Distributed Optimization for Non-Strongly Convex Regularizers

Master Thesis

Simone Forte

Wednesday 16\textsuperscript{th} September, 2015

Supervising Professor: Prof. Dr. Thomas Hofmann\textsuperscript{1}
Advisors: Dr. Martin Jaggi\textsuperscript{1}, Dr. Matthias Seeger\textsuperscript{2}, Virginia Smith\textsuperscript{3}

\textsuperscript{1} Department of Computer Science, ETH Zürich
\textsuperscript{2} Amazon Development Center Germany GmbH
\textsuperscript{3} Department of Computer Science, UC Berkley
Acknowledgements

I would like to thank Prof. Thomas Hofmann for the opportunity to work in his research group.
I would also like to express my deep gratitude to my advisors Dr. Martin Jaggi, Dr. Matthias Seeger and Virginia Smith for their indispensable guidance and advice and to Celestine Dünner for her useful suggestions.
Abstract

We develop primal-dual algorithms for distributed training of linear models in the Spark framework. We present the ProxCoCoA+ method which represents a generalization of the CoCoA+ algorithm and extends it to the case of general strongly convex regularizers. A primal-dual convergence rate analysis is provided along with an experimental evaluation of the algorithm on the problem of elastic net regularized logistic regression.

We also develop the PrimalCoCoA+ method, a method that allows certain non-strongly convex regularizers to be trained in the ProxCoCoA+ theoretical framework; the algorithm works under the assumption that this regularizers are linearly separable and box constrained. This allows for primal-dual convergence rates for $L_1$ regularized models, which are, to the best of our knowledge, the first of their kind; we also evaluate the practical efficiency of this method in the case of $L_1$ regularized logistic on two real world datasets.

Finally, we experimentally explore and prove the validity of ProxCoCoA+ Wild and PrimalCoCoA+ Wild, two new optimization methods that combine distributed and parallel optimization techniques and achieve significant speed-ups with respect to their non-wild variants.
## Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>1 Introduction</strong></td>
<td></td>
</tr>
<tr>
<td>1.1 Regularized Loss Minimization</td>
<td>1</td>
</tr>
<tr>
<td>1.1.1 Losses</td>
<td>1</td>
</tr>
<tr>
<td>1.1.2 Regularization</td>
<td>2</td>
</tr>
<tr>
<td>1.1.3 Gradient Descent</td>
<td>3</td>
</tr>
<tr>
<td>1.2 Distributed Optimization</td>
<td>4</td>
</tr>
<tr>
<td>1.2.1 The MapReduce Model</td>
<td>4</td>
</tr>
<tr>
<td>1.2.2 Apache Spark</td>
<td>5</td>
</tr>
<tr>
<td>1.2.3 Distributed Gradient Descent</td>
<td>5</td>
</tr>
<tr>
<td>1.3 Primal-Dual Methods</td>
<td>6</td>
</tr>
<tr>
<td>1.3.1 Convex Conjugates</td>
<td>6</td>
</tr>
<tr>
<td>1.3.2 The Dual Problem</td>
<td>7</td>
</tr>
<tr>
<td>1.3.3 Stochastic Dual Coordinate Ascent for L2 Regularization</td>
<td>8</td>
</tr>
<tr>
<td><strong>2 ProxCoCoA+</strong></td>
<td>10</td>
</tr>
<tr>
<td>2.1 Setting</td>
<td>11</td>
</tr>
<tr>
<td>2.2 Distributed Algorithmic Framework</td>
<td>11</td>
</tr>
<tr>
<td>2.2.1 Solving the Local Subproblems</td>
<td>15</td>
</tr>
<tr>
<td>2.3 Convergence Analysis for Lipschitz Losses</td>
<td>17</td>
</tr>
<tr>
<td>2.4 Convergence Analysis for Smooth Losses</td>
<td>18</td>
</tr>
<tr>
<td>2.5 Elastic Net</td>
<td>20</td>
</tr>
<tr>
<td><strong>3 Primal ProxCoCoA+</strong></td>
<td>21</td>
</tr>
<tr>
<td>3.1 Encoding Primal Problems in Dual Form</td>
<td>22</td>
</tr>
<tr>
<td>3.1.1 The Corresponding Primal Form</td>
<td>23</td>
</tr>
<tr>
<td>3.2 Convergence Analysis for L-Bounded Regularizers</td>
<td>24</td>
</tr>
<tr>
<td>3.3 Convergence Analysis for Strongly Convex Regularizers</td>
<td>25</td>
</tr>
<tr>
<td>3.4 Applications</td>
<td>27</td>
</tr>
</tbody>
</table>
The last several years have seen an enormous increase in the size and complexity of datasets across numerous domains. Social networks, search engines and electronic commerce websites, among many others, produce huge amounts of valuable data at an ever increasing rate, enabling machine learning applications hard to imagine only a few years ago. This has driven the creation and development of distributed computing infrastructures and frameworks, which are necessary in order to store and consume such a wealth of information in the most efficient way possible and requires us to adapt and rethink many common machine learning algorithms, whose often inherently sequential structure makes unfit to this new setting.

1.1 Regularized Loss Minimization

Many common methods in machine learning can be posed as a convex optimization problem having the following structure

$$\min_{w \in \mathbb{R}^d} \left[ \mathcal{P}(w) := g(w) + \sum_{i=1}^{n} \ell_i(x_i^T w) \right].$$

(1.1)

In this formulation, data are represented by $x_i$ vectors in $\mathbb{R}^d$ and we wish to learn a vector $w$ that minimizes the sum of convex loss functions $\ell_i$ and a regularization term $g$. Throughout this work we will refer to this formulation as a regularized loss minimization (RLM) problem.

1.1.1 Losses

Regression

A common application of this theoretical framework is linear regression. In this, given a data matrix $A := [x_1; \ldots; x_n] \in \mathbb{R}^{d \times n}$ with data examples as its
columns, and a label vector $\mathbf{y} \in \mathbb{R}^n$, we seek a linear function that interpolates this labels as "good as possible". The definition of "good" can be captured through several different convex loss functions with common choices being the squared error

$$\ell_i(x_i^T \mathbf{w}) = \frac{1}{2n}(x_i^T \mathbf{w} - y_i)^2$$

and the absolute error

$$\ell_i(x_i^T \mathbf{w}) = \frac{1}{2n}|x_i^T \mathbf{w} - y_i|.$$

### Support Vector Machines

Another widespread application of this is binary classification. The input data in this class of problems are represented by a data matrix $A := [\mathbf{x}_1; \ldots; \mathbf{x}_n] \in \mathbb{R}^{d \times n}$ and a label vector $\mathbf{y} \in \{-1, +1\}^n$ with loss functions $\ell_i$ parameterized by the $y_i$ entry. Among the most successful models in this field, we find the Support Vector Machines model [2]; this model is based on the simple of idea of finding a hyperplane with the highest possible margin, that is, maximizing the distance between the plane and the closest data points. This can be achieved by using the hinge loss function:

$$\ell_i(x_i^T \mathbf{w}) = \frac{1}{2n} \max (0, 1 - y_i x_i^T \mathbf{w}).$$

### Logistic Regression

Another common classification model is logistic regression; in this, although, instead of just trying to predict the most likely label, we seek to interpolate the probability of the labels, assuming that this probability can be expressed using the sigmoid function as

$$p(y_i = +1) = \frac{1}{1 + \exp(-x_i^T \mathbf{w})}.$$

Given a matrix $A$ and a set of observed labels $y_i$, it can be shown that the maximum likelihood solution for this problem is given by using

$$\ell_i(x_i^T \mathbf{w}) = \frac{1}{2n} \log (1 + \exp(-y_i x_i^T \mathbf{w}))$$

as a loss function.

### 1.1.2 Regularization

Widespread and critical in machine learning is the concept of model regularization; this refers to imposing constraints on the solution vector $\mathbf{w}$ in order to penalize too high coefficient values with the aim of controlling its complexity.
and avoid the overfitting problem.

In our framework we can achieve regularization by choosing an appropriate $g$ function with the most predominant choice being

$$g(w) = \frac{\lambda}{2} \|w\|^2_2.$$

Another common choice is $L_1$ regularization, which uses a

$$g(w) = \lambda \|w\|_1$$

regularizer. This has the property of giving very sparse models and thus easily interpretable $w$ weight vectors and it’s often used in combination with the squared error loss, with the resulting model going by the name of Lasso and having important applications in feature selection tasks [14]. Very relevant and with similar applications in feature selection is also the Elastic Net [19] regularizer, which is based on a convex combinations of the $L_1$ and $L_2$ norms.

### 1.1.3 Gradient Descent

In the case of differentiable loss and regularization terms - as for example in the logistic regression - a simple but often very effective strategy for RLM is using a gradient descent based approach on function (1.3). This class of algorithms work by constructing a sequence of vectors, $w_t$, which represent approximations of the optimal value of the function we wish to minimize, where the $w_{t+1}$ vector is obtained subtracting from it an approximation of the gradient of the previous iterate $w_t$ scaled by the step size $\eta_t$.

Several possible incarnations of gradient descent are possible; two commons ones are:

- **batch** gradient descent which, at every step, computes the exact gradient at the point $w_t$;

- **stochastic** gradient descent which approximates the gradient by using only one randomly chosen data point $x_t$.

The first algorithm will generally converge in a much smaller number of iterations than the stochastic gradient descent, but the much higher cost of every iteration makes it generally much slower to use in practice, making SGD the algorithm of choice for many RLM problems. Note here, that this is particularly true in the ”big data” setting that is of interest in this work.

### Choosing the Step Size

The sequence $\eta_t$ is generally chosen to be a decreasing function of $t$ with common choices being $\eta_t = 1/t$ or $\eta_t = 1/\sqrt{t}$ or even $\eta_t = \eta$ for some constant $\eta$ and needs to be tuned to achieve good performance. A $\eta_t$ that decreases too
1.2. Distributed Optimization

fast e.g. \(1/t\) can in fact severely slow down convergence while a bigger step can completely prevent the algorithm from converging. An alternative to defining a fixed \(\eta_t\) sequence, is to select a good step size at each step, using a line search algorithm; this approach is however limited to batch gradient descent algorithms, as a line search algorithm would need to compute the value of the whole objective function, making the complexity of every iteration at least linear in the size of the dataset.

<table>
<thead>
<tr>
<th>Algorithm 1 Batch Gradient Descent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: <strong>Input:</strong> Loss function (\ell_i), regularizer (g) and data matrix (A).</td>
</tr>
<tr>
<td>2: <strong>Output:</strong> (\arg\min_w P(w))</td>
</tr>
<tr>
<td>3: (w_0 := 0)</td>
</tr>
<tr>
<td>4: <strong>for</strong> (t = 1, 2, 3, \ldots) <strong>do</strong></td>
</tr>
<tr>
<td>5: (w_t = w_{t-1} - \eta_t \nabla P(w))</td>
</tr>
<tr>
<td>6: <strong>end for</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Algorithm 2 Stochastic Gradient Descent</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: <strong>Input:</strong> Loss function (\ell_i), regularizer (g) and data matrix (A).</td>
</tr>
<tr>
<td>2: <strong>Output:</strong> (\arg\min_w P(w))</td>
</tr>
<tr>
<td>3: (w_0 := 0)</td>
</tr>
<tr>
<td>4: <strong>for</strong> (t = 1, 2, 3, \ldots) <strong>do</strong></td>
</tr>
<tr>
<td>5: Pick a random data point (x_i)</td>
</tr>
<tr>
<td>6: (w_t = w_{t-1} - \eta_t \nabla [n\ell_i(x_i^T w) + g(w)])</td>
</tr>
<tr>
<td>7: <strong>end for</strong></td>
</tr>
</tbody>
</table>

1.2 Distributed Optimization

In this section we will look at some simple distributed optimization algorithms to better understand the power and limitations of this model of computation. When speaking about computation, especially distributed computation, it is important to specify the exact model we will be employing, as different models exist with different specifications and capabilities. In particular, in this work, we are interested in distributed computation according to the general MapReduce [4] programming paradigm, and will utilize Apache Spark [18], which is a popular instantiation of this.

1.2.1 The MapReduce Model

Presented in [4], MapReduce has been one of the first and most influential programming models for distributed computation. The model assumes that the data input to the computation are partitioned into several *splits* and spread
1.2. Distributed Optimization

across a set of machines; the data is also assumed to be a collection of key-value pairs. Computation is based on two simple operations inspired by the functional programming paradigm:

- **Map**: \((k_1, v_1) \rightarrow \text{list}(k_2, v_2)\): This operation, applied independently to all the input pairs, takes a key-value pair and maps it to a list of key-value pairs.

- **Reduce**: \((k_1, \text{list}(v_1)) \rightarrow \text{list}(v_2))\): Pairs from the map stage having the same key are then grouped together producing a set of lists with an associated key. Each of this lists are then independently fed to a reduce function which will finally output a list of values.

Despite its apparent simplicity the model is very powerful and many common algorithms can be formalized in it.

1.2.2 Apache Spark

Apache Spark is a framework for large-scale in-memory data processing [18] based on the MapReduce model. The model is based on the concept of Resilient Distributed Dataset (RDD) which can be thought of as collections of elements distributed over \(K\) partitions on \(M\) worker machines. In many scenarios the number of partitions in an RDD will correspond to the total number of cores collectively available on the machines. The Spark computational model supports several classes of operations on RDDs; these operations are orchestrated and always initiated by a driver machine. In this work we will mostly be concerned with the following three kinds of operations:

- **mapPartition**: This operation, takes the content of a partition and creates a new RDD obtained my mapping the content of each partition using some mapping function; we will assume that the resulting RDD keeps the original partitioning. A mapPartition, though still orchestrated by the driver, will only involve the worker machines in the actual computations and no data transfer will happen across machines.

- **reduce**: This aggregates the elements of an RDD by means of a commutative and associative operation (e.g. sum), returning the result of this aggregation back to the driver node.

- **broadcast**: This will be used to broadcast data from the master to all the machines; it is worth noticing that only one copy of the broadcast data is created per machine and it is thus shared across partitions present on the same machine.

1.2.3 Distributed Gradient Descent

We can now present a simple strategy to distribute a gradient descent algorithm for RLM that perfectly fits in the Spark computing system. We will thus
1.3 Primal-Dual Methods

assume that we are given a data matrix $A = [x_1, \ldots, x_n]$ partitioned column-wise over $K$ worker nodes. We will refer to the set of indices assigned to each partition $k \in [K]$ as $P_k$; in other words each partition $k$ will store data points $x_i$ for $\forall i \in P_k$. The simplest strategy to distribute Gradient Descent thus is to distribute over the computation of the gradient

$$\nabla P(w) = \nabla g(w) + \sum_{i=1}^{n} \nabla \ell_i(x_i^T w) = \nabla g(w) + \sum_{k=1}^{K} \sum_{i \in P_k} \nabla \ell_i(x_i^T w).$$

In order to compute this we will first of all broadcast the $w$ vector to all worker nodes. At this point, we can map every partition $P_k$ to the $\sum_{i \in P_k} \nabla \ell_i(x_i^T w)$ summation value; this only requires accessing data already available on the partition and the broadcast $w$ vector. We then sum the values returned by the partitions through a reduce operation and the master can now finally compute the gradient of the function by finally adding $\nabla g(w)$ to it. We can see all of this put together in Algorithm 3.

