi-task learning brings out another idea. Since there are a lot of interrelated databases[5], by linking them all together, we have a chance to get better results.

This idea is not restricted to the machine learning community. Statisticians, especially biometricians and econometricians, have used *panel data* or *longitudinal data* long ago (Laird and Ware 1982, Liang and Zeger 1986, Zeger and Liang 1986, Singer and Willett 2003, Diggle 2002). They use the *fixed effects* model or the *random effects* model to capture the variation caused by individuals and time. This is equivalent to linking several databases of different time (resp. of different individuals). By using the multi-task terminology, we can define the prediction for a specific time (resp. individual) as a task.

The above research domain is an excellent illustration of one of the two common approaches in multi-task learning – shared parameters. It uses the same set of parameters for each database, with the addition of *one* extra parameter for each single database. This technique greatly reduces the number of parameters and makes the previously data deficit situation data sufficient.

Sometimes, these shared parameters can appear as latent parameters. All databases share a common set of (latent) parameters, which transform the raw features to latent features. The number of latent features is sufficiently small, so that there will not be many (unshared) task-specific parameters mapping the latent features to the prediction. This is especially true for the *neural network* if we assume that each neuron at the output layer corresponds to a task (Caruana 1998). Other examples include shared parameter Gaussian process (Lawrence and Platt 2004) and common latent representation in nonparametric Bayesian models (Zhu et al. 2011).

The second approach is to add constraints on the parameters. Each task still has its own parameters, but the parameters of each task must be interrelated. The interrelation is imposed by adding a regularization term to the empirical risk. This approach is typically machine learning, and I have not found any equivalence in econometrics.

Evgeniou and Pontil (2004) assume that all tasks are related in that the models of all tasks come from a particular distribution. They hence suggest to penalize the deviation of each task from the mean

$$\min_{W} \quad \text{Loss}(W) + \lambda \sum_{t=1}^{T} \left\| w_t - \tfrac{1}{T} \sum_{s=1}^{T} w_s \right\|,$$

where each w_t is a column of W and corresponds to a task.

[5] See the third property of Big Data era in the preface.

Another way to capture the task relatedness is to constrain all parameters to share a common set of features. Zhou et al. (2011b) propose to use $\ell_{1,2}$-norm regularization

$$\min_{W} \quad \text{Loss}(W) + \lambda \sum_{i=1}^{p} \|W_i\|_2 \,,$$

where W_i is the ith row of the matrix W. Each W_i represents the weight for the ith feature across the tasks, and its ℓ_2-norm should not be too large. We sum these norms as the penalty, so that the solution will have few features. In other words, this approach imposes group sparsity – selecting few (common) features.

Besides the above regularization, people also use trace norm minimization (2.16). For example, Argyriou et al. (2008) use the trace norm as a proxy of $\ell_{1,2}$ norm, since group sparsity implies low-rank. In addition, low-rank matrices are an implicit way to impose the structure of shared latent parameters. Indeed, a low-rank matrix can be decomposed into two smaller-sized matrices. One can be the task-shared map from the raw features to the latent features, while the other can be the task-specific map from the latent features to the task prediction. More concretely, let us consider the prediction of the effect of a single drug on patients as one task. Since the drugs all have similar components, the effect of each drug can be see as the inner product between the effect of the (few) components and the percentage of the components in this drug. Therefore, the parameter matrix is a low-rank matrix, and the trace norm minimization naturally applies. Papers using trace norm minimization for multi-task learning include Ji and Ye (2009) and Pong et al. (2010).

2.4.2 Taxonomy

After a discussion on the approaches in last subsection, this subsection discusses the taxonomy of multi-task learning. According to the different nature of the tasks, it can be categorized as *multi-class learning*, *multi-label learning*, and general *multi-task learning* (Zhou et al. 2011a).

Multi-class learning consists in classifying several classes simultaneously. Given several or even many classes, we have to answer whether a sample belongs to a given class. Instead of replying yes or no for each class, which is the approach of multiple binary classification, multi-class learning replies directly which class the sample is from.

In multi-label learning, each instance can have multiple labels (*e.g.*, a man can simultaneously be handsome, rich and intelligent). If we consider the prediction of whether an instance has a specific label as one task, then the prediction of all labels this instance has is multi-label learning. If each instance is allowed to have only one label, it becomes multi-class learning.

In the above examples, the prediction is binary – belonging to a class or not or having a label or not. More generally, we have regression problems, where the prediction is a real number. Considering all these types of tasks, we call them all multi-task learning. In summary, multi-class learning is a special type of multi-label learning, and multi-label learning is a special type of multi-task learning.

2.4.3 Multi-task least square

In this and the next subsection, I will describe two instances of multi-task learning that I will use for my experiments. The first is multi-task least square regression defined by

$$\begin{aligned} \min_{W} \quad & \tfrac{1}{2} \|XW - Y\|_{\mathrm{F}}^2 \\ \text{s.t.} \quad & \|W\|_{\mathrm{tr}} \leq \theta, \end{aligned} \tag{2.19}$$

where $X_{n\times d}$ is the design matrix, *a.k.a.* the features, $Y_{n\times m}$ is the response matrix, and $W_{d\times m}$ is the parameter matrix. This problem is considered in Pong et al. (2010) and Toh and Yun (2010).

It can be considered as multiple least square regressions. Indeed, each row of X is a sample/record, each column of W is the weight for a single task, and each column of Y is the output of this task. The optimization (2.19) indicates the existence of m databases $(X, y_j), j = 1, \ldots, m$, where y_j is the jth column of the matrix Y. The prediction of each y_j corresponds to a single task, which is equivalent to infer each column w_j of W. The objective function is the sum of all tasks' empirical error, with equal importance on each task. In order to impose the relatedness of all tasks, we use the trace norm constraint. That is, all w_j's should be related so that they form a low-rank matrix.

2.4.4 Multinomial logistic regression

The response is a real value in the previous subsection, whereas it is binary in this subsection. In particular, we consider multi-class learning problems. Here, we discuss a specific model – *multinomial logistic regression* – which is the generalization of the traditional 2-class logistic regression model.

Multinomial logistic regression has several interpretations, among which I present, in this thesis, the *likelihood* one. This interpretation relies on the assumption of *independence of irrelevant alternatives* (IIA), which states that the odds of preferring one class over another do not depend on the presence or the absence of other "irrelevant" alternatives. This assumption allows the choice among m alternatives to be modeled as $m - 1$ independent binary choices: one alternative is chosen as the "pivot" (or reference), and the other $m - 1$ are compared against it, one at a time.

Suppose that $X_{n\times d}$ is the feature matrix, and that there are m classes noted by $1, 2, \ldots m$, with $C_{n\times 1}$ indicating the class of each sample. Using the mth class as the pivot class, we model the likelihood ratio of all classes against the pivot one:

$$\ln \frac{\Pr(C_i = j)}{\Pr(C_i = m)} = X_i w_j, \quad \forall i = 1, \ldots, n, \quad j = 1, \ldots, m-1,$$

where X_i is the ith row of X. The above probability should be understood as conditional probability (conditioned on X). Since the likelihood ratio is in $(0, +\infty)$, the left-hand side takes value from $\mathbb{R}$, which means that the above formula are well-defined. Besides, the likelihood of each class is separately defined, which is coherent with the IIA assumption.

By rearranging the terms, we get

$$\Pr(C_i = j) = \Pr(C_i = m)e^{X_i w_j}, \quad \forall i = 1, \ldots, n, \quad j = 1, \ldots, m-1.$$

Since the probability measure sums to 1, we get

$$\Pr(C_i = m) = \frac{1}{1 + \sum_{j=1}^{m-1} e^{X_i w_j}},$$

which gives the other probabilities

$$\Pr(C_i = j) = \frac{e^{X_i w_j}}{1 + \sum_{j=1}^{m-1} e^{X_i w_j}}, \quad \forall i = 1, \ldots, n, \quad j = 1, \ldots, m-1.$$

Furthermore, we can suppose that the parameter of the pivot class $w_m = 0$, which unifies the formula for all classes.

The negative log-likelihood takes this form

$$-\ln \prod_{i=1}^{n} \Pr(C_i) = \sum_{i=1}^{n} \left(-X_i w_{C_i} + \ln \sum_{j=1}^{m} e^{X_i w_j} \right). \tag{2.20}$$

In some literature, it is also reformulated as

$$\begin{aligned} -\ln \prod_{i=1}^{n} \Pr(C_i) &= \sum_{i=1}^{n} \ln \sum_{j=1}^{m} e^{X_i w_j - X_i w_{C_i}} \\ &= \sum_{i=1}^{n} \ln \left(1 + \sum_{j \neq C_i}^{m} e^{X_i w_j - X_i w_{C_i}} \right). \end{aligned} \tag{2.21}$$

Defining W_C in such a way that the ith column of W_C is equal to W_{C_i}, we can reformulate (2.20) in a more concise matrix form

$$\mathbf{1}^T \cdot \ln(e^{XW} \cdot \mathbf{1}) - \mathbf{tr}(XW_C), \tag{2.22}$$

where $\mathbf{1}$ is a vector whose coefficients are all 1, and the exponential and logarithmic function are defined elementwisely.

For multinomial logistic regression, people usually add a regularization term such as $\|W\|_{\mathrm{F}}^2$ and solve it with generalized iterative scaling (Darroch and Ratcliff 1972) or with iteratively reweighted least squares (Bishop 2006), by means of gradient-based optimization algorithms such as L-BFGS (Malouf 2002), or by specialized coordinate descent algorithms (Yu et al. 2011). In my thesis, however, I will use the trace norm $\|W\|_{\mathrm{tr}}$ as the constraint, as in Dudik et al. (2012) and Liu and Tsang (2017).

2.5 Distributed machine learning

This section describes distributed machine learning. Besides some general ideas such as parallel and distributed computing, large-scale machine learning, and frameworks,

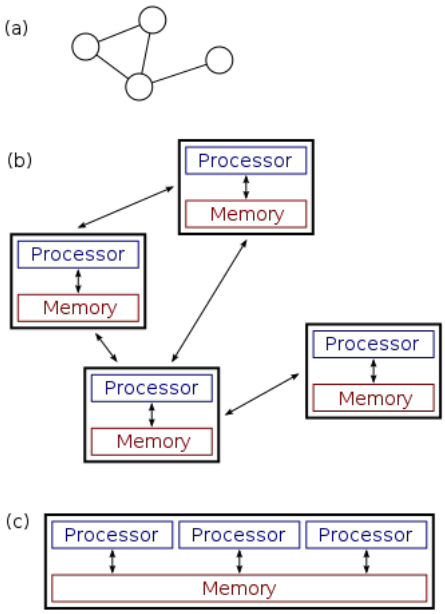


Figure 2.1 – (a), (b): a distributed system. (c): a parallel system. Figure from Wikimedia user: *Miym*.

it defines Bulk Synchronous Parallel and the star network, which are the premise of my study.

This section contains some of my personal reflection, which makes it slightly less rigorous. It is thus preferable to consider this section as a brief introduction into this domain and brainstorming, instead of being taken as a formal research work or an academic reference.

2.5.1 Parallel vs. distributed computing

Distributed machine learning uses distributing technologies to solve machine learning problems. It is often confused with another term, parallel computing, in that distributed computing is often parallel. There is a significant overlap between these two terms, and distributed computing indeed has a lot of "parallel" flavor.

However, many researchers including me use a narrow sense of parallel computing to make it apart from distributed computing. We refer with parallel computing to the scenario where all processors exchange states via a *shared memory* and by distributed computing to the scenario where each processor has its own private memory (*a.k.a. distributed memory*) and exchanges the states by passing messages.

This definition captures the logic layer of the problem instead of the underlying concrete complex physical systems. A multi-core laptop using two cores and one memory (with memory coherence) is parallel computing. Several computers using *distributed shared memory* (DSM) is also parallel computing, even though this memory is split among several computers and may suffer from severe *race conditions*. A group of computers, each managing its own memory and communicating by message

passing, is distributed computing. A single computer using two GPUs (each with its own memory) is also distributed computing. Furthermore, each core of the CPU can have its own cache, which is also a type of distributed computing if two caches need to pass messages to each other. Regardless of the actual physical computation object, we can call them all *nodes* or workers algorithmically without distinction.

With this logic layer, we can develop algorithms (both parallel and distributed) oblivious to the underlying physical infrastructure. Like all kinds of abstraction layers in computer science, this simplifies the description and development of algorithms and makes them applicable in various physical infrastructures. The performance depends on the underlying infrastructures though. Furthermore, by an analysis of the computation and communication/coordination complexity, we can have a rough idea of the performance on various physical infrastructures before the algorithm has even been implemented, which further eases the choice of the infrastructures. Conversely, if our job needs us to work on a specific infrastructure (working constraint), it will still be helpful to describe the algorithms on top of the logic layer, so that other researchers can hence draw inspiration for their own problems.

2.5.2 Large-scale machine learning

The previous subsection explains the broad applicability of distributed machine learning, whereas this subsection will focus on the use of clusters of commodity machines for large-scale machine learning. The majority of the material comes from Chih-Jen Lin's talk, *When and When Not to Use Distributed Machine Learning*[6], in the 2nd International Winter School on Big Data as well as from my personal reflection.

In traditional machine learning, where the bottleneck is the data, the scarcity of data gives birth to various sophisticated algorithms for the fullest use. Nowadays, the emergence of large volume of data, much more than can fit into an average memory, makes the memory and the computational resources the bottleneck, which raises the challenge of large-scale machine learning.

There are several approaches to this challenge. First of all, if the dataset is too large to fit into an average memory, we can just buy a sufficiently large, say, 1TB memory. It then becomes a traditional machine learning problem, with the possibility of parallel computing to accelerate the computation. The second approach is *out-of-core* learning, which refers to algorithms relying on the trick of the efficient write and read of data stored in slow bulk memory such as hard drives. A typical DRAM has an approximate transfer rate of 2-20GB/s, whereas a typical SSD has a rate of 50-200MB/s. The hard drive is therefore one to two orders of magnitude slower but still one to two orders of magnitude faster than the 1-40MB/s of LAN data rate. By smartly accessing from the disk only one subset of the data at a time, we can further reduce the time used for data loading. GraphChi (Kyrola et al. 2012) is a typical example of this approach. The third approach is distributed computing: if one memory is not enough, then we use many memories. The remaining problem is

[6] https://www.csie.ntu.edu.tw/~cjlin/talks/bigdata-bilbao.pdf

to design algorithms that do not incur significant communication cost among these memories.

In the machine learning community, some believe that distributed machine learning is not necessary for large-scale machine learning, claiming that the memory is becoming cheaper and cheaper and that there are few available datasets impossible to fit into a sufficiently large memory. Disagreeing with them, I will argue, in the remaining of this subsection, that distributed machine learning is still relevant and will continue to be so.

On the one hand, there are problems where the data are naturally distributed. Let us consider a multinational corporation owning data centers in several countries, with each data center collecting data continuously for the training of the model at the headquarter. Every week, the data centers do the data warehousing: extract, transform, load (ETL) the data, and then send them to the headquarter. The headquarter combines the new data with the old ones (dropping outdated data if necessary) and then retrain the model. This workflow not only needs significant work and particular coordination but also is doomed to result in offline algorithms. If we had used distributed machine learning, the ETL could have been done on the fly in a streaming manner, and an online algorithm could have been used. With distributed learning, a lot of workforces can be saved, and the model immediately and continuously benefits from the new data, which enables the corporation to take immediate actions. This advantage explains why streaming is one of the development focuses in Apache SPARK 2.0. As a second example, let us consider wireless body area network for medical care, where an immediate response to the health condition is critical. If we collect the data to a data center and analyze it later, we may already miss the timing, and the data collected has thus no longer any value. If we allow the various sensors to communicate with each other and to do some primary analysis, they may be able to detect some abnormalities and even to take measures.

On the other hand, distributed machine learning has several advantages compared with parallel machine learning on a single supercomputer. Firstly, it loads the data from the memory faster. If we have N machines, then the data transfer rate is N times the one of a single machine, which gives us a significant speed up. There are two reasons behind this speed up. The first reason is that the time to solve large-scale machine learning problem is often determined by the data loading instead of the model training. Given a dataset of l samples, the data loading time complexity will be $l \times C_{\text{load}}$, where C_{load} is a constant related to the transfer rate, and the model training time complexity is $l^q \times C_{\text{run}}$, where C_{run} is a constant related to the operation and $q \geq 1$. Since in large-scale machine learning, we tend to use algorithms scaling linearly with the sample size (*i.e.*, $q = 1$), the dominating factor will depend on the constants uniquely. If we run the model training algorithm only once, the model training time is highly likely to be outweighed by the data loading time. See Chih-Jen Lin's talk, as mentioned at the beginning of this subsection, for more explanation.

The second reason of the speed up is about data reloading. Usually, we run the model training algorithm several times (*e.g.*, with different hyperparameters), which justifies the need to load the entire dataset into the memory. However, the reality is often the contrary, and we have to occasionally reload the dataset, say,

during the prototyping. It is worth noting that we do not load the data into the memory *directly* but via some scientific software (*e.g.*, Matlab, R, Python, Julia). If the software in question crashes or, for whatever reasons, we have to restart the software, the memory controlled by the software will all be released, in which case we have to reload the dataset again. This is particularly true for the Julia language, whose workflow requires building an execution environment gradually and does not allow the "remove" of variables from the memory.[7]

If we intend to get the same transfer rate in a single machine as in the distributed one, we will have to resort to some unusual architectures (*e.g.*, multiple hard drives and *multibus*), which require money, time, and expertise. Distributed machine learning provides, alternatively, a cheap software-level solution. By using *distributed file systems*, we can easily scale up the transfer rate through widely available commodity machines.

The second advantage of distributed machine learning over using a single high-memory computer is less interference among multiple users. We all have the experience of sharing a supercomputer among the colleagues, and the supercomputer has all the time several researchers running applications on it. Occasionally, because of careless design (you can never be careful enough to avoid all caveats), some application just eats up all the memory, which causes the failure of all applications on that supercomputer. This failure can result in data/model loss and significant amounts of time wasted. Alternatively, with a cluster of commodity machines, we can allocate to each researcher the number of machines according to his memory requirement, so that the accidents of his applications will not destroy other researchers' work.

The above discussion also leads to the third advantage – *fault tolerance*. In a cluster, the failure of one machine should not impede the entire work. When a failure occurs, you only need to either restart the failed machine or transfer its work to the other ones. This fault tolerance is particularly important in the industry, where the program is supposed to work 24 hours a day and 7 days a week without interruption, which is unlikely to achieve with a supercomputer.

So far, I have given several arguments to support the use of clusters of commodity machines for large-scale machine learning, and more can be found in Peteiro-Barral and Guijarro-Berdiñas (2013, Section 2.2). I did not say that the cluster of commodity machines is definitely the only choice in the future; I simply said that it is still relevant in both laboratories and industries. If, in laboratories, the disadvantage of supercomputers can be mitigated by good practices, in industries, the low-cost, highly responsive, fault tolerant cluster is an excellent response to the large-scale machine learning challenges.

I will conclude this subsection with an anecdote. During NIPS 2016 held in Barcelona, I met a Belgian researcher, who was doing some research for a bank. The bank wanted to do some machine learning on Apache SPARK, so they contacted this researcher. They wanted him to prototype the algorithms on a small-scale cluster,

[7] In Julia, if we give another definition to an existing function, the latter will coexist with the new one – for multiple dispatch – instead of being replaced. During the prototyping phase, we try various constructions and discard the undesired ones either by restarting the software or by creating a new namespace. Either way, we will have to reload the dataset, which, consequently, costs us more time for the data loading than the model training.

say 8 machines, so that they can later deploy directly these algorithms on a large-scale cluster inside the bank. The moral of this story is that the academy should still study efficient algorithms working on clusters of commodity machines simply because of the industrial need.

2.5.3 Algorithm taxonomy

This subsection studies the taxonomy of the distributed machine learning algorithms. The majority of the material comes from Xing et al. (2015b).

One approach towards distributed machine learning is to modify an existing non-distributed sequential algorithm so that it can run on a distributed infrastructure. In this case, the logic of the distributed algorithm is the same as its counterpart, and we say that it is *serializable*. In other words, the algorithm will get the same results as its counterpart, and its theoretical convergence is also guaranteed. The downside of this approach is that it often incurs high communication cost, for its counterpart supposes that it has easy access to all parts of the memory.

The other approach is to develop novel distributed algorithms, which may not have obvious equivalent non-distributed counterparts. This approach takes the communication cost into account from the very beginning of the algorithms design, and we are thus more likely to get algorithms with low communication cost. However, the theoretical convergence of the algorithm is not apparent, and we should make extra effort to study it. One may question the possibility of two different algorithms having the same result. To understand this, one may recall that machine learning algorithms often have the aim to find a *unique* solution (*e.g.*, a local or global optimum) in the space, and that this solution is often an attractor in a vector field. Although the different algorithms may take different paths, they can still reach the same destination, as the old proverb, "all roads lead to Rome", claims.

Big data can have either large-scale data, or large-scale model (*i.e.*, parameter), or both. Similarly, there are two types of parallelisms in distributed machine learning – data parallelism and model parallelism. The former partitions the data into multiple nodes, with each node storing the whole model, while the latter partitions the model into multiple nodes, with each node storing the entire dataset. These two parallelisms are not mutually exclusive: one can apply both parallelisms simultaneously.

One central decision during the development of distributed machine learning algorithms is the *bridging model*, which dictates the way to bridge computation with inter-machine communication (Xing et al. 2015b). Programs following the *Bulk Synchronous Parallel* (BSP) model (Figure 2.2) alternate a computation phase and a communication phase, with the communication phase serving as a barrier separating the successive computation phases. We then call each computation phase an *epoch* as opposed to the term *iteration* commonly used in optimization.

The BSP model often entails a serializable algorithm and is the principal way to adapt an existing sequential algorithm to make it distributed. In spite of its straightforwardness, its downside is also evident: quick nodes must wait for slow nodes. This waiting can still happen even with balanced workload and nodes of

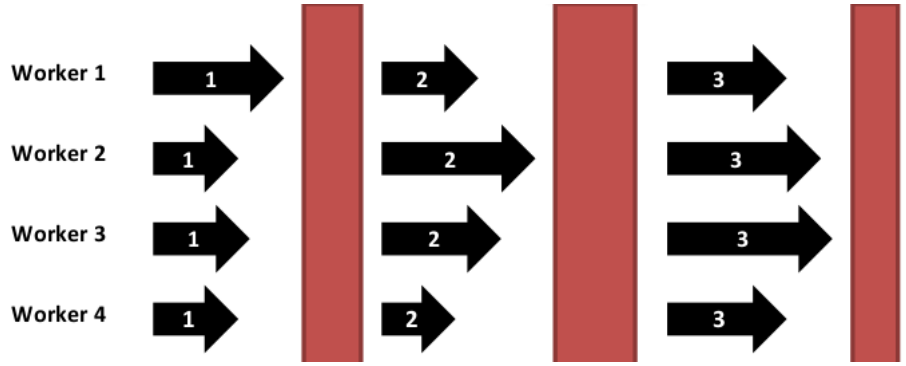


Figure 2.2 – Bulk Synchronous Parallel (BSP) Bridging model. The nodes wait at the end of every epoch for each other, and then exchange information during the synchronization barrier. Figure from Xing et al. (2015b).

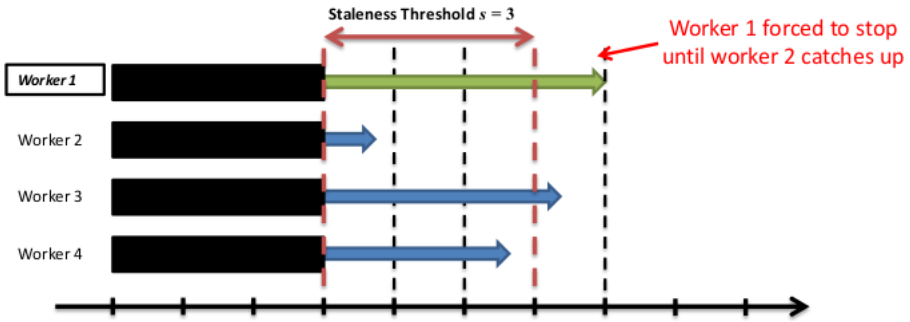


Figure 2.3 – Stale Synchronous Parallel (SSP) Bridging model. Figure from Xing et al. (2015b).

equal capacity. Indeed, it is impossible to eliminate this problem via the balance of workload, for the waiting nodes in one epoch can differ from the waiting nodes in another.

In contrast to BSP, the *Asynchronous Parallel* model lets each node work at its maximal speed. They send messages to each other when appropriate but never stop to wait for messages from other nodes. This model accelerates the computation (no wait) but can suffer from reduced per computation gain. It can even make the algorithm give an incorrect answer because some nodes may rely on "outdated" information from other nodes. This error can be arbitrarily large to the extent that the power of the attractor is overwhelmed by the induced error.

Sitting in the middle is the *Stale Synchronous Parallel* (SSP) model (Figure 2.3), where, compared with BSP, nodes may advance ahead of each other up to s iterations apart (where s is called the staleness threshold). Nodes getting too far ahead are forced to stop until slower nodes catch up. Like Asynchronous Parallel execution, information is exchanged asynchronously and continuously between nodes without the need for synchronization barriers. The advantage of SSP is that it behaves like Asynchronous Parallel execution most of the time but can also stop nodes, when needed, to ensure the convergence. An excellent work concerning SSP's convergence can be found in Tsitsiklis et al. (1986).

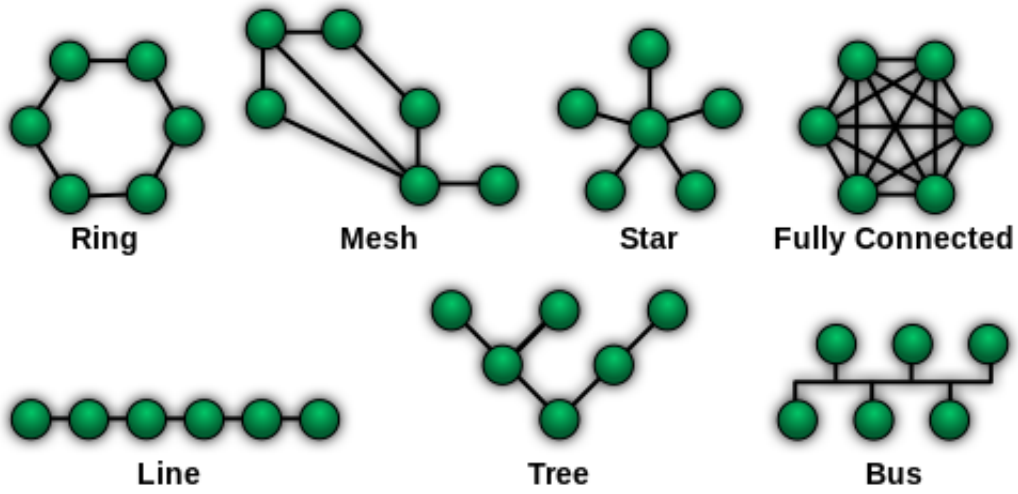


Figure 2.4 – Diagram of different network topologies. Figure from Wikimedia user: Malyszkz.

Another important aspect is the topology of the communication network. This term network stated here should be considered as a logical network instead of a physical one. Figure 2.4 shows various topologies of networks, among which the *star* network is the most relevant to my work. The center of a star network is often called *master*/*server*, and the other nodes are often called *slaves*/*clients*. Passing messages from the master to the slaves is called *broadcast*, and passing messages from the slaves to the master is called *convergecast*.[8] The associated algorithm is often called *centralized* machine learning algorithm. On the contrary, a *decentralized* machine learning algorithm works on a partially connected *mesh* network, where no node plays a leading role.

2.5.4 Frameworks

Researchers find that many parallel/distributed machine learning algorithms follow a similar "template". Instead of developing for each algorithm separately, we can develop an abstract *framework* which various individual algorithms can be fit into. This layer of abstraction frees the algorithm designers from nasty message passing details and fault tolerance management.

The first such framework is Google's MapReduce (Dean and Ghemawat 2008), which provides `Map` and `Reduce` functional (*a.k.a.* higher-order function) and hence gets its name. By feeding functions to these functionals, algorithm designers can easily invent and implement distributed algorithms. The original MapReduce uses hard drives to store computational states, which appears to be extremely slow. Apache SPARK (Zaharia et al. 2012a) takes this step further and uses *Resilient distributed datasets* (RDD) (Zaharia et al. 2012b) to make the computation in-memory, which gives nearly 100x speedup.

[8] The same terminology also applies to a *tree* network in a similar way.

Meanwhile, there have come out various type of frameworks. Pregel (Malewicz et al. 2010) and GraphLab (Low et al. 2012) are used for machine learning based on the graph representation. If SPARK is a Jack of all trades, ParameterServer (Li et al. 2014) and its associated OS Petuum (Xing et al. 2015a) is specialized on machine learning problems with large-scale parameters, and it can be 10-100x faster than SPARK. Tensorflow (Abadi et al. 2016), supported by Google, becomes one of the most popular frameworks nowadays, especially for its usage in *deep learning*. There are more, but here I restrain myself from further development.

Chapter 3

Frank-Wolfe Algorithms

This chapter is a centralized place to discuss Frank-Wolfe algorithms. It starts with a not-so-brief presentation of the state of the art, which includes the majority of research on Frank-Wolfe. Then, it describes the basic form of Frank-Wolfe to make the readers familiar with this algorithm as well as understand the intuitive idea behind. After this lightweight introduction is the presentation of the comprehensive general form, which includes most Frank-Wolfe variants as its special cases. Finally, it proceeds to the most important part – the *nondeterministic* convergence rate, which applies to, among others, solving the linear subproblem of trace norm minimization with power iteration.

The pedagogical writing style of this chapter does not highlight my contributions enough. To compensate this inconvenience, I list my discoveries about Frank-Wolfe here. The most important one is the sublinear convergence rate in expectation and with high probability for trace norm minimization. The second important is the general form of Frank-Wolfe, which serves as a guideline to develop a heavy-duty package. The third is the link between Frank-Wolfe and the steepest descent. The fourth, a minor one, is the clarification of the effect of Lipschitz continuity on the finiteness of the global curvature. I refer the readers to Section 3.1.2 for a detailed explanation.

3.1 Introduction

The Frank-Wolfe algorithm (*a.k.a.* conditional gradient method) is a first-order projection-free algorithm designed for constrained convex optimization problems of the following form

$$\min_{x \in \mathcal{D}} f(x), \tag{3.1}$$

where $\mathcal{D}$ is a convex constraint set (usually compact) in a finite-dimensional space (*e.g.*, $\mathbb{R}^n$) and f is a differentiable convex function. It does not require any projection like in projected gradient descent or proximal methods, where the projection/proximal operator may be expensive to compute. Instead, it proposes to min-

imize a linear function over the constraint set, which, in many cases, has efficient solvers.

Before proceeding on, let us first view some concrete examples of the above optimization problem. When $\mathcal{D}$ is an ℓ_1-norm ball in $\mathbb{R}^n$, it becomes the well-known Lasso (Tibshirani 1996, Koh et al. 2007). When $\mathcal{D}$ is an ℓ_∞-norm ball, the coefficients of its solution have a high probability of being integers instead of any arbitrary real numbers (Mangasarian and Recht 2011). When $\mathcal{D}$ is the trace norm ball, it becomes the trace norm minimization (Candès and Recht 2012, Candès and Tao 2010). When $\mathcal{D}$ is the ball induced by the matrix's max-norm, it becomes max-norm minimization (Goemans and Williamson 1995, Srebro et al. 2005, Lee et al. 2010).