**Algorithm 3 Distributed Gradient Computation**

1: **Input:** $x_i$ vectors distributed over $K$ partitions, $w$ vector
2: **Output:** $\nabla P(w)$
3: Broadcast $w$ to all workers
4: for $k \in \{1, 2, \ldots, K\}$ in parallel over workers do
5: Compute $\nabla P_k(w) := \sum_{i \in P_k} \nabla \ell_i(x_i^T w)$
6: end for
7: Reduce $\nabla P'(w) = \sum_{k=1}^{K} P_k(w)$
8: return $\nabla P'(w) + \nabla g(w)$

1.3 Primal-Dual Methods

In this section we will now turn to primal-dual based methods applied to RLM problems. This will serve as a foundation for the rest of this work, as the main concepts related to primal-dual methods will be here introduced and explained.

1.3.1 Convex Conjugates

A fundamental concept that will appear all throughout this work is the convex conjugate.

**Definition 1.** The convex conjugate of a function $f : \mathbb{R}^d \to \mathbb{R}$ is defined as

$$f^*(v) := \max_{u \in \mathbb{R}^d} v^T u - f(u) \quad (1.2)$$
1.3. Primal-Dual Methods

1.3.2 The Dual Problem

As in the previous section, we will be looking at RLM problems, defined as
\[
\min_{w \in \mathbb{R}^d} \mathcal{P}(w) := g(w) + \sum_{i=1}^n \ell_i(x_i^T w).
\]  

(1.3)

The main idea behind primal-dual based methods though is to maximize a quadratic lowerbound of (1.3) instead, which can be defined as
\[
\max_{\alpha \in \mathbb{R}^n} \mathcal{D}(\alpha) := -g^*(A\alpha) - \sum_{i=1}^n \ell_i^*(-\alpha_i).
\]  

(1.4)

Here we will now show as this problem arises naturally from (1.3). Beginning with the primal formulation, we introduce variables \( z_i = w^T x_i \) for \( i = 1, \ldots, n \). The optimization problem (1.3) thus becomes:
\[
\min_{w \in \mathbb{R}^d} \quad g(w) + \sum_{i=1}^n \ell_i(z_i) \quad \text{such that} \quad z_i = x_i^T w
\]  

(1.5)

Introducing dual variables \( \alpha = [\alpha_1, \ldots, \alpha_n] \), the Lagrangian is given by:
\[
L(w, z; \alpha) := g(w) + \sum_{i=1}^n \ell_i(z_i) + \sum_{i=1}^n \alpha_i (z_i - x_i^T w).
\]

The dual problem follows by taking the infimum with respect to \( w \) and \( z \):
\[
\inf_{w, z} L(w, z; \alpha) = \inf_w \left\{ g(w) - (A\alpha)^T w \right\} + \sum_{i=1}^n \inf_{z_i} \left\{ \ell_i(z_i) + \alpha_i z_i \right\}
\]
\[
= -\sup_w \left\{ (A\alpha)^T w - g(w) \right\} - \sum_{i=1}^n \sup_{z_i} \left\{ -\alpha_i z_i - \ell_i(z_i) \right\}
\]
\[
= -g^*(A\alpha) - \sum_{i=1}^n \ell_i^*(-\alpha_i)
\]

arriving at the dual formulation (1.4). From the above derivation it is also clear that, since
\[
\min_{w \in \mathbb{R}^d} \mathcal{P}(w) = \max_{\alpha \in \mathbb{R}^n} \mathcal{D}(\alpha)
\]

then for every \( w \in \mathbb{R}^d \) and \( \alpha \in \mathbb{R}^n \)
\[
G(\alpha) = \mathcal{P}(w) - \mathcal{D}(\alpha)
\]  

(1.6)

will always be positive and an upper bound to the suboptimality of a \( w \) solution for the primal problem \( \mathcal{P} \). Given \( \alpha \in \mathbb{R}^n \), a corresponding primal vector \( w \in \mathbb{R}^d \) can be obtained as:
\[
w := w(\alpha) := \nabla g^*(v(\alpha)).
\]
1.3.3 Stochastic Dual Coordinate Ascent for L2 Regularization

In this section we will look at SDCA, a well known algorithm to maximize problem (1.4) in the simplest possible case of $g(w) = \|w\|^2_2 / 2$. We will only give a high level overview of the method, but an extensive analysis of it can be found in [12] and a very popular implementation in the open source LIBLINEAR [5] library. In order to get the dual problem for this case, we will first of all compute the convex conjugate of $g$; we thus need to solve

$$g^*(v) := \max_{u \in \mathbb{R}^d} v^T u - \frac{\|u\|^2_2}{2}. \quad (1.7)$$

Given the convexity of $v^T u - \|u\|^2_2$ we can compute its maximum by finding the root of its derivative

$$v - u = 0 \iff v = u$$

and replacing it in (1.7) getting $g^*(v) = \|v\|^2_2$.

We will now look at an actual SDCA based algorithm for solving this problem; the idea behind this class of methods is to iteratively choose a random index $i \in [n]$ and maximize problem (1.4) with respect to $\alpha_i$ keeping all the other coordinates fixed. In other words, for a fixed $i$, we want to solve

$$\max_{\Delta \alpha_i \in \mathbb{R}} D(\alpha + \Delta \alpha_i \xi_i) = \max_{\Delta \alpha_i \in \mathbb{R}} \left[ -\frac{1}{2} \|A\alpha + \Delta \alpha_i \xi_i\|^2_2 - \ell_i^* (-\alpha_i - \Delta \alpha_i) \right]$$

and replacing $A\alpha$ with $v(\alpha)$ we get:

$$\max_{\Delta \alpha_i \in \mathbb{R}} \left[ -\frac{1}{2} \|v(\alpha) + \Delta \alpha_i \xi_i\|^2_2 - \ell_i^* (-\alpha_i - \Delta \alpha_i) \right]. \quad (1.8)$$

More precisely, the resulting optimization algorithm will be Algorithm 4. One important thing to notice is that, for efficiency reasons, the $v(\alpha_i)$ iterate will be computed incrementally at each iteration, given the iterate $v(\alpha_{i-1})$, as

$$v(\alpha_i) = A\alpha_i = A(\alpha_{i-1} + \Delta \alpha_i \xi_i) = v(\alpha_{i-1}) + \Delta \alpha_i \xi_i.$$ 

The SDCA algorithm we just presented has several advantages over the similar SGD algorithm:

- The algorithm has no parameter to be tuned; in particular there is no step size to be chosen.

- It is possible to use the duality gap as a sound certificate of suboptimality and it can thus be used as a well defined stopping criterion for the algorithm.

- Despite having similar theoretical convergence rates [12], it is reported that it can be superior to SGD in $L_2$ regularized SVM problems.
Algorithm 4 Stochastic Dual Coordinate Ascent

1: **Input**: Loss functions $\ell_i$, data matrix $A$.
2: **Output**: $\arg\max_\alpha D(\alpha)$ and $v = v(\alpha) = \arg\min_\nu P(v)$
3: $\alpha(0) := 0$, $w(0) := 0$.
4: for $t = 1, 2, 3, \ldots$ do
5: Pick a random data point $x_i$
6: Find $\Delta\alpha$ maximizing (1.8)
7: $\alpha(t) = \alpha(t-1) + \Delta\alpha e_i$
8: $v(t) = v(t-1) + \Delta\alpha x_i$
9: end for
Chapter 2

ProxCoCoA+

In the previous chapter we looked at different sequential and distributed optimization algorithms. In particular, we saw how to apply gradient descent based algorithms to the general form of RLM problems in a distributed environment, but we only saw a sequential version of SDCA and on the specific setting of $L_2$ regularization. Achieving the same generality for the SDCA approach is in fact possible and will be the main focus of this chapter and a main contribution of this thesis.

Previous work on SDCA has already, in part, dealt with the problem of applying SDCA to more general settings, both in term of distribution and generalization to different regularizers; in particular two methods have been studied that tried to bridge the gap with gradient descent methods:

- **ProxSDCA**: The proxSDCA method [11]: a generalization of the algorithm 4. The basic idea underlying the method is to optimize, during a coordinate step, a quadratic lowerbound of the actual dual function; convergence results are proven for general strongly convex regularizers.

- **CoCoA+**: The CoCoA+ method [10]: a distributed and communication efficient primal-dual optimization method for $L_2$ regularization. The algorithm assumes that the data are partitioned across several machines and works by optimizing surrogate dual local subproblems for every partition and then combining the results by scaling the independent partition updates in a suitable way; the local optimizations can be performed using arbitrary local solvers. A special case of CoCoA+ is the CoCoA method [8] which, in its common instantiation using an SDCA based local solver, can be thought of as performing SDCA on the original problem independently for every partition and then averaging the results.

In this chapter we will combine ideas from both works in order to develop ProxCoCoA+, a general method for primal-dual distributed optimization of RLM problems with strongly convex regularizers.
2.1 Setting

In this section we will formally present the setting for this work. Similarly as in the previous chapter we will be looking at models of the form

\[
\min_{w \in \mathbb{R}^d} \left[ P(w) := g(w) + \sum_{i=1}^{n} \ell_i(x_i^T w) \right].
\]  

(2.1)

In addition to this, in order to prove the convergence results, we will assume that the functions \( \ell_i, i \in [n] \) are convex, and the function \( g(.) \) is \( \tau \)-strongly convex, with \( \tau > 0 \).

Data Partitioning

As in section 1.2.3 we will assume that the data matrix \( A = [x_1, \ldots, x_n] \) is partitioned column-wise in \( P_k \) partitions over \( K \) worker nodes; the size of each partition will be denoted as \( n_k = |P_k| \) and the data in each partition will be denoted as \( A_{[k]} \). In addition we will also have one dual variable \( \alpha_i \) per data point, and will represent them as \( \alpha = [\alpha_1, \ldots, \alpha_n] \) vectors; this vector will have the same partitioning of the \( A \) matrix, and we will indicate each partition \( k \) as \( \alpha_{[k]} \).

2.2 Distributed Algorithmic Framework

The CoCoA Special Case

We will now look at what is perhaps the simplest strategy to apply an SDCA type of algorithm to a distributed setting. The approach we will use is based on and generalizes the CoCoA algorithm presented in [8]; this will serve us to build an intuition for when, later on, we will generalize the problem further. As with the previously seen SDCA method, we will work on the dual formulation

\[
\max_{\alpha \in \mathbb{R}^n} \left[ D(\alpha) := -g^*(v(\alpha)) - \sum_{i=1}^{n} \ell_i^*(-\alpha_i) \right].
\]  

(2.2)

instead than on the primal one. The algorithm will proceed iteratively looking at each round \( t \) for a better \( \alpha^{(t+1)} \) solution for the dual optimization problem, starting from a previous solution \( \alpha^{(t)} \). More precisely, each partition will only be responsible for the dual variable corresponding to the data points that reside on that partition; we will indicate the subset of dual variables for a partition as \( \alpha_{[k]} \). We will then start the algorithm with \( \alpha_{[0]} = 0 \) and thus a corresponding \( \nu = v(\alpha) = 0 \). Independently for each partition we will then look for a \( \Delta \alpha_{[k]} \) vector that optimizes the problem on partition \( k \) only with respect to the dual variables on that partition. Formally, if \( \alpha^{(t)} \) is the current iterate, partition \( k \) would solve the following local subproblem
\[ \max_{\Delta \alpha[k] \in \mathbb{R}^n} - \sum_{i \in P_k} \ell_i^* \left( - (\alpha + \Delta \alpha[k], i) \right) - g^* \left( v(\alpha^{(t)}) + \Delta \alpha[k] \right). \] (2.3)

Solving this problem might however not be easy for complex \( g^* \) functions [11]. Therefore, instead of solving this problem directly we will – inspired by the proxSDCA approach [11] – maximize a quadratic lower bound of it. In order to achieve this we will notice that – because of \( \tau \)-strong convexity assumption on \( g \) and using Proposition 2 – \( g^* \) is \((1/\tau)\)-smooth. This means that, by Definition 17, we have:

\[ g^*(v(\alpha^{(t)} + \Delta \alpha[k])) \leq g^*(v(\alpha^{(t)}) + [\nabla g^*(v(\Delta \alpha[k]))]^T v(\alpha^{(t)}) + \frac{1}{2\tau} \| v(\Delta \alpha[k]) \|^2 \]

and therefore we have that

\[ - \sum_{i \in P_k} \ell_i^* \left( - (\alpha + \Delta \alpha[k], i) \right) - g^* \left( v(\alpha^{(t)}) - w(\alpha^{(t)})^T v(\alpha^{(t)}) - \frac{1}{2\tau} \| v(\Delta \alpha[k]) \|^2 \right) \leq - \sum_{i \in P_k} \ell_i^* \left( - (\alpha + \Delta \alpha[k], i) \right) - g^* \left( v(\alpha^{(t)} + \Delta \alpha[k]) \right), \]

where the left hand side of the equation is the lower-bound we will optimize on. By optimizing this problem for every partition we will thus get \( k \) solution vectors \([\Delta \alpha[k], \ldots, \Delta \alpha[k]] =: \Delta \alpha\). We can now conclude an iteration by computing the new vector iterate summing \( \alpha^{(t)} \) and the \( \Delta \alpha \) vector update rescaled by \( \frac{1}{K} \):

\[ \alpha^{(t+1)} = \alpha^{(t)} + \frac{1}{K} \Delta \alpha. \]

It is shown in [8] – and will again be shown in a more general setting in this work – that the \( \frac{1}{K} \) scaling is a safe choice to ensure that the algorithm will convergence to the correct solution of the dual problem.

**Generalizing the Local Subproblems**

Based on the derivation from the previous section we will now define a slightly different and more general local problem, generalizing the one defined for CoCoA+ [10]. Similarly as before, this problem, defined for each partition \( k \), will only directly depend on data that reside on it and on a \( v = v(\alpha) \) vector shared among all partitions. This vector will allow us to account, in a communication efficient way, for the state of other partitions. A new parameter \( \sigma' \) that we’ll refer to as complexity parameter will also appear in the formula; its role will though only be explored in later sections and only after having presented the complete ProxCoCoA+ algorithm. We thus define the local subproblems as follows:

\[ \max_{\Delta \alpha[k] \in \mathbb{R}^n} G_{k}^{\sigma'} \left( \Delta \alpha[k]; v, \alpha[k] \right) \] (2.4)
2.2. Distributed Algorithmic Framework

where
\[
G_k'(\Delta \alpha_{(k)}; v, \alpha_{(k)}) := - \sum_{i \in P_k} \ell_i^*(\alpha + \Delta \alpha_{(k)}), - \frac{1}{K} g^*(v) - w^T v(\Delta \alpha_{(k)}) - \frac{\sigma'}{2\tau} \|v(\Delta \alpha_{(k)})\|^2
\]

(2.5)

and \(w\) is defined as \(w := \nabla g^*(v)\). This subproblem is more general than the CoCoA+ subproblem [10]. In order to obtain a subproblem formulation analogous to the one presented in CoCoA+ we just have to define \(g(w) = \|w\|_2^2\); we therefore have that the strong convexity parameter is 1 and is \(\tau\)-strongly convex, and \(w = v\). The obtained formula would however not be identical as we have no scaling factor \(\frac{1}{n}\) in front of the loss functions and no factor \(\lambda\) that multiplies the regularizer. Later in this chapter, we will however see how to adapt ProxCoCoA+ to the case in which losses and regularizers are rescaled.