Of course, Frank-Wolfe is not the only tool to solve these problems, but there *are* lots of research on this approach. Here is an non-exhaustive list of problems solved by Frank-Wolfe: Fukushima (1984), LeBlanc et al. (1985), Ouyang and Gray (2010), Lacoste-Julien et al. (2012), Ping et al. (2016).

To demonstrate the popularity of Frank-Wolfe as well as the research lineage, in the remaining of this section, I describe the state of the art. Due to the length and the limited knowledge of the author, I cannot discuss all of them in detail (I apologize in advance in case I forget some important papers). Instead, I recommend readers to use them as road signs to discover the work meeting their needs the most. If you are already familiar with all of them but still unsatisfied, there are also my contributions listed at the end of this section.

3.1.1 State of the art

The Frank-Wolfe algorithm (hereinafter Frank-Wolfe) was first proposed by Frank and Wolfe in 1956 (Frank and Wolfe 1956). It was used to minimize a quadratic function over a polytope by minimizing its (linear) supporting plane over the same polytope. This invention was right after the Danzig's simplex method for linear programming and well before the now well-known interior-point method. Its alternative name, conditional gradient method, was coined by Levitin and Polyak (1966).

Recently, Clarkson (2010) restudied this old algorithm and renewed people's interest within the machine learning community. He showed that Frank-Wolfe can be extended to optimize smooth convex function over the simplex. Moreover, it has a close relationship with the sparse greedy approximation and the coreset concept widely used in machine learning. Hazan (2008) used Frank-Wolfe to minimize a convex function over the bounded semidefinite cone. This work initialized researchers' interest in using Frank-Wolfe in some other spaces (*e.g.*, matrix spaces) than Euclidean space. Based on this idea, Jaggi (2013) further extended it by allowing the constraint set to be arbitrary convex and compact sets. In particular, it can be a convex hull of a set with finite or infinite number of elements.

Regarding the convergence rate, it is shown, in the above literature, that Frank-Wolfe converges in the order of $O(\frac{1}{t})$, where t is the iteration number. Garber and Hazan (2015) showed that, if the constraint set is strongly convex, Frank-Wolfe can converge at a rate of $O(\frac{1}{t^2})$. The various balls induced by ℓ_p-norms ($p > 1$), Schatten

p-norms ($p > 1$) and group norms are examples of strong convexity. The above rates are called sublinear rate and are considered slow (see Section 2.2.4). It is therefore interesting to find out whether Frank-Wolfe can achieve *linear* rate when the objective function is further *strongly* convex (see Section 2.1.4) as many other first-order algorithms do. In response to this question, researchers have tried various approaches.

Wolfe (1970) sketched a proof, and Guélat and Marcotte (1986) proved in detail that (1) Frank-Wolfe converges linearly provided that the optimum point is not located on the boundary of the constraint set and that (2) a variant of Frank-Wolfe using the *away step* converges linearly once it arrives at the smallest face containing the optimum point.

However, this analysis has two drawbacks. First, it relies on the *strict complementarity* assumption (Wolfe 1970), which may not be satisfied in certain cases. This drawback was pointed out by Damla Ahipasaoglu et al. (2008), Kumar and Yıldırım (2011), Ñanculef et al. (2014), and they proposed therefore to use *Robinson's condition* (Robinson 1982) instead. This condition is weaker than the strict complementarity (see Ñanculef et al. 2014, Section 4 for a discussion). Damla Ahipasaoglu et al. (2008) and Kumar and Yıldırım (2011) proved that Frank-Wolfe with the away step converges linearly, and Ñanculef et al. (2014) proved the same thing for a Frank-Wolfe variant with the *pairwise step* (Mitchell et al. 1974).

The second drawback is that the linear convergence rate is local: if the initial point is far away from the optimum point, the convergence rate can be much slower. Levitin and Polyak (1966, Section 6) and later works (Demianov and Rubinov 1970, Dunn 1979) proved the global linear convergence rate under the assumption that the constraint set is strongly convex and the assumption that the norm of the gradient is bounded away from zero. This assumption is very restrictive (see Garber and Hazan 2015, the end of Section 1.1 for a discussion). To improve their results, Garber and Hazan (2013a) proved that a modified Frank-Wolfe equipped with a *local* linear optimization oracle (instead of the global one) can achieve the global linear convergence rate, and they also provided some ways to construct this oracle.

Concerning Frank-Wolfe with *away steps* itself, the recent works of Beck and Shtern (2015), Lacoste-Julien and Jaggi (2015), Pena et al. (2016) established global linear convergence results when the objective function is strongly convex with Lipschitz continuous gradient and when the domain is a convex hull of a finite set of atoms. They also found that the performance of away steps is closely related to the geometry of the constraint set. To describe the geometry, Lacoste-Julien and Jaggi (2015) constructed the *pyramidal width*, Beck and Shtern (2015) constructed the *vertex-facet distance*, and Pena et al. (2016) construct the *restricted width*. Both Beck and Shtern (2015) and Pena et al. (2016) are "inspired by and relied upon key ideas and results" first introduced in a preliminary workshop version of Lacoste-Julien and Jaggi (2015) (Pena and Rodriguez 2015). Lacoste-Julien and Jaggi (2015) also gave an *affine invariant* proof of the global linear convergence rate for away-step Frank-Wolfe and pairwise Frank-Wolfe. Later, Pena and Rodriguez (2015) constructed the *facial distance*, which unifies all the three previous constructions.

There are further studies built on these works. Beck and Shtern (2015) applied it to a special type of non-convex functions. Freund and Grigas (2016) studied the effect of the step size. Mairal (2013) provided a unified viewpoint for several first-order optimization algorithms including Frank-Wolfe. Freund et al. (2017) provided an *in-face* variant. A recent work of Gidel et al. (2016) even used it for *saddle point* problem.

Besides proving faster convergence rates under stronger conditions, we can also move towards the opposite direction by proving slower rates under weaker conditions. Recently, Xu (2017) proved some results for objective functions having uniformly continuous Fréchet derivatives. The convergence rate is $O(\frac{1}{t^{\sigma-1}})$ when the objective function has a bounded curvature of order $\sigma \in (1, 2]$, and it becomes $O(\frac{1}{t^{\nu}})$ when its Fréchet derivative is ν-Hölder continuous for $\nu \in (0, 1]$.

Another approach to accelerate the computation other than proving faster convergence rate is to reduce the computation cost. Lan and Zhou (2016) proposed Conditional Gradient Sliding by skipping some gradient evaluations, and they proved that the original sublinear rate still holds.

Like many other descent algorithms, researchers have also invented stochastic versions of Frank-Wolfe (Lafond et al. 2015, Ouyang and Gray 2010, Hazan and Luo 2016, Goldfarb et al. 2017, Liu and Tsang 2017). For instance, Ouyang and Gray (2010) applied Stochastic Frank-Wolfe to the dual of SVM. Hazan and Luo (2016) invented the Stochastic Variance-Reduced Frank-Wolfe based on the variance reduction trick (Johnson and Zhang 2013, Mahdavi et al. 2013) and Stochastic Variance-Reduced Conditional gradient Sliding based on the Conditional Gradient Sliding algorithm (Lan and Zhou 2016). Goldfarb et al. (2017) proposed the Away-step Stochastic Frank-Wolfe and the Pairwise Stochastic Frank-Wolfe and proved that they all converge linearly in expectation with the technique of empirical processes and concentration inequalities. Liu and Tsang (2017) proposed a stratified progressive sampling method; in their work, the batch size used to calculate the stochastic gradient grows with the iteration, and the data are sampled in a stratified way according to the structure of the problem.

As to the online convex optimization (Hazan and others 2016) which intends to minimize the *regret* in lieu of the empirical risk, related researches can be viewed in Hazan and Kale (2012), Garber and Hazan (2013a,b), Garber et al. (2015). Also see Garber's PhD disseratation (Garber 2016b) for a full exposition.

3.1.2 Contributions

The major contribution of my work included in this chapter is the introduction of some new concepts into the Frank-Wolfe machinery. It consists of the cases where the linear subproblem is solved approximately in expectation and/or with high probability. These machinery address the situation where the linear subproblem cannot be approximately solved *deterministically*. Under these conditions, I prove that the correspondent Frank-Wolfe converges (sublinearly) in expectation and/or with high probability.

In particular, I prove that, for the trace norm minimization, if the linear subproblem is solved by power iteration, Frank-Wolfe converges in expectation and with high probability, and that the required number of power iterations grows linearly or quadratically according to the type of results desired. This is the first time that a rigorous convergence theory is established for Frank-Wolfe on the trace norm minimization problem.

This contribution is inspired by Jaggi (2011), who has already pointed out the type of results expectable. However, the extreme conciseness and the adoption of deterministic language therein makes it inappropriate in the nondeterministic context. I address this little imperfection and express it rigorously. One step further, I study for the general matrices, symmetric matrices, and positive semidefinite matrices. Except for the symmetric case, which is treated heuristically, all other cases are proven to enjoy the conclusion mentioned above.

There are also other minor contributions in this chapter:

- Drawing inspiration from Goldfarb et al. (2017), I give a general form of Frank-Wolfe (Algorithm 3), which makes most variants its special cases. This general form, made possible by the functional programming paradigm, may help the development of a Frank-Wolfe package.

- I point out the link between Frank-Wolfe and the steepest descent when the constraint set is an atomic norm ball (Section 3.2.4). As far as I know, I am the first to find this link.

- I clarify the effect of Lipschitz continuity on the finiteness of the global curvature (Theorem 3.4.1). The idea is already in Jaggi (2013) and Lacoste-Julien and Jaggi (2015). I only centralize the separated results and make it more digestible to the readers.

3.2 An invitation to Frank-Wolfe algorithms

This section presents the basic form of Frank-Wolfe algorithms and some concepts associated, which are the basis to understand all other variants. The concept of duality gap is essential to prove the sublinear convergence rate of Frank-Wolfe whether it is for deterministic cases as in Jaggi (2013) or for nondeterministic cases as in my dissertation. The concept of atomic norm makes Frank-Wolfe even more relevant in today's science and is the most important content in this section. It not only allows Frank-Wolfe to be as widely applicable as possible but also highlights the cases where the linear subproblem can be efficiently solved. Moreover, it helps address the relation between Frank-Wolfe and steepest descent, which is one of my minor contributions.

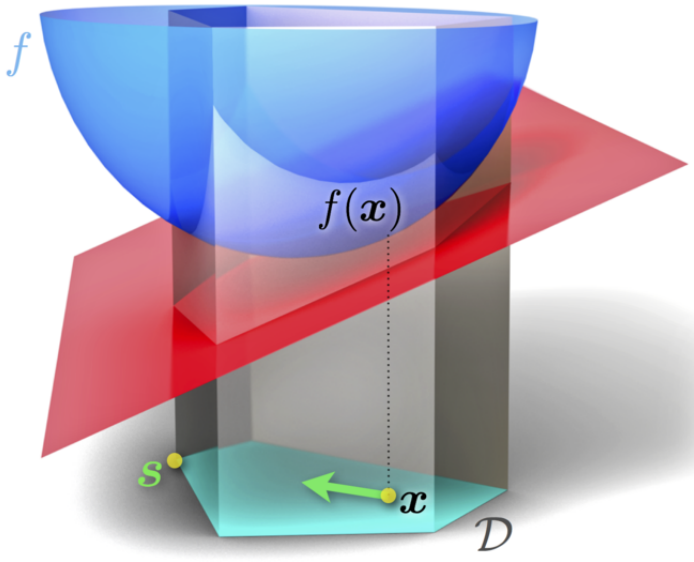


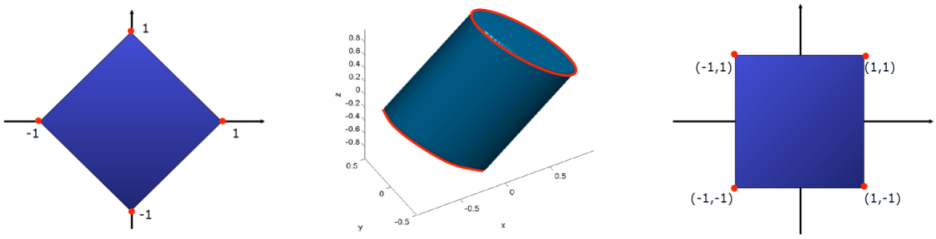
Figure 3.1 – Illustration of one step of Frank-Wolfe (Jaggi 2011), made by *Stephanie Stutz*.

3.2.1 Basic form

Let us consider the optimization problem (3.1). A standard Frank-Wolfe algorithm consists of the following steps in each iteration. First, since f is a convex function, we can calculate its supporting plane on the point x, and then we minimize along this plane over the constraint set and denote this solution as s. Since the supporting plane is a linearization of the original objective function, we usually call this step solving the linear subproblem (or the linear approximation problem). In many cases, this problem can be efficiently solved, much more efficient than projecting back to the constraint domain. Then, we compute a convex combination of the current point x and the destination point s according to the step size γ. This combination constitutes the new x in the next iteration. Jaggi (2013, Theorem 1) showed that this algorithm converges in the order of $O(\frac{1}{t})$ when the step size is chosen as $\frac{2}{2+t}$ or by the line search (t is the iteration number). The basic form of Frank-Wolfe is summarized in Algorithm 2, and an illustration is shown in Figure 3.1.

Algorithm 2. Basic form of Frank-Wolfe (1956)

Input: Objective function f, constraint set $\mathcal{D}$ and an initial point $x \in \mathcal{D}$
repeat
  $s \leftarrow \arg\min_{s \in \mathcal{D}} \langle s, \nabla f(x) \rangle$ ▷ solve the linear subproblem
  $\gamma \in [0, 1]$ ▷ choose a step size
  $x \leftarrow (1 - \gamma)x + \gamma s$. ▷ the update is a convex combination
until stopping criterion is satisfied.
Output: x, $f(x)$



(a) (b) (c)

Figure 3.2 – Unit balls of some atomic norms (Chandrasekaran et al. 2010). In each sub-figure, the set of atoms is graphed in red and the unit ball of the associated atomic norm is graphed in blue. In (a), the atoms are the unit-Euclidean-norm one-sparse vectors, and the atomic norm is the ℓ_1-norm. In (b), the atoms are the 2×2 symmetric unit-Euclidean-norm rank-one matrices, and the atomic norm is the trace norm. In (c), the atoms are the vectors $\{-1, +1\}^2$, and the atomic norm is the ℓ_∞-norm.

3.2.2 Duality gap and certificate

The *duality gap* is an important concept in optimization. It is usually defined as the difference between the minimum of the primal problem and the maximum of the dual problem. Here, instead, we give a lightweight definition of duality gap, which Jaggi (2011) named the "poor-man's duality [gap]":

$$g(x) := \max_{s \in \mathcal{D}} \langle x - s, \nabla f(x) \rangle \,. \tag{3.2}$$

The convexity of f implies that the linearization $f(x) + \langle s - x, \nabla f(x) \rangle$ always lies below the graph of f, which further implies that $g(x) \geq f(x) - f(x^*)$, where x^* is the minimum point of f. The difference $f(x) - f(x^*)$ is called *suboptimality error*, and the fact that it is upper bounded by $g(x)$ implies that the duality gap is a *certificate*.[1] For a more detailed introduction of the concept of duality gap, see Jaggi (2011, Section 2.2).

While the minimum $f(x^*)$ is generally unknown especially for problems from the real life, the duality gap $g(x)$ for the current point x is easy to calculate. Indeed, it is a by-product when solving the linear subproblem. If, furthermore, the duality gap decreases to zero, we can use it as a stop criterion in Algorithm 2. Jaggi (2013, Theorem 2) showed that, under suitable conditions, the duality gap converges to 0 in the order of $O(\frac{1}{t})$ just like the suboptimality error.

3.2.3 Atomic norms

This subsection presents the *atomic norm* and shows how efficient Frank-Wolfe can be when the constraint set is an atomic norm ball.

[1] In optimization, a certificate is a function that upper bounds the suboptimality error.

Given a vector space X and a subset K, let us define the *Minkowski functional* (Thompson 1996) $\Omega_K : X \to [0, +\infty]$ with regard to K by

$$\Omega_K(x) := \inf\{t \geq 0 : x \in tK\},$$

which is also called the *gauge* of K (Rockafellar 2015). Usually, we assume that $0 \in K$ and that the set $\{t \geq 0 : x \in tK\}$ is nonempty for all x's, so that Ω_K is finite everywhere ($\inf \emptyset = +\infty$ by convention).

One can prove that, if K is *absolutely convex*, that is, if it is convex and *balanced* (*i.e.*, $tK \subset K, \forall\, |t| \leq 1$), the Minkowski functional becomes a seminorm. If, furthermore, K is bounded, it becomes a norm. For example, if we let K be the unit ℓ_2-ball in $\mathbb{R}^n$, Ω_K will be exactly the Euclidean norm $\|\cdot\|_2$. Once it is a norm, its dual norm is given by $\Omega_K^*(y) := \sup_{x \in K} \langle x, y \rangle$ provided that X is an inner product space.

Now, let $\mathcal{A} \subset X$ be a set of *atoms*, whose meaning will be clear soon. Let us suppose that $\mathcal{A}$ satisfies the following conditions:

1. $\mathcal{A}$ is bounded;
2. for every $x \in \mathcal{A}$, we have $-x \in \mathcal{A}$;
3. 0 is in the interior of $\mathbf{conv}(\mathcal{A})$[2].

Through the definition of norm, we can easily prove that $\Omega_{\mathbf{conv}(\mathcal{A})}(\cdot)$ is a norm. Chandrasekaran et al. (2010) defines it as *atomic norm*. The dual norm of the atomic norm has an extremely simple expression: $\Omega^*_{\mathbf{conv}(\mathcal{A})}(\cdot) = \sup_{x \in \mathcal{A}} \langle x, \cdot \rangle$, where the supremum is taken on the atom set $\mathcal{A}$ instead of on the entire convex hull. Indeed, since the inner product is linear, the extreme value must be achieved at the extreme points (viz. atoms). With slight abuse of notation, we also note the atomic norm as $\Omega_{\mathcal{A}}(\cdot)$ and its dual norm as $\Omega^*_{\mathcal{A}}(\cdot)$.

Now let us view some examples (Figure 3.2). If $\mathcal{A}$ is $\{\pm e_i, i = 1, \ldots, n\}$ of $\mathbb{R}^n$, then the associated atomic norm is exactly the ℓ_1-norm. If $\mathcal{A}$ is the set of *sign-vectors* (whose entries take on value of ± 1), then the associated atomic norm is the ℓ_∞-norm. If $\mathcal{A}$ is the set of all rank-1 matrices, then the associated atomic norm is the trace norm. I refer to Chandrasekaran et al. (2010) for more examples.

The shape of the trace norm ball above is a bit confusing and is asked several times in various places (*e.g.*, Stack Exchange). Here, I give a detailed but still concise presentation about it. Indeed, this trace norm ball is a convex hull of two ellipses. To see this, let $\begin{bmatrix} a & b \\ b & c \end{bmatrix}$ be a symmetric unit-Euclidean-norm rank-one matrix, which means

$$\begin{cases} ac = b^2 \\ a^2 + 2b^2 + c^2 = 1. \end{cases}$$

A slight transformation yields

$$\begin{cases} (a - \frac{1}{2})^2 + b^2 = \frac{1}{4} \\ a + c = 1 \end{cases} \quad \text{or} \quad \begin{cases} (a + \frac{1}{2})^2 + b^2 = \frac{1}{4} \\ a + c = -1. \end{cases}$$

[2]Convex hull. See Page 8 for the definition.

The solution set is essentially two planes cutting two cylinders respectively.

Geometry tells us that, if we use the atomic norm ball as the constraint set, we shall get solutions with few atoms (*e.g.*, sparse vectors, low-rank matrices) (Chandrasekaran et al. 2010). This property is very useful when the measurements are not sufficient to recover the model in under-determined situations. This technique is widely used in *compressed sensing* and other sparse recovery tasks.

It remains the question how to solve this kind of optimization problems. Chandrasekaran et al. (2010) finds that they can generally be transformed into SDP problems. However, as pointed out in Section 2.3.4, SDP software cannot handle large number of parameters. That is how Frank-Wolfe comes to help. Indeed, the linear subproblem becomes $\arg\min_{s\in\mathcal{A}} \langle s, \nabla f(x)\rangle$, and the duality gap is essentially $\langle x, \nabla f(x)\rangle + \Omega^*_{\mathcal{A}}(-\nabla f(x))$.

When $\mathcal{A}$ has only $2n$ atoms (*e.g.*, the ℓ_1-norm ball), it can be solved by at most $2n$ inner products. When the number of atoms are infinite, it may also exist efficient algorithms. For instance, the trace norm ball has infinite atoms (two ellipses). However, we know that its dual norm is the spectral norm (Section 2.1.6). Noticing that the spectral norm is the largest singular value, we have many algorithms to calculate it. In general, whenever we have efficient algorithms to calculate the dual norm of the atomic norm in question, we can use Frank-Wolfe. I refer the reader to Jaggi (2013, Table 1) for more examples.

Besides the fact that Frank-Wolfe is suitable for this task, it also has other merits. For instance, Frank-Wolfe takes in only *one* atom per iteration, which means that its solution path is sparse. Jaggi (2013, Lemma 3) showed that the suboptimality error of any solution involving s atoms is lower bounded by $O(\frac{1}{s})$, which implies that Frank-Wolfe is optimal in the sense of sparsity (up to a constant factor).

3.2.4 Similarity with steepest descent

When the atomic norm ball is the constraint set, Frank-Wolfe is similar to the steepest descent equipped with the same atomic norm. As mentioned in Section 2.2.5, in unconstrained optimization, the steepest descent requires a predefined norm. It chooses the direction which minimizes the support plane over the unit norm ball, and then it descends along this direction. In particular, if it chooses the atomic norm defined by $\mathcal{A}$, the algorithm can be described as a loop of

$$\begin{cases} s \leftarrow \arg\min_{s\in\mathcal{A}} \langle s, \nabla f(x)\rangle \\ x \leftarrow x + \gamma s. \end{cases}$$

This is very similar to Frank-Wolfe, which can be expressed as a loop of

$$\begin{cases} s \leftarrow \arg\min_{s\in\mathcal{A}} \langle s, \nabla f(x)\rangle \\ x \leftarrow (1-\gamma)x + \gamma s, \end{cases}$$

except for the update step. In the steepest descent, s is a direction, whereas, in Frank-Wolfe, s is a destination. Frank-Wolfe takes a convex combination of the

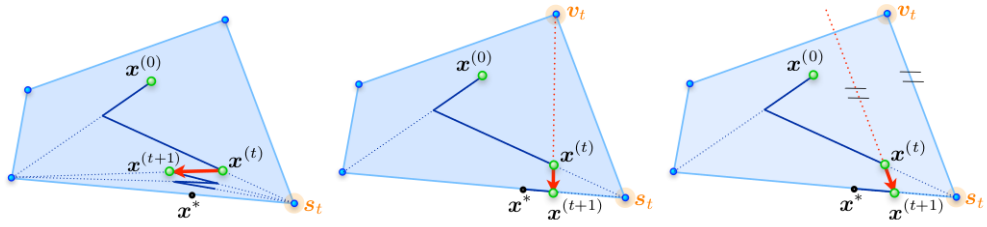


Figure 3.3 – Illustration of the away step and the pairwise step (Lacoste-Julien and Jaggi 2015). (left) The FW algorithm zig-zags when the solution x^* lies on the boundary. (middle) Adding the possibility of an away step attenuates this problem. (right) As an alternative, a pairwise FW step.

current iterate and the destination and makes it the next iterate, which thus ensures that the iterate stays always inside the constraint set. To my best knowledge, I am the first to discover this link.

3.3 Frank-Wolfe variants

This section describes the general form of Frank-Wolfe algorithms, which can express away step, pairwise step, stochastic variant, approximate subproblem, and so forth. It is made possible via the combination of both functional and procedural programming paradigm. With this minor contribution of mine, a comprehensive Frank-Wolfe package can be easily made.

3.3.1 General form

This subsection presents the general form of Frank-Wolfe (Algorithm 3). It can summarize most variants except the fully-corrective one. Besides, it offers a guideline to program a Frank-Wolfe package. The description of the algorithm adopts a combination of both the procedural and the functional paradigm. It appears much more sophisticated than the basic form (Algorithm 2) because I have integrated various possible variants into a single algorithm.

Let us analyze Algorithm 3 step by step. The algorithm needs a few inputs (Line 1–8). The quantity μ is a probability measure specifying the initial point, which is usually initiated as a Dirac measure concentrated on a single atom. In other words, there exists an atom $a \in \mathcal{A}$ so that $\mu(\{a\}) = 1$. With the optimization process going on, this measure gradually becomes less concentrated. `GEO`, `LMO` and `SSO` are all *oracles* (*a.k.a. virtual functions* in the programming terminology) and will be concretized during the runtime. The quantities η_1 and η_2 are small values specifying the required precision in the oracles. The variables `flag_away` and `flag_pair` are flags taking boolean values (*i.e.*, `True` or `False`), which dictates whether to use away-step, pairwise step, or the vanilla direction.

Algorithm 3. General form of Frank-Wolfe:
FW($\mathcal{A}$, μ, η_1, η_2, `GEO`, `LMO`, `SSO`, `flag_away`, `flag_pair`)

input: $\mathcal{A}$: atom set
μ: probability measure on $\mathcal{A}$
$\eta_1, \eta_2 > 0$: approximation error
`GEO`: gradient evaluation oracle
`LMO`: linear minimization oracle
`SSO`: step size oracle
`flag_away`: away flag
`flag_pair`: pairwise flag
Init: $x \leftarrow \int_{\mathcal{A}} x\mu(\mathrm{d}x)$ ▷ initial point
$S \leftarrow \{x \in \mathcal{A} : \mu(x) > 0\}$ ▷ initial active set
repeat
  $h \leftarrow \texttt{GEO}(x, \eta_1)$ ▷ evaluate the gradient
  $s \leftarrow \texttt{LMO}(h, \mathcal{A}, \eta_2)$ ▷ solve the linear subproblem
  $d^F \leftarrow s - x$ ▷ vanilla direction
  if `flag_away` **or** `flag_pair` **then**
    $v \leftarrow \arg\max_{v \in S} \langle h, v \rangle$
    $d^A \leftarrow x - v$ ▷ away direction
  end if
  if `flag_pair` **then**
    $d \leftarrow s - v$, and $\gamma_{\max} \leftarrow \mu(v)$ ▷ choose the pairwise direction
  else if `flag_away` **and** $\langle h, d^A \rangle < \langle h, d^F \rangle$ **then**
    $d \leftarrow d^A$, and $\gamma_{\max} \leftarrow \frac{\mu(v)}{1-\mu(v)}$ ▷ choose the away direction
  else
    $d \leftarrow d^F$, and $\gamma_{\max} \leftarrow 1$ ▷ choose the vanilla direction
  end if
  $\gamma \leftarrow \texttt{SSO}(x, d, \gamma_{\max})$ ▷ choose a step size
  $x \leftarrow x + \gamma d$ ▷ update the iterate
  `VRU`() ▷ update the representation of x
until stopping criterion is satisfied.
output: x

Algorithm 4. Vertex Representation Update (`VRU`, Goldfarb et al. 2017)

if $d == d^F$ **then** ▷ vanilla direction is chosen
  $\mu(u) \leftarrow (1-\gamma)\mu(u), \quad \forall u \in S$
  $\mu(s) \leftarrow \mu(s) + \gamma$
  if $\mu(s) == 1$ **then**
    $S \leftarrow \{s\}$
  else
    $S \leftarrow S \cup \{s\}$
  end if
else if $d == d^A$ **then** ▷ away direction is chosen
  $\mu(u) \leftarrow (1+\gamma)\mu(u), \quad \forall u \in S$
  $\mu(v) \leftarrow \mu(v) - \gamma$
  if $\mu(v) == 0$ **then**
    $S \leftarrow S \setminus \{v\}$
  end if
else ▷ pairwise direction is chosen
  $\mu(v) \leftarrow \mu(v) - \gamma$
  $\mu(s) \leftarrow \mu(s) + \gamma$
  $S \leftarrow S \cup \{s\}$
  if $\mu(v) == 0$ **then**
    $S \leftarrow S \setminus \{v\}$
  end if
end if

Given the measure μ, we can immediately calculate the associated point x (Line 9). Since we use a Dirac measure as input, along with the fact that Frank-Wolfe takes in at most one atom each iteration, μ is always a finite measure and can be hence computed exactly (ignoring the precision limit of the floating-point representation). The reverse is generally not true, except for some special situations (*e.g.*, simplex). The active set is just the support of the measure μ (Line 10). When $\mathcal{A}$ forms an atomic norm ball, we can also use the origin as the initial point. Indeed, we can add the origin as an extra atom without the risk of violating the nice property of $\mathcal{A}$ to form a norm.

Once we get the initial point x, we can ask for the gradient at this point via the *gradient evaluation oracle* (Line 12). This oracle can return either the exact gradient or a perturbed one if a nonzero η_1 is given. The latter is often called the stochastic gradient and widely used in various optimization applications.

Then, we request the destination atom s via the *linear minimization oracle* (Line 13). It takes the gradient, the atoms, and an extra argument η_2, which indicates the quality of the solution that the oracle is expected to give. When $\eta_2 = 0$, it means that the exact solution is expected. When $\eta_2 \neq 0$, it may have different meanings depending on how the oracle is concretized. Naturally, the vanilla Frank-Wolfe direction is $s - x$ (Line 14).

If we wish to use the away-step or the pairwise step variant, we will have to calculate $\arg\max_{v \in S} \langle h, v \rangle$ (Line 15–25). The atom obtained is the one yielding the largest *ascent* direction and is what we may want to remove from our current active

Table 3.1 – Virtual functions and concrete functions

Virtual functions	Concrete functions
`GEO`	`EGE`, `SGE` ...
`LMO`	`ELS`, `ALS` ...
`SSO`	`DSS`, `CSS`, `LSS` ...

set. By removing this atom, the iterate is pushed onto the face (if it is a simplex) opposite to the atom in question; it is hence called the away step. In general, an away step uses either the away direction or the vanilla direction, depending on which direction looks more "efficient". On the contrary, a pairwise step always combines the vanilla step and the away step together, which makes the maximal use of the available resources. Empirically, it often outperforms the away step (Lacoste-Julien and Jaggi 2015).

Then, we should choose a step size via the *step size oracle* (Line 26). It takes a starting point, a direction and a maximal step size allowed (the minimal step size is of course zero). This can be done by the line search or, in some literature (*e.g.*, Jaggi 2013), they use a default step size $\frac{2}{t+2}$, where t is the number of iteration. I recommend Freund and Grigas (2016) for an extensive study on the choice of step size.

Once the iterate moves (Line 27), the measure μ changes too. `VRU` (Algorithm 4) ensures that μ is updated accordingly. It is worth mentioning that, because of numeric precision, the equality may not be precisely satisfied. Some quantities which should be zero may end up with strict positive values. This numeric issue should be taken into account during the implementation.

3.3.2 Variants as special cases of the general form

This subsection shows how to use the above presented general form to express various concrete Frank-Wolfe examples. In particular, the virtual functions (*a.k.a.* oracles) will all be replaced by concrete functions. To help the readers recall the pairing, I have made Table 3.1.