The Algorithm

We will now formally present Algorithm 5. The basic idea is similar to what we have seen for the CoCoA special case and it is based on decomposing the problem in partition local subproblems and solving them with respect to a subset of dual variables. These solutions are then summed up, after being scaled by a \(\gamma\) parameter. A detailed analysis of the algorithm dependent on its parameters will be the main focus of the following sections.

**Algorithm 5** ProxCoCoA+ Algorithm for Problem (2.1)

1: **Input:** Regularizer \(g(v)\), loss functions \(\ell_i\). Data matrix \(A\) distributed column-wise according to partitions \(\{P_k\}_{k=1}^K\). Aggregation parameter \(\gamma \in (0, 1]\), parameter \(\sigma'\).
   Initialize \(\alpha^{(0)} := 0 \in \mathbb{R}^n, v^{(0)} := 0 \in \mathbb{R}^d\).
2: for \(t = 0, 1, 2, \ldots\) do
3:   for \(k \in \{1, 2, \ldots, K\}\) in parallel over partitions do
4:     find a \(\Theta\)-approximate solution \(\Delta \alpha_{(k)}\) for the local subproblem (2.5)
5:       using an arbitrary local solver
6:     update \(\alpha^{(t+1)}_{(k)} := \alpha^{(t)}_{(k)} + \gamma \Delta \alpha_{(k)}\)
7:   end for
8: reduce \(v^{(t+1)} := v^{(t)} + \gamma\sum_{k=1}^K \Delta v_k\)
9: end for

The Aggregation and Complexity Parameters

As we has seen in the previous sections, Algorithm 5 depends on two important parameters \(\gamma\) and \(\sigma'\). The aggregation parameter \(\gamma\), in particular, allows
2.2. Distributed Algorithmic Framework

the algorithm to interpolate between an averaging based and an adding based approach. The $\sigma'$ parameter instead serves as a measure of difficulty of partitioning and has to be chosen in a way that is dependent on the correlation between the partitions and on the aggregation strategy of the algorithm as determined by $\gamma$.

In order to clarify the role of this parameter we will now look at two $\sigma'$ choices for the $L_2$ regularized case (in this case we can assume $\tau = 1$):

- $\sigma' = 1$: in this case the local problem becomes

  $$\max_{\Delta \alpha_{[k]} \in \mathbb{R}^n} - \sum_{i \in P_k} \ell_i^* (- (\alpha + \Delta \alpha_{[k]})_i) - \frac{1}{2K} \|v\|^2 - w^T v(\Delta \alpha_{[k]}) - \frac{1}{2} \|v(\Delta \alpha_{[k]})\|^2$$

  Being the maximization problem independent of $v$ and $w = v$ we can get rid of the $1/K$ factor, obtaining the following equivalent maximization problem

  $$\max_{\Delta \alpha_{[k]} \in \mathbb{R}^n} - \sum_{i \in P_k} \ell_i^* (- (\alpha + \Delta \alpha_{[k]})_i) - \frac{1}{2} \|v + v(\Delta \alpha_{[k]})\|^2.$$  

  We can easily notice that this maximization problem is equivalent to maximizing the original problem for the $\alpha_{[k]}$ subset of dual variables. As we will prove in the following section, and as previously seen, a safe choice to combine the subproblem solutions in this case is to use a parameter $\gamma = \frac{1}{K}$ that is equivalent to a simple averaging of the primal solutions.

- $\sigma' = K$: in this case the local problem becomes

  $$\max_{\Delta \alpha_{[k]} \in \mathbb{R}^n} - \sum_{i \in P_k} \ell_i^* (- (\alpha + \Delta \alpha_{[k]})_i) - \frac{1}{2K} \|v\|^2 - w^T v(\Delta \alpha_{[k]}) - \frac{1}{2} \|v(\Delta \alpha_{[k]})\|^2$$

  Multiplying everything by $K$ we get that the previous is equivalent to the following

  $$\max_{\Delta \alpha_{[k]} \in \mathbb{R}^n} - \sum_{i \in P_k} \ell_i^* (- (\alpha + \Delta \alpha_{[k]})_i) - \frac{1}{2} \|v\|^2 - K v^T v(\Delta \alpha_{[k]}) - \frac{K^2}{2} \|v(\Delta \alpha_{[k]})\|^2 =$$

  $$\max_{\Delta \alpha_{[k]} \in \mathbb{R}^n} - \sum_{i \in P_k} \ell_i^* (- (\alpha + \Delta \alpha_{[k]})_i) - \frac{1}{2} \|v + K v(\Delta \alpha_{[k]})\|^2.$$

  Differently from the $\sigma' = 1$ case, we are in this case explicitly accounting for the contributions of the other machines by scaling the $v(\Delta \alpha_{[k]})$ vector by the number of partitions $K$. In other words we are looking for a solution to the optimization problem on the local partition by assuming that the other partitions will find a similar $v(\Delta \alpha_{[k]})$ parameter vector.
2.2. Distributed Algorithmic Framework

More generally and similarly as for the original CoCoA+ method, we will make the following assumption on the $\gamma$ and $\sigma'$ parameters.

**Assumption 1.** The $\gamma$ and $\sigma'$ parameters must be such that $\gamma \in [0, 1]$ and

\[
\sigma' \geq \sigma'_{\text{min}} := \gamma \max_{\alpha \in \mathbb{R}^n} \frac{\|A\alpha\|^2}{\sum_{k=1}^{K} \|A\alpha[k]\|^2}.
\]

(2.6)

Given this assumption we can in fact present and prove Lemma 2 – which is the direct equivalent of [10, Lemma 4] – which illustrates how different choices of aggregation and complexity parameters affects the gain that we get when combining different subproblems.

**Lemma 2.** Given for any $\alpha, \Delta \alpha \in \mathbb{R}^n$, $v = v(\alpha)$ and real values $\gamma, \sigma'$ as in Assumption 1, it holds that

\[
D\left(\alpha + \gamma \sum_{k=1}^{K} \Delta \alpha[k]\right) \geq (1 - \gamma)D(\alpha) + \gamma \sum_{k=1}^{K} G_{\sigma'}^k(\Delta \alpha[k]; v, \alpha[k]),
\]

(2.7)

Choosing the complexity parameter $\sigma'$ to be exactly $\sigma'_{\text{min}}$ as in (2.6) is though in general not always possible; however as shown in [10] it always holds that

\[
\sigma'_{\text{min}} \leq \gamma K.
\]

(2.8)

The value $\gamma K$ can thus be used as a safe value for $\sigma'$ in the algorithm.

For later use we will also define the following quantity:

**Definition 3 ([10, defined in Lemma 6]).**

\[
\sigma_k := \max_{\alpha[k] \in \mathbb{R}^n} \frac{\|A\alpha[k]\|^2}{\|\alpha[k]\|^2}.
\]

(2.9)

### 2.2.1 Solving the Local Subproblems

As seen, Algorithm 5 presented in the previous section does not specify or impose any specific method to optimize the local subproblems $G_{\sigma'}^k$. In fact, as in [10] we will make relatively weak assumptions about such solvers:

**Assumption 2 (\(\Theta\)-approximate solution).** We will require that a local solver always returns a $\Theta \in [0, 1)$ approximate solution; that is, whenever called and for $\forall k \in [K], \sigma', w, \alpha[k]$ it produces a (possibly) randomized approximate solution $\Delta \alpha[k]$, which satisfies

\[
\mathbb{E}\left[G_{\sigma'}^k(\Delta \alpha^*_k; w, \alpha[k]) - G_{\sigma'}^k(\Delta \alpha[k]; w, \alpha[k])\right] \\
\leq \Theta \left(G_{\sigma'}^k(\Delta \alpha^*_k; w, \alpha[k]) - G_{\sigma'}^k(0; w, \alpha[k])\right),
\]

(2.10)
where
\[
\Delta \alpha^*_{[k]} \in \arg \max_{\Delta \alpha \in \mathbb{R}^n} G'_k(\Delta \alpha_{[k]}; w, \alpha_{[k]}) \quad \forall k \in [K] \tag{2.11}
\]

By looking at the structure of the local subproblems (2.5), we can see that a very natural choice for a local solver, would be a Stochastic Dual Coordinate Ascent based algorithm, similar to Algorithm 4 – this is however just a possible choice we’ll present for ease of understanding, since any local solver conforming to Assumption 2 would be suitable. In order to solve this we immediately notice that we can leave out the \( g^*(v) \) term as it is independent of the \( \Delta \alpha_{[k]} \) variables. We therefore get that
\[
\max_{\Delta \alpha \in \mathbb{R}^n} G'_k(\Delta \alpha_{[k]}; v, \alpha_{[k]}) =
\]
\[
\max_{\Delta \alpha_{[k]} \in \mathbb{R}^n} \left[ -\sum_{i \in \mathcal{P}_k} \ell_i^*(\alpha_i - \Delta \alpha) - w^T v(\Delta \alpha_{[k]}) - \frac{\sigma'}{2\tau} \| v(\Delta \alpha_{[k]}) \|^2 \right].
\]

Similarly as in Algorithm 4, an SDCA algorithm for this problem will work by iteratively choosing a random index \( i \in \mathcal{P}_k \) and maximize the problem with respect to the \( i \)-th coordinate. That is, for a fixed \( i \) we get:
\[
\max_{\Delta \alpha \in \mathbb{R}^n} \left[ -\ell_i^*(\alpha_i - \Delta \alpha) - w^T v(\Delta \alpha_{[k]}) - \frac{\sigma'}{2\tau} \| v(\Delta \alpha_{[k]}) \|^2 \right]
\]
that we obtained by leaving out the \( \ell_i \) that do not depend on \( \Delta \alpha \). By expanding \( v(\Delta \alpha_{[k]}) \) we now get
\[
-\max_{\Delta \alpha \in \mathbb{R}} \left[ \ell_i^*(-\alpha_i - \Delta \alpha) - w^T (v(\Delta \alpha_{[k]}) + \Delta \alpha x_i) - \frac{\sigma'}{2\tau} \| v(\Delta \alpha_{[k]}) + \Delta \alpha x_i \|^2 \right],
\]
and by again leaving out terms independent of \( \Delta \alpha \) we arrive at
\[
-\max_{\Delta \alpha \in \mathbb{R}} \left[ \ell_i^*(-\alpha_i - \Delta \alpha) - \Delta \alpha w^T x_i - \frac{\sigma'}{2\tau} \| \Delta \alpha x_i \|^2 \right] =
\]
\[
\max_{\Delta \alpha \in \mathbb{R}} \left[ -\ell_i^*(-\alpha_i - \Delta \alpha) - \Delta \alpha x_i^T (w + \frac{\sigma'}{\tau} v(\Delta \alpha_{[k]})) - \frac{\sigma'}{2\tau} \| \Delta \alpha x_i \|^2 \right]. \tag{2.12}
\]

Just as in Algorithm 4, the \( \Delta v_{[k]} := v(\Delta \alpha_{[k]}) \) term will be updated at every local iteration in a similar way (as shown in Algorithm 6).
2.3 Convergence Analysis for Lipschitz Losses

Finally, we can give the following theorem for the general case of Lipschitz continuous losses.

**Theorem 4** (Generalization of [10, Theorem 8] for general $g(\cdot)$ functions). Let $L$-Lipschitz continuous $\ell_i(\cdot)$ functions, a $\tau$-strongly convex $g$ function and an arbitrary local solver satisfying Assumption 2 be given. Let $\gamma, \sigma'$ be defined as in Assumption 1. Then, if $\epsilon_G > 0$ is the desired duality gap, after $T$ iterations of Algorithm 5 with

\[
T \geq T_0 + \max\left\{ \frac{1}{\gamma(1-\Theta)}, \frac{4L^2\sigma'\gamma(1-\Theta)}{\tau\epsilon_G\gamma(1-\Theta)} \right\},
\]

\[
T_0 \geq t_0 + \left( \frac{2}{\gamma(1-\Theta)} \left( \frac{8L^2\sigma'\gamma}{\tau\epsilon_G} - 1 \right) \right)_+,\n\]

\[
t_0 \geq \max(0, \left[ \frac{1}{\gamma(1-\Theta)} \log\left( \frac{2(\mathcal{D}(\alpha^*) - \mathcal{D}(\alpha^{(0)}))}{4L^2\sigma'\gamma} \right) \right]),
\]

the expected duality gap will satisfy

\[
\mathbb{E}[\mathcal{P}(w(\bar{\alpha})) - \mathcal{D}(\bar{\alpha})] \leq \epsilon_G,
\]

at the averaged iterate

\[
\bar{\alpha} := \frac{1}{T-T_0} \sum_{t=T_0+1}^{T-1} \alpha^{(t)}
\]

(2.14)

In order to compare this to previous work on distributed convex optimization we will now look at the more common RLM structure, that is:

\[
\max_{w \in \mathbb{R}^d} \frac{1}{n} \sum_{i=1}^{n} \ell_i^*(w^T x_i) + \lambda g^*(w)
\]
2.4 Convergence Analysis for Smooth Losses

with the assumption that the $\ell'_i$ are $L'$-Lipschitz continuous and the function $g'(\mathbf{w})$ is 1-strongly convex. It is quite easy to see that by defining loss functions

$$\ell_i(w^T x_i) = \frac{1}{n} \ell'_i(w^T x_i)$$

and regularizer

$$g(w) = \lambda g'(w)$$

the problem

$$\max_{\mathbf{w} \in \mathbb{R}^d} \sum_{i=1}^{n} \ell_i(w^T x_i) + \lambda g(w)$$

now fits in the required form. In particular we now have that the $\ell_i$ functions are ($\frac{L'}{n}$)-Lipschitz and $g$ is $\lambda$-strongly convex. This is sufficient to prove Corollary 5 which exactly matches the convergence rates for the CoCoA+ method.

**Corollary 5.** Let $L'$-Lipschitz continuous $\ell'_i(\cdot)$ functions, a 1-strongly convex $g'(\cdot)$ function and an arbitrary local solver satisfying Assumption 2 be given. Let $\gamma, \sigma'$ be defined as in Assumption 1. Then, if $\epsilon_G > 0$ is the desired duality gap, after $T$ iterations of Algorithm 5 with

$$T \geq T_0 + \max\left\{ \frac{1}{\gamma(1 - \Theta)}, \frac{4L'^2\sigma'\gamma}{\lambda n^2 \epsilon_G (1 - \Theta)} \right\},$$

$$T_0 \geq t_0 + \left( \frac{2}{\gamma(1 - \Theta)} \left( \frac{8L'^2\sigma'\gamma}{\lambda n^2 \epsilon_G} - 1 \right) \right),$$

$$t_0 \geq \max(0, \left\lceil \frac{1}{\gamma(1 - \Theta)} \log\left( \frac{2\lambda n^2 (D(\alpha^*) - D(\alpha(0)))}{4L'^2\sigma'} \right) \right\rceil),$$

the expected duality gap will satisfy

$$\mathbb{E}[P(\mathbf{w}(\overline{\alpha})) - D(\overline{\alpha})] \leq \epsilon_G,$$

at the averaged iterate

$$\overline{\alpha} := \frac{1}{T - T_0} \sum_{t=T_0+1}^{T} \alpha^{(t)}. \quad (2.16)$$

2.4 Convergence Analysis for Smooth Losses

The analysis in the previous chapter applies to general continuous Lipschitz continuous losses. If the function is smooth we can however show that we can achieve a better convergence rate.

**Theorem 6** (Generalization of [10, Theorem 10] for general $g(.)$ functions). Let $(1/\mu)$-smooth $\ell_i(\cdot)$ functions, a $\tau$-strongly convex $g$ function and an arbitrary local solver satisfying Assumption 2 be given. Let $\gamma, \sigma'$ be defined as in
Assumption 1 and define $\sigma_{\text{max}} = \max_{k \in [K]} \sigma_k$. Then, if $\epsilon_D > 0$ is the desired maximum dual suboptimality, after $T$ iterations of Algorithm 5 with

$$T \geq \frac{1}{\gamma(1-\Theta)} \frac{\mu' n \lambda + \sigma_{\text{max}} \sigma'}{\mu' n \lambda} \log \frac{1}{\epsilon_D},$$

it holds that

$$\mathbb{E}[D(\alpha^*) - D(\alpha^{(T)})] \leq \epsilon_D.$$ 

Furthermore, given the desired duality gap $\epsilon_G$, after $T$ iterations with

$$T \geq \frac{1}{\gamma(1-\Theta)} \frac{\mu' n \lambda + \sigma_{\text{max}} \sigma'}{\mu' n \lambda} \log \left( \frac{1}{\gamma(1-\Theta)} \frac{1}{\epsilon_G} \right),$$

the expected duality gap is such that

$$\mathbb{E}[P(w(\alpha^{(T)})) - D(\alpha^{(T)})] \leq \epsilon_G.$$ 

**Proof.** The proof of this closely follows the one of [10, Theorem 10].

Similarly as what we have done for Lipschitz continuous losses, we can also present the following lemma that gives us a convergence rate for the more common case

$$\max_{w \in \mathbb{R}^d} \sum_{i=1}^n \frac{1}{n} \ell'_i(w^T x_i) + \lambda g'(w)$$

with the assumption that the $\ell'_i$ are $(1/\tau')$-smooth continuous and the function $g'(w)$ is 1-strongly convex. We easily see that the losses $\frac{1}{n} \ell'_i(w^T x_i)$ are $(1/(\mu'n))$ smooth and $\lambda g$ regularizer is $\lambda$-strongly convex. This is enough to prove Lemma 7 which matches the corresponding convergence rate from [10].

**Corollary 7.** Let $(1/\mu')$-smooth $\ell'_i(\cdot)$ functions, a $\tau'$-strongly convex $g'$ function and an arbitrary local solver satisfying Assumption 2 be given. Let $\gamma, \sigma'$ be defined as in Assumption 1 and define $\sigma_{\text{max}} = \max_{k \in [K]} \sigma_k$. Then, if $\epsilon_D > 0$ is the desired maximum dual suboptimality, after $T$ iterations of Algorithm 5 with

$$T \geq \frac{1}{\gamma(1-\Theta)} \frac{\mu' n \lambda + \sigma_{\text{max}} \sigma'}{\mu' n \lambda} \log \frac{1}{\epsilon_D},$$

it holds that

$$\mathbb{E}[D(\alpha^*) - D(\alpha^{(T)})] \leq \epsilon_D.$$ 

Furthermore, given the desired duality gap $\epsilon_G$, after $T$ iterations with

$$T \geq \frac{1}{\gamma(1-\Theta)} \frac{\mu' n \lambda + \sigma_{\text{max}} \sigma'}{\mu' n \lambda} \log \left( \frac{1}{\gamma(1-\Theta)} \frac{1}{\epsilon_G} \right),$$

the expected duality gap is such that

$$\mathbb{E}[P(w(\alpha^{(T)})) - D(\alpha^{(T)})] \leq \epsilon_G.$$
A very natural and immediate application of the framework we developed in this chapter is given by Elastic Net regularization; we will, however, only consider the case when it does not degenerate to $L_1$ regularization, as we have seen that a strong convex regularizer is a necessary requirement for primal-dual methods. Something similar to what we’ll see, was also done in the [11] paper, however for a non-distributed settings and applied to the problem of approximating $L_1$ regularization. Formally, we will therefore be looking at models with a regularization term having the following form:

$$h(w) = \frac{\eta}{2} \|w\|^2_2 + \eta \|w\|_1.$$ 

with $\eta > 0$.

**Lemma 8.** For $\eta \in (0, 1]$, the convex conjugate of the elastic net function

$$f(\alpha) := \frac{\eta}{2} \alpha^2 + (1 - \eta)|\alpha|$$

is

$$f^*(x) := \frac{1}{2\eta}(\lfloor |x| - (1 - \eta) \rfloor^+ + \eta)^2,$$

where $[.]^+$ is the positive part operator, $[s]^+ = s$ for $s > 0$, and zero otherwise. Also, the conjugate $f^{**}$ of $f^*$ is equal to $f$.

More generally, just like we did for $L_2$ regularization, we can also allow a scaling factor $\lambda$, thus giving us a regularizer

$$g(w) = \lambda h(w).$$

Given a convex loss function $\ell_i$, and depending on its Lipschitz $L$-continuity or $(1/\mu)$-smoothness we can now quite easily apply any of the convergence results proven in this chapter to an Elastic Net regularized model by simply noticing that the strong convexity parameter $\tau$ of $g$ would in this case simply be $\lambda \eta$. 

---

20
Primal ProxCoCoA+

In the previous chapter, we have developed a framework capable of handling problems of primal form

$$\min_{\mathbf{w} \in \mathbb{R}^d} \left[ \mathcal{P}(\mathbf{w}) := g(\mathbf{w}) + \sum_{i=1}^{n} \ell_i(x_i^T \mathbf{w}) \right]$$

with the assumption of $g$ being $\tau$-strongly convex with $\tau > 0$. The main application of this framework that we have explored has been Elastic Net. However, we faced an important limitation: as the relative contribution of the $L_1$ term with respect to the $L_2$ term became bigger, the problem complexity increased asymptotically, allowing us to only approximate $L_1$ regularized problems. The main problem we were facing was that as we approached $L_1$, the problem became less and less strongly-convex which is a necessary requirement for applying ProxCoCoA+. To overcome this, we will need a complete change of perspective: whereas traditional approaches were based on optimizing a problem that fit into the primal form and exploits the dual mapping as a mere algorithmic tool, here we will look at problems that fit directly into the dual form and optimize on it directly. We will call this method Primal ProxCoCoA+ or just PrimalCoCoA+ since the resulting algorithm will run on the primal of the original problem and the whole optimization problem will be distributed according to the primal coordinates, not the dual ones as in the previous sections. In summary, the roles of primal and dual problems will be completely swapped. The main idea behind the method will be to encode problems that are typically solved as primal problems in a dual formulation, suitable for the ProxCoCoA+ algorithm, and then use the ProxCoCoA+ algorithm to compute the actual solution.
3.1 Encoding Primal Problems in Dual Form

In this section we will now look at how to encode typical primal problems in a form that is solvable directly in the dual. The models we are interested in solving are again problems in RLM-dual form and have therefore the following formulation:

\[
\max_{\alpha \in \mathbb{R}^n} \left[ D(\alpha) := -\sum_{j=1}^{d} \phi^*_j(y_j^T\alpha) - r^*(\alpha) \right]
\]  

(3.2)

where the training examples \( y_j \) are the rows of an \( A := [y_1; \ldots; y_d]^T = [x_1; \ldots; x_n] \in \mathbb{R}^{d \times n} \) data matrix - it is important to keep in mind here that training examples are put as rows of the data matrix, instead of being columns as it is more commonly the case. We will also assume that the \( \phi^*_i \) functions are \((1/\tau)\)-smooth. Clearly, doesn’t look exactly like the standard dual formulation (1.4); in order to achieve this we will have to make the additional assumption of \( r^* \) being separable i.e. expressable as a sum of functions each depending on one coordinate. This is not a very restrictive assumption, as most regularizers - \( L_2, L_1 \) or elastic net - have in fact this form. We thus assume that \( r^* \) has the following form

\[
r^*(\alpha) = \sum_{i=1}^{n} \ell_i^*(-\alpha_i).
\]

As for the remaining part of the equation, we just need to adjust the notation. Using the now familiar primal-dual mapping notation

\[
v = v(\alpha) = A\alpha
\]

we can define the following function

\[
g^*(v) := \sum_{j=1}^{d} \phi^*_j(v_j) = \sum_{j=1}^{d} \phi^*_j((A\alpha)_j) = \sum_{j=1}^{d} \phi^*_j(y_j^T\alpha)
\]

thus getting

\[
\max_{\alpha \in \mathbb{R}^n} \left[ D(\alpha) := -g^*(v) - \sum_{i=1}^{n} \ell_i^*(-\alpha_i) \right].
\]  

(3.3)

We will later see that, under the smoothness assumption of the \( \phi^*_i \) function, we can prove that the primal corresponding to this formulation has a strongly convex \( g \) regularizer. We will also later see assumptions on the \( \ell_i^* \) that will give us either Lipschitz continuous or smooth \( \ell_i \) functions, thus fulfilling the conditions for applying ProxCoCoA+.
3.1.Encoding Primal Problems in Dual Form

3.1.1 The Corresponding Primal Form

In this section we will now work on deriving the primal form of (3.1). This formula is actually not entirely necessary to optimize on (3.1), as we are encoding and solving our models directly in a dual type of formulation and can be ignored in a purely implementative setting. However this form will come in handy to compute the duality gap, which allows us to have a certificate of how far we are from an optimal solution. This will also be useful as an analysis tool as we will be able to reuse most of the results in the previous section to study the convergence rate of the duality gap which in turns gives us also a convergence rate for the dual problem only. In order to do this we need to be able to compute the conjugate of

\[ g^*(v) := \sum_{j=1}^{d} \phi_j^*(v_j). \]

To get the primal form we need to find a \( g \), such that \( g^* \) is its convex conjugate. As it turns out this form is simply

\[ g(w) = \sum_{j=1}^{d} \phi_j(w_j). \]

We can see this by applying the definition of convex conjugate for \( g \), exploiting the fact that the coordinate are, in this case, all independent from each other:

\[
\begin{align*}
  g^*(v) &= \max_{w \in \mathbb{R}^d} [v^T w - g(w)] = \max_{w \in \mathbb{R}^d} \sum_{j=1}^{d} (v_j w_j - \phi_j(w_j)) \\
  &= \sum_{j=1}^{d} \max_{w_j \in \mathbb{R}} (v_j w_j - \phi_j(w_j)) = \sum_{j=1}^{d} \phi_j^*(v_j).
\end{align*}
\]

We can now write the primal of \( D(\alpha) \) as

\[ P(w) = g(w) + \sum_{i=1}^{n} \ell_i(w^T x_i). \]

Finally, we notice that, given the \((1/\tau)\)-smoothness assumption on the \( \phi_j^* \) functions, the \( g \) we have just derived is \( \tau \)-strongly convex (from Proposition 2). Having derived its primal form, we can now trivially present an algorithm for optimizing equation (3.1) which simply works by reducing the problem to a ProxCoCoA+ solvable problem. Algorithm 7 precisely reflects what we have just seen in this section; the only additional point to notice is that Algorithm 7 takes as input an \( A \in \mathbb{R}^{n \times d} \) matrix having the data examples as its columns – as it is generally more commonly the case in machine learning – and only then transpose it to get the actual matrix \( A \) that we have used when defining the problem.
3.2 Convergence Analysis for $L$-Bounded Regularizers

Algorithm 7: PrimalCoCoA+ Algorithm for primal form (2.1)

1: **Input:** Data matrix $\hat{\mathbf{A}} = [\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_d] \in \mathbb{R}^{n \times d}$; Regularizer $r^*(\alpha) = \sum_{i=1}^{n} \ell_i^*(\alpha_i)$, loss functions $\phi_i^*$. Aggregation parameter $\gamma \in (0, 1]$, subproblem parameter $\sigma'$ for the local subproblems
2: Compute matrix $A = \hat{\mathbf{A}}^T$
3: Define function $g^*(v) := \sum_{j=1}^{d} \phi_j^*(v_j)$
4: Run Algorithm 5 on matrix $A$, with $g := (g^*)^*$, $\ell_i = (\ell_i^*)^*$, aggregation parameter $\gamma$ and subproblem parameter $\sigma'$.
5: **Output:** The $\alpha$ vector as computed by Algorithm 5

3.2 Convergence Analysis for $L$-Bounded Regularizers

In this section we will now analyze the convergence of Algorithm 7 on the previously defined problem

$$\max_{\alpha \in \mathbb{R}^n} \left[ D(\alpha) := -\sum_{j=1}^{d} \phi_j^*(y_j^T \alpha) - r^*(\alpha) \right]. \quad (3.4)$$

in the case of $r^*(\alpha) := \sum_{i=1}^{n} \ell_i^*(-\alpha_i)$ being such that the $\ell_i^*$ functions are $L$-bounded.

**Definition 9** ($L$-bounded function). A function $f : \mathbb{R}^d \to \mathbb{R}$ is called $L$-bounded, if

$$f(u) \geq +\infty \quad \forall u : \|u\|_2 \geq L \quad (3.5)$$

As previously mentioned, we will also work under the assumption that the the $\phi_j^*$ functions are $(1/\tau)$-smooth, so that $g$ is $\tau$-strongly convex; also, by Proposition 3 we have that the convex conjugates the $\ell_i$ functions are $L$-Lipschitz. This allows us to state Corollary 10.

**Corollary 10.** Let $\phi_j^*(\cdot)$ be $(1/\tau)$-smooth, $r^*(\alpha) := \sum_{i=1}^{n} \ell_i^*(-\alpha_i)$ with the $\ell_i^*(\cdot)$ functions being $L$-bounded. Let $\gamma, \sigma'$ be as in Assumption 1 and an arbitrary local solver satisfying Assumption 2 be given. Then, if $\epsilon_G > 0$ is the desired duality gap, after $T$ iterations of Algorithm 7 with

$$T \geq T_0 + \max\{\left[\frac{1}{\gamma(1-\Theta)} \right]^2, \frac{4L^2\sigma\sigma'}{\tau\epsilon_G \gamma(1-\Theta)}\}, \quad (3.6)$$

$$T_0 \geq t_0 + \left(\frac{2}{\gamma(1-\Theta)} \left(\frac{8L^2\sigma\sigma'}{\tau\epsilon_G} - 1\right)\right)_+,$$

$$t_0 \geq \max(0, \left[\frac{1}{\gamma(1-\Theta)} \log\left(\frac{2(D(\alpha^*) - D(\alpha^{(0)})}{4L^2\sigma\sigma'}\right)\right]),$$

the expected duality gap will satisfy

$$\mathbb{E}[P(\mathbf{w}(\mathbf{a})) - D(\mathbf{a})] \leq \epsilon_G,$$
3.3. Convergence Analysis for Strongly Convex Regularizers

at the averaged iterate

$$\bar{\alpha} := \frac{1}{T-T_0} \sum_{t=T_0+1}^{T-1} \alpha^{(t)}. \quad (3.7)$$

Similarly to the previous chapter, and in order to compare this result to the result from other works [10, 11] we are now also interested in looking at the convergence rate for the case in which the optimization problem takes the following more common form:

$$\max_{\alpha \in \mathbb{R}^n} \left[ D(\alpha) := -\frac{1}{d} \sum_{j=1}^{d} \phi_j^*(y_j^T \alpha) - \lambda r^*(\alpha) \right]. \quad (3.8)$$

with $\lambda > 0$, $\phi_j^*(\cdot)$ being $(1/\tau')$-smooth and $r^*(\alpha) := \sum_{i=1}^{n} \ell_i^*(\cdot)$ being such that the $\ell_i^*(\cdot)$ functions are $L'$-bounded. Very easily we can now see that by defining $\phi_j^* = \frac{1}{d} \phi_j^*$ and $\ell_i^* = \lambda \ell_i^*(\cdot)$ (with $r^*(\alpha) := \sum_{i=1}^{n} \ell_i^*(-\alpha_i)$) we can reduce our problem to (3.4). This also gives us that $\phi_j^*$ are now $(1/(d\tau'))$-smooth while the $\ell_i^*$ clearly stay $L'$-bounded; we can therefore finally state Corollary 11.