First, let us introduce some concrete algorithms for `GEO`. Algorithm 5 takes the current iterate as input and outputs the (exact) gradient at this point. If $f = \sum_{i=1}^{n} f_i$, then its time complexity is proportional to n. We can also consider inexact gradient information as in Jaggi (2013, Section 3). When a perturbed gradient is enough, we can use the *stochastic gradient evaluation* (Algorithm 6) to accelerate the computation. If each f_i is an independent stochastic process from the same distribution, it can be done by sampling a subset of $\{f_i\}_{i=1}^{n}$. Then, the time complexity is proportional to the size of this subset. This approach is extensively used in stochastic Frank-Wolfe (Lafond et al. 2015, Ouyang and Gray 2010, Hazan and Luo 2016, Goldfarb et al. 2017, Liu and Tsang 2017). For an inexact but non-stochastic gradient evaluation, I refer to the δ-model of d'Aspremont (2008).

Then, let us have a close look at $\texttt{LMO}(h, \mathcal{A}, \eta_2)$. This oracle is problem dependent. For instance, if $\mathcal{A}$ has finite elements, we can use the simplex method. When it is

Algorithm 5. Exact gradient evaluation: `EGE`(x)

Require: ∇f
Input: x
Output: $\nabla f(x)$

Algorithm 6. Stochastic gradient evaluation: `SGE`$(x,\ \eta)$

Require: Stochastic gradient $f'(a,b) := \nabla f(a) + b\xi$, where $\xi \sim \mathcal{N}(0,1)$
Input: $x,\ \eta$
Output: $f'(x,\eta)$

infinite, it may require some domain specific knowledge. In case the exact solution is computationally expensive, we can use an approximate solution. Algorithm 7 shows how it should look like for an exact linearization solver. Algorithm 8 shows how an approximate linearization solver looks like (Jaggi 2013).

Algorithm 7. Exact linearization solver: `ELS`$(h,\ \mathcal{A})$

Input: $h,\ \mathcal{A}$
Output: $\arg\min_{s\in\mathbf{conv}(\mathcal{A})} \langle s, h\rangle$

Algorithm 8. Approximate linearization solver: `ALS`$(h,\ \mathcal{A},\ \eta)$

Input: $h,\ \mathcal{A},\ \eta$
Find $s \in \mathbf{conv}(\mathcal{A})$ s.t. $\langle s, h\rangle \leq \min_{s\in\mathbf{conv}(\mathcal{A})} \langle s, h\rangle + \eta$
Output: s

Finally, we have `SSO`. For Frank-Wolfe, the line search (Algorithm 9: `LSS`) is always preferred, especially when it has a closed form. Besides the line search, we can use the default step size $\frac{2}{2+t}$ defined in Jaggi (2013) (Algorithm 10: `DSS`). However, it often results in fluctuations. Another approach is to use constant step size (Algorithm 11: `CSS`). Freund and Grigas (2016) prove that Frank-Wolfe converges to a neighborhood of the optimum point when `CSS` is used.

Algorithm 9. Line search step: `LSS`$(x,\ d,\ \gamma_{\max})$

Input: $x,\ d,\ \gamma_{\max}$
Output: $\arg\min_{\gamma\in[0,\gamma_{\max}]} f(x+\gamma d)$

Algorithm 10. Default step size: `DSS`$(\gamma_{\max})$

Require: t
Input: $\gamma_{\max}$
Output: $\frac{2}{2+t} \wedge \gamma_{\max}$

Algorithm 11. Constant step size: `CSS`$(\gamma_{\max})$

Require: c
Input: $\gamma_{\max}$
Output: $c \wedge \gamma_{\max}$

`GEO` provides the freedom to use the exact or the stochastic gradient; `LMO` provides the freedom to solve the linear subproblem exactly or approximately; `SSO` provides the freedom to choose the step size; `flag_away` and `flag_pair` provide the freedom to use away-step or pairwise Frank-Wolfe. With their combination, we can express many Frank-Wolfe variants except for fully-corrective Frank-Wolfe.

For example,

$$\texttt{FW}(\{0, \pm e_i\}_{i=1}^n, \delta_0, 0.1, 0, \texttt{SGE}, \texttt{ELS}, \texttt{LSS}, \texttt{True}, \texttt{False})$$

expresses solving Lasso via stochastic away-step Frank-Wolfe by solving the exact linear subproblem and using the line search step size. It is not restrictive for us to use the unit ball here since we can always scale the feature space so that the constraint set is a unit ball.

For fully-corrective variant, replace all contents in the loop after the linear minimization oracle by Algorithm 12.

Algorithm 12. Fully-corrective variant
Replace Line 14–28 in Algorithm 3 by

$x \leftarrow \arg\min_{x \in \mathbf{conv}(S \cup \{s\})} f(x)$
update S accordingly

3.4 Deterministic and nondeterministic convergence rates

This section presents the sublinear convergence rate of Frank-Wolfe and my efforts to extend it. It starts with the deterministic rate when the subproblem is solved approximately but deterministically, which is a classic result about Frank-Wolfe. Then, to present my contributions, it introduces the stochastic approximate linearization solvers, which solve the subproblem approximately and nondeterministically. Lastly, it proves that, when the approximate linearization solvers are used, Frank-Wolfe converges sublinearly in expectation or with high probability. Along the presentation, I also gives a proof of the boundedness of the global curvature when the objective function is Lipschitz continuous.

3.4.1 Deterministic convergence rates

In this subsection, I will present the sublinear convergence rate of Frank-Wolfe under the least restrictive condition, which is enough for the remaining of this dissertation. This dissertation is *not* an encyclopedic analysis of all convergence rates. For readers curious to learn all about the theory, Section 3.1.1 serves as a guide.

For a convex and differentiable function $f : \mathbb{R}^n \to \mathbb{R}$, we define its *global curvature* with regard to a compact domain $\mathcal{D}$ as

$$C_{\mathcal{D}}^f := \sup_{\substack{x, s \in \mathcal{D} \\ \gamma \in [0,1] \\ y = x + \gamma(s - x)}} \tfrac{2}{\gamma^2}(f(y) - f(x) - \langle y - x, \nabla f(x) \rangle). \tag{3.3}$$

This definition is slightly different from the *curvature* in differential geometry, which is defined as the length of the second derivative.

Curvature is a local property and varies through affine transformation of the space, whereas *global* curvature is a global property characterizing both the function f and the domain $\mathcal{D}$ and is invariant to affine transformation. To see this, let $y = \lambda y'$ and $x = \lambda x'$. Then, the domain becomes $\frac{1}{\lambda}\mathcal{D}$, without affecting γ, and we have

$$\begin{aligned}
&f(y) - f(x) - \langle y - x, \nabla f(x)\rangle) \\
=&f(\lambda y') - f(\lambda x') - \left\langle \lambda(y' - x'), \nabla_{\lambda x'} f(\lambda x')\right\rangle \\
=&f(\lambda y') - f(\lambda x') - \left\langle \lambda(y' - x'), \tfrac{1}{\lambda}\nabla_{x'} f(\lambda x')\right\rangle \\
=&f(\lambda y') - f(\lambda x') - \left\langle y' - x', \nabla_{x'} f(\lambda x')\right\rangle .
\end{aligned}$$

Indeed, $f(y) - f(x) - \langle y - x, \nabla f(x)\rangle$ is the *Bregman divergence* $D_f(y, x)$ between y and x. It represents the error to approximate $f(y)$ by the linearization of f at x. When the function f is itself linear, the (global) curvature becomes 0. When $f(x) = \frac{1}{2}\|x\|_2^2$ on $\mathbb{R}^n$, the Bregman divergence is exactly the squared Euclidean distance, and global curvature is the squared (Euclidean) diameter of the domain $\mathcal{D}$.

More generally, Theorem 3.4.1 gives an upper bound about the global curvature. This theorem is proved in Jaggi (2013, Lemma 7) with ambiguous statement and claimed in Lacoste-Julien and Jaggi (2015) without proof.

Theorem 3.4.1. *Let f be a convex and differentiable function with its gradient ∇f being L-Lipschitz continuous on $\mathcal{D}$ with regard to some arbitrary norm $\|\cdot\|$ in dual pairing, then*

$$C_{\mathcal{D}}^f \leq L \operatorname{diam}_{\|\cdot\|}(\mathcal{D})^2, \tag{3.4}$$

where $\operatorname{diam}_{\|\cdot\|}$ *is the diameter of a set according to the norm* $\|\cdot\|$.

Proof. Use the proof in the appendix of Jaggi (2013, Lemma 7) and replace the reference (Nesterov, 2004, Lemma 1.2.3) therein by Theorem 2.1.8. Here, for the self-completeness, I reproduce the proof.

By Theorem 2.1.8, we have that for any $x, y \in \mathcal{D}$,

$$f(y) - f(x) - \langle y - x, \nabla f(x)\rangle \leq \tfrac{L}{2}\|y - x\|^2 .$$

Observing that for any $x, s \in \mathcal{D}$, we have that also $y := x + \gamma(s - x) \in \mathcal{D}$ and $\frac{1}{\gamma^2}\|y - x\|^2 = \|s - x\|^2$, we can therefore upper bound the curvature as

$$C_f \leq \sup_{x,s,\gamma} \tfrac{2}{\gamma^2}\tfrac{L}{2}\|y - x\|^2 = \sup_{x,s} L\|s - x\|^2 \leq L \operatorname{diam}_{\|\cdot\|}(\mathcal{D})^2,$$

which is the claimed bound. □

From this, we can see that global curvature incorporates both the geometry of the function and the constraint set. It characterizes the smoothness of the problem, and it is thus natural that global curvature is affine invariant.

The upper boundedness of the global curvature is important in that it is the constant in the sublinear convergence rate. The following proposition shows that after $O(\frac{1}{\varepsilon})$ many iterations, the iterate x_t of Frank-Wolfe is an ε-approximate solution. We call this convergence rate *sublinear*, which is generally considered slow. Besides the convergence rate, we observe that it is proportional to the global curvature. This illustrates that the global curvature reflects the complexity of the problem.

Proposition 3.4.2 (Theorem 1 of Jaggi (2013))**.** *Let f be a differentiable convex function with a finite global curvature $C_{\mathcal{D}}^f$ on the constraint set $\mathcal{D}$. Let $\{x_t\}_{t\geq 0}$ be the iterates of Frank-Wolfe, and $\delta \geq 0$ be a small constant. The iterates generated by Frank-Wolfe using $\eta_t = \frac{\delta C_{\mathcal{D}}^f}{t+2}$ in Algorithm 8: `ALS` (Algorithm 7: `ELS` if $\delta = 0$) satisfy*

$$f(x_t) - f(x^*) \leq \frac{2C_{\mathcal{D}}^f}{t+2}(1+\delta), \quad \forall t > 0, \tag{3.5}$$

if any of the following three methods are used:

1. *using Algorithm 10: `DSS` (default step size) as the step size oracle;*
2. *using Algorithm 9: `LSS` (line search step size) as the step size oracle;*
3. *Algorithm 12: fully-corrective Frank-Wolfe is used.*

The proof can be found in Jaggi (2013, Appendix A) and will be omitted here. We shall see a generalization of this proposition soon, and the proof for that generalization is valid here too.

Remark. Among the above results is missing Algorithm 11: `CSS` (constant step size). In fact, Freund and Grigas (2016) proves that it converges to a neighborhood of the optimum instead of to the optimum itself.

3.4.2 Stochastic approximate linearization solver

The approximate linearization solver requires that the error *must* be inferior to η, which is unrealistic in some cases. Imagine that we are calculating the largest eigenvalue of a positive semi-definite matrix via the power iteration. If the initial vector is unfortunately independent of the eigenvector associated with the largest eigenvalue, we can by no means get the correct largest eigenvalue. This difficulty motivates us to find a *stochastic* approximate linearization solver and to establish dedicated convergence theory for it. From this section on, if not explicitly mentioned, I use the vanilla direction, and I suppose that the gradient is exact.

Definition 3.4.1. Let $\langle \cdot, \cdot \rangle$ be the inner product of the space in question and random variable $x \in \mathcal{D}$ the iterate. For any $\eta \geq 0$, we say that the linear subproblem is approximately solved *deterministically* if we find a deterministic relation $s^* := s^*(x) \in \mathcal{D}$ subject to

$$\langle s^*, \nabla f(x) \rangle \leq \min_{s \in \mathcal{D}} \langle s, \nabla f(x) \rangle + \eta. \tag{3.6}$$

It is solved *in expectation* if we find a random variable $\hat{s}$ subject to

$$\langle \mathbb{E}[\hat{s}|x], \nabla f(x) \rangle \leq \min_{s \in \mathcal{D}} \langle s, \nabla f(x) \rangle + \eta. \tag{3.7}$$

It is solved *with high probability* if, for any $\alpha > 0$, we find a random variable $\hat{s}$ subject to

$$\Pr\left(\langle \hat{s}, \nabla f(x)\rangle > \min_{s\in\mathcal{D}}\langle s, \nabla f(x)\rangle + \eta\right) < \alpha. \tag{3.8}$$

Here, we take the exact gradient. Algorithmically, it is not a big issue to use the stochastic gradient; my theorem will only cover the exact gradient case though. (3.6) is exactly the one used in Algorithm 8: `ALS`. (3.7) means sometimes the error of the output $\hat{s}$ may be lower than η; other times it may be higher than η; but, in average, the expectation conditional on the current iterate is lower than η. Algorithm 13: `SALS1` uses this solver. (3.8) means that the probability of the solution failing to get an error lower than η is lower than ε. In practice, by the *law of total probability*, we can enforce a stronger condition by using the conditional probability:

$$\Pr\left(\langle \hat{s}, \nabla f(x)\rangle > \min_{s\in\mathcal{D}}\langle s, \nabla f(x)\rangle + \eta \middle| x\right) < \alpha. \tag{3.9}$$

Algorithm 14: `SALS2` uses this solver.

Algorithm 13. Stochastic approximate linearization solver I: `SALS1`$(h, \mathcal{A}, \eta)$

Input: h, $\mathcal{A}$, η
Find $s \in \mathcal{D} := \mathbf{conv}(\mathcal{A})$ s.t. $\langle \mathbb{E}[s|h], h\rangle \leq \min_{s\in\mathcal{D}}\langle s, h\rangle + \eta$
Output: s

Algorithm 14. Stochastic approximate linearization solver II: `SALS2`$(h, \mathcal{A}, \eta, \alpha)$

Input: h, $\mathcal{A}$, η, α
Find $s \in \mathcal{D} := \mathbf{conv}(\mathcal{A})$ s.t. $\Pr(\langle s, h\rangle > \min_{s\in\mathcal{D}}\langle s, h\rangle + \eta) < \alpha$
Output: s

3.4.3 Nondeterministic convergence rates

With these stochastic solvers, I can extend Proposition 3.4.2. The following theorem means that, if the linear subproblem is solved in expectation, Frank-Wolfe will converge in expectation; and that, if the linear subproblem is solved with high probability, Frank-Wolfe will converges with high probability. It constitutes, along with the next two sections, my major contribution presented in this chapter.

Theorem 3.4.3. *Let f be a differentiable convex function with a finite global curvature $C_{\mathcal{D}}^f$ on the constraint set $\mathcal{D}$. Let $\{x_t\}_{t\geq 0}$ be the iterates of Frank-Wolfe, and $\delta \geq 0$ be a small constant. Suppose that we use Algorithm 10: `DSS` (default step size) as the step size oracle, which gives $\gamma_t = \frac{2}{t+2}$. The following statements are true.*

1. *If Algorithm 13: `SALS1`$(\nabla f, \mathcal{A}, \frac{\delta C_{\mathcal{D}}^f}{t+2})$, $t \geq 0$ are used as the linear minimization oracle, then the algorithm converges sublinearly in expectation*

$$\mathbb{E}[f(x_t)] - f(x^*) \leq \frac{2C_{\mathcal{D}}^f}{t+2}(1+\delta), \quad \forall t > 0. \tag{3.10}$$

2. *For any* $\alpha \in [0,1]$ *and any finite horizon* T*, if Algorithm 14:* `SALS2`$(\nabla f, \mathcal{A}, \frac{\delta C_{\mathcal{D}}^f}{t+2}, \frac{\alpha}{T})$*,* $t = 1, \ldots, T$ *are used as the linear minimization oracle, then the algorithm converges sublinearly* in probability *over a finite horizon*

$$\Pr\left(f(x_t) - f(x^*) > \frac{2C_{\mathcal{D}}^f}{t+2}(1+\delta), \quad t = 1, \ldots, T\right) \leq \alpha. \tag{3.11}$$

3. *For any* $\alpha \in [0,1]$*, if Algorithm 14:* `SALS2`$(\nabla f, \mathcal{A}, \frac{\delta C_{\mathcal{D}}^f}{t+2}, \frac{\alpha}{2^t})$*,* $t > 0$ *are used as the linear minimization oracle, then the algorithm converges sublinearly* in probability *over an infinite horizon*

$$\Pr\left(f(x_t) - f(x^*) > \frac{2C_{\mathcal{D}}^f}{t+2}(1+\delta), \quad t > 0\right) \leq \alpha. \tag{3.12}$$

Proof. Let us begin with the first statement. The proof is a direct adaption of Jaggi (2013, Lemma 5 and Theorem 1). For any step $x_{t+1} := x_t + \gamma_t(s_t - x_t)$ with the destination point s_t generated by `LMO`, we have

$$f(x_{t+1}) \leq f(x_t) + \gamma_t \langle s_t - x_t, \nabla f(x_t)\rangle + \tfrac{\gamma_t^2}{2} C_{\mathcal{D}}^f$$

by the definition of global curvature.

Take conditional expectation on both sides, we get

$$\begin{aligned}
\mathbb{E}[f(x_{t+1})|x_t] &\leq f(x_t) + \gamma_t \langle \mathbb{E}[s_t|x_t] - x_t, \nabla f(x_t)\rangle + \tfrac{\gamma_t^2}{2} C_{\mathcal{D}}^f \\
&\leq f(x_t) + \gamma_t \left(\min_{s\in\mathcal{D}} \langle s - x_t, \nabla f(x_t)\rangle\right) + \gamma_t \eta_t + \tfrac{\gamma_t^2}{2} C_{\mathcal{D}}^f \\
&= f(x_t) + \gamma_t \left(\min_{s\in\mathcal{D}} \langle s - x_t, \nabla f(x_t)\rangle\right) + \tfrac{\gamma_t^2}{2} C_{\mathcal{D}}^f (1+\delta) \\
&= f(x_t) - \gamma_t g(x_t) + \gamma_t^2 C
\end{aligned}$$

where $g(x_t)$ is the duality gap at x_t and $C := \frac{C_{\mathcal{D}}^f}{2}(1+\delta)$.

Define $b(x) := f(x) - f(x^*) \leq g(x)$, which is the suboptimality error. By subtracting $f(x^*)$ on both sides, we get

$$\begin{aligned}
\mathbb{E}[b(x_{t+1})|x_t] &\leq b(x_t) - \gamma_t g(x_t) + \gamma_t^2 C \\
&\leq b(x_t) - \gamma_t b(x_t) + \gamma_t^2 C \\
&= (1-\gamma_t) b(x_t) + \gamma_t^2 C.
\end{aligned}$$

Now, we shall use induction over t to prove the sublinear convergence in expectation (3.10), *i.e.*,

$$\mathbb{E}[b(x_t)] \leq \tfrac{4C}{t+2}, \quad t = 1, 2, \ldots$$

When $t = 1$, we have $\gamma_0 = \frac{2}{0+2} = 1$ and $x_1 = s_0 \in \mathcal{D}$. For any $x \in \mathcal{D}$, we have $b(x) \leq \frac{C_{\mathcal{D}}^f}{2} < C < \frac{4}{3}C$. This proves the case of $t = 1$.

If it is true for t, then

$$\begin{aligned}\mathbb{E}[b(x_{t+1})] &= \mathbb{E}[\mathbb{E}[b(x_{t+1})|x_t]] \\ &\leq (1-\gamma_t)\mathbb{E}[b(x_t)] + \gamma_t^2 C \\ &\leq \left(1 - \tfrac{2}{t+2}\right)\tfrac{4C}{t+2} + \left(\tfrac{2}{t+2}\right)^2 C.\end{aligned}$$

Simply rearranging the terms gives

$$\mathbb{E}[b(x_{t+1})] \leq \tfrac{4(t+1)C}{(t+2)^2} < \tfrac{4(t+1)C}{(t+1)(t+3)} = \tfrac{4C}{t+3}.$$

This concludes the convergence in expectation.

Now, let us prove the convergence with high probability. For any $\alpha \geq 0$, `SALS2` generates a finite sequence for the finite horizon, so that each subproblem fails with the probability $\frac{\alpha}{T}$; it generates an infinite sequence for the infinite horizon, so that each subproblem fails with the probability $\frac{\alpha}{2^t}$. Therefore, with probability $1-\alpha$, all subproblems are solved for both the finite horizon and the infinite horizon. This is all what we need for the convergence in probability. □

In (3.11) and (3.12), the time index t is inside the probability measure, which means that all these inequalities are *simultaneously* satisfied. For finite horizons, this is not a big difference. However, for the infinite horizon, it is a much stronger result than moving $t > 0$ out of the probability measure.

Although this theorem claims only for `DSS`, it is not difficult to prove its validity for the line search and the fully-corrective variant. Indeed, when using the line search Frank-Wolfe, each descent must be lower than when using the default step size. Therefore, it converges no less than the above rate. For the fully-corrective Frank-Wolfe, the descent is even lower. Indeed, it can be regarded as a search over the entire subspace generated by the active set.

This theorem is a generalization of Proposition 3.4.2. To see this, either let s_t be deterministic about x_t, or let the parameter α be 0 in the theorem. It recovers Proposition 3.4.2 as a special case.

Remark. Among these step size oracles is missing the constant step size. In fact, one can prove that, under appropriate conditions, Frank-Wolfe converges to a neighborhood of the optimum in expectation and with high probability with the help of Freund and Grigas (2016, Theorem 5.1).

3.5 Frank-Wolfe for trace norm minimization

In this section, we consider the trace norm minimization problem, which is a special case of (3.1). In particular, we are in a matrix space with the inner product defined by $\langle A, B\rangle := \mathrm{Tr}(A^T B)$, where Tr is the trace of a matrix. Let f be a convex differentiable matrix function. We consider the optimization

- on the general matrix space

$$\begin{aligned} \min_{W\in\mathbb{R}^{m\times n}} \quad & f(W) \\ \text{s.t.} \quad & \|W\|_{\mathtt{tr}} \leq \theta, \end{aligned} \tag{3.13}$$

- on the symmetric matrix space

$$\begin{aligned} \min_{W\in\mathbb{S}^{n}} \quad & f(W) \\ \text{s.t.} \quad & \|W\|_{\mathtt{tr}} \leq \theta, \end{aligned} \tag{3.14}$$

  where $\mathbb{S}^n := \{W \in \mathbb{R}^{n\times n} : W^T = W\}$,

- and on the positive semidefinite (PSD) matrix cone

$$\begin{aligned} \min_{W\in\mathbb{S}^{n}_{+}} \quad & f(W) \\ \text{s.t.} \quad & \|W\|_{\mathtt{tr}} \leq \theta, \end{aligned} \tag{3.15}$$

  where $\mathbb{S}^n_+ := \{W \in \mathbb{S}^n : W \succeq 0\}$.

I separate these three cases because they each entail a different linear subproblem. Although they are all treated with power iteration, each of them requires a slightly different micro-management.

They each have their applications. For example, (3.13) is useful for recommender systems (Koren et al. 2009), multi-task learning (Caruana 1998) and so forth; (3.14) is useful for learning a Boltzmann machine (Ackley et al. 1985, Ping et al. 2016), where the weights may be symmetric, and for approximating a Hessian matrix, which is symmetric but not necessarily PSD; (3.15) is useful for metric learning (Yao et al. 2014, Liu et al. 2015, Huo et al. 2016), where any PSD matrix W defines a distance through the form $D_W(x_1, x_2) := (x_1 - x_2)^T W(x_1 - x_2)$, and for low-rank output kernel learning (Dinuzzo and Fukumizu 2011), where the kernel is PSD. Besides, some researchers transform (3.13) to (3.15) via the method described in Page 20.

It may be worth mentioning that the above three constraint sets are not the only possibilities. We may impose orthogonality, normality, or instability as well (Higham 1988). I will only study the above three scenarios in my dissertation though.

3.5.1 General matrix space

The linear subproblem associated with (3.13) is

$$\min_{Q\in\mathbb{R}^{m\times n},\, \|Q\|_{\mathtt{tr}}\leq\theta} \quad \langle Q, \nabla f(W)\rangle \,. \tag{3.16}$$

If $\theta = 1$, the minimum will be the negative of the dual norm of the gradient $-\|\nabla f(W)\|_{\mathtt{sp}}$ (recall that the spectral norm is the dual norm of the trace norm). Let $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_{m\wedge n} \geq 0$ be the singular values of the gradient; the optimum is just $-\sigma_1$. Notice that $\nabla f(x)$ has its singular value decomposition $\nabla f(x) = \sum_{i=1}^{m\wedge n} \sigma_i u_i v_i^T$,

Algorithm 15. Solve subproblems in $\mathbb{R}^{m\times n}$

1: **Input:** $\nabla f(W), \theta, K$
2: $A \leftarrow \nabla f(W)$
3: $v \sim \text{Unif}(\mathcal{S}^n)$ ▷ uniformly sample on the unit sphere
4: **for** $k = 1, \dots, K$ **do** ▷ power iteration
5: $u \leftarrow Av$
6: $u \leftarrow \frac{u}{\|u\|_2}$ ▷ left singular vector
7: $v \leftarrow A^T u$
8: $v \leftarrow \frac{v}{\|v\|_2}$ ▷ right singular vector
9: **end for**
10: $u \leftarrow Av$
11: $u \leftarrow \frac{u}{\|u\|_2}$ ▷ left singular vector once more
12: **Output:** $Q \leftarrow -\theta u v^T$

where u_i and v_i are the left and right singular vectors associated with σ_i. To achieve the optimum, it suffices to take $Q = -u_1 v_1^T$. If $\theta \neq 1$, we will take $Q = -\theta u_1 v_1^T$, and the optimum is $-\theta\sigma_1$.

It remains the question how to calculate the singular vectors u_1, v_1 numerically. For this purpose, I adapt the power iteration algorithm, so that it can calculate the largest singular value and the associated singular vectors now.[3] The same algorithm is also documented in *Edo Liberty*'s lecture note.[4]

Without loss of generality, we can suppose that $m \geq n$. The algorithm (Algorithm 15) starts with sampling a random vector uniformly on the sphere and uses it as the initial guess of the right singular vector (Line 3). Then, the algorithm multiplies it with the gradient repeatedly to update the guess of the left and right singular vectors as well as normalizes them (Line 4–11). The number of iterations K will be determined in the next section.

The intuition behind is that $\nabla f(W)^T \nabla f(W)$ is a positive semidefinite matrix of size $n \times n$, whose largest eigenvalue is the *square* of the largest singular value of $\nabla f(W)$, and we can apply the classic power iteration on it. Here, I initiate the right singular vector instead of the left one. If $m < n$, we may initiate the left singular vector instead. This is equivalent to applying the power iteration on $\nabla f(W) \nabla f(W)^T$, which is also a positive semidefinite matrix.

3.5.2 Symmetric matrix space

The linear subproblem associated with (3.14) is

$$\min_{Q\in\mathbb{S}^n,\, \|Q\|_{\text{tr}}\leq\theta} \quad \langle Q, \nabla f(W)\rangle \,. \tag{3.17}$$

[3] Power iteration is originally used to calculate the largest eigenvalue and the associated eigenvector of a positive semidefinite matrix. See Section 3.6.1 for the detail.

[4] http://www.cs.yale.edu/homes/el327/datamining2013aFiles/07_singular_value_decomposition.pdf

For the symmetric matrix space, there is some difficulty: the gradient matrix may not be symmetric. As a counter example, let $f(X) = \frac{1}{2}(x_{11}^2 + 2x_{12}^2 + 3x_{21}^2 + 4x_{22}^2)$, and then $\nabla f(X) = \begin{bmatrix} x_{11} & 2x_{12} \\ 3x_{21} & 4x_{22} \end{bmatrix}$ is not symmetric except for the origin. This inconvenience makes the solution to (3.17) inevident.

However, if f satisfies $f(X) = f(X^T)$ not only for symmetric matrices but also for any $X \in \mathbb{R}^{n\times n}$, then its gradient will be symmetric. Indeed, for any i, j, take the partial gradient on both sides

$$\frac{\partial}{\partial x_{ij}} f(X) = \frac{\partial}{\partial x_{ij}} f(X^T).$$

According to the chain rule, the right-hand side equals to $[\nabla f(X^T)]_{ji}$. If X is symmetric, then it is equal to $[\nabla f(X)]_{ji}$. The gradient is then symmetric for symmetric matrices.

With this condition, we get $\nabla f(X) = \sum_{i=1}^n \sigma_i v_i v_i^T$ for any symmetric X, where $\sigma_1 \geq \sigma_2 \geq \ldots \geq \sigma_n$ are the eigenvalues, and v_i are eigenvectors (with a little abuse of notation). Contrary to the singular values, eigenvalues are not necessarily positive. The key here is the eigenvalue with the largest absolute value, which can be either σ_1 or σ_n. If σ_1 has the largest absolute value, we can take $Q = -\theta v_1 v_1^T$; otherwise, we can take $Q = \theta v_n v_n^T$. The optimum is $-\theta\sigma_1$ and $\theta\sigma_n$ respectively (in the latter case, σ_n must be negative).

For symmetric matrices, the power iteration does not necessarily get us the eigenvalue with the largest absolute value. Consider the matrix $\mathrm{diag}\{1, -1\}$, which has eigenvectors $(1, 0)^T$ and $(0, 1)^T$. If we take $(1, 1)$ as the initial vector for power iteration, we will get $(1, -1)^T$ and $(1, 1)^T$ alternately, which means that power iteration does not converge. However, if we suppose that the matrix has only one eigenvalue with the largest absolute value, we can prove that power iteration converges. Therefore, let us suppose that the set of all Ws whose gradient $\nabla f(W)$ has multiple eigenvalues with the largest absolute value is a *null set* (*i.e.*, measure equal to 0).[5]

The algorithm (Algorithm 16) starts with sampling a random vector uniformly on the sphere and uses it as the initial guess of the eigenvector (Line 3). Then, the algorithm multiplies it with the gradient repeatedly as well as normalizes it (Line 4–7). Lastly, it uses the sign of $v^T \nabla f(W) v$ as an estimator of the sign of the eigenvalue (Line 8). If the estimated eigenvalue is positive, we will return $-\theta v v^T$ (the negative direction is the descent direction). Otherwise, we will return $\theta v v^T$ (the positive direction is the descent direction).

3.5.3 Positive semidefinite cone

The linear subproblem associated with (3.15) is

$$\min_{Q\in\mathbb{S}_+^n, \|Q\|_{\mathrm{tr}}\leq\theta} \langle Q, \nabla f(W)\rangle . \tag{3.18}$$

[5]This problem cannot be solved by combining every two power iterations as a pair, which makes the algorithm converge but not necessarily to an eigenvector.