**Corollary 11.** Let $\phi_j^*(\cdot)$ be $(1/\tau')$-smooth, $r^*(\alpha) := \sum_{i=1}^{n} \ell_i^*(-\alpha_i)$ with the $\ell_i^*(\cdot)$ functions being $L'$-bounded. Let $\gamma, \sigma'$ be as in Assumption 1 and an arbitrary local solver satisfying Assumption 2 be given. Then, if $\epsilon_G > 0$ is the desired duality gap, after $T$ iterations of Algorithm 7 with

$$T \geq T_0 + \max\left\{ \frac{1}{\gamma(1-\Theta)}, \frac{4L'^2\sigma'}{d\tau'\epsilon_G\gamma(1-\Theta)} \right\}, \quad (3.9)$$

$$T_0 \geq t_0 + \left( \frac{2}{\gamma(1-\Theta)} \left( \frac{8L'^2\sigma'}{d\tau'\epsilon_G} - 1 \right) \right)^+, \quad$$

$$t_0 \geq \max(0, \left[ \frac{1}{\gamma(1-\Theta)} \log\left( \frac{2d\tau'(D(\alpha^*)-D(\alpha^{(0)}))}{4L'^2\sigma'} \right) \right]),$$

the expected duality gap will satisfy

$$\mathbb{E}[P(w(\bar{\alpha})) - D(\bar{\alpha})] \leq \epsilon_G,$$

at the averaged iterate

$$\bar{\alpha} := \frac{1}{T-T_0} \sum_{t=T_0+1}^{T-1} \alpha^{(t)}. \quad (3.10)$$

### 3.3 Convergence Analysis for Strongly Convex Regularizers

In this section we will instead analyze the convergence of Algorithm 7 in the case of strongly convex regularizers; in order to do so, we will apply
the convergence analysis for smooth losses developed in the previous chapter. More formally we will assume a model such as (3.4) but with \( \mu \)-strongly convex \( \ell_i^* \) functions. This gives us, from Proposition 2, that the \( \ell_i \) functions are \((1/\mu)\) smooth. We can thus easily state Corollary 12.

**Corollary 12.** Let \( \phi_j^* (\cdot) \) be \((1/\tau)\)-smooth, \( r^* (\alpha) := \sum_{i=1}^n \ell_i^* (-\alpha_i) \) with the \( \ell_i^* (\cdot) \) functions being \( \mu \)-strongly convex. Let \( \gamma, \sigma \) be defined as in Assumption 1 and define \( \sigma_{\text{max}} = \max_{k \in [K]} \sigma_k \). Then, if \( \epsilon_D > 0 \) is the desired maximum dual suboptimality, after \( T \) iterations of Algorithm 7 with

\[
T \geq \frac{1}{\gamma (1 - \Theta)} \frac{\mu + \sigma_{\text{max}} \sigma'}{\mu^*} \log \frac{1}{\epsilon_D},
\]

it holds that

\[
\mathbb{E} [D(\alpha^*) - D(\alpha^{(T)})] \leq \epsilon_D.
\]

Furthermore, given the desired duality gap \( \epsilon_G \), after \( T \) iterations with

\[
T \geq \frac{1}{\gamma (1 - \Theta)} \frac{\mu + \sigma_{\text{max}} \sigma'}{\mu^*} \log \left( \frac{1}{\gamma (1 - \Theta)} \frac{\mu + \sigma_{\text{max}} \sigma'}{\mu^*} \frac{1}{\epsilon_D} \right),
\]

the expected duality gap is such that

\[
\mathbb{E} [\mathcal{P}(w(\alpha^{(T)})) - D(\alpha^{(T)})] \leq \epsilon_G.
\]

As done for the \( L \)-bounded case, we will also look at the averaged loss case as in equation (3.8) but under the \( \mu' \)-strong convexity assumption of the \( \ell_i' \) functions; as in the previous section we thus define \( \ell_i = \lambda \ell_i' \) getting that the \( \ell_i \) are now \( \lambda \mu' \)-strongly convex.

**Corollary 13.** Let \( \phi_j' (\cdot) \) be \((1/\tau')\)-smooth, \( r'^* (\alpha) := \sum_{i=1}^n \ell_i'^* (-\alpha_i) \) with the \( \ell_i'^* (\cdot) \) functions being \( \mu' \)-strongly convex. Let \( \gamma', \sigma' \) be defined as in Assumption 1 and define \( \sigma_{\text{max}} = \max_{k \in [K]} \sigma_k \). Then, if \( \epsilon_D > 0 \) is the desired maximum dual suboptimality, after \( T \) iterations of Algorithm 7 with

\[
T \geq \frac{1}{\gamma (1 - \Theta)} \frac{\lambda \mu \mu' + \sigma_{\text{max}} \sigma'}{\lambda \mu' \mu^*} \log \frac{1}{\epsilon_D},
\]

it holds that

\[
\mathbb{E} [D(\alpha^*) - D(\alpha^{(T)})] \leq \epsilon_D.
\]

Furthermore, given the desired duality gap \( \epsilon_G \), after \( T \) iterations with

\[
T \geq \frac{1}{\gamma (1 - \Theta)} \frac{\lambda \mu \mu' + \sigma_{\text{max}} \sigma'}{\lambda \mu' \mu^*} \log \left( \frac{1}{\gamma (1 - \Theta)} \frac{\lambda \mu \mu' + \sigma_{\text{max}} \sigma'}{\lambda \mu' \mu^*} \frac{1}{\epsilon_D} \right),
\]

the expected duality gap is such that

\[
\mathbb{E} [\mathcal{P}(w(\alpha^{(T)})) - D(\alpha^{(T)})] \leq \epsilon_G.
\]
3.4 Applications

3.4.1 $L_1$ regularization

In order to achieve $L_1$ regularization, we’ll simply have to define the $\ell^*_i$ functions as

$$\ell^*_i(\alpha_i) = |\alpha_i|$$

where this gets us the dual problem

$$\max_{\alpha \in \mathbb{R}^n} \left[ D(\alpha) := -\frac{1}{d} \sum_{j=1}^d \phi_j^*(y_j^T\alpha) - \lambda \|\alpha\|_1 \right]$$

(3.11)

which should be easy to recognize as a typical $L_1$-RLM problem. The function $\ell_i$ that has $|\cdot|$ as its conjugate is the indicator of the unit interval

$$\ell_i(x) := \begin{cases} 0 & : x \in [-1,1] \\ +\infty & : \text{otherwise} \end{cases}.$$  

(3.12)

As this $\ell$ function (we will drop the $i$ index from now on) is neither Lipschitz continuous nor smooth, it won’t be possible to immediately apply the convergence results proved in the previous chapter. A possible solution to this problem is to add a small $L_2$ regularization term to the $L_1$ term as done in [11], thus using elastic net to approximate $L_1$ regularization; this approach has however the drawback of destroying the sparsity property of the norm. The alternative approach that we take here is to replace the $L_1$ function with an $L$-bounded surrogate, which preserves an almost identical solution space as the original problem and preserves the sparsity property. More specifically, we will modify the function $|\cdot|$ by imposing an additional constraint, which is only active outside the region of interest. Formally, we replace $\ell^*(\cdot) = |\cdot|$ with

$$\bar{\ell}^*(\alpha) := \begin{cases} |\alpha| & : \alpha \in [-B,B] \\ +\infty & : \text{otherwise} \end{cases}.$$

Lemma 14. The convex conjugate of $\bar{\ell}^*$ as defined above is

$$\bar{\ell}(x) = \begin{cases} 0 & : x \in [-1,1] \\ B(|x| - 1) & : \text{otherwise} \end{cases},$$

and is $B$-Lipschitz.

With this modified $L_1$-regularizer, the optimization problem becomes

$$\max_{\alpha \in \mathbb{R}^n} -\frac{1}{d} \sum_{i=1}^n \bar{\ell}^*(\alpha_i) - \lambda g^*(v(\alpha)) .$$

(3.13)

It is not hard to see that for a large enough choice of the value $B$, this problems yields the same solution as the original Lasso objective. An easy way to derive
3.4. Applications

A big enough value for $B$ is as follows. We first of all notice that, since we start the algorithm with $\alpha = 0$, if we use a monotonic optimization algorithm - e.g., SDCA, then all the $\alpha$ iterates encountered during the execution of the algorithm will never have an objective value worse than $D(0)$. Formally, this means that for every $i$:

$$\frac{|\alpha_i|}{d} \leq D(0) = \lambda g^*(0) \implies |\alpha_i| \leq d\lambda g^*(0)$$

and this $d\lambda g^*(0)$ can be used as a safe value for $B$; this value will be safe in the sense that the modified problem will be guaranteed to have the same optimal solution(s) as the original problem.\footnote{Uniqueness of solution does not necessarily hold.}

It is worth noticing that this trick only affects the convergence theory, in that it allows us to present a strong primal-dual theory for Lasso and other $L_1$-regularized problems. This modification of $\ell_i$, however, does not affect the algorithms for the original problems whenever a monotone optimizer is used, as we will never encounter solutions that violate the $B$ constraint; we can therefore run the algorithm on the original unmodified problem.

To finally get a primal dual convergence rate for $L_1$ regularized models we can simply apply Corollary 11 with $L'$ equal to $d\lambda g^*(0)$.

3.4.2 Elastic Net

The same approach used for $L_1$ regularization can also be applied to Elastic Net regularized models. The approach is very similar and simply requires us to define $\ell_i^*$ functions

$$\ell_i^*(\alpha_i) = \frac{\eta}{2}\alpha_i^2 + (1 - \eta)|\alpha_i|$$

that will act as per-coordinate regularizers.

**Lemma’ 8.** For $\eta \in (0, 1]$, the convex conjugate of the elastic net function

$$f(\alpha) := \frac{\eta}{2}\alpha^2 + (1 - \eta)|\alpha|$$

is

$$f^*(x) := \frac{1}{2\eta^2}([|x| - (1 - \eta)]_+)^2,$$

where $[.]_+$ is the positive part operator, $[s]_+ = s$ for $s > 0$, and zero otherwise. Also, the conjugate $f^{**}$ of $f^*$ is equal to $f$.

Here, in contrast to the $L_1$ case above, we have that since the Elastic Net function is $\eta$-strongly convex we don’t need to go through defining an $L$-bounded surrogate; all we need to do to obtain a convergence rate result is applying Corollary 13 with the strong convexity parameter $\mu'$ of the $\ell_i^*$ functions equal to $\eta$.\footnote{Uniqueness of solution does not necessarily hold.}
3.5 Related Work

In this section we will briefly look at some of the existing work related to the PrimalCoCoA+ algorithm when applied to $L_1$ regularization as Section 3.4.1. The recent approach of [15] for sparse logistic regression is closely related to our method. Inspired by glmnet and [17], [15] they use a block-diagonal Hessian upper approximation. On the theory side, its analysis [15, 17] is not an explicit convergence rate but only asymptotic, as the quadratic upper bounds are not explicitly controlled for safety as with our $\sigma'$. [15, 17] assume the quadratic subproblems to be solved exactly. In contrast, we do allow arbitrary weak accuracy $\Theta$ of the local solver, allowing trade-off between communication and computation.

Shotgun (parallel stochastic coordinate descent, for the case of $L_1$-regularizers) was proposed in [3], and is among the state-of-the-art solvers in the parallel setting. Our framework reduces to shotgun as a special case when the internal solver is one single coordinate update on the subproblem (2.5), $\gamma = 1$, and a suitable $\sigma'$. However, shotgun is not covered by our convergence theory, since it uses a potentially unsafe upper bound $\beta$ instead of $\sigma'$, not necessarily satisfying condition (2.6).

Parallel coordinate descent on the $L_1$-primal objective directly has recently been analyzed in [6, 13], but not in the communication-efficient setting.
In this chapter we will now finally look at how the methods developed and analyzed in the previous chapters actually perform in practice. In particular we will be looking at the problem of Logistic Regression with Elastic Net regularization (included the case in which it degenerates to $L_1$ regularization) and compare several different approaches on different regularization parameters. Formally, given $n$ datapoints $x_i$ we will therefore be solving a logistic regression classification problem with target labels $y_i$, by means of optimizing the following function:

$$\min_{w \in \mathbb{R}^d} \left[ \mathcal{P}(w) := \lambda g(w) + \frac{1}{n} \sum_{i=1}^{n} \ell_i(x_i^T w) \right],$$

where the loss functions are defined as

$$\ell_i(x_i^T w) = \log (1 + y_i x_i^T w)$$

and the regularizer $g$ is defined as

$$g(w) = \eta \|w\|_2^2 + (1 - \eta) \|w\|_1.$$

We will also assume that our data are distributed on $K$ partitions.

### 4.1 Experimental Infrastructure

The experimental infrastructure will be based on the MapReduce computational model, with all implementations written in Spark. We will therefore assume that we have $M$ machines each having $C$ cores each, thus giving us a total of $K := M \cdot C$ partitions. More precisely all experiments have been carried out in an Amazon EC2 cluster composed of 1 master and 8 worker machines of type $m3.2xlarge$, all of which having 8 cores and 30GB of main memory each.
4.2 Datasets

All the experiments will be run on the following classification datasets:

- **RCV1** [9]: The RCV1 classification dataset is a text classification dataset obtained by preprocessing the RCV1 category annotated collection of news articles. The number of datapoints is 677,399 with a total number of 47,236 features. The sparsity of this dataset is 0.16%.

- **webspam** [16]: The webspam dataset is a binary classification dataset where every data point is constructed by taking trigrams of a document and it has a spam/non-spam label. It contains a total of 350,000 datapoints and 16,609,143 features, of which only 680,715 are nonzero. This dataset is much sparser than the previous one, with a sparsity of 0.02%.

4.3 Compared Algorithms

Before showing any results, we will now list the methods we will be comparing, quickly reiterating on some of the methods we presented in previous sections and presenting some new ones.

4.3.1 Proximal CoCoA+ (ProxCoCoA+)

By ProxCoCoA+ in experiments we will refer to Algorithm 5 with data $x_i$ distributed column-wise over the $K$ partitions. The number of partitions $K$ will be $M \cdot C$, where $M$ is the number of machines and $C$ is the number of cores per machine; more precisely, in our case will therefore be equal to 64. It is important to notice here that this method will only be applicable to the Elastic Net regularized linear model with $\eta > 0$, since strong convexity is required for the method to work. The additive version of the algorithm will be used by setting $\gamma = 1$; the $\sigma'$ value will instead be set to the corresponding safe value $K$ as it can be derived from (2.8). Depending on the actual $\sigma'_{\text{min}}$ value, $\sigma'$ could be set to bigger values, thus yielding better performances; this is often especially true on very sparse datasets. More aggressive choice will however not be explored, as it goes beyond the scope of this work.