Algorithm 16. Solve subproblems in $\mathbb{S}^n$

Input: $\nabla f(W), \theta, K$
$A \leftarrow \nabla f(W)$ ▷ A is symmetric
$v \sim \text{Unif}(\mathcal{S}^n)$ ▷ uniformly sample on the unit sphere
for $k = 1, \ldots, K$ **do** ▷ power iteration
    $v \leftarrow Av$
    $v \leftarrow \frac{v}{\|v\|_2}$
end for
Output: $Q \leftarrow -\operatorname{sgn}(v^T Av)\theta vv^T$ ▷ the opposite sign of the eigenvalue

For the positive semi-definite cone, the situation is even more complicated. With the condition the same as the above, the gradient is symmetric but not necessarily positive semi-definite. As a counter example, taking $f(X) = \frac{1}{2}[(x_{11} - 1)^2 + x_{22}^2]$, the gradient is thus $\nabla f(X) = \text{diag}\{x_{11} - 1, 1\}$, and then $\nabla f\left(\begin{bmatrix} 0 & 0 \\ 0 & 1 \end{bmatrix}\right) = \begin{bmatrix} -1 & 0 \\ 0 & 1 \end{bmatrix}$, which is not a positive semi-definite matrix.

However, we do not need the gradient to be positive semi-definite; the symmetry suffices. Indeed, we shall pick the smallest eigenvalue σ_n and the associated eigenvector v_n. If σ_n is negative, we shall take $Q = \theta v_n v_n^T \in \mathbb{S}_+^n$, and the optimum will be $\theta\sigma_n$ as above. Otherwise, we shall let $Q = 0$ be the origin, and the optimum will thus be 0.

It remains the problem of obtaining the smallest eigenvalue instead of the largest eigenvalue. For this, we transform the smallest eigenvalue to the largest by transforming a symmetric matrix to a positive semidefinite one. Indeed, if f is a continuous differentiable and is L-Lipschitz continuous with regard to the trace norm, then according to Theorem 2.1.6, $\|\nabla f(W)\|_{\text{sp}} \leq L$, which implies that the largest eigenvalue in absolute value is less than L. Therefore, $LI - \nabla f(W)$, where I is the identity matrix, is a positive semidefinite matrix, and its eigenvector associated to the largest eigenvalue is the one associated to the smallest eigenvalue of the original matrix.

The algorithm (Algorithm 17) starts with transforming the gradient to a positive semidefinite matrix. Then, it samples a random vector uniformly on the sphere

Algorithm 17. Solve subproblems in $\mathbb{S}_+^n$

Input: $\nabla f(W), L, \theta, K$
$A \leftarrow LI - \nabla f(W)$ ▷ A is PSD
$v \sim \text{Unif}(\mathcal{S}^n)$ ▷ uniformly sample on the unit sphere
for $k = 1, \ldots, K$ **do** ▷ power iteration
    $v \leftarrow Av$
    $c \leftarrow \|v\|_2$
    $v \leftarrow \frac{v}{c}$
end for
if $c \leq L$ **then** ▷ the smallest eigenvalue is larger than 0
    $v \leftarrow 0$ ▷ return the origin
end if
Output: $Q \leftarrow \theta vv^T$

and uses it as the initial guess of the eigenvector (Line 3). Then, the algorithm multiplies it with the gradient repeatedly as well as normalizes it (Line 4–8). The number of iterations K will be determined in the next section. If the eigenvalue of the transformed matrix is smaller than L, which implies that the eigenvalue of the original matrix is positive, we shall take $Q = 0$ (Line 9–11). Otherwise, we just take θvv^T (Line 12).

3.6 Nondeterministic convergence rates for trace norm minimization

The previous section presented algorithms solving the linear subproblem associated with trace norm minimization but without proving anything about the convergence rate. In this section, I prove that, for general matrix space (3.13) and positive semidefinite cone (3.15), with carefully chosen number of power iterations, Frank-Wolfe converges sublinearly both in expectation and with high probability. For symmetric matrix space (3.14), the convergence remains an open question. These results are the most important contribution of this chapter, and they lay the theoretical foundation of combining power iteration and Frank-Wolfe to solve trace norm minimization, which is one of the premises of the next chapter. I will start with a brief introduction of power iteration and Lanczos iteration. Then, I will use the classic results about power iteration described in Kuczyński and Woźniakowski (1992) to prove my theory.

3.6.1 Power iteration vs. Lanczos iteration

Concerning calculating the eigenvalues and eigenvectors, Power iteration and Lanczos iteration (Lanczos 1950) are the two dedicated tools for this purpose. They are mainly invented for positive semidefinite matrices.

The best way to understand the commonness and difference of these two methods is via the *Krylov* subspace. For an $n \times n$ positive semi-definite matrix A and a vector b, the associated order-r Krylov subspace is defined by $\mathcal{K}_r(A, b) = \text{span}\{b, Ab, A^2b, \dots, A^{r-1}b\}$. It is easy to see that the Krylov subspace contains some information about the matrix A. In particular, when the order-n Krylov subspace has dimension n, it becomes exactly the column space of A, which implies that it contains full information about A. Of course, our task is not to replicate A but to estimate the largest eigenvalue σ_1, which is partial information about A. Therefore, we can expect to obtain an accurate estimator of the eigenvalue from a low-order Krylov subspace.

With the notion of Krylov space, the estimator via power iteration can be expressed as

$$\phi(A, b, r) := \frac{\langle Ax, x\rangle}{\langle x, x\rangle} \quad \text{with} \quad x = A^{r-1}b, \tag{3.19}$$

and the Lanczos iteration can be expressed as

$$\psi(A, b, r) := \max\left\{ \frac{\langle Ax, x\rangle}{\langle x, x\rangle} : 0 \neq x \in \mathcal{K}_r(A, b) \right\} \tag{3.20}$$

(Kuczyński and Woźniakowski 1992). Power iteration uses only the last vector in the generating vectors of the Krylov subspace, whereas Lanczos iteration uses the entire subspace. If the Krylov subspace is the entire space (*i.e.*, has the dimension n), it becomes the definition of the eigenvalue. This is why Lanczos iteration is more accurate than power iteration (also more computationally expensive).

It remains the question how accurate these methods are for the estimation of the largest eigenvalue. If we take $b = v_n$, the eigenvector associated with the smallest eigenvalue, the Krylov space will be one-dimensional $\operatorname{span}\{v_n\}$. It is clearly not possible to find the largest eigenvalue, no matter how many iterations we do. This failure implies that a fixed initial vector b is not a good idea. It is preferred to use randomly initiated vector b, so that the chance that it takes "bad" value is zero. The intuition is that b should (almost surely) *not* be linearly independent of v_1.

Moreover, with the order of the Krylov subspace fixed, it is obvious that the more correlated b and v_1 are, the more accurate the estimator will be. In particular, if $b = v_1$, order-1 Krylov subspace is enough to get the exact eigenvalue. This randomness implies that the accuracy of the estimator itself is a random variable and must be evaluated accordingly. We define the *average relative error* of the estimator ξ as

$$e(\xi) := \int_{\xi} \left| \frac{\xi - \sigma_1}{\sigma_1} \right| \mu(\mathrm{d}\xi), \tag{3.21}$$

and the probabilistic ε relative failure as

$$e'(\xi) := \mu \left\{ \xi : \left| \frac{\xi - \sigma_1}{\sigma_1} \right| > \varepsilon \right\}, \tag{3.22}$$

where μ is the underlying probability of ξ.

It is clear that it is the *direction* of b that determines the Krylov subspace, not its length. Without loss of generality, we can therefore sample b on the unit sphere. The following proposition is from Kuczyński and Woźniakowski (1992, Theorem 3.1, 3.2, 4.1, 4.2), which shows that, with sampling b on the unit sphere $\mathcal{S}^n$ uniformly, the power iteration has roughly an error in the order of $O(\frac{1}{r})$, and the Lanczos iteration has roughly an error in the order of $O(\frac{1}{r^2})$, regardless of the distribution of the eigenvalues, where r is the order of Krylov subspace.

Proposition 3.6.1. *Let $A_{n\times n}$ be any positive semi-definite matrix, b be a random vector sampled uniformly on the unit sphere $\mathcal{S}^n$.*

- *When $n \geq 8$, for power iteration with order-r Krylov subspace, the average relative error*

$$e(\phi) \leq \frac{\ln n}{r-1}, \tag{3.23}$$

  the probabilistic ε relative failure

$$e'(\phi) \leq \sqrt{n}(1-\varepsilon)^{r-1/2}. \tag{3.24}$$

- *For Lanczos iteration with order-r Krylov subspace, the average relative error*

$$e(\psi) \leq 3\left(\frac{\ln n}{r-1}\right)^2, \tag{3.25}$$

the probabilistic ε relative failure

$$e'(\psi) \leq 2\sqrt{n}e^{-\sqrt{\varepsilon}(2r-1)}. \tag{3.26}$$

In my dissertation, I use only the results about power iteration and build the theory only for it; an extension to Lanczos iteration is evident though.

3.6.2 General matrix space

Theorem 3.4.3 states that, if the linear subproblem is solved in expectation and/or with high probability, Frank-Wolfe will also converge in expectation and/or with high probability. Proposition 3.6.1 states that, for positive semidefinite matrices, power iteration indeed solves the subproblem in expectation and with high probability. By combining these two results and making some adaption for the general matrices, I get the following theorem.

Theorem 3.6.2. *Suppose that $m \geq n$ and $f : \mathbb{R}^{m\times n} \to \mathbb{R}$ is L-Lipschitz continuous with regard to the trace norm, differentiable, convex, and has finite global curvature $C_{\mathcal{D}}^f$. $b_0, b_1, b_2, \ldots,$ are independently sampled from the unit sphere $\mathcal{S}^n$. At Iteration t of Frank-Wolfe, solve the linear subproblem by Algorithm 15 with the initial vector b_t and K_t power iterations. The following statements are true.*

1. *If $K_t \geq \frac{L\theta(t+2)\ln n}{\delta C_{\mathcal{D}}^f}$ for any t, then (3.10) holds.*
2. *If $K_t \geq \frac{L\theta(t+2)\ln \frac{nT}{\alpha}}{\delta C_{\mathcal{D}}^f}$ for any t, then (3.11) holds.*
3. *If $K_t \geq \frac{L\theta(t+2)(t\ln 2+\ln \frac{n}{\alpha})}{\delta C_{\mathcal{D}}^f}$ for any t, then (3.12) holds.*

Proof. To simplify the notation, I omit the time subscript t. Denote $A := \nabla f(W)$ as the gradient matrix and $\sigma \leq L$ as the largest singular value.

Let us start with the first statement. Applying Proposition 3.6.1 to A^TA, we get

$$\mathbb{E}\left|\frac{v^TA^TAv - \sigma^2}{\sigma^2}\right| \leq \frac{\ln n}{K}.$$

Therefore,

$$\mathbb{E}\left|\frac{\|Av\|_2}{\sigma} - 1\right| \leq \mathbb{E}\left|\frac{\|Av\|_2}{\sigma} - 1\right|\left|\frac{\|Av\|_2}{\sigma} + 1\right| = \mathbb{E}\left|\frac{\|Av\|_2^2}{\sigma^2} - 1\right| \leq \frac{\ln n}{K}.$$

Rearranging the terms, we get

$$\sigma - \mathbb{E}\,\|Av\|_2 \leq \tfrac{L\ln n}{K}.$$

Therefore,

$$\mathbb{E}\left\langle uv^T, A\right\rangle = \mathbb{E}[u^TAv] = \mathbb{E}\left[\frac{v^TA^TAv}{\|Av\|_2}\right] = \mathbb{E}\,\|Av\|_2 \geq \sigma - \frac{L\ln n}{K}.$$

Multiplied on both sides with θ, it shows that the sublinear problem is solved with an error of $\frac{\theta L \ln n}{K}$. To make it less than $\frac{\delta C_{\mathcal{D}}^f}{t+2}$, we only need $K \geq \frac{L\theta(t+2)\ln n}{\delta C_{\mathcal{D}}^f}$. Since the subproblem is appropriately solved in expectation, by Theorem 3.4.3, this proves the first statement.

Now let us prove the second and third statement together. Also according to Proposition 3.6.1, for any $\varepsilon > 0$, we have

$$\Pr\left\{\left|\frac{v^T A^T A v - \sigma^2}{\sigma^2}\right| > \varepsilon\right\} \leq \sqrt{n}(1-\varepsilon)^K.$$

Using the same computation as above, we get

$$\begin{aligned}
\Pr\left\{\left\langle -\theta u v^T, A\right\rangle > -\theta\sigma + \varepsilon L\theta\right\} &= \Pr\left\{-\left\|Av\right\|_2 > -\sigma + \varepsilon L\right\} \\
&\leq \Pr\left\{-\left\|Av\right\|_2 > -\sigma + \varepsilon\sigma\right\} \\
&= \Pr\left\{\left|\frac{\left\|Av\right\|_2}{\sigma} - 1\right| > \varepsilon\right\} \\
&= \Pr\left\{\left|\frac{v^T A^T A v - \sigma^2}{\sigma^2}\right| > \varepsilon\right\} \\
&\leq \sqrt{n}(1-\varepsilon)^K.
\end{aligned}$$

Or equivalently, for any $\tilde{\alpha} \in (0,1)$, we have

$$\Pr\left\{\left\langle -\theta u v^T, A\right\rangle > -\theta\sigma + L\theta\left[1 - \left(\frac{\tilde{\alpha}}{\sqrt{n}}\right)^{1/K}\right]\right\} \leq \tilde{\alpha}.$$

In order to make the error less than $\frac{\delta C_{\mathcal{D}}^f}{t+2}$, we need $L\theta\left[1 - \left(\frac{\tilde{\alpha}}{\sqrt{n}}\right)^{1/K}\right] \leq \frac{\delta C_{\mathcal{D}}^f}{t+2}$. Rearranging the terms, we get

$$K \geq \frac{L\theta(t+2)\ln\frac{n}{\tilde{\alpha}}}{\delta C_{\mathcal{D}}^f}.$$

By replacing $\tilde{\alpha}$ by either $\frac{\alpha}{T}$ or $\frac{\alpha}{2^t}$, the subproblem is appropriately solved with high probability, either in finite horizon or infinite horizon. By Theorem 3.4.3, this proves the second and third statement. □

This theorem implies that to get the nondeterministic convergence rate, we should increase the number of power iterations per Frank-Wolfe iteration t. For the convergence in expectation and the one with high probability in finite horizon, linearly growing number of power iterations $O(t)$ suffices. For the convergence with high probability in infinite horizon, quadratically growing number of power iterations $O(t^2)$ suffices.

Note that the independence condition of $b_0, b_1, b_2, \ldots$ cannot be omitted. In fact, if we let $\xi = b_0 = b_1 = b_2 = \cdots$, W_t will be a function of b_t, which makes b_t no longer independent of W_t. Hence, the assumption of Proposition 3.6.1 can no longer be satisfied.

3.6.3 Positive semidefinite cone

Theorem 3.4.3 states that, if the linear subproblem is solved in expectation and/or with high probability, Frank-Wolfe will also converge in expectation and/or with high probability. Proposition 3.6.1 states that, for positive semidefinite matrices, power iteration indeed solves the subproblem in expectation and with high probability.

Nonetheless, as we have seen in the previous sections, the gradient is generally not positive semidefinite, even for positive semidefinite matrices. Therefore, Proposition 3.6.1 still cannot be applied directly, but, after making some adaption, we can prove the following theorem.

Theorem 3.6.3. *Suppose that $f : \mathbb{S}_+^n \to \mathbb{R}$ is L-Lipschitz continuous with regard to the trace norm, differentiable, convex, and has finite global curvature $C_f^{\mathcal{D}}$. $b_0, b_1, b_2, \ldots,$ are independently sampled from the unit sphere $\mathcal{S}^n$. At Iteration t of Frank-Wolfe, solve the linear subproblem by Algorithm 17 with the initial vector b_t and K_t power iterations.*

1. *If $K_t \geq \frac{2L\theta(t+2)\ln n}{\delta C_f^{\mathcal{D}}}$ for any t, then (3.10) holds.*
2. *If $K_t \geq \frac{2L\theta(t+2)\ln \frac{nT}{\alpha}}{\delta C_f^{\mathcal{D}}}$ for any t, then (3.11) holds.*
3. *If $K_t \geq \frac{2L\theta(t+2)(\ln \frac{n}{\alpha}+t\ln 2)}{\delta C_f^{\mathcal{D}}}$ for any t, then (3.12) holds.*

Proof. Let us begin with the first statement. Algorithm 17 applies the power iteration to the transformed gradient. According to Proposition 3.6.1, we have

$$\mathbb{E}\left|\frac{v^T(LI-\nabla f(W))v-(L-\sigma_n)}{L-\sigma_n}\right| \leq \frac{\ln n}{K}$$

before v has ever been moved to 0, where σ_n is the smallest eigenvalue. By simply rearranging the terms, we get

$$\mathbb{E}\left|v^T\nabla f(W)v-\sigma_n\right| \leq \tfrac{2L\ln n}{K}.$$

Then, according to the different values of σ_n, we have two cases, each of which corresponds to a different treatment. To simplify the notation, let us denote $\zeta := \left\langle vv^T, \nabla f(W)\right\rangle = v^T\nabla f(W)v$. If $\sigma_n < 0$, it is obvious that

$$\mathbb{E}\left|\zeta\wedge 0-\sigma_n\right| \leq \mathbb{E}\left|\zeta-\sigma_n\right|.$$

If $\sigma_n \geq 0$, then

$$\begin{aligned}\mathbb{E}\left|\zeta\wedge 0-0\right| &= \mathbb{E}\left|\zeta-0\right|1_{\{\zeta<0\}}+\mathbb{E}\left|0-0\right|1_{\{\zeta\geq 0\}}\\ &\leq \mathbb{E}\left|\zeta-\sigma_n\right|.\end{aligned}$$

Therefore, in both cases, the linear subproblem (3.18) is approximately solved with an error of $\frac{2L\theta\ln n}{K}$ in expectation. In order to make it less than $\frac{\delta C_f^{\mathcal{D}}}{t+2}$, we only need

$K \geq \frac{2L\theta(t+2)\ln n}{\delta C_{\mathcal{D}}^f}$. Since the subproblem is appropriately solved in expectation, by Theorem 3.4.3, this proves the first statement.

Now, let us prove the second and the third statement. Also according to Proposition 3.6.1, for any $\varepsilon > 0$, we have

$$\Pr\left\{\left|\frac{v^T(LI - \nabla f(W))v - (L - \sigma_n)}{L - \sigma_n}\right| > \varepsilon\right\} \leq \sqrt{n}(1-\varepsilon)^K.$$

By rearranging the terms, we get

$$\Pr\left\{\left|v^T \nabla f(W) v - \sigma_n\right| > 2L\varepsilon\right\} \leq \sqrt{n}(1-\varepsilon)^K.$$

Here again we have two cases. If $\sigma_n < 0$, then

$$|\zeta \wedge 0 - \sigma_n| \leq |\zeta - \sigma_n| \quad a.s.$$

So

$$\Pr\left\{|\zeta \wedge 0 - \sigma_n| > 2L\varepsilon\right\} \leq \sqrt{n}(1-\varepsilon)^K.$$

If $\sigma_n \geq 0$, then

$$\begin{aligned}
\Pr\{|\zeta \wedge 0 - 0| > 2L\varepsilon\} &= \Pr\{\zeta \wedge 0 > 2L\varepsilon\} + \Pr\{\zeta \wedge 0 < -2L\varepsilon\} \\
&= 0 + \Pr\{\zeta < -2L\varepsilon\} \\
&\leq \Pr\{\zeta < \sigma_n - 2L\varepsilon\} \\
&\leq \sqrt{n}(1-\varepsilon)^K.
\end{aligned}$$

Therefore, in both cases, the linear subproblem (3.18) is approximately solved with an error at most $2L\theta\varepsilon$ and with a probability no less than $1 - \sqrt{n}(1-\varepsilon)^K$. Or equivalently, for any $\tilde{\alpha} \in (0,1)$, it is solved with an error at most $2L\theta[1 - (\frac{\tilde{\alpha}}{n})^{1/K}]$ and with a probability no less than $1 - \tilde{\alpha}$.

To make this error less than the desired precision:

$$\begin{aligned}
2L\theta\left[1 - \left(\frac{\tilde{\alpha}}{n}\right)^{1/K}\right] &\leq \frac{\delta C_{\mathcal{D}}^f}{t+2} \\
\left(\frac{\tilde{\alpha}}{n}\right)^{1/K} &\geq 1 - \frac{\delta C_{\mathcal{D}}^f}{2L\theta(t+2)} \\
\frac{1}{K}\ln\frac{\tilde{\alpha}}{n} &\geq \ln\left(1 - \frac{\delta C_{\mathcal{D}}^f}{2L\theta(t+2)}\right).
\end{aligned}$$

Using the fact that $\ln(1+x) \leq x$, we can ask for the sufficient condition:

$$\frac{1}{K}\ln\frac{\tilde{\alpha}}{n} \geq -\frac{\delta C_{\mathcal{D}}^f}{2L\theta(t+2)}.$$

By simply rearranging the terms, we get

$$K \geq \frac{2L\theta(t+2)\ln\frac{n}{\tilde{\alpha}}}{\delta C_{\mathcal{D}}^f}.$$

By replacing $\tilde{\alpha}$ by either $\frac{\alpha}{T}$ or $\frac{\alpha}{2^t}$, the subproblem is appropriately solved with high probability, either in finite horizon or infinite horizon. By Theorem 3.4.3, this proves the second and third statement. □

This theorem implies that, to get the nondeterministic convergence rate, we should increase the number of power iterations per Frank-Wolfe iteration t. For the convergence in expectation and the one with high probability in finite horizon, linearly growing number of power iterations $O(t)$ suffices. For the convergence with high probability in infinite horizon, quadratically growing number of power iterations $O(t^2)$ suffices.

Note that the independence condition of $b_0, b_1, b_2, \ldots$ cannot be omitted. In fact, if we let $\xi = b_0 = b_1 = b_2 = \cdots$, W_t will be a function of b_t, which makes b_t no longer independent of W_t and, consequently, violates the assumption of Proposition 3.6.1.

3.7 Conclusion

Trace norm minimization is a problem of great importance in machine learning, statistics, computer vision and so forth. Many researchers solve it with Frank-Wolfe algorithms without a sound theoretic basis. In this chapter, I laid the theoretical foundation of using power iteration (as well as Lanczos iteration) to solve the linear subproblem in Frank-Wolfe algorithms, indirectly making their work rigorous.

I studied the general matrix space, symmetric matrix space, and the positive semidefinite matrix cone and provided heuristic or nonheuristic algorithms, which is the first time these problems are so thoroughly studied. Each of these three cases corresponds to important applications: we can foresee works using Frank-Wolfe to solve high-dimensional trace norm minimization problems in, among others, multi-task learning, recommender systems, Boltzmann machine, metric learning, and kernel learning.

The only drawback in this work is that I failed to prove the nondeterministic convergence for symmetric matrix space: the theoretical convergence of using Algorithm 16 for the subproblem remains an open question.

Chapter 4

Distributed Frank-Wolfe for Trace Norm Minimization

After a long preparation of the theoretical foundation of using power iteration to solve the linear subproblem of Frank-Wolfe, we finally move on to the exciting part of this study – distributed Frank-Wolfe. This study will empower you to use Frank-Wolfe on datasets large both on dimensions and samples.

In this chapter, I will show you how to conduct each step of Frank-Wolfe (*i.e.*, solving the linear subproblem, determining the step size, and updating the iterate) within the Bulk Synchronous Parallel model. Concerning the linear subproblem, like in the previous chapter, I will discuss three cases – general matrix space, symmetric matrix space, and positive semidefinite cone. I have designed distributed versions for all of them. Regarding the update, relying on the property of rank-1 update per Frank-Wolfe iteration, I have designed efficient ways to calculate the gradient recursively. As to the step size, a distributed version of the line search is sometimes feasible.

The above ideas are empirically evaluated on two problems: multi-task least square and multinomial logistic regression, both of which belong to the general matrix scenario. I have developed a package for these problems and evaluated the proposed algorithms on Apache SPARK.

4.1 Introduction

Frank-Wolfe is an important algorithm to solve trace norm minimization. Each of its iterations solves a simple linear subproblem, which has a closed-form solution expressed by the eigenvector (resp. singular vectors) associated with the extreme eigenvalue (resp. singular value) of the gradient. By using power iteration, whose complexity is characterized by matrix-vector multiplications, we can solve high-dimensional (*i.e.*, lots of parameters) machine learning problems.

However, there are two difficulties about this approach. First, for high-dimensional problems, we often need a massive dataset. Although trace norm minimization looks for a low-rank solution, which is by nature less demanding about the data volume, it is still tempting to use the entire dataset when it is available, especially when we are not sure about the low-rank nature of the ground truth. Such kind of massive datasets raises challenges for the computation of the gradient. Its time complexity is proportional to the sample size, which can potentially be one billion. Concerning the space complexity, it is preferred to store the whole dataset in the memory during the entire optimization process so that the gradient can be recomputed efficiently. These two aspects challenge both the memory capacity and the computation power of the computer. Second, the subproblem becomes difficult to solve or store in the memory when the dimensions are extremely large. The time complexity of the calculation of the matrix-vector multiplication for a matrix of size $10^6 \times 10^6$ will be 10^{12}, which is both computationally prohibitive and expensive to fit in the memory of a single machine.

I propose here a distributed Frank-Wolfe algorithm for trace norm minimization. Although it is a distributed algorithm, many of the ideas fit directly in a parallel infrastructure (*i.e.*, shared memory) and can be even more efficient than a naive parallel algorithm. My algorithms are a response to the above two difficulties. They can be *generally*[1] categorized as data parallel algorithms (see Page 31) in that the data are split among several nodes. Splitting the data is equivalent to splitting the loss function. Mathematically, let f be the loss function characterized by the entire dataset. For the i-th portion of the dataset, we denote its correspondent loss function as f_i. Naturally, we have $f = \sum_i f_i$, which is common in statistics and machine learning. Then, each node calculates the gradient of f_i, denoted as ∇f_i, locally. Since the derivative is a linear operator, we have $\nabla f = \sum_i \nabla f_i$, which implies that we can sum all local gradients to get the whole gradient. While this is straightforward in a parallel system, it can be communicationally expensive in a distributed system when the gradient has high dimensions. Therefore, three other low communication cost strategies are designed for this purpose.

At first glance, this simple data parallelism may not be able to handle the second challenge, where the parameter dimension is extremely large. I shall show that, with a particular implementation (which I called low-rank representation) incorporating the parameters into the data, this difficulty vanishes. In other words, the parameter is digested by the data and then be distributed among the nodes.

In particular, I consider the Bulk Synchronous Parallel (BSP) model in a star network. The central node of the network is called *master*, and the other nodes are called *slaves*, with the slave i holding the function f_i. The slaves communicate with the master to solve the trace norm minimization problem via Frank-Wolfe in a distributed manner. Like in the centralized Frank-Wolfe, the distributed Frank-Wolfe consists of three steps: solving the linear subproblem, determining the step size, and updating. They are executed with the concerted effort of the master and the slaves (Algorithm 18). They initialize the parameter matrix with 0 (Line 1) and then enter the Frank-Wolfe loop. In each loop, *a.k.a. epoch*, they solve the linear subproblem approximately with a precision η (Line 3), determine the step size (Line 4), and

[1] There exist tricks to integrate the parameters into data.

Algorithm 18. General form of distributed Frank-Wolfe

1: **Master & Slaves:** $W \leftarrow 0$ ▷ initialize the parameter matrix W with 0
2: **repeat**
3: **Master & Slaves:** $Q \leftarrow \text{Subproblem}(f, W, \eta)$
4: **Master & Slaves:** Determine step size γ
5: **Master & Slaves:** $W \leftarrow (1-\gamma)W + \gamma Q$
6: **until** stopping criterion is satisfied.
7: **Output:** W

update it accordingly (Line 5). They repeat this process until the stopping criterion is met (Line 6). In my dissertation, I consider a fixed number of epochs.

Among the three steps above, the linear subproblem is the most difficult since no slave knows the entire gradient. A naive solution is to centralize all local gradients to the master, which is obviously communicationally expensive. In my dissertation, I study four strategies – gradient centralizing, average mixture, distributed power iteration, and distributed power iteration with warm start – for this problem and discuss their advantages and disadvantages. Recalling that the subproblem comes with slightly different flavor depending on the scenario, I will investigate the general matrix space

$$\begin{aligned} \min_{W \in \mathbb{R}^{d \times m}} \quad & f(W) \\ \text{s.t.} \quad & \|W\|_{\mathtt{tr}} \leq \theta, \end{aligned} \tag{4.1}$$

the symmetric matrix space

$$\begin{aligned} \min_{W \in \mathbb{S}^{d}} \quad & f(W) \\ \text{s.t.} \quad & \|W\|_{\mathtt{tr}} \leq \theta, \end{aligned} \tag{4.2}$$

and the positive semidefinite cone

$$\begin{aligned} \min_{W \in \mathbb{S}^{d}_{+}} \quad & f(W) \\ \text{s.t.} \quad & \|W\|_{\mathtt{tr}} \leq \theta. \end{aligned} \tag{4.3}$$

The four strategies are adapted for all three scenarios.

Having noticed that, in each Frank-Wolfe update, W increases its rank by 1, I developed much more efficient ways to carry out the three Frank-Wolfe steps. Instead of calculating the gradient from scratch in each epoch, I store the gradient and some auxiliary information so as to update the gradient in a *recursive* way. This trick significantly reduces the computational cost per Frank-Wolfe epoch and makes it outperform other descent methods. In particular, I will study two concrete cases: multi-task least square and multinomial logistic regression. The recursive update trick applies to both cases by varying the auxiliary information stored. Moreover, as we shall see, it is possible to conduct the line search in a distributed manner for multi-task least square, where the auxiliary information also helps.

To store the auxiliary information, we have two options, which I call dense representation and low-rank representation. The former consists in storing everything

as dense matrices, whereas the latter tends to store them as low-rank matrices. These two representations are a trade-off regarding the dataset and the computation resources. The former is more suitable for datasets with average dimensions and clusters with few cores but large memory size, whereas the latter is more suitable for datasets with high dimensions and clusters with many cores but average memory size.

To illustrate the feasibility of the above ideas, I have conducted some experiments on Apache SPARK. I have tested the four aforementioned strategies for multi-task least square and multinomial logistic regression via the dense representation. The results show that, in low dimension, gradient centralizing and average mixture suffice, whereas, in high dimension, distributed power iteration with or without warm start outperform. Although these experiments are conducted on SPARK, the experiment results are expectable on other infrastructures. Meanwhile, surprisingly, one or two power iterations per epoch is already enough: increasing the number of power iterations has little or no marginal gain. This result appears to be much more optimist than the linearly or quadratically increasing iteration number stated in Theorem 3.6.3.

This chapter is organized as follows. Section 4.2 describes the four strategies for the linear subproblem. Section 4.3 presents how to update the gradient in a recursive manner to avoid the expensive gradient evaluation. Since this technique is problem dependent, I shall discuss multi-task least square and multinomial logistic regression in particular. Section 4.4 introduces and compares the dense and low-rank representation. Section 4.5 presents the implementation. Section 4.6 shows the experiments.

4.1.1 Related works

There has been some effort to make Frank-Wolfe algorithm distributed. Bellet et al. (2014) proposed a generic Frank-Wolfe framework for ℓ_1-norm minimization. Wang et al. (2016) proposed a parallel and distributed version of the Block-Coordinate Frank-Wolfe algorithm (Lacoste-Julien et al. 2012). Wai et al. (2017a) proposed a decentralized Frank-Wolfe framework for general problems, which is best when the parameter in each node is sparse. My work focuses on standard Frank-Wolfe for trace norm minimization and uses the BSP model in a star network. Moharrer and Ioannidis (2017) are particularly interested in the map-reduce type framework. They identified two properties that can make the map-reduce applicable. My work has two differences from theirs: on the one hand, without being implementation-picky, my framework can be implemented by either map-reduce or message passing; on the other hand, I am particularly interested in matrix parameter instead of vector parameter as in their work. Concerning the implementation, we share a lot in common: we both use SPARK and map-reduce and make use of SPARK's particular properties to accelerate the calculation. For instance, the core concept, called *common information* in their paper, also enjoys a recursive update.