As a local solver we will use a variant of the SDCA local solver presented as Algorithm 6; the algorithm will differ in that the data points will be sampled without replacement i.e. the algorithm iterates on the data points following a random permutation of the data matrix columns $[x_1, \ldots, x_n]$ and does exactly one pass over the data.

4.3.2 Primal Proximal CoCoA+ (PrimalCoCoA+)

This refers to Algorithm 7 with local solver and parameters $\gamma$ and $\sigma'$ as in Section 4.3.1. The partitioning scheme will also be the same as in Section 4.3.1.
4.3. Compared Algorithms

The problem we will be looking at will therefore be \(-D(\alpha)\) with \(D\) defined as

\[
D(\alpha) := -\frac{1}{n} \sum_{j=1}^{d} \phi^*_j(y_j^T \alpha) - \lambda r^*(\alpha)
\]

(4.2)

with

\[
\phi^*_i(x_i^T \alpha) = \log(1 + y_i x_i^T \alpha)
\]

and

\[
r^*(\alpha) = \eta \|\alpha\|_2^2 + (1 - \eta)\|\alpha\|_1
\]

which is clearly equivalent to the primal formulation (4.1). It is however crucial to understand how this precisely relates to Algorithm 7, by noticing that the \(A\) matrix will in this case be equal to \(A\) (as defined in this section) and thus in \(\mathbb{R}^{d \times n}\) (and not in \(\mathbb{R}^{n \times d}\)) hence also why there is a \(\frac{1}{n}\) term (instead of \(\frac{1}{d}\)). The time required by Algorithm 7 to transpose matrix \(A\) will not be accounted in the running time of the algorithm. It is worth noting that this transposition could be avoided by storing the data directly in a row-wise format.

4.3.3 Orthant-Wise Limited-Memory Quasi-Newton Optimizer (OWLQN)

The Orthant-Wise Limited-memory Quasi-Newton Optimizer [1] is a Quasi-Newton method based on the L-BFGS method for training of L1-regularized models. As other descent based methods, when minimizing a function \(f(w)\), OWLQN will compute a sequence of solution vectors \(w(t)\) as

\[
w(t+1) = w(t) - \gamma [Hf(w(t))]^{-1} \nabla f(w(t))
\]

where \(Hf(w(t))\) is an approximation of the true Hessian matrix of \(f\) at \(w(t)\). As for other Quasi-Newton methods this approximation will be computed by only using first order information as given by the history of the last \(m\) gradient evaluations. Furthermore \(Hf(w(t))\) won’t be computed explicitly - as its size is quadratic in the size of the \(w\) vector, but a search direction will rather be implicitly computed. The step size \(\gamma\) will then be chosen though a line search method.

Thus, given a problem like (4.1) the distribution of the algorithm will happen by simply computing this gradient in a distributed fashion as in Algorithm 3 (both for the gradient computation and for the line search); the computation of the search direction using the gradient history will instead happen entirely on the master. The algorithm thus strongly resembles the Batch Gradient Descent algorithm except that the search direction will be found using a smarter strategy.

4.3.4 Wild ProxCoCoA+ and PrimalCoCoA+

In section 4.3.1 we saw a set up for the ProxCoCoA+ method where we used \(K = M \cdot C\), i.e. every machine has 1 partition per each core. However, as
4.3. Compared Algorithms

the number of partitions increases the number of iterations required by the ProxCoCoA+ method tends to increase and it is thus sensible to try to keep the number of partitions to the smallest possible number, that is, the number of machines in the cluster $M$. A better approach could therefore be to only have one partition per machine, thus $K = M$, and then use a parallel local solver in order to solve the local subproblem defined on the partition, thus exploiting all the $C$ cores available on the machine. The main difference with what we did in section 4.3.1 is therefore that we now have that the $C$ threads on every machine are working together to solve the same subproblem more efficiently, instead of each of them solving a different subproblem independently. By using a parallel SDCA method, this results in a cost per iteration identical to the setup from section 4.3.1 but with $K = M$ as opposed to $K = M \cdot C$ and thus possibly with a faster convergence.

PASSCoDe

The PASSCoDe wild algorithm, presented in [7] is a simple SDCA parallel variant. The algorithm assumes a machine with $C$ cores, all having access to a shared main memory where the $A = [x_1, \ldots, x_n]$ data matrix is stored and accessible to every thread in its entirety. In the case of a model such as (4.1) with $L_2$ regularization, every thread of execution will have access to all the $x_i$ data points and will simply execute a procedure identical to Algorithm 4 on them; the only difference will be that the $w$ vector will be unique and shared between the $C$ threads. Different synchronization rules for updating the shared $w$ are presented in [7], but we will only explore an adaptation of the Wild variant, in which no synchronization happens between threads. This means that dirty and lost updates are possible and that the $w = A\alpha$ relation will hold only approximately.

Wild Local Solver

In this section we now present Algorithm 8, a parallel local solver based on the PASSCoDe algorithm. This algorithm, though very similar, has two main differences from the original PASSCoDe algorithm:

- Each thread $c \in \{1, \ldots, C\}$ won’t have access to the entire partition data matrix $A_{[k]}$, but only to a subpartition $A_{[k][c]}$ of partition $A_{[k]}$; we will also refer with $P_{k,c} \in P_k$ to the the subpartition indices. This modification is necessary to better fit the Spark computational model and, though formally different from the original PASSCoDe we’ll see from the experiments that is still works quite well in practice. The algorithm still retains the original ”wild” nature since the weight vector $w$ is shared among the threads.

- Although the PASSCoDe algorithm as presented in [7], only supported $L_2$ regularization, we will nonetheless apply to our more general strongly
4.3. Compared Algorithms

... convex regularizer setting. Experiments seem to confirm that the algorithm should support this setting too.

As the number of partitions $K$ can now be assumed to be $M$, a safe choice when $\gamma = 1$, will be $\sigma' = M - \alpha$ instead of $\sigma' = C \cdot M$.

Later in the experiments, we will refer to the ProxCoCoA+ method using the wild local solver (and with $\sigma' = M$ and $\gamma = 1$) as the ProxCoCoA+Wild algorithm. Also, when the ProxCoCoA+Wild is used in Algorithm 7, we will refer to the resulting algorithm as PrimalCoCoA+Wild.

Algorithm 8 Wild SDCA Local Solver

1: Input: Loss functions $\ell_i$, data matrix $A_k$, $\sigma' > 0$, $v \in \mathbb{R}^d$, $\alpha[k] \in \mathbb{R}^{nk}$
2: $w(0) = \nabla g^*(v)$
3: $\alpha[0] := \alpha[k]$
4: for each thread $c \in \{1, \ldots, C\}$ in parallel do
5: Pick a random permutation $R_{k,c}$ of the subpartition indices $P_{k,c}$.
6: for $i \in R_k$ do
7: $\Delta\alpha = \arg \max_{\Delta\alpha} \ell_i^* (-\alpha_i - \Delta\alpha) - \Delta\alpha x_i^T w(t-1) - \frac{\sigma'}{2}\Delta\alpha^2 ||x_i||^2$
8: $\alpha(t) = \alpha(t-1) + \Delta\alpha e_i$
9: $w(t) = w(t-1) + \frac{\sigma'}{2}\Delta\alpha x_i$
10: end for
11: end for
12: return $\Delta\alpha[k] := \alpha^{(n)}[k] - \alpha[k]$

Implementation Details

As seen in the previous section, we are assuming that our data are partitioned into $M$ partitions and each of this partitions has $C$ subpartitions each. The reason for this choice is due to some limitations of the Spark model. The algorithm was in fact implemented by partitioning the RDD containing the data points into $C \cdot M$ partitions and each machine was assigned by Spark exactly $C$ of this partitions and one Spark task was run for each of this partitions; this $C$ partitions are what we called subpartitions in the previous section. In order for this partitions to share a common $w$ weight vector, we then used the broadcast construct to broadcast the weight vector to each machine; the semantics of this operation will therefore make sure that only one copy of the vector is present per machine. All tasks will then write on this same vector concurrently. It is important to mention that, although the Spark computational model is based on immutability and tasks are not supposed to modify the content of broadcasted data, this is however possible when mutable data e.g. a standard java array are broadcast. We warn that this is not the intended usage of the broadcast construct and could be considered a hack and...
the Spark fault-tolerance guarantees might not hold as strongly anymore as a result. This however works very well in practice and will serve us well at least as a proof-of-concept implementation.

4.4 Experiments

We can now finally look at the results we achieved in our experiments on logistic regression problems for elastic net and l1 regularization. All the results will be presented in graphs having time in seconds on the x-axis and primal suboptimality on the y-axis; this will be computed as the difference between the current objective value and an approximation of the optimal one. This approximation is obtained as the minimum objective value found by any of the methods.

4.4.1 Elastic Net Regularization

In this section we will now look at a comparison of several methods on an Elastic Net regularized logistic regression problem with $\eta = \frac{1}{2}$. From experiments on both webspam and rcv1 and for both $\lambda = 10^{-4}$ and $\lambda = 10^{-6}$ (Figure 4.1, 4.2, 4.3 and 4.4) we can immediately see that the two clear winners are the proxCoCoA+ – along with its wild variant – and the OWLQN algorithm. The performances of primalCoCoA+ are instead sharply inferior to the other two methods. A superior performance of proxCoCoA+ method can also be observed when using bigger values of $\lambda$ and it is excepted behavior and consistent with the convergence rate formula.

Worth noticing is also the contribution of the wild modification to the performance of the proxCoCoA+Wild and primalCoCoA+Wild algorithms. Especially in the latter case, we can see that the wild approach makes the algorithm substantially better for both values of $\lambda$. In the case of proxCoCoA+, however, even though a speed-up is still provided by the algorithm during the initial rounds, the objective value experiences an oscillating behavior once it reaches a high enough accuracy; this behavior seems consistent with the convergence result from [7] for PASSCoDe-Wild, which states that the algorithm does not convergence to the actual solution of the optimization problem but to a solution with a perturbed regularizer.

Comparing the experiments on the two different datasets, we can also notice that ProxCoCoA+ performs much better than OWLQN on webspam while being almost on par on the rcv1 dataset; we believe this can be attributed to the higher sparsity of the webspam dataset which would tend to favor an SDCA approach.
4.4.2 \( L_1 \) Regularization

In this section we look instead at \( L_1 \) regularized logistic regression; as previously indicated, we will not use algorithm ProxCoCoA+ and its wild variant on this problem as the \( L_1 \) norm is not a strongly convex function. As opposed to the previous section, in this case we observe a very sharp difference between the webspam and the rcv1 experiments. In fact, from the experiments in Figure 4.7 and 4.8 on rcv1, we see a much better performance of the OWLQN algorithm both against PrimalCoCoA+ and PrimalCoCoA+Wild; the picture looks however very different when the algorithm is run on the webspam datasets (Figure 4.5 and Figure 4.6) where can see that PrimalCoCoA+ Wild appears to be substantially faster than OWLQN in the initial phases though still being outperformed by OWLQN in the long run; this means that PrimalCoCoA+ Wild could therefore still be a better choice than OWLQN in scenarios where very high accuracy is not required and thus especially in machine learning, where it’s known that very high training accuracy is often not required for a good test accuracy.
4.4. Experiments

Figure 4.1: webspam - $\lambda = 1.0E - 4, \eta = 0.5$

Figure 4.2: webspam - $\lambda = 1.0E - 6, \eta = 0.5$
4.4. Experiments

Figure 4.3: $\text{rcv1} - \lambda = 1.0 \times 10^{-4}, \eta = 0.5$

Figure 4.4: $\text{rcv1} - \lambda = 1.0 \times 10^{-6}, \eta = 0.5$
4.4. Experiments

Figure 4.5: webspam - $\lambda = 1.0E - 4, \eta = 0.0$

Figure 4.6: webspam - $\lambda = 1.0E - 6, \eta = 0.0$
4.4. Experiments

Figure 4.7: rcv1 - $\lambda = 1.0 \times 10^{-4}, \eta = 0.0$

Figure 4.8: rcv1 - $\lambda = 1.0 \times 10^{-6}, \eta = 0.0$
Chapter 5

Conclusion

In this work we developed primal-dual algorithms for regularized loss minimization in a distributed setting. We started by presenting ProxCoCoA+, a generalization of the CoCoA+ [10] algorithm that is able to handle general strongly convex regularizers and we saw convergence results analogous to the the one presented in [10], both for the case of smooth and Lipschitz continuous losses. This allowed us to apply the ProxCoCoA+ method to elastic net regularized problems; in particular we saw experiments for the logistic regression loss, on which the method proved to be very effective and competitive with the OWLQN algorithm. We then showed how the ProxCoCoA+ method is able to handle non-strongly convex regularizers, by means of reversing the role of primal and dual problems; this resulted in the PrimalCoCoA+ method, a method capable of solving RLM-problems directly using their primal formulation and represents a major contribution of this work. This method allows us to obtain primal-dual convergence rates for $L_1$ regularized problems and this result is, to the best of our knowledge, the first of its kind. Experiments on $L_1$ regularized logistic regression showed that this method is competitive, in certain settings, with the more complex OWLQN algorithm. We finally saw an experimental study for a new parallel SDCA local solver, based on PASSCoDe-Wild [7], which allows the ProxCoCoA+ and PrimalCoCoA+ method to exploit the benefits of parallel computation between threads running on the same machine and gave very promising results.

5.1 Future Work

This work opens some possible lines of future research:

- The PrimalCoCoA+ method we developed, because of its reduction-based nature, only handles separable regularizers. It would be worth investigating whether a more general method could be developed capable of handling non-separable regularizers.
• The Wild SDCA local solver presented in the experiments’ section, though based on the PASSCoDe Wild algorithm, was applied to the more general strongly convex regularizer setting and it proved successful. However, in [7] we only see a proof of convergence of the algorithm on $L_2$ regularized problems; studying its behavior, from a theoretical standpoint, on this more general setting would thus be important to investigate.
Chapter 6

Internship At Amazon

As part of my thesis, I also worked in the Amazon Development Center in Berlin from June 1, 2015 to August 15, 2015.
The work produced during my internship there is currently not included in this document.
Definition 15 (\(\mu\)-strong convexity). A function \(g : \mathbb{R}^d \to \mathbb{R}\) is called \(\mu\)-strongly convex, for \(\mu \geq 0\), if
\[
g(u) \geq g(w) + \langle \nabla g(w), u - w \rangle + \frac{\mu}{2} \|u - w\|^2 \quad \forall u, w \in \mathbb{R}^d. \tag{A.1}
\]

Definition 16 (\(L\)-Lipschitz continuity). A function \(f : \mathbb{R}^d \to \mathbb{R}\) is called \(L\)-Lipschitz continuous, if
\[
|f(u) - f(v)| \leq L \|u - v\|_2 \quad \forall u, v \in \mathbb{R}^d \tag{A.2}
\]

Definition 17 ((1/\(\mu\))-smoothness). A function \(f : \mathbb{R}^d \to \mathbb{R}\) is called (1/\(\mu\))-smooth if its differentiable and its gradient is (1/\(\mu\))-Lipschitz continuous. An equivalent condition for smoothness (from [11]) is that:
\[
f(u) \leq f(v) + \nabla f(v)^T (u - v) + \frac{1}{2\mu} \|u - v\|_2^2 \quad \forall u, v \in \mathbb{R}^d
\]

Proposition 1 (Fenchel–Young inequality). Given a function \(f : \mathbb{R}^d \to \mathbb{R}\) and its convex conjugate \(f^* : \mathbb{R}^d \to \mathbb{R}\) it holds that
\[
x^T y \leq f(x) + f^*(y) \quad \forall x, y \in \mathbb{R}^d
\]

Proposition 2. The conjugate \(f^*\) of a (1/\(\mu\))-smooth \(f\) function for \(\mu > 0\) is \(\mu\)-strongly convex, and vice versa, the conjugate \(f^*\) of a \(\mu\)-strongly convex \(f\) function for \(\mu > 0\) is (1/\(\mu\))-smooth. [11]

Proposition 3. The conjugate \(f^* : \mathbb{R} \to \mathbb{R}\) of an \(L\)-bounded function \(f : \mathbb{R} \to \mathbb{R}\) function is \(L\)-Lipschitz.

Proof. In order to prove \(L\)-Lipschitzness we need to upper bound the following term; we start by applying the definition of convex conjugate and noticing that
\[ f^*(u) - f^*(v) = \sup_{|x| \leq L} |ux - f(x) - f^*(v) - f^*(v)| \]

\[ \left(\frac{1}{|x|} \right) \leq L|x| \leq L|u - v| \]

\[ |f^*(u) - f^*(v)| = \left| \sup_{|x| \leq L} (ux - f(x) - f^*(v)) \right| \leq L|u - v| \]

\[ A = -\sum_{k=1}^{K} \left( \sum_{i \in P_k} \ell_i^*(-\alpha_i - \gamma(\Delta \alpha_{[k]})) \right) \]

\[ = -\sum_{k=1}^{K} \left( \sum_{i \in P_k} \ell_i^*(-(1 - \gamma)\alpha_i - \gamma(\alpha + \Delta \alpha_{[k]})) \right) \]

\[ \geq -\sum_{k=1}^{K} \left( \sum_{i \in P_k} (1 - \gamma)\ell_i^*(-\alpha_i) + \gamma \ell_i^*(-(\alpha + \Delta \alpha_{[k]})) \right) \]
where the last inequality is obtained by applying Jensen’s inequality. We then proceed at bounding $B$ exploiting the $(1/\tau)$-smoothness of $g^*$ and Assumption 2.6 on $\sigma'$:

$B = g^*(v(\alpha + \gamma \sum_{k=1}^{K} \Delta \alpha_{[k]})) = g^*(v(\alpha) + \gamma \sum_{k=1}^{K} v(\Delta \alpha_{[k]}))$

\[ \text{smoothness of } g^* \leq g^*(v(\alpha)) + \sum_{k=1}^{K} \gamma \nabla g^*(v(\alpha))^T v(\Delta \alpha_{[k]}) + \frac{\gamma^2}{2\tau} \| \sum_{k=1}^{K} v(\alpha_{[k]}) \|^2 \]

\[ \text{definition of } w(.) \leq g^*(v(\alpha)) + \sum_{k=1}^{K} \gamma v(\Delta \alpha_{[k]})^T w(\alpha) + \frac{\gamma^2}{2\tau} \| \sum_{k=1}^{K} v(\alpha_{[k]}) \|^2 \]

\[ \text{assumption (2.6)} \leq g^*(v(\alpha)) + \sum_{k=1}^{K} \gamma v(\Delta \alpha_{[k]})^T w(\alpha) + \frac{\gamma}{2}\sigma' \sum_{k=1}^{K} \| v(\alpha_{[k]}) \|^2 \]

We can now plug $A$ and $B$ back into (A.4) obtaining:

$D(\alpha + \gamma \sum_{k=1}^{K} \Delta \alpha_{[k]})$

\[ \geq - \sum_{k=1}^{K} \left( \sum_{i \in P_k} (1 - \gamma) \ell^*_i (-\alpha_i) + \gamma \ell^*_i (-\alpha + \Delta \alpha_{[k]}i) \right) \]

\[ - \left( g^*(v(\alpha)) + \sum_{k=1}^{K} \gamma v(\Delta \alpha_{[k]})^T w(\alpha) + \frac{\gamma}{2\tau} \| \sum_{k=1}^{K} v(\alpha_{[k]}) \|^2 \right) \]

\[ = - \sum_{k=1}^{K} \left( \sum_{i \in P_k} (1 - \gamma) \ell^*_i (-\alpha_i) \right) - (1 - \gamma) g^*(v(\alpha)) \]

\[ + \gamma \sum_{k=1}^{K} \left( -\frac{1}{n} \sum_{i \in P_k} \ell^*_i (-\alpha + \Delta \alpha_{[k]}i) - \frac{1}{K} g^*(v(\alpha)) - v(\Delta \alpha_{[k]})^T w(\alpha) - \frac{\sigma'}{2\tau} \| v(\alpha_{[k]}) \|^2 \right) \]

\[ \overset{(2.5)}{=} (1 - \gamma) D(\alpha) + \gamma \sum_{k=1}^{K} \mathcal{G}_{k}^*(\Delta \alpha_{[k]}; v). \]

where the last equality is by the definition of the dual objective $D(.)$ and the subproblem objective $\mathcal{G}_{k}^*(.)$ as in (2.5).

### A.1.2 Preliminary Results

Before proving the main convergence result of Theorem 4 we will present the following lemma, that characterizes the effect of an iteration of Algorithm 5 for any local solver compliant to Assumption 1.
Lemma 18. Let \( f : \mathbb{R}^d \to \mathbb{R} \) be a continuous proper convex function and \( f^* : \mathbb{R}^d \to \mathbb{R} \) its conjugate. Then:

\[
y \in \partial f(x) \iff f(x) + f^*(y) = x^T y
\]

(A.5)

and

\[
x \in \partial f^*(y) \iff f(x) + f^*(y) = x^T y
\]

(A.6)

Proof. We will begin by proving A.5. We first of all notice that Fenchel’s inequality gives us that

\[
f(x) + f^*(y) \geq x^T y;
\]

we thus only need to prove the reverse. We begin by noting that for every subgradient \( y \in \partial f(x) \) it holds

\[
f(z) \geq f(x) + (z - x)^T y \quad \forall z \in \mathbb{R}^d.
\]

By now applying the definition of convex conjugate we get:

\[
f(x) + f^*(y) = f(x) + \sup_z [z^T y - f(z)]
\]

\[
\leq f(x) + \sup_z [z^T y - (f(x) + (z - x)^T y)]
\]

\[
= \sup_z x^T y = x^T y
\]

Equation A.6 can be similarly proven by treating \( f \) as the biconjugate \( f^{**} \) of \( f \); this is possible since \( f = f^{**} \) which is given by the fact that \( f \) is a proper and continuous convex function.

Lemma 19. Let \( \ell^*_i \) be a \( \mu \)-strongly convex function with \( \mu \geq 0 \) and \( g \) be a \( \tau \)-strongly convex function with \( \tau > 0 \). Then for any iteration \( t \) of Algorithm 5 under Assumption 2 on \( \gamma, \sigma' \) and Assumption 1 on the local solver, for any \( s \in [0, 1] \) it holds that

\[
\mathbb{E}[D(\alpha^{(t+1)}) - D(\alpha^{(t)})] \geq \gamma (1 - \Theta) \left( sG(\alpha^{(t)}) - \frac{\sigma'}{2\tau} s^2 R^{(t)} \right),
\]

(A.7)

where

\[
R^{(t)} := -\frac{\tau \mu (1-s)}{\sigma' s} \|u^{(t)} - \alpha^{(t)}\|^2 + \sum_{k=1}^K \|A(u^{(t)} - \alpha^{(t)}_k)\|^2,
\]

(A.8)

for \( u^{(t)} \in \mathbb{R}^n \) with

\[
- u^{(t)}_i \in \partial \ell_i(x_i^T w(\alpha^{(t)})).
\]

(A.9)

Proof. The line of thought follows the one of [10, Lemma 5]. The main difference will be the extension to our generalized subproblems \( \mathcal{G}^*_k(\cdot; v, \alpha_k) \) along with the general mappings \( w(\alpha) := \nabla g^*(v(\alpha)) \) with \( v(\alpha) := A\alpha \).

For sake of notation, we will fix an iteration index \( t \) and drop it from the notation, writing \( \alpha, v, u \) instead of \( \alpha^{(t)}, v^{(t)}, u^{(t)} \).
We now begin by applying the update formula $\alpha^{(t+1)} := \alpha^{(t)} + \gamma \sum_k \Delta \alpha_{[k]}$ for the dual variables as used in Algorithm 5; we thus have:

$$E[\mathcal{D}(\alpha^{(t)}) - \mathcal{D}(\alpha^{(t+1)})] = E \left[ \mathcal{D}(\alpha) - \mathcal{D} \left( \alpha + \gamma \sum_{k=1}^K \Delta \alpha_{[k]} \right) \right]$$

(by Lemma 2)

$$\leq E \left[ \mathcal{D}(\alpha) - (1 - \gamma) \mathcal{D}(\alpha) - \gamma \sum_{k=1}^K G_{k}^{\sigma'}(\Delta \alpha_{[k]}^*; \nu, \alpha_{[k]}) \right]$$

$$= \gamma E \left[ \mathcal{D}(\alpha) - \sum_{k=1}^K G_{k}^{\sigma'}(\Delta \alpha_{[k]}^*; \nu, \alpha_{[k]}) \right]$$

(by Assumption 2 on the local solver $\Theta$-approximation quality)

$$\leq \gamma \left( \mathcal{D}(\alpha) - \sum_{k=1}^K G_{k}^{\sigma'}(\Delta \alpha_{[k]}^*; \nu, \alpha_{[k]}) + \Theta \left( \sum_{k=1}^K G_{k}^{\sigma'}(\Delta \alpha_{[k]}^*; \nu, \alpha_{[k]}) - \sum_{k=1}^K G_{k}^{\sigma'}(0; \nu, \alpha_{[k]}) \right) \right)$$

$$= \gamma (1 - \Theta) \left( \mathcal{D}(\alpha) - \sum_{k=1}^K G_{k}^{\sigma'}(\Delta \alpha_{[k]}^*; \nu, \alpha_{[k]}) \right) + C \left( \mathcal{D}(\alpha) \right)$$

(A.10)

where the last equality has been obtained by noticing that $\sum_{k=1}^K G_{k}^{\sigma'}(0; \nu, \alpha_{[k]}) = -g^*(\nu(\alpha)) - \sum_{i=1}^n \ell_{i}^*(\alpha_i) = \mathcal{D}(\alpha)$.

We will now upper bound term $C$. In the remaining of the proof we will be denoting $\Delta \alpha^* = \sum_{k=1}^K \Delta \alpha_{[k]}^*$. We start by applying the dual objective definition $\mathcal{D}$ and the definition of the local subproblems (2.5); we thus have
A.1. Convergence Analysis

that

\[ C = \sum_{i=1}^{n} (\ell^*_i(-\alpha_i - \Delta \alpha^*_i) - \ell^*_i(-\alpha_i)) + v(\Delta \alpha^*)^T w(\alpha) + \sum_{k=1}^{K} \frac{1}{2\tau} \sigma' \| v(s \alpha[k]) \|^2 \]

(By the optimality of \( \Delta \alpha^* \))

\[ \leq \sum_{i=1}^{n} (\ell^*_i(-\alpha_i - s(u_i - \alpha_i)) - \ell^*_i(-\alpha_i)) + v(s(\mathbf{u} - \mathbf{\alpha}))^T \mathbf{w}(\alpha) + \sum_{k=1}^{K} \frac{1}{2\tau} \sigma' \| v(s(\mathbf{u} - \mathbf{\alpha})) \|^2 \]

(By the \( \mu \)-strong convexity of \( \ell^* \))

\[ \leq \sum_{i=1}^{n} \left( s \ell^*_i(-u_i) + (1-s) \ell^*_i(-\alpha_i) - \frac{\mu}{2} (1-s) s(u_i - \alpha_i)^2 - \ell^*_i(-\alpha_i) \right) \\
+ v(s(\mathbf{u} - \mathbf{\alpha}))^T \mathbf{w}(\alpha) + \sum_{k=1}^{K} \frac{1}{2\tau} \sigma' \| v(s(\mathbf{u} - \mathbf{\alpha})) \|^2 \]

= \sum_{i=1}^{n} \left( s \ell^*_i(-u_i) - s \ell^*_i(-\alpha_i) - \frac{\mu}{2} (1-s) s(u_i - \alpha_i)^2 \right) \\
+ v(s(\mathbf{u} - \mathbf{\alpha}))^T \mathbf{w}(\alpha) + \sum_{k=1}^{K} \frac{1}{2\tau} \sigma' \| v(s(\mathbf{u} - \mathbf{\alpha})) \|^2 \]

By now applying Lemma 18 we get

\[ \ell^*_i(-u_i) = -u_i x_i^T w(\alpha) - \ell_i(x_i^T w(\alpha)). \quad \text{(A.11)} \]

Using the primal and dual problems definitions (1.3) and (1.4) we now write the duality gap as:

\[ G(\alpha) := P(\mathbf{w}(\alpha)) - D(\mathbf{\alpha}) \overset{(1.3),(1.4)}{=} \sum_{i=1}^{n} (\ell_i(x_i^T \mathbf{w}(\alpha)) + \ell^*_i(-\alpha_i)) + (g(\mathbf{w}(\alpha)) + g^*(v(\mathbf{\alpha}))) \]

= \sum_{i=1}^{n} (\ell_i(x_i^T \mathbf{w}(\alpha)) + \ell^*_i(-\alpha_i)) + (g(\nabla g^*(v(\mathbf{\alpha}))) + g^*(v(\mathbf{\alpha})))

(by Lemma 18)

= \sum_{i=1}^{n} (\ell_i(x_i^T \mathbf{w}(\alpha)) + \ell^*_i(-\alpha_i)) + v(\mathbf{\alpha})^T \mathbf{w}(\mathbf{\alpha})

= \sum_{i=1}^{n} (\ell_i(x_i^T \mathbf{w}(\mathbf{\alpha})) + \ell^*_i(-\alpha_i) + \alpha_i x_i^T \mathbf{w}(\mathbf{\alpha})). \quad \text{(A.12)}
A.1. Convergence Analysis

Hence,

\[ C \leq \sum_{i=1}^{n} \left( -su_i x_i^T w(\alpha) - s\ell_i(x_i^T w(\alpha)) - s\ell_i^*((-\alpha_i) \right) \\
- sa_i x_i^T w(\alpha) + sa_i x_i^T \w(\alpha) - \frac{\mu}{2} (1-s)s(u_i - \alpha_i)^2 \right) \\
+ v(s(u - \alpha))^T w(\alpha) + \sum_{k=1}^{K} \frac{1}{2\tau} \|v(s(u - \alpha)_{[k]})\|^2 \\
= \sum_{i=1}^{n} \left( -s\ell_i(x_i^T w(\alpha)) - s\ell_i^*((-\alpha_i) - sx_i^T w(\alpha)\alpha_i) \right) \\
+ \sum_{i=1}^{n} \left( sx_i^T w(\alpha)(\alpha_i - u_i) - \frac{\mu}{2} (1-s)s(u_i - \alpha_i)^2 \right) \\
+ v(s(u - \alpha))^T w(\alpha) + \sum_{k=1}^{K} \frac{1}{2\tau} \|v(s(u - \alpha)_{[k]})\|^2 \\
= -sG(\alpha) - \frac{\mu}{2} (1-s)s\|u - \alpha\|^2 + \frac{\sigma'}{2\tau} s^2 \sum_{k=1}^{K} \|A(u - \alpha)_{[k]}\|^2. \tag{A.13} \]

Finally, plugging this last equation (A.10) into (A.7) yields the improvement bound (A.7).