Based on the work of Wai et al. (2017a), for the trace norm minimization problem, the same authors further incorporated a decentralized power method to solve the FW linear subproblem with the aim of reducing the communication cost (Wai

et al. 2017b). Although we all study trace norm minimization and use the power method, my work has six differences from theirs. First, their algorithm is based on the Gossip protocol and works on any network topology, whereas I assume the availability of a master-slaves star network and can hence have much less communication overhead. Second, they transform asymmetric matrices to symmetric ones via the method documented in Fazel et al. (2001) (see Page 20 of this dissertation as well), whereas I work on asymmetric matrices directly, which entails less memory and communication overhead. Third, they prove the convergence in probability for their Gossip-based algorithm, whereas I prove the convergence in both probability and in expectation for the master-slave scenario. Fourth, they are more interested in the matrix completion application with the use of MovieLens100k dataset[2], whereas I am more interested in the multi-task learning application with the use of the ImageNet dataset (Deng et al. 2009). Fifth, in their paper, they provide some primary experiment results with a single-thread MATLAB environment, whereas I demonstrate my algorithm on a physical cluster and with larger-scale datasets. Last but not least, they use more than 6 power iterations per FW iteration, whereas I use as few as 1 or 2 power iterations.

Multinomial logistic regression is an important problem in machine learning, and researchers have applied Frank-Wolfe to it (on a single machine). Dudik et al. (2012) proposed a lifted coordinate descent algorithm, which is equivalent to Frank-Wolfe. Their experiments on both synthetic dataset and a subset of ImageNet illustrate that, regarding the throughput, Frank-Wolfe is as good as accelerated proximal gradient (Toh and Yun 2010) and iterative rescaling and alternating minimization, if not better than. Liu and Tsang (2017) targeted the particular structure of the multi-class classification problem and proposed a stratified progressive sampling Frank-Wolfe. Their experiments illustrate that their methods outperform stochastic Frank-Wolfe and stochastic variance-reduction Frank-Wolfe regarding throughput on most real datasets.

4.1.2 Contributions

This chapter is composed of my contributions.

- I designed distributed strategies to solve the linear subproblem of trace norm minimization. These strategies work for the general matrix space, the symmetric matrix space, and the positive semidefinite cone.
- For two concrete models – multi-task least square and multinomial logistic regression, I designed dedicated recursive update schemes, which avoid the expensive full gradient evaluation at each epoch. This technique significantly accelerates the algorithms, and it works both on a distributed system and on a typical single-node system.
- In distributed systems, we need to store the parameters in each node. Regarding the manner to store the gradient, I designed two representations – dense and low-rank. The former is adept at the datasets with medium parameter

[2] http://grouplens.org/datasets/movielens/100k/

size or at the clusters with a low node/memory ratio, whereas the latter is adept at the datasets with large parameter size or at the cluster with a high node/memory ratio.

- To carry out my study in a systematic way, I developed a PySpark package, which is modularized and extensible. Through this package, users can implement their own objective functions by following the API and profit from my research fruit. In particular, I have implemented the objective functions for multi-task least square and multinomial logistic regression.

4.2 Distributing strategies

This section presents four distributing strategies to solve the linear subproblem. As we have seen in Section 3.5, for trace norm minimization, the subproblem is to calculate the eigenvector (resp. singular vectors) associated with the extreme eigenvalue (resp. singular value). The four strategies thus consist in calculating these eigenvectors (resp. singular vectors) in a distributed setting.

These strategies were initially proposed by Aurélien Bellet in an unofficial meeting in LIP6/UPMC and then concretized and made feasible by me afterward. All of these four strategies require (logically) a star network, whose center node is called *master* and other nodes *slaves*. Although the algorithms follow a BSP model, it is evident that it will not hurt the algorithms substantially by ignoring some untimely information sent from a few slow slaves.

4.2.1 Gradient centralizing

As the name suggests, *gradient centralizing* consists in centralizing all local gradients from the slaves to the master. Once the gradients are centralized and summed, the master can solve the subproblem with whatever classic algorithms. Algorithm 19 describes this strategy, whose description is generic and thus applicable to all three scenarios: (4.1), (4.2), and (4.3).

Centralizing the gradients is not equivalent to centralizing the samples. The gradient contains only the local information of the objective function (and hence these of the samples). We are therefore not able to recover the objective function by gradients at few points except in some extremely simple cases. Besides, the gradient has fixed size (i.e., it does not increase with the number of samples), and it is normally much smaller than the dataset. In general, for a gradient matrix of size $d \times m$, this strategy requires one communication round and dm communication volume per epoch.

4.2.2 Average mixture

Average mixture (Mcdonald et al. 2009, Zinkevich et al. 2010) is a common strategy in distributed machine learning. For a statistic of interest on the whole dataset,

we can estimate it by the (weighted) average of the statistics in question on the local datasets. If the statistic is a linear functional, this estimator will be equal to the global statistic; it is not, otherwise (see Jensen's inequality in Section 2.1.3). This strategy helps to reduce the variance but not the bias (Mcdonald et al. 2009, Zinkevich et al. 2010).

For our problem, we can calculate the local solution Q_i at each slave and then calculate their average at the master. However, the resulted solution matrix may no longer be of rank-one. Indeed, it, almost surely, has a rank equal to the number of the slaves. Recalling that, in Frank-Wolfe, each update is precisely a rank-one matrix, we lose consequently the most remarkable property of Frank-Wolfe algorithms. Moreover, this excess of ranks will result in more communication volume at the update step.

Alternatively, we choose not to average the solution but the eigenvectors / singular vectors instead. However, these vectors can be either positive or negative (*i.e.*, its opposite direction), which can essentially cancel each other. We need therefore to ensure that these vectors are more or less in the same direction, which can be done by, say, normalizing the coefficient with the largest absolute value to 1. Indeed, if all vectors are close to each other, their coefficients with the largest absolute value will be likely at the same place. This normalization will essentially align them to the same direction. Of course, after this average, the obtained eigenvector / singular vectors may no longer have unit length and thus require one more normalization.

Let us define the function $\text{apogee}(x) := x_{\arg\max_i\{|x_i|\}}$ as the entry of x with the largest absolute value.[3] In the general matrix scenario (Algorithm 20), the slaves divide *both* the (local) left and right singular vector by the apogee of the *right* singular vector (Line 2–3), and then the master uses the averaged and normalized version (Line 4–6). In the other two scenarios (Algorithm 21), the slaves divide the (local) eigenvector by its apogee (Line 2), and then the master uses the averaged and normalized version (Line 3–5). It is worth mentioning that the master needs c^i for the estimation of the sign of the eigenvalue, which allows it to take the sign opposite to the one of the eigenvalue. For positive semidefinite cone $\mathbb{S}^d_+$, the evaluation and communication of c^i is optional because they are always non-positive.[4] All three scenarios require one round of communication and $d+m$ communication volume per epoch, which is much cheaper than gradient centralizing.

Since this strategy is a variance reduction technique, the higher the sample-parameter ratio, the higher the precision of this strategy will be; and the more slaves are used, the less precise this strategy will be (because of more bias). As an extreme case, when the sample size on each slave is not sufficient for each slave to learn

[3] This is not equal to the ℓ_∞-norm, since it can be negative.
[4] If it is positive, just take 0 as in Algorithm 17.

Algorithm 19. Gradient centralizing

Slave i: $\nabla f_i(W)$ $\Uparrow$ ▷ slaves send all local gradients to the master
Master: Calculates Q $\Downarrow$ ▷ master sends the update to all slaves
Note: $\Uparrow$ *stands for convergecast.* $\Downarrow$ *stands for broadcast. See Page 32 for the definition.*

independently, their eigenvectors / singular vectors can be very far away from each other, which will break this strategy.

Algorithm 20. Average mixture for $\mathbb{R}^{d\times m}$

1: **Slave** i**:** solve singular vectors u^i, v^i of interest
2: **Slave** i**:** $c^i \leftarrow \text{apogee}(v^i)$
3: **Slave** i**:** $u^i,\ v^i \leftarrow u^i/c^i,\ v^i/c^i$ $\Uparrow$
4: **Master:** $u,\ v \leftarrow \sum_i u^i,\ \sum_i v^i$
5: **Master:** $u,\ v \leftarrow u/\left\|u\right\|_2,\ v/\left\|v\right\|_2$
6: **Master:** $Q \leftarrow -\theta uv^T$ $\Downarrow$

Algorithm 21. Average mixture for $\mathbb{S}^d$ and $\mathbb{S}^d_+$

1: **Slave** i**:** solve eigenvector v^i and eigenvalue c^i of interest
2: **Slave** i**:** $c^i,\ v^i \leftarrow c^i,\ v^i/\,\text{apogee}(v^i)$ $\Uparrow$
3: **Master:** $c,\ v \leftarrow \sum_i c^i,\ \sum_i v^i$
4: **Master:** $v \leftarrow v/\left\|v\right\|_2$
5: **Master:** $Q \leftarrow -\,\text{sgn}(c)\theta vv^T$ $\Downarrow$

4.2.3 Distributed power iteration

Gradient centralizing is accurate but communicationally expensive, whereas average mixture is communicationally cheap but may be inaccurate when the per-slave sample size is not enough. Distributed power iteration allows us to achieve the precision of gradient centralizing while still maintaining a moderate communication cost.

By analyzing the computation procedure of power iteration, we can find that it is composed of two alternating phases – matrix-vector multiplication and normalization. Since matrix multiplication is a linear operator, we can execute it *locally*, sum them and normalize it at the master, and then send it back to the slaves; this strategy is what I mean by *distributed power iteration* here.

Algorithm 22 describes the general matrix scenario, which is a distributed way to do the same thing as in Algorithm 15. The power iteration comes in pairs, with each pair composed of a left matrix multiplication (Line 4) and a right matrix multiplication (Line 7). With one calculating the left singular vector and the other calculating the right singular vector, each pair needs two rounds of communication. In the description, the algorithm starts with the *right* singular vector, uniformly sampled from the sphere (Line 2). This choice is optional: we can start with the left singular vector. In practice, it is preferred to start with a vector in lower dimension, which will lead to a better result.

Algorithm 23 describes the symmetric matrix scenario, which is a distributed way to do the same thing as in Algorithm 16. Here, we need to store the eigenvector of the last iteration v' (Line 5). By calculating the inner product of v and v', we are actually evaluating $v'^T Av'$ as in Algorithm 16.

Algorithm 24 describes the positive semidefinite matrix scenario, which is a distributed way to do the same thing as in Algorithm 17. We suppose here that each

Algorithm 22. Distributed power iteration for $\mathbb{R}^{d\times m}$

Slave i**:** $A_i \leftarrow \nabla f_i(W)$
Master: $v \sim \mathrm{Unif}(\mathcal{S}^m)$ $\Downarrow$.
for $k = 1, \ldots, K$ **do**
  Slave i**:** $u^i \leftarrow A_i v$ $\Uparrow$ ▷ left multiplication
  Master: $u \leftarrow \sum_i u^i$
  Master: $u \leftarrow \frac{u}{\|u\|_2}$ $\Downarrow$
  Slave i**:** $v^i \leftarrow A_i^T u$ $\Uparrow$ ▷ right multiplication
  Master: $v \leftarrow \sum_i v^i$
  Master: $v \leftarrow \frac{v}{\|v\|_2}$ $\Downarrow$
end for
Master & Slaves: $Q \leftarrow -\theta u v^T$

Algorithm 23. Distributed power iteration for $\mathbb{S}^d$

Slave i**:** $A_i \leftarrow \nabla f_i(W)$
Master: $v \sim \mathrm{Unif}(\mathcal{S}^d)$ $\Downarrow$.
for $k = 1, \ldots, 2K$ **do**
  Slave i**:** $v^i \leftarrow A_i v$ $\Uparrow$
  Master & Slaves: $v' \leftarrow v$ ▷ store the old eigenvector
  Master: $v \leftarrow \sum_i v^i$
  Master: $v \leftarrow \frac{v}{\|v\|_2}$ $\Downarrow$
end for
Master & Slaves: $Q \leftarrow -\operatorname{sgn}(v^T v')\theta v v^T$

Algorithm 24. Distributed power iteration for $\mathbb{S}^d_+$

Slave i**:** $A_i \leftarrow L_i I - \nabla f_i(W)$ ▷ make it positive semidefinite
Master: $v \sim \mathrm{Unif}(\mathcal{S}^d)$ $\Downarrow$
for $k = 1, \ldots, 2K$ **do**
  Slave i**:** $v^i \leftarrow A_i v$ $\Uparrow$
  Master: $v \leftarrow \sum_i v^i$
  Master: $c \leftarrow \|v\|_2$
  Master: $v \leftarrow \frac{v}{c}$ $\Downarrow$
end for
if $c \le \sum_i L_i$ **then** ▷ the smallest eigenvalue is positive
  Master: $v \leftarrow 0$ $\Downarrow$ ▷ return the origin
end if
Master & Slaves: $Q \leftarrow \theta v v^T$

local f_i is L_i-Lipschitz continuous with regard to the trace norm, so that f is $\sum_i L_i$-Lipschitz continuous. Equivalent, letting $L := \sum_i L_i$, we can also multiply LI at the master instead of doing it locally at each slave.

All of these algorithms in this subsection need $2K$ communication rounds and $K(d+m)$ communication volume.

4.2.4 Distributed power iteration with warm start

One may ask that, if we do several matrix-vector multiplications within each communication round, whether the algorithms will converge faster? Unfortunately, the modified version is not equivalent to the original one, and it may lead to a statistical bias. However, this idea is closely related to the idea of warm start, which consists in using the average-mixture eigenvector / singular vectors as the initial guess, instead of sampling from the unit sphere. It can be expected that the warm-start guess is closer to the true eigenvector / singular vectors, and thus it needs fewer power iterations afterward.

The detailed description is omitted; one just inserts the average mixture algorithms (removing the last line) into the correspondent distributed power iteration by replacing the second line. This algorithm needs $2K + 1$ (one for warm start) communication rounds and $m + K(d + m)$ communication volume. See Table 4.1 for a summary.

	Comm. volume	Comm. rounds
central	dm	1
avgmix	$d + m$	1
power	$K(d + m)$	$2K$
pow/ws	$m + K(d + m)$	$1 + 2K$

Table 4.1 – The cost to solve the subproblem of a matrix of dimension $d \times m$ in a distributed setting. $2K$ is the number of power iterations. `central` stands for gradient centralizing. `avgmix` stands for average mixture. `power` stands for distributed power iteration. `pow/ws` stands for distributed power iteration with warm start.

4.3 Recursive update

The expensive gradient evaluation motivates the parallel or distributed gradient computation. In addition to this approach, this section presents a recursive way to update the gradient instead of repeatedly evaluating it from scratch. Since each Frank-Wolfe epoch only adds a rank-one matrix to the solution, which implies that the new gradient will not be too far away from the old one, by updating the gradient recursively, we can significantly reduce the computation cost.

Since this update scheme varies according to the problem, I will discuss, for illustration, two specific problems: multi-task least square and multinomial logistic regression. It is applicable to both non-distributed system, as in this section, and the distributed system, as in the next section. Last but not least, not only does this trick apply to the gradient, but also the line search step size (if available) benefits from it, which will be illustrated in multi-task least square.

4.3.1 Multi-task Least square

The intuition of multi-task least square is described in Section 2.4.3, and we will proceed directly to the computation here.

Recall that the objective function is

$$f(W) = \tfrac{1}{2} \|XW - Y\|_{\mathrm{F}}^2 = \tfrac{1}{2} \sum_{i,j} (X_i w_j - y_{ij})^2, \tag{4.4}$$

where $X_{n\times d}$ and $Y_{n\times m}$ are data records; $W_{d\times m}$ is the regression coefficients, which is supposed to be a low-rank matrix (*i.e.*, $\operatorname{rank}(W) \ll d \wedge m$); X_i is the i-th line of X; w_j is the j-th column of W; and y_{ij} is the (i,j)-th entry of Y.

Its gradient can be expressed as

$$\nabla f(W) = X^T(XW - Y) = \sum_i (X_i^T X_i)W - \sum_i X_i^T Y_i, \tag{4.5}$$

where Y_i is the i-th line of Y. It bears both a concise matrix formulation and a summation formulation.

For multi-task least square, there exists a closed form for the line search step size. Let Q be the solution provided by the linear minimization oracle (`LMO`). The line search step size is the solution of

$$\arg\max_{\gamma\in[0,1]} \{\tilde{f}(\gamma) := f(W + \gamma(Q - W))\}. \tag{4.6}$$

The one-dimensional function $\tilde{f}$ is a composition of a convex function and an affine mapping and thus is itself a convex function (Boyd and Vandenberghe 2004, Section 2.2.2).

Taking derivative on γ, by the chain rule, we get

$$\begin{aligned}\tfrac{\mathrm{d}}{\mathrm{d}\gamma}\tilde{f}(\gamma) &= \left\langle X^T X(W + \gamma(Q - W)) - X^T Y, Q - W \right\rangle \\ &= \left\langle X^T XW - X^T Y, Q - W \right\rangle + \gamma \left\langle X^T X(Q - W), Q - W \right\rangle.\end{aligned}$$

Making it equal to zero, we get the line search step size

$$\gamma = \frac{\langle -\nabla f(W), Q - W\rangle}{\langle X^T X(Q - W), Q - W\rangle} = \frac{\left\langle \sum_i X_i^T Y_i - \sum_i (X_i^T X_i)W, Q - W \right\rangle}{\left\langle \sum_i X_i^T X_i (Q - W), Q - W \right\rangle}. \tag{4.7}$$

It is a generalization of (3.11) in Jaggi (2011); indeed, by letting $W = I$, we recover the formula described therein. Since the step size should be in $[0, 1]$, but the step size given above can exceed one (it cannot be negative because it is a descent direction), we will have to take $\gamma \wedge 1$.

The gradient is a *polynomial* of W, which gives us the opportunity to update it recursively.

In particular, the gradient at the next epoch is

$$X^T X((1 - \gamma)W + \gamma Q) - X^T Y,$$

Algorithm 25. Recursive update for multi-task least square

1: $\Phi,\ \Psi \leftarrow X^TX,\ X^TY$ ▷ constant matrices
2: $A \leftarrow 0$ ▷ $A := X^TXW$
3: $a,\ b \leftarrow 0,\ 0$ ▷ $a, b := \langle X^TXW, W\rangle, \langle X^TY, W\rangle$
4: **loop**
5: Send $A - \Psi$ to `LMO` ▷ the gradient
6: Get u, v from `LMO` ▷ $uv^T = Q$
7: $B \leftarrow (\Phi u)v^T$ ▷ $B := X^TXQ$
8: $c,\ d,\ e \leftarrow \langle B, Q\rangle,\ \langle \Psi, Q\rangle,\ \langle A, Q\rangle$
9: $\gamma \leftarrow \frac{e-a-d+b}{c-2e+a} \wedge 1$ ▷ line search step size
10: $A \leftarrow (1-\gamma)A + \gamma B$ ▷ update A
11: $a \leftarrow (1-\gamma)^2 a + 2\gamma e + \gamma^2 c$ ▷ update a
12: $b \leftarrow (1-\gamma)b + \gamma d$ ▷ update b
13: **end loop**

or equivalently

$$(1-\gamma)X^TXW + \gamma X^TXQ - X^TY.$$

If we have the value of X^TXW at the current epoch, then by simply calculating X^TXQ, we can obtain the value of X^TXW at the next epoch. Subtracting X^TY, which is constant across the epochs, we get the gradient of the next epoch. The computation of X^TXQ is much cheaper than X^TXW since Q is a rank-one matrix. Supposing $Q = uv^T$, we can calculate the matrix-vector multiplication of X^TX and u and then the outer product of X^TXu and v. This trick incurs $O(d^2 + dm)$ time complexity instead of $O(d^2m)$ as in the full gradient evaluation.

It is the same for the line search step size. To calculate the current step size, we have to evaluate $\langle X^TXW, W\rangle$, $\langle X^TY, W\rangle$, $\langle X^TXQ, Q\rangle$, $\langle X^TY, Q\rangle$, and $\langle X^TXW, Q\rangle$ (equal to $\langle X^TXQ, W\rangle$). While the 3rd, 4th and 5th must be calculated at each epoch, the 1st and 2nd can be stored and updated recursively. Supposing that we have already the five quantities above, the 1st and 2nd quantity of the next epoch can be updated by

$$\begin{aligned}&\left\langle X^TX((1-\gamma)W + \gamma Q), (1-\gamma)W + \gamma Q\right\rangle \\ =&(1-\gamma)^2\left\langle X^TXW, W\right\rangle + 2\gamma(1-\gamma)\left\langle X^TXW, Q\right\rangle + \gamma^2\left\langle X^TXQ, Q\right\rangle,\end{aligned}$$

and

$$\left\langle X^TY, (1-\gamma)W + \gamma Q\right\rangle = (1-\gamma)\left\langle X^TY, W\right\rangle + \gamma\left\langle X^TY, Q\right\rangle.$$

However, the benefit of this update is marginal, which does not reduce the computation complexity. One meaningful benefit may be that it avoids the materialization of Q by using its low-rank property. Whether we use this update or not, the complexity is always $O(dm)$.

The algorithm is summarized in Algorithm 25, which is presented as a separated thread communicating with the `LMO`. In other words, it receives the output of the `LMO` (Line 6) and sends the gradient to the `LMO` (Line 5). Despite its particular form, one can easily integrate it into any Frank-Wolfe variants.

4.3.2 Multinomial logistic regression

The intuition of multinomial logistic regression and its three forms of the objective function are presented in Section 2.4.4, and I will therefore proceed directly to the computation of the gradient in this subsection.

Let us take the objective function

$$-\ln\prod_{i=1}^{n}\Pr(C_i)=\sum_{i=1}^{n}\left(-X_i w_{C_i}+\ln\sum_{j=1}^{m}e^{X_i w_j}\right) \tag{4.8}$$

as the starting point. Ignoring the outermost summation, it is composed of two parts. Let us first consider the second part. By the chain rule, we get

$$\frac{\mathrm{d}}{\mathrm{d}W}\ln\sum_j e^{X_i w_j}=\frac{X_i^T e^{X_i W}}{\sum_j e^{X_i w_j}}, \tag{4.9}$$

where the exponential of a vector is defined component-wisely.

Define the *softmax* function from $\mathbb{R}^n$ to $\mathbb{R}_+^n$ as $\operatorname{softmax}(x):=\frac{\exp(x)}{\sum_i\exp(x_i)}$. Each component of its value is in $[0,1]$, and the sum is equal to 1. We call it softmax because it has a close relation with the max function. Indeed, when $\lambda\to+\infty$, $\operatorname{softmax}(\lambda x)$ tends to the max function (*i.e.*, with the component with the largest value equal to 1, and 0 for others); when $\lambda=0$, $\operatorname{softmax}(\lambda x)$ becomes a vector with all components equal. It is a smooth alternative to the max function. By denoting the line vector $S_i:=\operatorname{softmax}(X_iW)$, (4.9) can be expressed compactly as $X_i^T S_i$.

Now let us consider the first part of (4.8). Let C_i be the class number of the i-th sample, denote the line vector $Y_i=Y_i(j):=\mathbf{1}_{[j==C_i]}$. Then,

$$\frac{\mathrm{d}}{\mathrm{d}W}X_i w_{C_i}=X_i^T Y_i.$$

Combining the above two parts, we get

$$\nabla f(W)=\sum_i X_i^T(S_i-Y_i)=X^T(S-Y), \tag{4.10}$$

where $S:=[S_1;S_2;\ldots;S_n]$ and $Y:=[Y_1;Y_2;\ldots;Y_n]$ are line vector stacking (*i.e.*, matrices with line vectors S_i and Y_i respectively). Just like (4.5), for multinomial logistic regression, we also get both a concise matrix formulation and a summation formulation.

We may also want to get the line search step size. However, it does not have a closed form for multinomial logistic regression because γ appears both in and outside the exponential in the derivative, which makes it impossible to solve the equation.

Although the gradient is not a polynomial of W, we can still obtain a recursive update, for it is the composition of a general function and a polynomial. Algorithm 26 shows the update scheme. To update the XW, we can use the one-rank property of Q as in multi-task least square. However, to calculate X^TS, we are still faced with

Algorithm 26. Recursive update for multinomial logistic regression

1: $A \leftarrow 0$ ▷ $A := XW$
2: **loop**
3: $S \leftarrow \text{softmax}(A)$ ▷ softmax is defined line-wisely
4: Send $X^T(S - Y)$ to `LMO` ▷ gradient
5: Get u, v from `LMO` ▷ $uv^T = Q$
6: $A \leftarrow (1 - \gamma)A + \gamma(Xu)v^T$ ▷ update A
7: **end loop**

a matrix-matrix multiplication. In general, S cannot be a low-rank matrix, and the complexity of the update is consequently $O(ndm)$ unless X is a low-rank or sparse matrix. Without further conditions, the benefit of the recursive update is limited in multinomial logistic regression.

4.4 Dense vs. low-rank representation

The previous section describes how to do the recursive update on a single machine, and this section will discuss how to do it in a distributed system. According to the manner we store variables, it can be categorized into two solutions – dense representation and low-rank representation. The former is for datasets with medium-sized parameters, and the latter is for datasets with large-sized parameters.

Both the dense representation and the low-rank representation are problem specific. I will first discuss the multi-task least square and then multinomial logistic regression in each representation. Throughout this section, I suppose that there are N data samples distributed to M slaves, with each slave hosting $n := \frac{N}{M}$ samples.

4.4.1 Dense representation

For multi-task least square, the gradient can be expressed as a summation across the samples as in (4.5). With a little abuse of notation, we can generalize it by considering X_i and Y_i as matrices correspondent to the data of the i-th slave, and the equation (4.5) still holds. This viewpoint shift suggests a straightforward adaption of Algorithm 25, as described in Algorithm 27.

Algorithm 27 uses an important trick to calculate the line search step size (Line 9–10). The formula for the line search step size (4.7) is not linear with regard to the dataset, but its numerator and denominator *are*. By summing the numerators and denominators respectively via the master, we finish the task.

To avoid the heavy notation, I will drop the subscript i during the complexity analysis. The initialization of X^TX and X^TY needs $O(nd^2 + nmd)$ operations and $O(d^2 + md)$ space (Line 1). The evaluation of the gradient needs $O(md)$ operations (Line 5). During the `LMO` procedure, all strategies except gradient centralizing require some computation. Supposing that they all use power iteration either locally or distributedly, each pair of power iteration will take $O(md)$ time complexity. The

Algorithm 27. Dense representation for multi-task least square

1: **Slave** i: $\Phi_i,\ \Psi_i \leftarrow X_i^T X_i,\ X_i^T Y_i$ ▷ $O(nd^2 + nmd)$
2: **Slave** i: $A_i \leftarrow 0$
3: **Slave** i: $a_i,\ b_i \leftarrow 0,\ 0$
4: **loop**
5: **Slave** i: Send $A_i - \Psi_i$ to `LMO` ▷ $O(md)$
6: **Slave** i: Get u, v from `LMO`
7: **Slave** i: $B_i \leftarrow (\Phi_i u)v^T$ ▷ $O(d^2 + md)$
8: **Slave** i: $c_i,\ d_i,\ e_i \leftarrow \langle B_i, Q\rangle,\ \langle \Psi_i, Q\rangle,\ \langle A_i, Q\rangle$ ▷ $O(md)$
9: **Slave** i: $\alpha_i,\ \beta_i \leftarrow e_i - a_i - d_i + b_i,\ c_i - 2e_i + a_i$ ⇑
10: **Master:** $\gamma \leftarrow \frac{\sum_i \alpha_i}{\sum_i \beta_i} \wedge 1$ ⇓
11: **Slave** i: $A_i \leftarrow (1-\gamma)A_i + \gamma B_i$ ▷ $O(md)$
12: **Slave** i: $a_i \leftarrow (1-\gamma)^2 a_i + 2\gamma e_i + \gamma^2 c_i$
13: **Slave** i: $b_i \leftarrow (1-\gamma)b_i + \gamma d_i$
14: **end loop**

Algorithm 28. Dense representation for multinomial logistic regression

1: **Slave** i: $A_i \leftarrow 0$
2: **Slave** i: $S_i \leftarrow \frac{1}{m}E$ ▷ E is a matrix whose all entries equal to 1
3: **Slave** i: Send $X_i^T S_i - X_i^T Y_i$ to `LMO` ▷ $O(nd + md)$
4: **loop**
5: **Slave** i: Get u, v from `LMO`
6: **Slave** i: $A_i \leftarrow (1-\gamma)A_i + \gamma(X_i u)v^T$ ▷ $O(nd + nm)$
7: **Slave** i: $S_i \leftarrow \mathrm{softmax}(A_i)$ ▷ $O(nm)$
8: **Slave** i: Send $X_i^T(S_i - Y_i)$ to `LMO` ▷ $O(nmd)$
9: **end loop**

update of $X^T XW$ (including the computation of $X^T XQ$) needs $O(d^2 + md)$ operations (Line 7). The line search (Line 9–10) takes one communication round, and the communication volume is $O(1)$. Regarding the computation cost, it will take $O(md)$ operation to calculate those inner products (Line 8). Here again, the recursive update of those inner products are optional, which may not reduce the computation time and only reduces the space by avoiding materializing Q. It does not reduce the space complexity though, which is lower bounded by $O(d^2 + md)$.

For multinomial logistic regression, Algorithm 28 is a straightforward adaption of Algorithm 26 with one exception: I put the gradient evaluation of the first epoch out of the loop, as an initialization (Line 2–3). This change is because it is much cheaper to calculate at the first epoch. Indeed, at the first epoch, $S = \frac{1}{m}E$, where E is a matrix whose all entries equal to 1. Therefore, to calculate $X^T S$, we only need to multiply X^T to the first column of S ($O(nd)$ operations) and then replicate it $m-1$ times ($O(md)$ operations). To calculate $X^T Y$, since Y is a sparse matrix with only n non-zero entry, we need $O(nd)$ operations. To sum up, the whole initialization needs $O(nd + md)$ operations. On the contrary, the gradient evaluation in the following epochs needs ($O(nmd)$) operations each, which is much more expensive and also more expensive than the quadratic complexity of multi-task least square. Regarding the space complexity, it needs $O(nd + md)$ to store XW and the gradient.

4.4.2 Low-rank representation

The major disadvantage of the dense representation is that it has to generate the gradient matrix for *each* slave, which incurs a total space complexity of $O(Mmd)$. If either m and d are huge (*e.g.*, $\geq 10^6$) or there are many slaves (*e.g.*, ≥ 1000), this can be prohibitively expensive. That said, can we use Frank-Wolfe if the gradient itself is not generated?

If it were a general gradient descent method, it would be impossible without generating the gradient. However, for Frank-Wolfe, the answer is *yes*: we *can* do that. The intuition is that what we want is not the gradient itself, but its extreme eigenvector / singular vectors. Since we shall use power iteration to get these vectors, we can use any representation of the gradient allowing us to do power iteration.

The gradient $X^T(XW-Y)$ of multi-task least square and $X^T(S-Y)$ of multinomial logistic regression both have the form of a $d \times n$ matrix X^T being multiplied by another $n \times m$ matrix. If $n \ll d \wedge m$, then the gradient is a low-rank matrix, and the low-rank decomposition is explicitly given with no cost. As the name suggests, the low-rank representation keeps this decomposition as the gradient, instead of evaluating it and carrying it as a dense matrix. For the power iteration, we can multiply the guessed vector with these two low-rank matrices of the low-rank representation successively, which needs $O(nd+nm)$ operation per iteration.

To use this representation, the other parts of the algorithm should be designed accordingly. In more detail, multi-task least square is described in Algorithm 29. The main difference compared with Algorithm 27 is that here we store XW instead of X^TXW (Line 2), which requires $O(nm)$ space complexity and is coherent to the low-rank representation. Although the high dimension of W is troublesome at the first place, XW successfully assimilates W into the data and is then distributed across the nodes. For the line search update, (4.7) can be reformulated as

$$\gamma = \frac{\langle Y - XW, XQ - XW \rangle}{\langle XQ - XW, XQ - XW \rangle}. \tag{4.11}$$

The recursive update scheme is analogous to Algorithm 27.

Algorithm 30 describes the low-rank representation for the multinomial logistic regression, which is not much different from Algorithm 28. Since we cease to calculate X^TS, the computation complexity becomes quadratic instead of cubic, which is an accidental benefit brought by the low-rank representation.

Apparently, the low-rank representation can also be used for parallel systems, and it provides not only the parallel preparation of gradients but also parallel power iterations.

4.4.3 Comparison

Table 4.2 is a summary of all complexities. For multi-task least square, the dense representation will use $O(Md(d+m))$ space and $O(d(d+m))$ time, ignoring the expensive initialization; and the low-rank representation will use $O(N(d+m))$ space

Algorithm 29. Low-rank representation for multi-task least square

Slave i**:** $X_i,\ Y_i$
Slave i**:** $A_i \leftarrow 0$ ▷ $A_i := X_iW$
Slave i**:** $a_i,\ b_i \leftarrow 0,\ 0$ ▷ $a_i, b_i := \langle X_iW, X_iW\rangle, \langle X_iW, Y_i\rangle$
loop
  Slave i**:** Send $(X_i, A_i - Y_i)$ to `LMO` ▷ $O(nd)$
  Slave i**:** Get u, v from `LMO`
  Slave i**:** $B_i \leftarrow (X_iu)v^T$ ▷ $B_i := X_iQ$, $O(nd + nm)$
  Slave i**:** $c_i,\ d_i,\ e_i \leftarrow \langle B_i, B_i\rangle, \langle Y_i, B_i\rangle, \langle A_i, B_i\rangle$ ▷ $O(nm)$
  Slave i**:** $\alpha_i,\ \beta_i \leftarrow e_i - a_i - d_i + b_i,\ c_i - 2e_i + a_i$ ⇑
  Master: $\gamma \leftarrow \frac{\sum_i \alpha_i}{\sum_i \beta_i} \wedge 1$ ⇓
  Slave i**:** $A_i \leftarrow (1-\gamma)A_i + \gamma B_i$ ▷ $O(nm)$
  Slave i**:** $a_i \leftarrow (1-\gamma)^2 a_i + 2\gamma e_i + \gamma^2 c_i$
  Slave i**:** $b_i \leftarrow (1-\gamma)b_i + \gamma d_i$
end loop

Algorithm 30. Low-rank representation for multinomial logistic regression

Slave i**:** $X_i,\ Y_i$
Slave i**:** $A_i \leftarrow 0$ ▷ X_iW
loop
  Slave i**:** $S_i \leftarrow \text{softmax}(A_i)$ ▷ $O(nm)$
  Slave i**:** Send $(X_i, S_i - Y_i)$ to `LMO` ▷ $O(m)$
  Slave i**:** Get u, v from `LMO`
  Slave i**:** $A_i \leftarrow (1-\gamma)A_i + \gamma(X_iu)v^T$ ▷ $O(nd + nm)$
end loop

and $O(n(d+m))$ time, which suggests that, when the sample size of each slave n is smaller than the feature dimension d, the low-rank representation is cheaper in both space and time complexity. Therefore, for CPU clusters, whose cores are limited and memory is abundant, it is preferable to use the dense representation, especially for low dimensional data; for GPU clusters, whose cores are abundant and memory is limited, it is preferable to use the low-rank representation, especially for high dimensional data.

For multinomial logistic regression, the low-rank representation is almost always better than the dense representation except when the data are in low dimension and when lots of power iterations are required.

4.5 Implementation

The four distributed strategies described in Section 4.2 are generic: they do not depend on any model or any infrastructure. The independence of the model implies that they apply to either multitask least square, or multinomial logistic regression, or any other loss functions. The independence of the infrastructure implies that they apply to any physical computation units and communication protocol.

	Multi-task least square		Multinomial logistic regression	
	Dense	Low-rank	Dense	Low-rank
Memory	$O(d^2 + md)$	$O(nd + nm)$	$O(nd + md)$	$O(nd + nm)$
Initialization	$O(nd^2 + nmd)$	0	$O(nd + md)$	0
Update	$O(d^2 + md)$	$O(nd + nm)$	$O(nmd)$	$O(nd + nm)$
Power	$O(md)$	$O(nd + nm)$	$O(md)$	$O(nd + nm)$
Line search	$O(md)$	$O(nm)$	—	—

Table 4.2 – The space and time complexities of various operations for different representations and problems. n is the sample size on a single slave, d and m are the row and column size of W. "Memory" represents the space complexity to store the variables. "Initialization" is the time complexity of the initialization of these variables. Since low-rank representation uses the raw data at the first epoch, its complexity is zero. "Update" represents the time complexity of the update of the variables and the preparation of the gradient at each epoch. "Power" represents the time complexity of each power iteration. "Line search" represents the extra time complexity of the line search at each epoch if applicable.

As mentioned in the previous section, the recursive update avoids repeated calculation of the gradient from scratch and hence accelerates the computation. Despite that the specific scheme of recursive update depends on the model (viz. loss function) in question, it is still oblivious to the underlying infrastructure: it can be implemented for any physical computation units and communication protocol.

These different layers of abstraction give us the maximal freedom and make it widely applicable. In this section, I will briefly discuss several infrastructures and then focus on a specific one – clusters of commodity machines deployed with Apache SPARK.

4.5.1 Infrastructure

The word *infrastructure* in this section has a strict meaning, which indicates the physical computation units involved and the communication protocol among them. As mentioned in Section 2.5, the hypothesis of distributed machine learning is the unavailability of a shared memory: each unit has its own local memory. These units can be several commodity machines connected in a local area network (LAN), or thousands of personal computers connected into the Internet, or several CPUs with unshared caches on the same motherboard, or several independent video cards on the same motherboard. In my thesis, I am particularly interested in commodity machines in LAN. Other infrastructures can also be interesting and have their own application situations though.

For a cluster of commodity machines in LAN, they can communicate with each other by *message passing*. Indeed, a Message Passing Interface (MPI) exists for distributed computing application, and the earliest distributed machine learning researches also benefit a lot from MPI. Nevertheless, with the prevalence of distributed machine learning, researchers find some common patterns across various applications, which gives birth to *frameworks*. For example, Google's *map-reduce* framework only

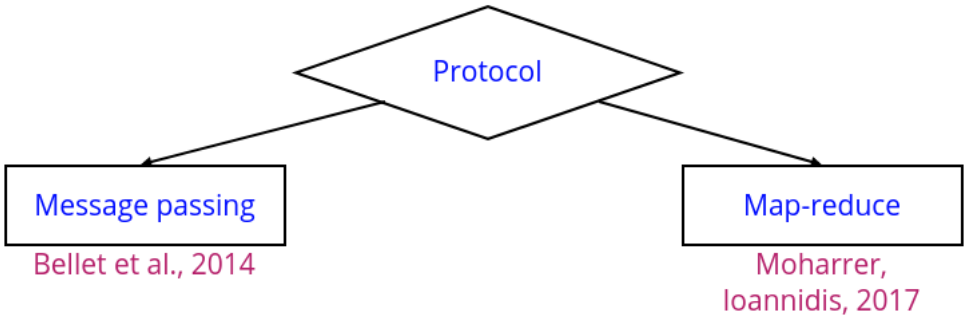


Figure 4.1 – Different protocols possible.

requires the researcher to implement the *map* function and the *reduce* function, which can already express a wide range of machine learning algorithms. In addition, it relieves the researchers from annoying tasks such as fault tolerance. I refer to Section 2.5.4 for more examples of frameworks.

It is possible to implement my distributed Frank-Wolfe framework with either MPI or frameworks. Concerning general Frank-Wolfe research, different researchers have made different choices. Bellet et al. (2014) use MPI for Lasso type of problems, whereas I choose a map-reduce framework, precisely Apache SPARK, for the implementation. A concurrent research project of Moharrer and Ioannidis (2017) also expresses their algorithms via map-reduce and implements it on Apache SPARK. They give a condition on the expressibility of algorithms via map-reduce and, with a device called *common information*, their algorithms can be expressed with a relatively simple and easy-to-update data structure. My recursive update scheme described in the previous section has the same flavor. Nevertheless, instead of Lasso type of problems as in Moharrer and Ioannidis (2017) as well as in Bellet et al. (2014), I focus on trace norm minimization.

SPARK assumes a master/slaves-like star network and can imitate the BSP model, which is coherent to the nature of my distributed Frank-Wolfe algorithms. Although SPARK provides a rich API containing many functions (*e.g.*, `join`), I implement my algorithms on SPARK with only the `map` function and the `reduce` function, which are already enough to express my algorithms in their integrity. With the popularity and accessibility of SPARK, my implementation can benefit all SPARK users over the world, and enthusiasts of Frank-Wolfe can conveniently test my implementation.

Rigorously, SPARK is more complicated than a trivial master/slaves clusters. In SPARK, several entities could be regarded as slaves: machines, cores and partitions are all potential slave candidates. At the high level, each machine can be seen as a slave, which employs its cores as *sub-slaves* to do the work in parallel; or, at the low level, each core can be seen as a slave, which is directly responsible to the master for completing the tasks due; or, at the data structure level, each partition can be seen as a (passive) slave, which forms a queue, waiting for the cores to serve them.

In addition to the nuance of master/slaves interpretation, SPARK may also transfer data across slaves, which cannot happen in a genuine master/slaves infrastruc-

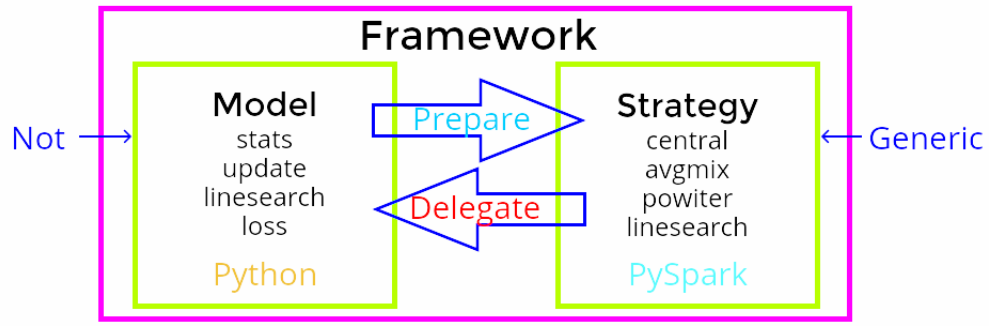


Figure 4.2 – Design pattern of the framework

ture. To make the entire system *resilient*, SPARK automatically transfers the tasks of failed or extremely slow machines to the remaining working machines. My algorithms are simple and straightforward, but specific cares should be taken for the underlying infrastructure during the implementation.

4.5.2 Package

SPARK is a *framework*, which can express many algorithms with a single support. My ambition to build a distributed Frank-Wolfe framework (as a package based on SPARK) implies that it should support many models (viz. loss functions) without significant modification of itself. This brings challenges. As mentioned earlier, recursive updates can accelerate the computation. Nevertheless, recursive updates are model specific: each model needs its own implementation, which contradicts the philosophy of frameworks, which must be stable and generic. To solve this problem, I resort to *design patterns*.

The idea is to break the framework/package into two parts: one for the generic distributed Frank-Wolfe framework, which is model agnostic; the other is a library of various models, which is not generic. These two parts link with each other via an interface.

The model interface has `stats`, `update`, `linesearch`, and `loss` as virtual methods. Each concrete subclass of the abstract model class implements the `stats` method, which initializes some variables (used for recursive updates) including the gradient, and the `update` method, which updates these variables after each epoch. It can also optionally implement the `linesearch` method, which provides line search information, and the `loss` method, which provides the objective function value. The model is purely implemented in the Python language, and SPARK knowledge is thus not required to implement new models. I have currently two model modules available: multi-task least square and multinomial logistic regression.

The generic Frank-Wolfe framework manipulates the interface via the model module's virtual methods. The framework itself implements all four aforementioned strategies as well as the method to determine the step size. If the model in question implements the `linesearch` method, it can delegate the job to the model. This part

is written in PySpark (SPARK's Python API), and extensive knowledge of SPARK is required to extend it.

Currently, my package[5] contains the dense representation for both multi-task least square and multinomial logistic regression. The implementation is roughly the same as described in the previous section with one exception: for multinomial logistic regression, my code has cubic complexity for the initialization instead of quadratic because I have not yet done any optimization in using the special properties described.

4.6 Experiments

In this section, I present the results of the experiments. My implementation is made in the way that the algorithms are as fast as possible; indeed, it *is* fast, but the main goal of these experiments is not to show that I have made the fastest system in the world. Instead, they serve as a proof of concept, which verifies the feasibility of the algorithms and shows insight on the possible behavior when they are implemented in other systems.

This section is organized as follows. I first explain the experiment environment and then present the experiment results about multi-task least square and multinomial logistic regression, where the latter includes the ImageNet dataset. Lastly, I measure the time needed for various computation steps, when using different numbers of cores.

4.6.1 General settings

I have run my implementation on a small cluster with 5 identical machines, with SPARK deployed in the standalone mode. One machine is used for the driver, and the other four for the executors. Each machine uses 2 Intel Xeon E5645 CPUs; each CPU has 6 physical cores; each core has 2 threads. Therefore, I have 96 logical cores available in total with each core being able to serve as an independent computation unit. The main frequency of the CPU is 2.40GHz. Each machine has 64G RAM with DDR3 at 1600MHz. My SPARK deployment is configured to use 60G, and the executors thus use 240G in total. The network card has a speed of 1Gb/s. The SPARK version is 1.6. Python version is 3.5.1. The BLAS version does not enable multithreading. The SPARK cluster is configured to use all 96 logical cores unless otherwise stated.

I tested all four distributing strategies. The label `central` stands for the gradient centralizing. The label `avgmix` stands for average mixture. The label `power#` stands for power iteration, where the placeholder `#` stands for the iteration number of power method, with each power iteration taking 2 communication rounds. In my experiments, I tested only `power1` and `power2` because I found that this small number of power iterations is already sufficient. The label `powlog` stands for power iteration

[5] https://github.com/WenjieZ/distributed-frank-wolfe

using a logarithmically increasing communication rounds with regard to the epoch. The label `pow#ws` stands for power iteration with warm start, where # has the same meaning as above.

4.6.2 Multi-task least square

The first experiment is about multi-task least square. I tested it on two synthetic datasets[6]: one small dataset containing 10^5 samples, 300 features, and 300 tasks; and one large dataset containing 10^5 samples, 1000 features, and 1000 tasks. In both datasets, the ground truth W has a rank of 10 and a trace norms equal to 1. It is generated by multiplying two arbitrary orthogonal matrices on a sparse diagonal matrix. X is generated randomly, with each coefficient following a Gaussian distribution. Y is obtained accordingly without further applying noise. The algorithm uses for its hyperparameter θ the same value as the ground truth (viz. 1) and uses the line search step size.

The experiment results are shown in Figure 4.3 and Figure 4.4. The first quick observation is that the program is fast – each epoch only takes seconds. The second is that the algorithm converges well. Then, let us have a detailed analysis of the performance.

Figure 4.3 shows the result for the low-dimensional dataset. Regarding the number of epochs, the strategy `central` is the most efficient as expected because it solves precisely the linear subproblem. The strategy `pow1ws` is almost equally good; we can hardly distinguish any difference between these two curves. The strategy `power2` has one more round of communication than `pow1ws` but is slightly worse. The strategy `power1` is worse than `power2` as expected because the more power iterations are run, the more accurately the subproblem will be solved. We can also notice that `powlog` first follows `power1` and then catches up with `power2`. The strategy `avgmix` has good performance at the beginning but loses to other methods (except `power1`) later. Regarding the runtime, despite the high communication volume, `central` is still the fastest. This can be explained by the fact that `central` needs the least rounds of communication and that this advantage outweighs the disadvantage of communication volume for low-dimensional data.

Figure 4.4 shows the result for the high-dimensional dataset. Regarding the number of epochs, all methods except `avgmix` behave similarly as in the previous case. This time, `avgmix` preforms significantly worse than other strategies because, with the increase of the dimension, the local gradient becomes a less accurate estimator of the global gradient. Regarding the runtime, `central` takes nearly twice the time of other strategies. The burden of communication volume finally outweighs the advantage of few communication rounds. Except for `avgmix` and `central`, the other strategies have similar performance vs. runtime behavior.

[6]There is no real datasets in this experiment because I have yet found any datasets containing thousands of tasks. The lack of these kinds of datasets makes an Apache SPARK implementation less appealing, but the experiment conducted on the synthetic data still sheds light upon the possible behavior of these distributing strategies when they are implemented for other types of clusters.

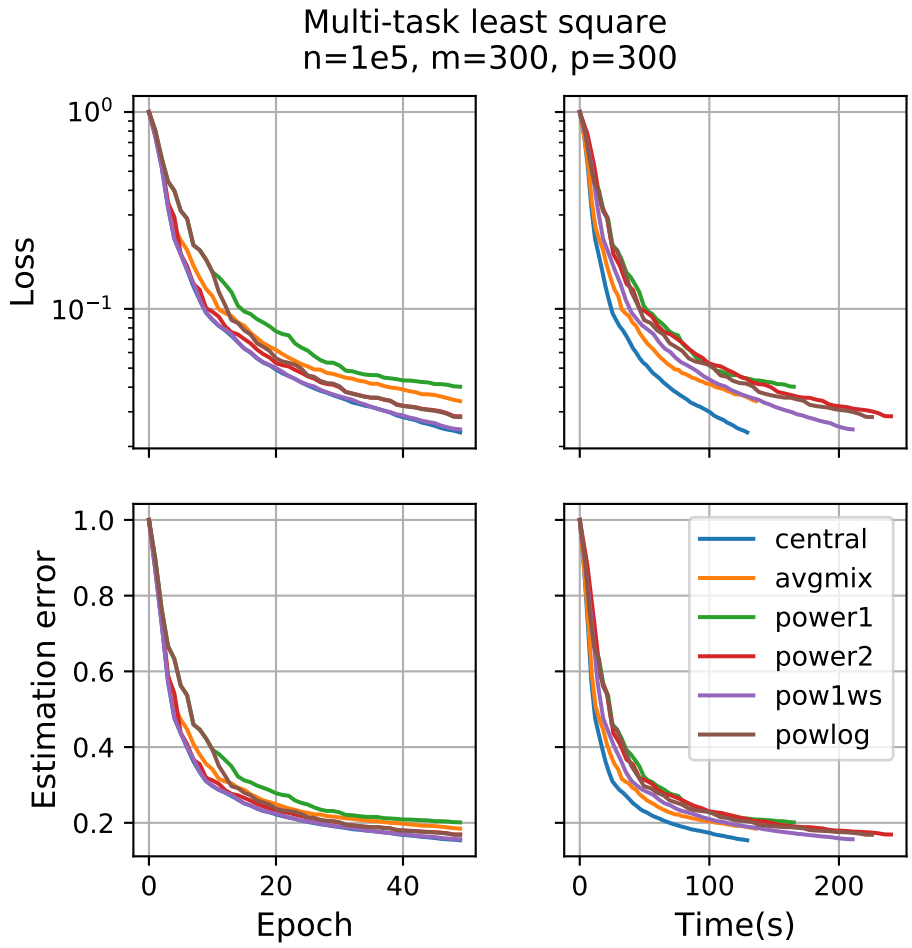


Figure 4.3 – The dataset contains 100,000 samples, 300 features and 300 tasks. Estimation error indicates the normalized Euclidean distance between the estimated parameter matrix W and the ground truth.

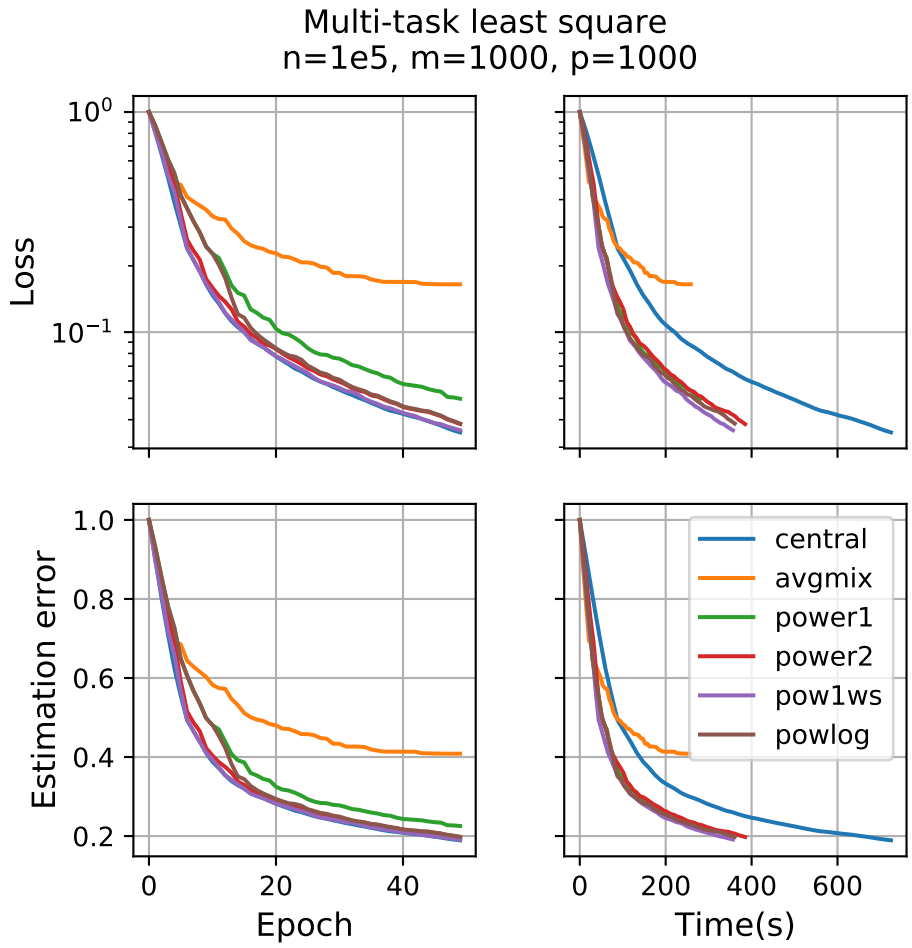


Figure 4.4 – The dataset contains 100,000 samples, 1000 features and 1000 tasks. Estimation error indicates the normalized Euclidean distance between the estimated parameter matrix W and the ground truth.

From these two datasets, we observe a regime shift about the relative behavior of `central`. When the dataset is low-dimensional, `central` is optimal; when the dataset is high-dimensional, `central` is not. For square parameter matrices W, the dimension threshold on Apache SPARK is somewhere between 300 and 1000. This threshold is system specific: for other systems penalizing more the communication volume and less the communication rounds, this threshold can be earlier.

4.6.3 Multinomial logistic regression

The second experiment is about multinomial logistic regression. I tested it on a synthetic dataset and a real dataset – ImageNet. The former is described in this subsection, and the latter will be in the next. The synthetic data, both the training set and the test set, have 10^5 samples, 1000 features, and 1000 classes. The generation of W and X is the same as multi-task least square, and the class is chosen as the one yielding the highest score. I tested the algorithms for various choices of θ, and I used fixed step size 0.01 for these experiments.

The experiment results are shown in Figure 4.5, 4.6, and 4.7. Although, the general behaviors of these results are similar to the ones in multi-task least square regression, there are some remarkable properties to point out here. Firstly, we hardly notice any difference between the training set and the test set. With a close look, the results on the test set are tinily worse than the ones on the training set. This difference is logical and universal in all machine learning problems. The tiny difference here and the nearly identical shapes (i.e., increasing and decreasing simultaneously) imply that Frank-Wolfe algorithms are not likely to overfit the training set.

Secondly, the behavior of the objective function and the one of the misclassification rate may not always be the same. In the experiment of $\theta = 10$, the strategy `central`'s objective function value is lower than `power1`, but its misclassification rate is higher. This implies that, although maximum likelihood estimator is efficient (i.e., achieves the Cramér-Rao lower bound), a higher likelihood (i.e., lower objective function) does not necessarily imply a better classification when the domain is constrained.

Besides, `powlog` works especially well regarding the runtime. At the early stage, when there is no performance difference between `power1` and `power2`, `powlog` follows `power1`. At the later stage, when `power1` is not accurate enough to solve the linear subproblem, `powlog` follows `power2`. It ends with the same error as `power2`, but with slightly less time.

4.6.4 ImageNet

The ImageNet dataset comes from the ILSVRC2012 challenge (Russakovsky et al. 2015), which has 1,281,167 pictures and 1000 classes. Since the dataset is composed of only photos without features, I used the learned features extracted from ResNet50 (He et al. 2016). The implementation of ResNet50 used in the experiment is provided by Keras (Chollet 2015), which contains 2048 features. My experiment uses 24 cores,

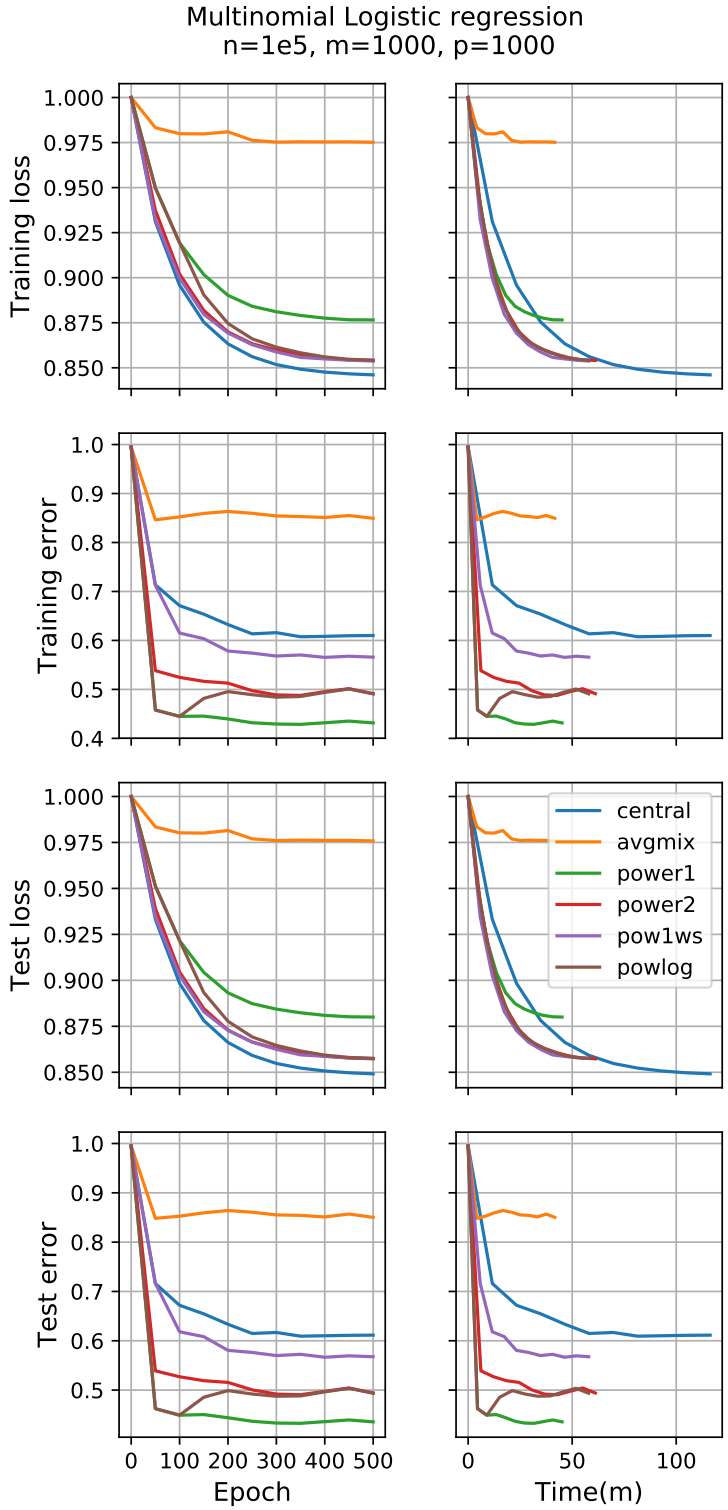


Figure 4.5 – Multinomial logistic regression: $\theta = 10$. The error stands for the Top-5 misclassification rate.

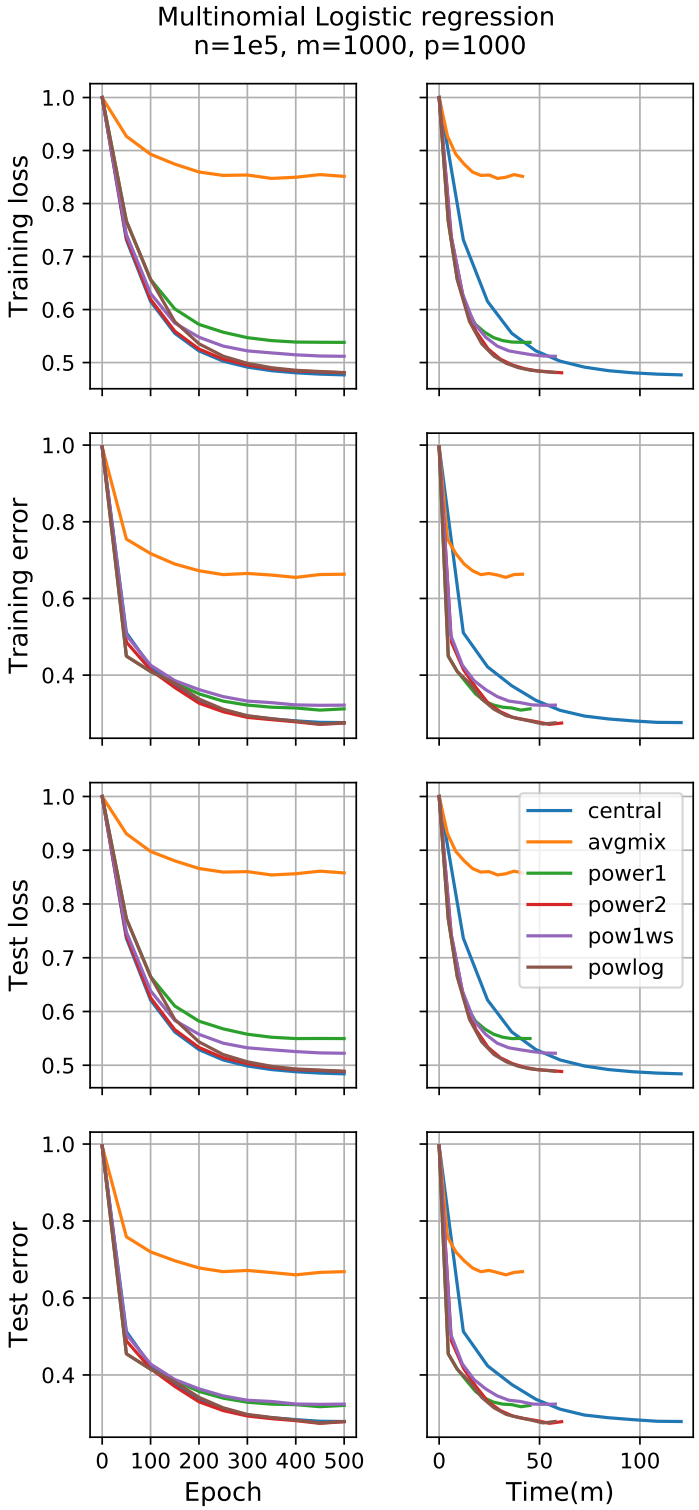


Figure 4.6 – Multinomial logistic regression: $\theta = 50$. The error stands for the Top-5 misclassification rate.