The following Lemma provides a uniform bound on \( R(t) \):

**Lemma 20.** If \( \ell_i \) are \( L \)-Lipschitz continuous for all \( i \in [n] \), then

\[ \forall t : R(t) \leq 4L^2 \sum_{k=1}^{K} \sigma_k n_k, \tag{A.14} \]

with \( \sigma_k \) as in (2.9).

**Proof.** [10, Proof of Lemma 6]

\[ \square \]

A.1.3 Proof of Convergence Rate for General Lipschitz Losses

**Theorem’ 4.** Let \( L \)-Lipschitz continuous \( \ell_i(\cdot) \) functions, a \( \tau \)-strongly convex \( g \) function and an arbitrary local solver satisfying Assumption 2 be given. Let
\( \gamma, \sigma' \) be defined as in Assumption 1. Then, if \( \epsilon_G > 0 \) is the desired duality gap, after \( T \) iterations of Algorithm 5 with

\[
T \geq T_0 + \max\left\{ \frac{1}{\gamma(1 - \Theta)}, \frac{4L^2\sigma'}{\tau \epsilon_G \gamma(1 - \Theta)} \right\}, \tag{A.15}
\]

\[
T_0 \geq t_0 + \left( \frac{2}{\gamma(1 - \Theta)} \left( \frac{8L^2\sigma'}{\tau \epsilon_G} - 1 \right) \right),
\]

\[
t_0 \geq \max\left(0, \left\lfloor \frac{1}{\gamma(1 - \Theta)} \log\left( \frac{2\tau(D(\alpha^*) - D(\alpha^{(0)}))}{4L^2\sigma'} \right) \right\rfloor \right),
\]

the expected duality gap will satisfy

\[
E[D(w(\bar{\alpha})) - D(\bar{\alpha})] \leq \epsilon_G,
\]
at the averaged iterate

\[
\bar{\alpha} := \frac{1}{T - T_0} \sum_{t = T_0 + 1}^{T-1} \alpha^{(t)} \tag{A.16}
\]

**Proof.** The proof of this theorem closely follows the proof of Theorem 8 from [10]. We start by estimating the expectation of the change in dual feasibility. Using Lemma 19 we get:

\[
E[D(\alpha^*) - D(\alpha^{(t + 1)})]
\]

\[
= E[D(\alpha^*) - D(\alpha^{(t)} + D(\alpha^{(t)}) - D(\alpha^{(t)})]
\]

\[
\leq D(\alpha^*) - D(\alpha^{(t)}) - \gamma(1 - \Theta)sG(\alpha^{(t)}) + \gamma(1 - \Theta)\frac{\sigma^2}{2\tau}R^{(t)}
\]

\[
= D(\alpha^*) - D(\alpha^{(t)}) - \gamma(1 - \Theta)s(P(w(\alpha^{(t)})) - D(\alpha^{(t)})) + \gamma(1 - \Theta)\frac{\sigma^2}{2\tau}R^{(t)}
\]

\[
\leq D(\alpha^*) - D(\alpha^{(t)}) - \gamma(1 - \Theta)s(D(\alpha^*) - D(\alpha^{(t)})) + \gamma(1 - \Theta)\frac{\sigma^2}{2\tau}R^{(t)}
\]

\[
\leq (1 - \gamma(1 - \Theta)s)(D(\alpha^*) - D(\alpha^{(t)})) + \gamma(1 - \Theta)\frac{\sigma^2}{2\tau}4L^2\sigma. \tag{A.17}
\]

We can now recursively apply (A.17) to get

\[
E[D(\alpha^*) - D(\alpha^{(t)})]
\]

\[
= (1 - \gamma(1 - \Theta)s)^{t-1} (D(\alpha^*) - D(\alpha^{(0)})) + \gamma(1 - \Theta)\frac{\sigma^2}{2\tau}4L^2\sigma \sum_{j=0}^{t-1} (1 - \gamma(1 - \Theta)s)^j
\]

\[
= (1 - \gamma(1 - \Theta)s)^{t-1} (D(\alpha^*) - D(\alpha^{(0)})) + \gamma(1 - \Theta)\frac{\sigma^2}{2\tau}4L^2\sigma \frac{1 - (1 - \gamma(1 - \Theta)s)^t}{\gamma(1 - \Theta)s}
\]

\[
\leq (1 - \gamma(1 - \Theta)s)^{t} (D(\alpha^*) - D(\alpha^{(0)})) + s\frac{4L^2\sigma\sigma'}{2\tau}. \tag{A.18}
\]

By choosing \( s = 1 \) and \( t = t_0 := \max\{0, \left\lfloor \frac{1}{\gamma(1 - \Theta)} \log\left( \frac{2\tau(D(\alpha^*) - D(\alpha^{(0)}))/(4L^2\sigma')}{\tau} \right) \right\rfloor \} \)

we can see that

\[
E[D(\alpha^*) - D(\alpha^{(t)})] \leq (1 - \gamma(1 - \Theta)^{t_0} (D(\alpha^*) - D(\alpha^{(0)})) + \frac{4L^2\sigma\sigma'}{2\tau} \tag{A.19}
\]

\[
\leq \frac{4L^2\sigma\sigma'}{2\tau} + \frac{4L^2\sigma\sigma'}{2\tau} = \frac{4L^2\sigma\sigma'}{\tau}. \tag{A.20}
\]
We’ll now proceed by induction in order to show
\[ \forall t \geq t_0 : \mathbb{E}[D(\alpha^*) - D(\alpha^{(t)})] \leq \frac{4L^2 \sigma s'}{\tau (1 + \frac{1}{2} \gamma (1 - \Theta)(t-t_0))}. \quad (A.21) \]

First of all we notice that the base case \( t = t_0 \) is readily implied by (A.19) We can show that, by assuming that (A.21) holds for a fixed \( t \geq t_0 \), it also holds for step \( t+1 \). We thus define \( s \) as a function of \( t \) as follows
\[ s = \frac{1}{1 + \frac{1}{2} \gamma (1 - \Theta)(t-t_0)} \in [0, 1] \quad (A.22) \]
and get
\[ \mathbb{E}[D(\alpha^*) - D(\alpha^{(t+1)})] \leq (1 - \gamma (1 - \Theta)s) (D(\alpha^*) - D(\alpha^{(t)})) + \gamma (1 - \Theta) \frac{s'^2}{2\tau} 4L^2 \sigma \]
\[ \text{and get} \]
\[ \mathbb{E}[D(\alpha^*) - D(\alpha^{(t+1)})] \leq (1 - \gamma (1 - \Theta)s) \frac{4L^2 \sigma s'}{\tau (1 + \frac{1}{2} \gamma (1 - \Theta)(t-t_0))} + \gamma (1 - \Theta) \frac{s'^2}{2\tau} 4L^2 \sigma \]
\[ \leq \frac{4L^2 \sigma s'}{\tau} \left( 1 + \frac{1}{2} \gamma (1 - \Theta)(t-t_0) - \frac{1}{2} \gamma (1 - \Theta) + \gamma (1 - \Theta) \frac{1}{2} \right) \]
\[ = \frac{4L^2 \sigma s'}{\tau} \left( 1 + \frac{1}{2} \gamma (1 - \Theta)(t-t_0) - \frac{1}{2} \gamma (1 - \Theta) \right) \]
\[ \leq 1 \]

We can now upperbound \( D \) using the fact that the geometric mean is less or equal than the arithmetic mean:
\[ D = \frac{1}{1 + \frac{1}{2} \gamma (1 - \Theta)(t+1-t_0)} \]
\[ \leq \frac{1}{1 + \frac{1}{2} \gamma (1 - \Theta)(t+1-t_0)}. \]

We now look at the expected duality gap for the averaged iterate \( \bar{\alpha} \) defined as in (A.16):
\[ \mathbb{E}[G(\bar{\alpha})] = \mathbb{E} \left[ G \left( \sum_{t=T_0}^{T-1} \frac{1}{T-T_0} \alpha^{(t)} \right) \right] \leq \frac{1}{T-T_0} \mathbb{E} \left[ \sum_{t=T_0}^{T-1} G \left( \alpha^{(t)} \right) \right] \]
\[ \leq \frac{1}{T-T_0} \mathbb{E} \left[ \sum_{t=T_0}^{T-1} \left( \frac{1}{\gamma (1 - \Theta)s} D(\alpha^{(t+1)}) - D(\alpha^{(t)}) + \frac{4L^2 \sigma s'}{2\tau} \right) \right] \]
\[ = \frac{1}{\gamma (1 - \Theta)s} \frac{1}{T-T_0} \mathbb{E} \left[ D(\alpha^{(T)}) - D(\alpha^{(T_0)}) \right] + \frac{4L^2 \sigma s'}{2\tau} \]
\[ \leq \frac{1}{\gamma (1 - \Theta)s} \frac{1}{T-T_0} \mathbb{E} \left[ D(\alpha^*) - D(\alpha^{(T_0)}) \right] + \frac{4L^2 \sigma s'}{2\tau}. \quad (A.23) \]
Assuming \( T \geq \left[ \frac{1}{\gamma(1-\Theta)} \right] + T_0 \) with \( T_0 \geq t_0 \) we get

\[
\mathbb{E}[G(\bar{\alpha})] \leq \frac{1}{\gamma(1-\Theta)s} \left( \frac{4L^2\sigma' \tau}{1 + \frac{1}{2} \gamma(1-\Theta)(T_0 - t_0)} + \frac{L^2\sigma' s}{2\tau} \right) + \frac{4L^2\sigma' s}{2\tau}.
\]

(A.24)

Choosing

\[
s = \frac{1}{(T - T_0)\gamma(1-\Theta)} \in [0, 1]
\]

(A.25)

we then obtain

\[
\mathbb{E}[G(\bar{\alpha})] \leq \frac{4L^2\sigma' \tau}{1 + \frac{1}{2} \gamma(1-\Theta)(T_0 - t_0)} + \frac{1}{(T - T_0)\gamma(1-\Theta)} \leq \frac{1}{2} \epsilon_G,
\]

(A.27)

We conclude by choosing a \( T \) and \( T_0 \) that make (A.26) smaller than the desired gap \( \epsilon_G \):

\[
4L^2\sigma' \left( \frac{1}{1 + \frac{1}{2} \gamma(1-\Theta)(T_0 - t_0)} + \frac{1}{2} \right) \leq \frac{1}{2} \epsilon_G.
\]

(A.28)

Therefore, using

\[
t_0 + \frac{2}{\gamma(1-\Theta)} \left( \frac{8L^2\sigma' \tau}{\epsilon_G} - 1 \right) \leq T_0,
\]

\[
T_0 + \frac{4L^2\sigma' \tau}{\epsilon_G \gamma(1-\Theta)} \leq T,
\]

(A.27) and (A.28) are both satisfied.

\[\square\]

### A.2 Specific Conjugates

**Lemma’ 14.** The convex conjugate of \( \tilde{\ell}_i^* \) as defined above is

\[
\tilde{\ell}(x) = \begin{cases} 
0 & : x \in [-1, 1] \\
B(|x| - 1) & : otherwise
\end{cases}
\]

and is \( B\)-Lipschitz.
A.2. Specific Conjugates

Proof. We start by applying the definition of convex conjugate, that is:

\[ \ell^\star(\alpha) = \sup_{x \in \mathbb{R}} [\alpha x - \ell(x)] \]

We begin by looking at the case in which \( \alpha \geq B \); in this case it’s easy to see that when \( x \to +\infty \), we have that:

\[ \alpha x - B(|x| - 1) = (\alpha - B)x - B \to +\infty \]

as \( \alpha - B \geq 0 \). Similarly one can show the case \( \alpha \leq -B \). We’ll now look at the case \( \alpha \in [0, B] \); in this case it is clear that an optimal \( x^\star \) needs to be \( \geq 0 \). Also it must hold that \( x^\star \leq 1 \) since \( \alpha x - B(x - 1) < 0 \) for every \( x > 1 \). Therefore the maximization becomes:

\[ \ell^\star(\alpha) = \sup_{x \in [0,1]} \alpha x \]

which has maximum \( \alpha \) at \( x = 1 \). Analogously one can show the remaining \( \alpha \in [-B, 0] \) case.

\[ f(\alpha) = \frac{\eta}{2} \alpha^2 + (1 - \eta)|\alpha| \]

Lemma’ 8. For \( \eta \in (0, 1] \), the convex conjugate of the elastic net function

\[ f(\alpha) = \frac{\eta}{2} \alpha^2 + (1 - \eta)|\alpha| \]

is

\[ f^\star(x) = \frac{1}{2\eta} \left( \left[|x| - (1 - \eta)\right]_+ \right)^2 , \]

where \( s_+ \) is the positive part operator, \( [s]_+ = s \) for \( s > 0 \), and zero otherwise. Also, the conjugate \( f^{**} \) of \( f^\star \) is equal to \( f \).

Proof. We start by applying the definition of convex conjugate, that is:

\[ f^\star(x) = \max_{\alpha \in \mathbb{R}} [x\alpha - f(\alpha)] = \max_{\alpha \in \mathbb{R}} \left[ x\alpha - \frac{\eta}{2} \alpha^2 - (1 - \eta)|\alpha| \right] \]

We’ll now distinguish the cases in which the optimal \( \alpha^\star \geq 0 \), \( \alpha^\star < 0 \). For the first case we get that

\[ f^\star(x) = \max_{\alpha \in \mathbb{R}} \left[ x\alpha - \frac{\eta}{2} \alpha^2 - (1 - \eta)\alpha \right] \]

and thus setting the derivative of it to 0 we get that \( \alpha^\star = \frac{x - (1 - \eta)}{\eta} \) and thus, in order for \( \alpha^\star \) to be \( \geq 0 \) this can only hold when \( x \geq 1 - \eta \). Replacing with \( \alpha^\star \) we thus get:

\[ f^\star(x) = \alpha^\star(x - \frac{1}{\eta}\alpha^\star - (1 - \eta)) = \alpha^\star \left( x - \frac{1}{\eta}(x - (1 - \eta)) - (1 - \eta) \right) = \]
A.2. Specific Conjugates

\[ \frac{1}{2} \alpha^* (x - (1 - \eta)) = \frac{1}{2\eta} (x - (1 - \eta))^2 \]

Similarly we can show that for \( x \leq -(1 - \eta) \)

\[ f^*(x) = \frac{1}{2\eta} (x + (1 - \eta))^2. \]

Finally, by the fact that \( f^*(\cdot) \) is convex, always positive and that \( f^*(-(1 - \eta)) = f^*(1 - \eta) = 0 \), it follows that \( f^*(x) = 0 \) for every \( x \in [-1 - \eta, 1 - \eta] \).

We conclude by noticing that, being the \( f \) function convex and continuous we can apply the Fenchel–Moreau theorem to prove that \( f^{**} = f \). \( \square \)
Bibliography


Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor’s thesis, Master’s thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

Distributed Optimization for Non-Strongly Convex Regularizers

Author(s) (in block letters):

For papers written by groups the names of all authors are required.

Name(s): Forte

First name(s): Simone

With my signature I confirm that
- I have committed none of the forms of plagiarism described in the ‘Citation etiquette’ information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date

Zürich, 16/09/2015

Signature(s)

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.