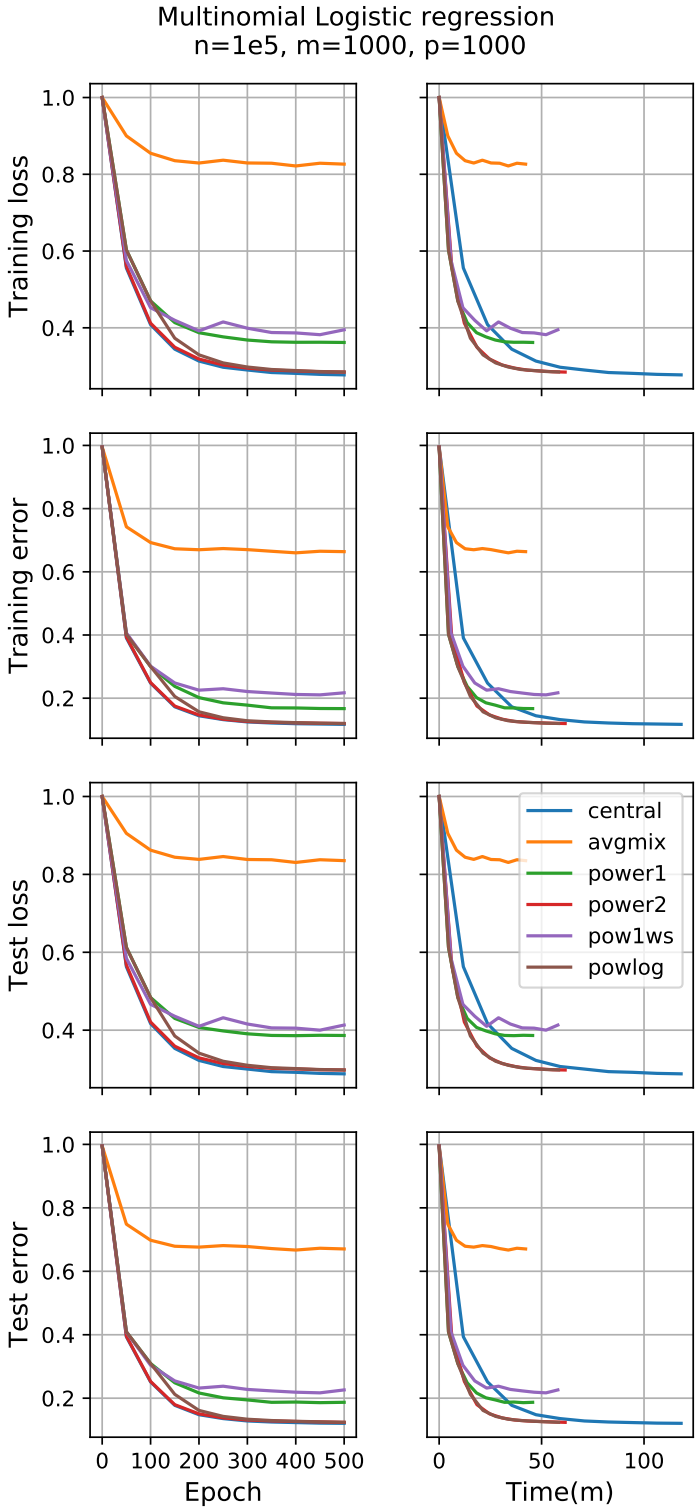


Figure 4.7 – Multinomial logistic regression: $\theta = 100$. The error stands for the Top-5 misclassification rate.

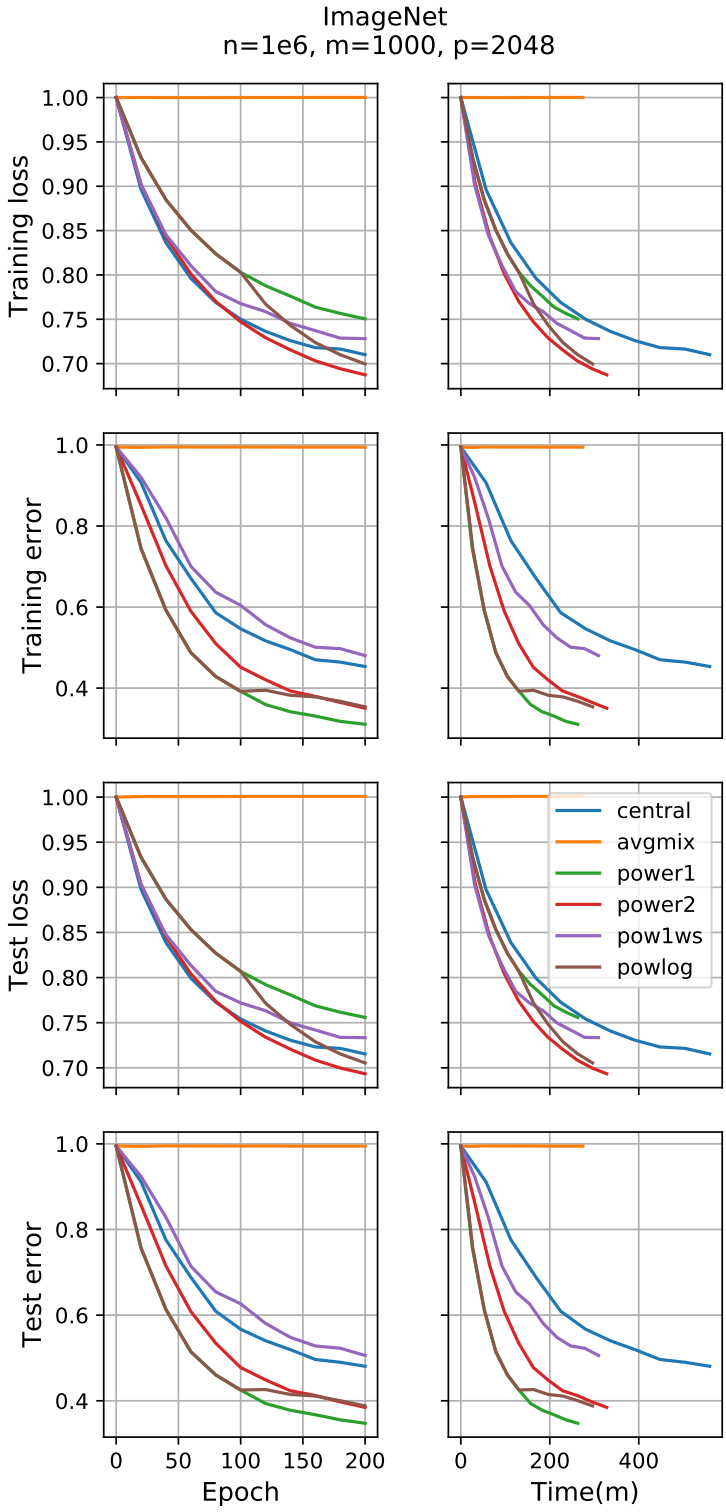


Figure 4.8 – Multinomial logistic regression on ImageNet. It has 1,281,167 samples, 2048 features and 1000 classes. The feature is learned by ResNet. The error stands for the Top-5 misclassification rate.

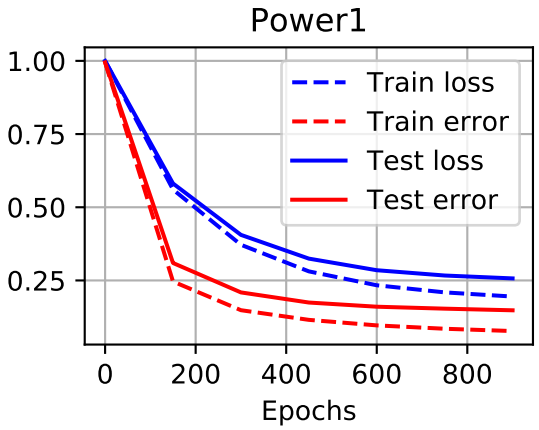


Figure 4.9 – `power1`'s performance on ImageNet. The blue dashed curve is the normalized objective function on the training set. The red dashed curve is the Top-5 misclassification rate on the training set. The blue solid curve is the normalized objective function on the test set. The red solid curve is the top-5 misclassification rate on the test set. `Test error` reaches nearly 0.13 after 900 epochs.

and the hyperparameter θ is chosen arbitrarily to be 30 without much caring about the performance.

Generally, Frank-Wolfe works well. Within only 200 epochs (several hours), the misclassification rate has dropped to 0.3 for certain strategies (Figure 4.8). Besides, we observe the same opposite behaviors between the objective function value and the misclassification rate as previous. Moreover, the strategy `power1`, with as few as one power iteration, outperforms all other strategies. These results illustrate how powerful Frank-Wolfe can be in solving real life problems.

Careful readers may discover that, in the loss vs. epoch subplot, `power2` outperforms `central`. They may think that this contradicts the fact that `central` is the most accurate solution of the linear subproblem. Moreover, this abnormality does not appear in the above experiments, which strengthens their doubt. However, this phenomenon can happen, and it is normal. The main reason is that, unlike the multi-task least square, multinomial logistic regression cannot do the line search. In consequence, the optimal step size for one strategy may not be the optimal one for another. We might have consequently chosen a step size that is suboptimal for `central`.

Figure 4.9 shows the experiment where `power1` is run for more than 800 epochs, with the hyperparameter $\theta = 400$. The Top-5 misclassification rate on the test set has almost reached 0.13, which is already very close to the optimal value because the pretrained ResNet50 provided by Keras, itself, has a rate of 0.12.

4.6.5 Impact of number of cores and partitions

My distributing strategies work in a master/slaves network. It is hence natural to investigate the impact of the number of slaves on the performance. Although this idea seems straightforward, special attention should be payed to the nuance of actual systems. As mentioned in Section 4.5.1, for Apache SPARK, there are several concepts (viz. machine, core, partition) that can be mapped to the concept of slave. There are thus three configurations that we can investigate. In my dissertation, I only investigate two of them: cores and partitions.[7]

A principled way in scientific experimentation is to vary a single variable when controlling all others. Nonetheless, this method cannot be applied as is here, for it is inappropriate to use more cores than partitions. Instead, I either vary the number of cores in controlling the total number of partitions or vary the number of cores in controlling the number of partitions *per core*.

The experiment is conducted on the multi-task least square high-dimensional dataset and a subset of the ImageNet dataset with 10^5 samples. The runtime is measured for various *unit* operations involved in any kinds of computation or communication. There are four types of behaviors about the runtime when increasing the number of cores. They are I) decreasing, II) increasing, III) firstly decreasing and then increasing, and IV) no much difference.

Table 4.3 presents the runtime for multi-task least square when using one partition per core. The type I behavior occurs for the initialization and the type II behavior occurs for all other operations. This result is coherent with the complexity analysis in Table 4.2, which says that the initialization complexity scales with the number of samples, while other operations do not. By increasing the number of cores, each core processes fewer samples, and hence the initialization is faster. Meanwhile, for other operations, each core takes up the same size of memory and has the same workload no matter how many cores used; this disadvantage makes the parallelism useless.

Table 4.4 presents the runtime for multi-task least square when using 96 partitions fixed. When using fewer cores than partitions, each core has to process more than one partition. Naturally, this circumstance favors the use of more cores, and the experiment result conveys the same message. Then, let us temporarily ignore the

[7]The number of cores can be configured by SPARK's `-total-executor-cores` property. The number of partitions can be configured within the program.

Table 4.3 – Multi-task least square: partition = core.

	96	48	32	24	16	8
init	42	66	143	165	316	996
line search	1.0	0.8	0.6	0.5	0.4	0.3
update	2.1	1.1	1.0	0.9	0.9	0.7
warm start	1.1	0.9	0.6	0.5	0.4	0.2
dist. power	1.9	1.2	0.9	0.7	0.6	0.4
centralize	10.0	5.0	3.0	2.8	2.0	1.0

Table 4.4 – Multi-task least square: 96 partitions fixed.

	96	48	32	24	16	8
init	42	31	31	34	36	49
line search	1.0	1.3	2.0	2.0	2.0	2.3
update	2.1	2.5	3.0	3.3	4.0	6.0
warm start	1.1	2.0	2.0	2.2	2.0	2.0
dist. power	1.9	2.0	2.1	3.2	3.5	4.0
centralize	10.0	9.0	8.8	9.2	9.0	9.0

Table 4.5 – Multinomial logistic regression: partition = core

	96	48	36	24	12	8
init	27	39	53	86	258	511
update	5.0	4.3	4.5	4.8	5.9	6.5
warm start	2.0	1.0	1.1	1.0	1.0	1.0
dist. power	2.0	1.9	1.8	1.6	1.8	2.0
centralize	25.8	13.3	10.2	7.7	4.3	3.6

Table 4.6 – Multinomial logistic regression: 96 partitions fixed

	96	48	36	24	12	8
init	27	22	25	25	30	37
update	5.0	6.0	7.8	8.6	10.9	14.1
warm start	2.0	2.4	3.0	3.3	3.9	4.0
dist. power	2.0	2.6	4.0	4.0	4.9	6.1
centralize	25.8	26.8	27.0	27.4	24.6	25.0

96-core case. The initialization, line search, update, and distributed power iteration are all accelerated by using more cores. However, the runtime for the gradient centralization is generally constant. This is due to huge communication overhead on the one hand and is because having more cores does not either help or damage the performance on the other hand. The runtime for the warm start is also almost constant. I have currently no explanation for that, but it is *not* an error of time measuring. Figure 4.10 confirms that, when more cores are used, the warm start time for each partition also increases. Concerning the 96-core case, the runtime is contrary to the intuition because 96 is the number of logical cores, but we have only 48 physical cores actually.

Table 4.5 presents the runtime for multinomial logistic regression when using one partition per core. The behaviors of the initialization and of the gradient centralization are the same as in Table 4.3. Nevertheless, type I behavior, instead of type II, occurs for the update. In fact, for multinomial logistic regression, the update complexity scales with the sample size per core, as shown in Table 4.2. Therefore, the more cores we use, the quicker the update will be. In addition to this new behavior, type III behavior also occurs. The runtime for the distributed power iteration first decreases and then increases with the increase of the number of cores. Compared with the pure type II behavior in Table 4.3, I suspect this type III behavior is actually a hybrid combination of both type I behavior and type II behavior. The cause of the potential type I component may be due to the fact that the memory

complexity scales with the sample size per core for multinomial logistic regression (see Table 4.2).

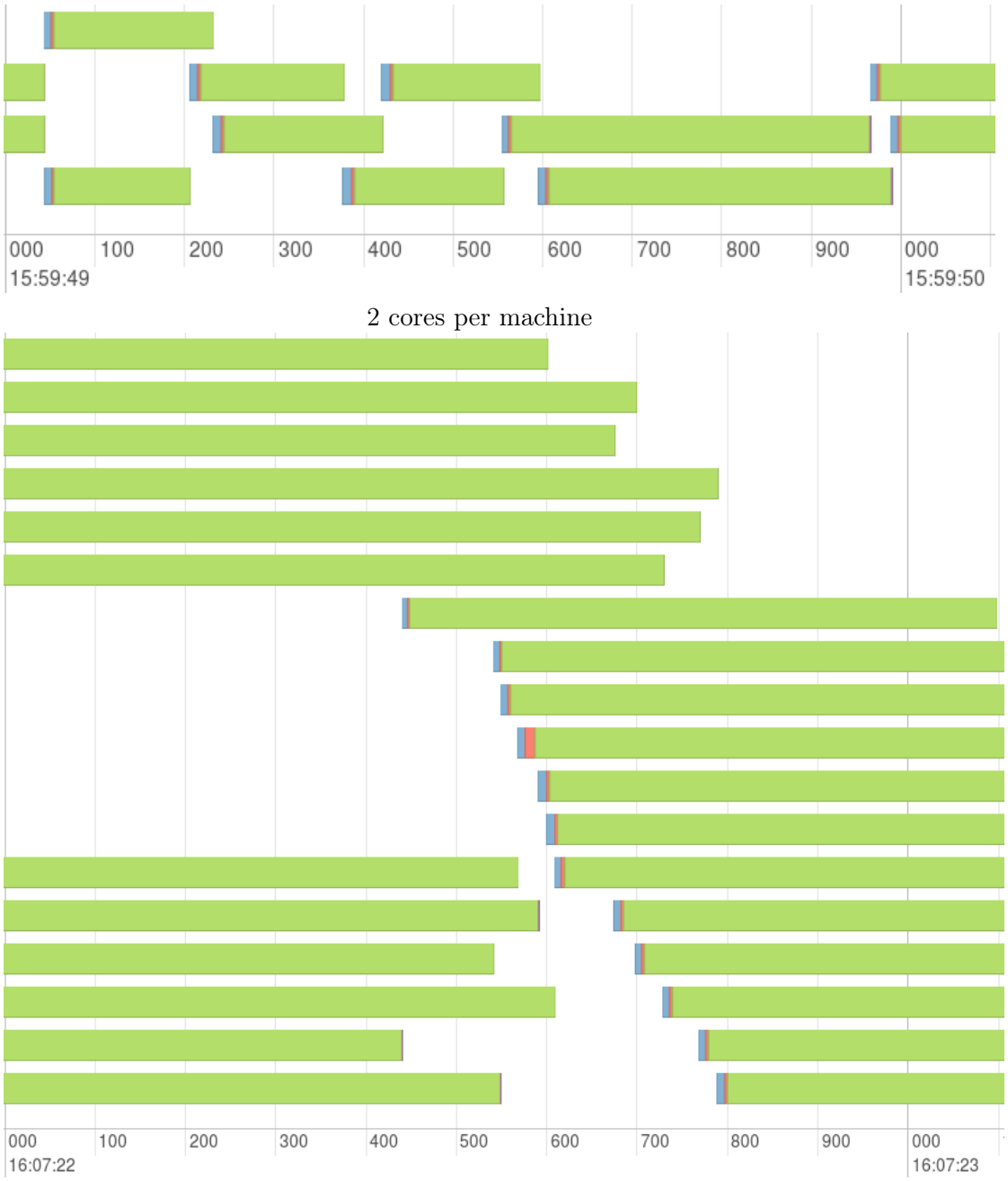
Table 4.6 presents the runtime for multinomial logistic regression when using 96 partitions fixed. It can be explained in the same way as Table 4.4, and thus I do not make any further explanation here.

4.7 Conclusion

This chapter explored the feasibility of executing Frank-Wolfe on a distributed infrastructure. It not only solved the Frank-Wolfe subproblem in a distributed manner but also proposed a recursive update scheme, which significantly accelerates the algorithm. Besides, it also provided ideas to conduct the line search in the same distributed infrastructure.

For the recursive update, some auxiliary information is stored, either as dense matrices or as low-rank matrices. I have developed an Apache SPARK package and done experiments to illustrate the feasibility of the dense representation through the multi-task least square and the multinomial logistic regression problem. Nonetheless, I have not had the luxury to experiment with the low-rank representation.

Experiments show that, in low dimension, gradient centralizing and average mixture suffice, whereas, in high dimension, distributed power iteration with or without warm start outperform. Furthermore, one or two power iterations are already enough, which is a much more encouraging fact than as stated in the previous chapter.



2 cores per machine

12 cores per machine

Figure 4.10 – It shows what is happening in a single machine during one second of warm start. The machine in the upper penal uses 2 cores, and the machine in the lower penal uses 12 cores.

Chapter 5

Conclusion

The goal of this research was to investigate the possibility of executing the Frank-Wolfe algorithms for the trace norm minimization problem in a star network via the bulk synchronous parallel (BSP) model. This research was composed of three parts.

- In the first part, I studied the theoretical foundation of using power iteration (or Lanczos iteration) to solve Frank-Wolfe's linear subproblem. I discovered that, although the subproblem cannot be solved in a deterministic manner, it *can* be solved in expectation and with high probability, which leads to what I called the *nondeterministic* sublinear convergence rate.
- In the second part, I studied the possibility to execute Frank-Wolfe on a distributed infrastructure. In particular, I have designed strategies to solve the subproblem, to conduct the line search, and to update. Meanwhile, I discovered that the rank-one update of Frank-Wolfe is especially suitable for the recursive update of the gradient, which significantly accelerates the computation.
- In the third part, I tested my approach on the problem of multi-task least square and of multinomial logistic regression on Apache SPARK. The experiments show that, among the four strategies to solve the subproblem in a distributed manner, gradient centralizing and average mixture are better in low-dimensional cases, and distributed power iteration with or without warm start outperforms in high-dimensional cases.

This research has several important implications.

- Researchers have been using power iteration or Lanczos iteration to solve the Frank-Wolfe subproblems for trace norm minimization. Although the empirical results are generally satisfying, few questioned the theoretical soundness of these methods. As we have seen in this dissertation, the subproblems can be solved only in expectation and with high probability. The convergence theory of Frank-Wolfe should therefore adopt, as in this dissertation, a probabilistic language. Although I treated only the simplest cases (*i.e.*, sublinear rate for convex objectives), similar results may be expected for all cases (*e.g.*, linear rate for strongly convex objectives with several Frank-Wolfe variants). In this

way, I laid the theoretical foundation for Frank-Wolfe when the subproblem can only be solved in expectation or with high probability.

- It is well known that the key to the Frank-Wolfe subproblem for trace norm minimization is the top eigenvector or the top singular vectors, but there is little analysis as detailed as in my dissertation. This dissertation studied it systematically by further dividing the problem into three cases – the asymmetric matrix space, the symmetric matrix space, and the positive semidefinite cone. In some of these cases, the answer is not as straightforward as we previously thought. The detailed analysis of these cases in this dissertation will be a handy tool when we apply Frank-Wolfe to recommender systems, multi-task learning, Boltzmann machine, metric learning, kernel learning, and so forth.

- Since Frank-Wolfe is computationally cheap, the evaluation of the gradient becomes the bottleneck. With the parallel or distributed algorithms in this dissertation, we can further accelerate this approach and make it match the efficiency of the parallel or distributed matrix factorization approach. Because of the possibility of the regularization path algorithm for Frank-Wolfe, which alleviates the burden of hyperparameter choice, we can eventually achieve better efficiency than matrix factorization.

- My experiments involve only the cluster of commodity machines and Apache SPARK, but the proposed algorithms work for *any* infrastructure compatible with the BSP model and the star network topology. The BSP model and the star network topology may be the most popular configuration and are widely available, especially in industries. My solutions can thus be reproduced on various infrastructures, such as GPU clusters and Google's TPU clusters, and be naturally integrated into the research or the service of other groups. Meanwhile, in the experiments, we observed a *phase transition* – distributed power iteration can be better or worse according to the dimension of the dataset. The same phenomenon can happen for other infrastructure as well, though the phase transition can be at a different point.

- As the experiment on ImageNet shows, as few as one or two power iterations are already enough for the convergence of Frank-Wolfe. Although the theoretical reason is still unclear, the excellent empirical performance encourages us to use Frank-Wolfe for deep learning. For instance, we can use Frank-Wolfe for the weight of every layer, which can potentially revolutionize how we train artificial neural network.

This study is mainly a proof of concept improving our understanding of the nature of Frank-Wolfe and its behavior when implemented on distributed infrastructures. Following the path its sets up, we can foresee the use of Frank-Wolfe to solve various problems in the academia and the industry.

Afterword

Doing a PhD is like rolling an immense boulder up to a hill, only to watch it come back to hit me, repeating this action for eternity.

Estragon: *Let's go.*

Vladimir: *We can't.*

Estragon: *Why not?*

Vladimir: *We are doing a PhD.*

Here, I would like to thank *all* who have helped me or given knowledge to me in my life. If it had not been for those who had made me the person I am now, I would not have had the motivation to finish my PhD. I wanted to show them that the education I had received from them and the help they had given me were so powerful that I could use it to make a positive impact, however little it was, on the knowledge tree. I felt the *calling* of using the acquired knowledge for the benefit of human being, so that they could be satisfied with their achievement and would feel the glory. It would not be exaggerating to say that this dissertation was dedicated to all who had helped me in various ways.

This dissertation was my first and maybe the last opportunity to thank them. Indeed, I had wanted to include an ultra-comprehensive name list here, but finally gave up because of some technical difficulties. On the one hand, I do not have a flawless memory: forgetting someone important in an ultra-comprehensive list will be *unforgivable*. On the other hand, many people are unknown to me: what are the names of those in the school canteens who have been giving me extra food because I am too thin? Therefore, in hoping that the missing ones could "automatically" feel my gratitude, I decided to write down only the names of the pivotal persons.

Among those teachers or professors, there are four who have the far most influence on me. I would like to thank Weidong Yin for the enlightenment on programming, which has been one of my greatest weapons since. I would like to thank Cheng Xu for showing me how mathematics can be used in a flexible and magical way, which is also what I alway strive to achieve. I would like to thank Ke Xu for solving many of my doubts toward the society and the world, and I appreciate our email exchange. I would like to thank Arnak Dalalyan for guiding me to the world of statistics and machine learning, which has been my favorite domain since, and for the precious scientific training I received from him.

They represent all who have helped me, in a form or another, and have played a positive role in my life. To dedicate this thesis to them, I tried my best to make it as valuable as possible. If there still remains any mistake or flaw in this thesis, the author, I, is the only person to blame because of his laziness and procrastination.

Bibliography

Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, and others. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016. URL https://arxiv.org/abs/1603.04467.

David H. Ackley, Geoffrey E. Hinton, and Terrence J. Sejnowski. A learning algorithm for Boltzmann machines. *Cognitive science*, 9(1):147–169, 1985.

AD Aleksandorov. Almost everywhere existence of the second differential of a convex function and some properties of convex functions. *Leningrad Univ. Ann.*, 37:3–35, 1939. URL http://ci.nii.ac.jp/naid/10006416975/.

Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. Convex multi-task feature learning. *Machine Learning*, 73(3):243–272, 2008. URL http://link.springer.com/article/10.1007/s10994-007-5040-8.

Francis R. Bach. Consistency of trace norm minimization. *Journal of Machine Learning Research*, 9(Jun):1019–1048, 2008. URL http://www.jmlr.org/papers/v9/bach08a.html.

Afonso S. Bandeira, Nicolas Boumal, and Vladislav Voroninski. On the low-rank approach for semidefinite programs arising in synchronization and community detection. *arXiv:1602.04426 [math]*, February 2016. URL http://arxiv.org/abs/1602.04426. arXiv: 1602.04426.

Amir Beck and Shimrit Shtern. Linearly convergent away-step conditional gradient for non-strongly convex functions. *Mathematical Programming*, pages 1–27, 2015. URL http://link.springer.com/article/10.1007/s10107-016-1069-4.

Amir Beck and Marc Teboulle. A fast iterative shrinkage-thresholding algorithm for linear inverse problems. *SIAM journal on imaging sciences*, 2(1):183–202, 2009. URL http://epubs.siam.org/doi/abs/10.1137/080716542.

Aurélien Bellet, Yingyu Liang, Alireza Bagheri Garakani, Maria-Florina Balcan, and Fei Sha. Distributed Frank-Wolfe algorithm: A unified framework for communication-efficient sparse learning. *CoRR, abs/1404.2644*, 2014. URL http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.750.5951&rep=rep1&type=pdf.

Srinadh Bhojanapalli, Behnam Neyshabur, and Nati Srebro. Global optimality of local search for low rank matrix recovery. In *Advances in Neural Information Processing Systems*, pages 3873–3881, 2016.

Christopher M. Bishop. Pattern recognition. *Machine Learning*, 128:1–58, 2006. URL http://www.academia.edu/download/30428242/bg0137.pdf.

Stephen Boyd and Lieven Vandenberghe. *Convex optimization*. Cambridge university press, 2004.

Herbert Busemann and Willy Feller. Krümmungseigenschaften konvexer Flächen. *Acta Mathematica*, 66(1):1–47, 1936. URL http://link.springer.com/content/pdf/10.1007/BF02546515.pdf.

R. Cabral, F. D. L. Torre, J. P. Costeira, and A. Bernardino. Unifying Nuclear Norm and Bilinear Factorization Approaches for Low-Rank Matrix Decomposition. In *2013 IEEE International Conference on Computer Vision*, pages 2488–2495, December 2013. doi: 10.1109/ICCV.2013.309.

Ricardo S. Cabral, Fernando Torre, Joao P. Costeira, and Alexandre Bernardino. Matrix Completion for Multi-label Image Classification. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 190–198. Curran Associates, Inc., 2011. URL http://papers.nips.cc/paper/4419-matrix-completion-for-multi-label-image-classification.pdf.

Jian-Feng Cai, Emmanuel J. Candès, and Zuowei Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journal on Optimization*, 20(4):1956–1982, 2010. URL http://epubs.siam.org/doi/abs/10.1137/080738970.

E. Candès, Y. Eldar, T. Strohmer, and V. Voroninski. Phase Retrieval via Matrix Completion. *SIAM Journal on Imaging Sciences*, 6(1):199–225, January 2013. doi: 10.1137/110848074. URL http://epubs.siam.org/doi/abs/10.1137/110848074.

Emmanuel Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Communications of the ACM*, 55(6):111–119, 2012. URL http://dl.acm.org/citation.cfm?id=2184343.

Emmanuel J. Candès and Yaniv Plan. Matrix completion with noise. *Proceedings of the IEEE*, 98(6):925–936, 2010. URL http://ieeexplore.ieee.org/abstract/document/5454406/.

Emmanuel J. Candès and Terence Tao. The power of convex relaxation: Near-optimal matrix completion. *IEEE Transactions on Information Theory*, 56 (5):2053–2080, 2010. URL http://ieeexplore.ieee.org/abstract/document/5452187/.

Rich Caruana. Multitask learning. In *Learning to learn*, pages 95–133. Springer, 1998. URL http://link.springer.com/chapter/10.1007/978-1-4615-5529-2_5.

Venkat Chandrasekaran, Benjamin Recht, Pablo A. Parrilo, and Alan S. Willsky. The convex algebraic geometry of linear inverse problems. In *Communication, Control, and Computing (Allerton), 2010 48th Annual Allerton Conference on*, pages 699–703. IEEE, 2010. URL http://ieeexplore.ieee.org/abstract/document/5706975/.

Yudong Chen, Srinadh Bhojanapalli, Sujay Sanghavi, and Rachel Ward. Coherent Matrix Completion. In *PMLR*, pages 674–682, January 2014. URL http://proceedings.mlr.press/v32/chenc14.html.

François Chollet. Keras. https://github.com/fchollet/keras, 2015.

Kenneth L. Clarkson. Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm. *ACM Transactions on Algorithms (TALG)*, 6(4):63, 2010. URL http://dl.acm.org/citation.cfm?id=1824783.

S. Damla Ahipasaoglu, Peng Sun, and Michael J. Todd. Linear convergence of a modified Frank–Wolfe algorithm for computing minimum-volume enclosing ellipsoids. *Optimisation Methods and Software*, 23(1):5–19, 2008. URL http://www.tandfonline.com/doi/full/10.1080/10556780701589669.

John N. Darroch and Douglas Ratcliff. Generalized iterative scaling for log-linear models. *The annals of mathematical statistics*, pages 1470–1480, 1972. URL http://www.jstor.org/stable/2240069.

Alexandre d'Aspremont. Smooth optimization with approximate gradient. *SIAM Journal on Optimization*, 19(3):1171–1183, 2008.

Jeffrey Dean and Sanjay Ghemawat. MapReduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008. URL http://dl.acm.org/citation.cfm?id=1327492.

Vladimir Fedorovich Demianov and Aleksandr Moiseevich Rubinov. *Approximate methods in optimization problems*, volume 32. Elsevier Publishing Company, 1970.

Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009. URL http://ieeexplore.ieee.org/abstract/document/5206848/.

Peter Diggle. *Analysis of longitudinal data*. Oxford University Press, 2002. URL https://books.google.fr/books?hl=en&lr=&id=JCwSDAAAQBAJ&oi=fnd&pg=PP1&dq=+Longitudinal+Data&ots=jT7aYEwuMF&sig=ukEzA2FZeQKQZPxGdVsy1pyjSLI.

F. Dinuzzo and K. Fukumizu. Learning low-rank output kernels. In *PMLR*, pages 181–196, November 2011. URL http://proceedings.mlr.press/v20/dinuzzo11.html.

Miroslav Dudik, Zaid Harchaoui, and Jérôme Malick. Lifted coordinate descent for learning with trace-norm regularization. In *Artificial Intelligence and Statistics*, pages 327–336, 2012. URL http://www.jmlr.org/proceedings/papers/v22/dudik12/dudik12.pdf.

Joseph C. Dunn. Rates of convergence for conditional gradient algorithms near singular and nonsingular extremals. *SIAM Journal on Control and Optimization*, 17(2):187–211, 1979. URL http://epubs.siam.org/doi/abs/10.1137/0317015.

Theodoros Evgeniou and Massimiliano Pontil. Regularized multi–task learning. In *Proceedings of the tenth ACM SIGKDD international conference on Knowledge*

discovery and data mining, pages 109–117. ACM, 2004. URL http://dl.acm.org/citation.cfm?id=1014067.

M. Fazel, H. Hindi, and S. Boyd. Rank minimization and applications in system theory. In *Proceedings of the 2004 American Control Conference*, volume 4, pages 3273–3278 vol.4, June 2004.

Maryam Fazel, Haitham Hindi, and Stephen P. Boyd. A rank minimization heuristic with application to minimum order system approximation. In *American Control Conference, 2001. Proceedings of the 2001*, volume 6, pages 4734–4739. IEEE, 2001. URL http://ieeexplore.ieee.org/abstract/document/945730/.

Herbert Federer. *Geometric measure theory*. Springer, 2014.

Marguerite Frank and Philip Wolfe. An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110, March 1956. ISSN 1931-9193. doi: 10.1002/nav.3800030109. URL http://onlinelibrary.wiley.com/doi/10.1002/nav.3800030109/abstract.

Robert M. Freund and Paul Grigas. New analysis and results for the Frank–Wolfe method. *Mathematical Programming*, 155(1-2):199–230, 2016. URL http://link.springer.com/article/10.1007/s10107-014-0841-6.

Robert M. Freund, Paul Grigas, and Rahul Mazumder. An Extended Frank–Wolfe Method with "In-Face" Directions, and Its Application to Low-Rank Matrix Completion. *SIAM Journal on Optimization*, 27(1):319–346, 2017. URL http://epubs.siam.org/doi/abs/10.1137/15M104726X.

Masao Fukushima. A modified Frank-Wolfe algorithm for solving the traffic assignment problem. *Transportation Research Part B: Methodological*, 18(2): 169–177, 1984. URL http://www.sciencedirect.com/science/article/pii/0191261584900298.

Dan Garber. Faster Projection-free Convex Optimization over the Spectrahedron. In *Advances in Neural Information Processing Systems*, pages 874–882, 2016a.

Dan Garber. *Projection-free Algorithms for Convex Optimization and Online Learning*. PhD thesis, Technion-Israel Institute of Technology, Faculty of Industrial and Management Engineering, 2016b.

Dan Garber and Elad Hazan. A linearly convergent conditional gradient algorithm with applications to online and stochastic optimization. *arXiv preprint arXiv:1301.4666*, 2013a. URL https://arxiv.org/abs/1301.4666.

Dan Garber and Elad Hazan. Playing non-linear games with linear oracles. In *Foundations of Computer Science (FOCS), 2013 IEEE 54th Annual Symposium on*, pages 420–428. IEEE, 2013b. URL http://ieeexplore.ieee.org/abstract/document/6686178/.

Dan Garber and Elad Hazan. Faster Rates for the Frank-Wolfe Method over Strongly-Convex Sets. In *PMLR*, pages 541–549, June 2015. URL http://proceedings.mlr.press/v37/garbera15.html.

Dan Garber, Elad Hazan, and Tengyu Ma. Online Learning of Eigenvectors. In *PMLR*, pages 560–568, June 2015. URL http://proceedings.mlr.press/v37/garberb15.html.

R.F. Gariepy and W.P. Ziemer. *Modern Real Analysis*. PWS Pub., 1995. ISBN 9780534944049. URL https://books.google.fr/books?id=CBQNNQAACAAJ.

Rong Ge, Furong Huang, Chi Jin, and Yang Yuan. Escaping From Saddle Points — Online Stochastic Gradient for Tensor Decomposition. *arXiv:1503.02101 [cs, math, stat]*, March 2015. URL http://arxiv.org/abs/1503.02101. arXiv: 1503.02101.

Rong Ge, Jason D. Lee, and Tengyu Ma. Matrix completion has no spurious local minimum. In *Advances in Neural Information Processing Systems*, pages 2973–2981, 2016. URL http://papers.nips.cc/paper/6048-matrix-completion-has-no-spurious-local-minimum.

Gauthier Gidel, Tony Jebara, and Simon Lacoste-Julien. Frank-Wolfe Algorithms for Saddle Point Problems. *arXiv preprint arXiv:1610.07797*, 2016. URL https://arxiv.org/abs/1610.07797.

Joachim Giesen, Martin Jaggi, and Sören Laue. Optimizing over the growing spectrahedron. *Algorithms–ESA 2012*, pages 503–514, 2012. URL http://www.springerlink.com/index/5636G3271781L702.pdf.

Michel X. Goemans and David P. Williamson. Improved approximation algorithms for maximum cut and satisfiability problems using semidefinite programming. *Journal of the ACM (JACM)*, 42(6):1115–1145, 1995. URL http://dl.acm.org/citation.cfm?id=227684.

Andrew Goldberg, Ben Recht, Junming Xu, Robert Nowak, and Xiaojin Zhu. Transduction with Matrix Completion: Three Birds with One Stone. In J. D. Lafferty, C. K. I. Williams, J. Shawe-Taylor, R. S. Zemel, and A. Culotta, editors, *Advances in Neural Information Processing Systems 23*, pages 757–765. Curran Associates, Inc., 2010. URL http://papers.nips.cc/paper/3932-transduction-with-matrix-completion-three-birds-with-one-stone.pdf.

Donald Goldfarb, Garud Iyengar, and Chaoxu Zhou. Linear Convergence of Stochastic Frank Wolfe Variants. *arXiv preprint arXiv:1703.07269*, 2017. URL https://arxiv.org/abs/1703.07269.

Edouard Grave, Guillaume R. Obozinski, and Francis R. Bach. Trace lasso: a trace norm regularization for correlated designs. In *Advances in Neural Information Processing Systems*, pages 2187–2195, 2011. URL http://papers.nips.cc/paper/4277-trace-lasso-a-trace-norm-regularization-for-correlated-designs.

David Gross. Recovering low-rank matrices from few coefficients in any basis. *IEEE Transactions on Information Theory*, 57(3):1548–1566, 2011. URL http://ieeexplore.ieee.org/abstract/document/5714248/.

David Gross, Yi-Kai Liu, Steven T. Flammia, Stephen Becker, and Jens Eisert. Quantum state tomography via compressed sensing. *Physical review letters*, 105 (15):150401, 2010. URL https://journals.aps.org/prl/abstract/10.1103/PhysRevLett.105.150401.

Jacques Guélat and Patrice Marcotte. Some comments on Wolfe's 'away step'. *Mathematical Programming*, 35(1):110–119, 1986. URL http://www.springerlink.com/index/xx184071t262341j.pdf.

Z. Harchaoui, M. Douze, M. Paulin, M. Dudik, and J. Malick. Large-scale image classification with trace-norm regularization. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3386–3393, June 2012. doi: 10.1109/CVPR.2012.6248078.

Elad Hazan. Sparse approximate solutions to semidefinite programs. In *Latin American Symposium on Theoretical Informatics*, pages 306–316. Springer, 2008. URL http://link.springer.com/chapter/10.1007/978-3-540-78773-0_27.

Elad Hazan and Satyen Kale. Projection-free online learning. *arXiv preprint arXiv:1206.4657*, 2012. URL https://arxiv.org/abs/1206.4657.

Elad Hazan and Haipeng Luo. Variance-reduced and projection-free stochastic optimization. In *International Conference on Machine Learning*, pages 1263–1271, 2016. URL http://www.jmlr.org/proceedings/papers/v48/hazana16.pdf.

Elad Hazan and others. Introduction to online convex optimization. *Foundations and Trends® in Optimization*, 2(3-4):157–325, 2016. URL http://ftp.nowpublishers.com/article/Details/OPT-013.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.

Juha M. Heinonen. Lectures on lipschitz analysis. 2005.

Nicholas J. Higham. *Matrix nearness problems and applications*. University of Manchester. Department of Mathematics, 1988.

Zhouyuan Huo, Feiping Nie, and Heng Huang. Robust and Effective Metric Learning Using Capped Trace Norm: Metric Learning via Capped Trace Norm. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 1605–1614, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939853. URL http://doi.acm.org/10.1145/2939672.2939853.

Martin Jaggi. Sparse convex optimization methods for machine learning. 2011. URL http://e-collection.library.ethz.ch/eserv/eth:5262/eth-5262-02.pdf.

Martin Jaggi. Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization. In *ICML 2013 - Proceedings of the 30th International Conference on Machine Learning*, 2013. URL http://www.jmlr.org/proceedings/papers/v28/jaggi13-supp.pdf.

Martin Jaggi, Marek Sulovsk, and others. A simple algorithm for nuclear norm regularized problems. In *Proceedings of the 27th International Conference on Machine Learning (ICML-10)*, pages 471–478, 2010. URL http://machinelearning.wustl.edu/mlpapers/paper_files/icml2010_JaggiS10.pdf.

H. Ji, C. Liu, Z. Shen, and Y. Xu. Robust video denoising using low rank matrix completion. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1791–1798, June 2010. doi: 10.1109/CVPR.2010.5539849.

Shuiwang Ji and Jieping Ye. An accelerated gradient method for trace norm minimization. In *Proceedings of the 26th annual international conference on machine learning*, pages 457–464. ACM, 2009. URL http://dl.acm.org/citation.cfm?id=1553434.

Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. In *Advances in neural information processing systems*, pages 315–323, 2013.

Bhargav Kanagal and Vikas Sindhwani. Rank selection in low-rank matrix approximations: A study of cross-validation for NMFs. In *Proc Conf Adv Neural Inf Process*, volume 1, pages 10–15, 2010.

Kwangmoo Koh, Seung-Jean Kim, and Stephen Boyd. An interior-point method for large-scale l1-regularized logistic regression. *Journal of Machine learning research*, 8(Jul):1519–1555, 2007. URL http://www.jmlr.org/papers/v8/koh07a.html.

Vladimir Koltchinskii, Karim Lounici, and Alexandre B. Tsybakov. Nuclear-norm penalization and optimal rates for noisy low-rank matrix completion. *The Annals of Statistics*, pages 2302–2329, 2011. URL http://www.jstor.org/stable/41713579.

Yehuda Koren, Robert Bell, and Chris Volinsky. Matrix Factorization Techniques for Recommender Systems. *Computer*, 42(8):30–37, August 2009. ISSN 0018-9162. doi: 10.1109/MC.2009.263. URL http://dx.doi.org/10.1109/MC.2009.263.

J. Kuczyński and H. Woźniakowski. Estimating the largest eigenvalue by the power and Lanczos algorithms with a random start. *SIAM journal on matrix analysis and applications*, 13(4):1094–1122, 1992. URL http://epubs.siam.org/doi/abs/10.1137/0613066.

Piyush Kumar and E. Alper Yıldırım. A linearly convergent linear-time first-order algorithm for support vector classification with a core set result. *INFORMS Journal on Computing*, 23(3):377–391, 2011. URL http://pubsonline.informs.org/doi/abs/10.1287/ijoc.1100.0412.

Aapo Kyrola, Guy E. Blelloch, and Carlos Guestrin. Graphchi: Large-scale graph computation on just a pc. USENIX, 2012. URL https://www.usenix.org/system/files/conference/osdi12/osdi12-final-126.pdf.

Simon Lacoste-Julien and Martin Jaggi. On the global linear convergence of Frank-Wolfe optimization variants. In *Advances in Neural Information Processing Systems*, pages 496–504, 2015. URL http://papers.nips.cc/paper/5925-on-the-global-linear-convergence-of-frank-wolfe-optimizati.

Simon Lacoste-Julien, Martin Jaggi, Mark Schmidt, and Patrick Pletscher. Block-coordinate Frank-Wolfe optimization for structural SVMs. *arXiv preprint arXiv:1207.4747*, 2012. URL https://arxiv.org/abs/1207.4747.

Jean Lafond, Hoi-To Wai, and Eric Moulines. Convergence analysis of a stochastic projection-free algorithm. *stat*, 1050:5, 2015. URL https://pdfs.semanticscholar.org/2e0c/5ba2aa5d2cf5592274ec1f6dba5e079aca1d.pdf.

J. L. (Joseph Louis) Lagrange. *Mécanique analytique*. Paris, Ve Courcier, 1811. URL http://archive.org/details/mcaniqueanalyt01lagr.

Nan M. Laird and James H. Ware. Random-effects models for longitudinal data. *Biometrics*, pages 963–974, 1982. URL http://www.jstor.org/stable/2529876.

Guanghui Lan and Yi Zhou. Conditional gradient sliding for convex optimization. *SIAM Journal on Optimization*, 26(2):1379–1409, 2016. URL http://epubs.siam.org/doi/abs/10.1137/140992382.

Cornelius Lanczos. *An iteration method for the solution of the eigenvalue problem of linear differential and integral operators*. United States Governm. Press Office Los Angeles, CA, 1950. URL http://www.cs.umd.edu/~oleary/lanczos1950.pdf.

Neil D. Lawrence and John C. Platt. Learning to learn with the informative vector machine. In *Proceedings of the twenty-first international conference on Machine learning*, page 65. ACM, 2004. URL http://dl.acm.org/citation.cfm?id=1015382.

Henri Lebesgue. *Sur l'intégration des fonctions discontinues*. Gauthier-Villars, 1910.

Larry J. LeBlanc, Richard V. Helgason, and David E. Boyce. Improved efficiency of the Frank-Wolfe algorithm for convex network programs. *Transportation Science*, 19(4):445–462, 1985. URL http://pubsonline.informs.org/doi/abs/10.1287/trsc.19.4.445.

Jason D. Lee, Ben Recht, Nathan Srebro, Joel Tropp, and Ruslan R. Salakhutdinov. Practical large-scale optimization for max-norm regularization. In *Advances in Neural Information Processing Systems*, pages 1297–1305, 2010. URL http://papers.nips.cc/paper/4124-practical-large-scale-optimization-for-max-norm-regularization.

E. S. Levitin and B. T. Polyak. Constrained minimization methods. *USSR Computational Mathematics and Mathematical Physics*, 6(5):1–50, January 1966. ISSN 0041-5553. doi: 10.1016/0041-5553(66)90114-5. URL http://www.sciencedirect.com/science/article/pii/0041555366901145.

Mu Li, David G. Andersen, Jun Woo Park, Alexander J. Smola, Amr Ahmed, Vanja Josifovski, James Long, Eugene J. Shekita, and Bor-Yiing Su. Scaling Distributed Machine Learning with the Parameter Server. In *OSDI*, volume 1, page 3, 2014. URL https://www.usenix.org/system/files/conference/osdi14/osdi14-paper-li_mu.pdf.

Kung-Yee Liang and Scott L. Zeger. Longitudinal data analysis using generalized linear models. *Biometrika*, pages 13–22, 1986. URL http://www.jstor.org/stable/2336267.

Wei Liu, Cun Mu, Rongrong Ji, Shiqian Ma, John R. Smith, and Shih-Fu Chang. Low-rank Similarity Metric Learning in High Dimensions. In *Proceedings of the*

Twenty-Ninth AAAI Conference on Artificial Intelligence, AAAI'15, pages 2792–2799, Austin, Texas, 2015. AAAI Press. ISBN 978-0-262-51129-2. URL http://dl.acm.org/citation.cfm?id=2886521.2886710.

Zhuanghua Liu and Ivor Tsang. Approximate Conditional Gradient Descent on Multi-Class Classification. In *AAAI*, pages 2301–2307, 2017. URL http://www.aaai.org/ocs/index.php/AAAI/AAAI17/paper/download/14853/14408.

Francesco Locatello, Rajiv Khanna, Michael Tschannen, and Martin Jaggi. A Unified Optimization View on Generalized Matching Pursuit and Frank-Wolfe. In *PMLR*, pages 860–868, April 2017. URL http://proceedings.mlr.press/v54/locatello17a.html.

Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M. Hellerstein. Distributed GraphLab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012. URL http://dl.acm.org/citation.cfm?id=2212354.

Shiqian Ma, Donald Goldfarb, and Lifeng Chen. Fixed Point and Bregman Iterative Methods for Matrix Rank Minimization. *arXiv:0905.1643 [cs, math]*, May 2009. URL http://arxiv.org/abs/0905.1643. arXiv: 0905.1643.

Mehrdad Mahdavi, Lijun Zhang, and Rong Jin. Mixed optimization for smooth functions. In *Advances in Neural Information Processing Systems*, pages 674–682, 2013. URL http://papers.nips.cc/paper/4941-mixed-optimization-for-smooth-functions.

Julien Mairal. Optimization with first-order surrogate functions. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 783–791, 2013. URL http://www.jmlr.org/proceedings/papers/v28/mairal13.pdf.

Grzegorz Malewicz, Matthew H. Austern, Aart JC Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: a system for large-scale graph processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of data*, pages 135–146. ACM, 2010. URL http://dl.acm.org/citation.cfm?id=1807184.

Robert Malouf. A comparison of algorithms for maximum entropy parameter estimation. In *proceedings of the 6th conference on Natural language learning-Volume 20*, pages 1–7. Association for Computational Linguistics, 2002. URL http://dl.acm.org/citation.cfm?id=1118871.

Olvi L. Mangasarian and Benjamin Recht. Probability of unique integer solution to a system of linear equations. *European Journal of Operational Research*, 214 (1):27–30, 2011. URL http://www.sciencedirect.com/science/article/pii/S0377221711003511.

Ryan Mcdonald, Mehryar Mohri, Nathan Silberman, Dan Walker, and Gideon S. Mann. Efficient large-scale distributed training of conditional maximum entropy models. In *Advances in Neural Information Processing Systems*, pages 1231–1239, 2009.

Gilles Meyer, Silvere Bonnabel, and Rodolphe Sepulchre. Linear regression under fixed-rank constraints: a Riemannian approach. In *Proceedings of the 28th international conference on machine learning*, 2011. URL http://orbi.ulg.ac.be/handle/2268/90930.

B. F. Mitchell, Vladimir Fedorovich Demyanov, and V. N. Malozemov. Finding the point of a polyhedron closest to the origin. *SIAM Journal on Control*, 12(1):19–26, 1974. URL http://epubs.siam.org/doi/pdf/10.1137/0312003.

Armin Moharrer and Stratis Ioannidis. Distributing frank-wolfe via map-reduce. In *ICDM*, 2017.

Cun Mu, Yuqian Zhang, John Wright, and Donald Goldfarb. Scalable Robust Matrix Recovery: Frank–Wolfe Meets Proximal Methods. *SIAM Journal on Scientific Computing*, 38(5):A3291–A3317, 2016. URL http://epubs.siam.org/doi/abs/10.1137/15M101628X.

Yurii Nesterov. *Introductory lectures on convex optimization: A basic course*, volume 87. Springer Science & Business Media, 2013.

Yurii Nesterov and others. *Gradient methods for minimizing composite objective function*. Core Louvain-la-Neuve, 2007. URL http://www.ucl.be/cps/ucl/doc/core/documents/coredp2007_76.pdf.

Hua Ouyang and Alexander Gray. Fast stochastic Frank-Wolfe algorithms for nonlinear SVMs. In *Proceedings of the 2010 SIAM International Conference on Data Mining*, pages 245–256. SIAM, 2010. URL http://epubs.siam.org/doi/abs/10.1137/1.9781611972801.22.

Javier Pena and Daniel Rodriguez. Polytope conditioning and linear convergence of the Frank-Wolfe algorithm. *arXiv preprint arXiv:1512.06142*, 2015. URL https://arxiv.org/abs/1512.06142.

Javier Pena, Daniel Rodríguez, and Negar Soheili. On the von Neumann and Frank–Wolfe Algorithms with Away Steps. *SIAM Journal on Optimization*, 26(1):499–512, 2016. URL http://epubs.siam.org/doi/abs/10.1137/15M1009937.

Diego Peteiro-Barral and Bertha Guijarro-Berdiñas. A survey of methods for distributed machine learning. *Progress in Artificial Intelligence*, 2(1):1–11, March 2013. ISSN 2192-6352, 2192-6360. doi: 10.1007/s13748-012-0035-5. URL https://link.springer.com/article/10.1007/s13748-012-0035-5.

Wei Ping, Qiang Liu, and Alexander T. Ihler. Learning Infinite RBMs with Frank-Wolfe. In *Advances in Neural Information Processing Systems*, pages 3063–3071, 2016. URL http://papers.nips.cc/paper/6341-learning-infinite-rbms-with-frank-wolfe.

Ting Kei Pong, Paul Tseng, Shuiwang Ji, and Jieping Ye. Trace norm regularization: Reformulations, algorithms, and multi-task learning. *SIAM Journal on Optimization*, 20(6):3465–3489, 2010. URL http://epubs.siam.org/doi/abs/10.1137/090763184.

Benjamin Recht. A simpler approach to matrix completion. *Journal of Machine Learning Research*, 12(Dec):3413–3430, 2011. URL http://www.jmlr.org/papers/v12/recht11a.html.

Stephen M. Robinson. Generalized equations and their solutions, Part II: Applications to nonlinear programming. *Optimality and Stability in Mathematical Programming*, pages 200–221, 1982. URL http://www.springerlink.com/index/L2J88715135N0408.pdf.

Ralph Tyrell Rockafellar. *Convex analysis*. Princeton university press, 2015. URL https://books.google.fr/books?hl=en&lr=&id=jzpzBwAAQBAJ&oi=fnd&pg=PP1&dq=rockefeller+convex+analysis&ots=aHH93nQm5d&sig=Uciw_w9aFbdeCNsfqNdTohr9_k0.

Walter Rudin and others. *Principles of mathematical analysis*, volume 3. McGraw-Hill New York, 1964. URL http://faculty.ksu.edu.sa/fawaz/481files/books/rudin.pdf.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

Judith D. Singer and John B. Willett. *Applied longitudinal data analysis: Modeling change and event occurrence*. Oxford university press, 2003. URL https://books.google.fr/books?hl=en&lr=&id=PpnA1M8VwR8C&oi=fnd&pg=PA1&dq=+Longitudinal+Data&ots=N4uawA5ypG&sig=J0jdlRJepBQV2q_bMdqcWwOKmT0.

Nathan Srebro, Jason Rennie, and Tommi S. Jaakkola. Maximum-margin matrix factorization. In *Advances in neural information processing systems*, pages 1329–1336, 2005. URL http://papers.nips.cc/paper/2655-maximum-margin-matrix-factorization.pdf.

Jos F. Sturm. Using SeDuMi 1.02, a MATLAB toolbox for optimization over symmetric cones. *Optimization methods and software*, 11(1-4):625–653, 1999. URL http://www.tandfonline.com/doi/abs/10.1080/10556789908805766.

Ju Sun, Qing Qu, and John Wright. When Are Nonconvex Problems Not Scary? *arXiv:1510.06096 [cs, math, stat]*, October 2015. URL http://arxiv.org/abs/1510.06096. arXiv: 1510.06096.

Anthony C. Thompson. *Minkowski geometry*, volume 63. Cambridge University Press, 1996. URL https://books.google.fr/books?hl=en&lr=&id=9QBfSBWjt7EC&oi=fnd&pg=PR9&dq=Minkowski+Geometry&ots=DToOYZjMQV&sig=--Oo2JtD2BRVXlFIP2VxsBI21Xk.

Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996. URL http://www.jstor.org/stable/2346178.

Kim-Chuan Toh and Sangwoon Yun. An accelerated proximal gradient algorithm for nuclear norm regularized linear least squares problems. *Pacific Journal of optimization*, 6(615-640):15, 2010. URL http://www.math.nus.edu.sg/~mattohkc/papers/mc11.pdf.

Kim-Chuan Toh, Michael J. Todd, and Reha H. Tütüncü. SDPT3—a MATLAB software package for semidefinite programming, version 1.3. *Optimization methods and software*, 11(1-4):545–581, 1999. URL http://www.tandfonline.com/doi/abs/10.1080/10556789908805762.

John Tsitsiklis, Dimitri Bertsekas, and Michael Athans. Distributed asynchronous deterministic and stochastic gradient optimization algorithms. *IEEE transactions on automatic control*, 31(9):803–812, 1986. URL http://ieeexplore.ieee.org/abstract/document/1104412/.

Bart Vandereycken. Low-rank matrix completion by Riemannian optimization. *SIAM Journal on Optimization*, 23(2):1214–1236, 2013. URL http://epubs.siam.org/doi/abs/10.1137/110845768.

Hoi-To Wai, Jean Lafond, Anna Scaglione, and Eric Moulines. Decentralized Frank-Wolfe Algorithm for Convex and Non-convex Problems. *IEEE Transactions on Automatic Control*, 2017a. URL http://ieeexplore.ieee.org/abstract/document/7883821/.

Hoi-To Wai, Anna Scaglione, Jean Lafond, and Eric Moulines. Fast and privacy preserving distributed low-rank regression. In *Acoustics, Speech and Signal Processing (ICASSP), 2017 IEEE International Conference on*, pages 4451–4455. IEEE, 2017b.

Yu-Xiang Wang, Veeranjaneyulu Sadhanala, Wei Dai, Willie Neiswanger, Suvrit Sra, and Eric Xing. Parallel and distributed block-coordinate Frank-Wolfe algorithms. In *International Conference on Machine Learning*, pages 1548–1557, 2016. URL http://www.jmlr.org/proceedings/papers/v48/wangd16.pdf.

Philip Wolfe. Convergence theory in nonlinear programming. *Integer and nonlinear programming*, pages 1–36, 1970.

Eric P. Xing, Qirong Ho, Wei Dai, Jin Kyu Kim, Jinliang Wei, Seunghak Lee, Xun Zheng, Pengtao Xie, Abhimanu Kumar, and Yaoliang Yu. Petuum: A new platform for distributed machine learning on big data. *IEEE Transactions on Big Data*, 1(2):49–67, 2015a. URL http://ieeexplore.ieee.org/abstract/document/7239545/.

Eric P. Xing, Qirong Ho, Pengtao Xie, and Wei Dai. Strategies and Principles of Distributed Machine Learning on Big Data. *arXiv:1512.09295 [cs, stat]*, December 2015b. URL http://arxiv.org/abs/1512.09295. arXiv: 1512.09295.

Hong-Kun Xu. Convergence Analysis of the Frank-Wolfe Algorithm and Its Generalization in Banach Spaces. *arXiv:1710.07367 [math]*, October 2017. URL http://arxiv.org/abs/1710.07367. arXiv: 1710.07367.

B. Yao, Z. Zhao, and K. Liu. Metric learning with trace-norm regularization for person re-identification. In *2014 IEEE International Conference on Image Processing (ICIP)*, pages 2442–2446, October 2014. doi: 10.1109/ICIP.2014.7025494.

Hsiang-Fu Yu, Fang-Lan Huang, and Chih-Jen Lin. Dual coordinate descent methods for logistic regression and maximum entropy models. *Machine Learning*, 85(1):41–75, 2011. URL http://www.springerlink.com/index/U661126L07176787.pdf.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy Mccauley, M. Franklin, Scott Shenker, and Ion Stoica. Fast and interactive analytics over Hadoop data with Spark. *USENIX Login*, 37(4):45–51, 2012a. URL https://www.usenix.org/system/files/login/articles/zaharia.pdf?spm=5176.100239.blogcont37396.229.RGRYGj&file=zaharia.pdf.

Matei Zaharia, Mosharaf Chowdhury, Tathagata Das, Ankur Dave, Justin Ma, Murphy McCauley, Michael J. Franklin, Scott Shenker, and Ion Stoica. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing. In *Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation*, pages 2–2. USENIX Association, 2012b. URL http://dl.acm.org/citation.cfm?id=2228301.

Scott L. Zeger and Kung-Yee Liang. Longitudinal data analysis for discrete and continuous outcomes. *Biometrics*, pages 121–130, 1986. URL http://www.jstor.org/stable/2531248.

J. Zhou, J. Chen, and J. Ye. *MALSAR: Multi-tAsk Learning via StructurAl Regularization*. Arizona State University, 2011a. URL http://www.public.asu.edu/~jye02/Software/MALSAR.

Jiayu Zhou, Lei Yuan, Jun Liu, and Jieping Ye. A multi-task learning formulation for predicting disease progression. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 814–822. ACM, 2011b. URL http://dl.acm.org/citation.cfm?id=2020549.

Jiayu Zhou, Jianhui Chen, and Jieping Ye. Multi-task learning: Theory, algorithms, and applications. In *U RL https://www. siam. org/meetings/sdm12/zhou_ chen_ ye. pdf*, 2012. URL http://www.public.asu.edu/~jye02/Software/MALSAR/MTL-SDM12.pdf.

Jun Zhu, Ning Chen, and Eric P. Xing. Infinite latent SVM for classification and multi-task learning. In *Advances in neural information processing systems*, pages 1620–1628, 2011. URL http://papers.nips.cc/paper/4365-infinite-latent-svm-for-classification-and-multi-task-learning.

Martin Zinkevich, Markus Weimer, Lihong Li, and Alex J. Smola. Parallelized stochastic gradient descent. In *Advances in neural information processing systems*, pages 2595–2603, 2010. URL http://papers.nips.cc/paper/4006-parallelized-stochastic-gradient-descent.

Ricardo Ñanculef, Emanuele Frandi, Claudio Sartori, and Héctor Allende. A novel Frank–Wolfe algorithm. Analysis and applications to large-scale SVM training. *Information Sciences*, 285:66–99, November 2014. ISSN 0020-0255. doi: 10.1016/j.ins.2014.03.059. URL http://www.sciencedirect.com/science/article/pii/S0020025514003508.