Distributed Machine Learning: Iterative Convex Optimization Methods

A Project Report Submitted
in Partial Fulfillment of Requirements
for the Degree of

Bachelor of Technology

by
Venkata Krishna Koundinya Pillutla

Department of Computer Science and Engineering
Indian Institute of Technology Bombay
Mumbai 400076, India

Abstract

Distributed convex optimization techniques try to augment the objective function minimized at each machine by adding a linear or a quadratic correction. Different theoretical treatments lead to different methods such as functional approximation and Iterative Parameter Mixing (IPM), Lagrange duality leads to an application of Alternating Direction Method of Multipliers (ADMM), Fenchel duality leads to yet another algorithm. All these algorithms are similar in that every global iteration requires communication of at least a vector of size equal to the number of features. We explore an ADMM derivative that tries to reduce the communication further. Experiments show that such a method, is indeed, promising. Further, we evaluate, theoretically and experimentally, IPM and its rate of convergence.

Acknowledgements

I would like to thank my advisor, Prof Saketha Nath J, for his patient guidance, encouragement and useful reviews of my work. His enthusiastic participation has been very much appreciated. My grateful thanks are also extended to Sundararajan Sellamanickam and Dhruv Mahajan of Microsoft Research, my guides during the summer internship there, in the summer of 2013. The project was originally their idea. Further, I would like to thank them for their guidance and all the fruitful discussions we had in the summer and during the course of the semester. Special thanks to Prof Soumen Chakrabarti for granting me access to the cluster and Research Assistant Shashank Gupta for all the help in running code and hadoop and his infinite patience in helping me. I would like to thank Pratik Jawanpuria for the discussions and with setting me up on a server.

Honor Code

I certify that we have properly cited any material taken from other sources and have obtained permission for any copyrighted material included in this report. I take full responsibility for any code submitted as part of this project and the contents of this report.

Author: Venkata Krishna Koundinya Pillutla

Certificate

It is certified that the B. Tech. project Iterative Parameter Mixing has been done by: Venkata Krishna Koundinya Pillutla under my supervision. This report has been submitted towards partial fulfillment of B. Tech. degree requirements.

Saketha Nath J
Advisor
Assistant Professor, Computer Science and Engineering
Indian Institute of Technology Bombay
Mumabi-400076, Maharashtra, India

Contents

Notation

- f is the objective function, $f(w) = \sum_{i=1}^{N} l(y_i w^T x_i) + \gamma R(w)$ where l is the loss function and $R(w)$ is the regularizer. Here, we only use the ℓ-2 regularizer, i.e $R(w) = \frac{1}{2} w^T w$. Here, we have N training instances.

IPM

- $w_i^{(k,r)}$ or $w_i^{(k)(r)}$ both represents the model at node i, in the r^{th} inner iteration of the k^{th} outer iteration.
- $w_i^{(k)}$ represents the **final** model at node i, in the k^{th} outer iteration.
- $w^{(k)}$ refers to the global model at the **beginning** of the k^{th} outer iteration.
- We have, for convex f,

$$f(y) \leq \hat{f}^w(y) = f(w) + \nabla f(w)^T (y - w) + \frac{L}{2} \|y - w\|^2$$

  where L is the Lipschitz Constant of ∇f
- $f_{-i}(w) = \sum_{j \neq i} f_j(w)$.
- The objective function for inner iterations of gradient descent at node i in outer iteration k is

$$\tilde{f}_{i,k}(w) = f_i(w) + \hat{f}_{-i}^{w_k}(w)$$

- c is the fixed number of times the inner iterations are performed, p is the number of nodes.

ADMM

- $x[i]$ repesents component i of vector x.
- w_i^t (or $w_i^{(t)}$) represents the model at node i in the outer iteration number t.
- λ_i is the (Lagrange) dual variable corresponding to the constraint between w_i and w_{i+1}. If it enforces k equalities, we have, $\lambda_i \in \mathbb{R}^k$.
- We define coefficient matrices (denoted by S) as follows: To relax a dimension-wise equality constraint $x = y$, $x, y \in \mathbb{R}^n$ to a set of $k \leq n$ constrains of the form $x_{I_1} = y_{I_1}, \ldots, x_{I_k} = y_{I_k}$ where $x_I = \sum_{j \in I} x[j]$. These constraints can be succintly represented as $S^T x = S^T y$ where $n \times k$ matrix S is constructed by the rule $S_{ij} = 1 \Leftrightarrow i \in I_j$. For instance, if the constraints are not relaxed, S is the identity matrix. On the other hand, under full relaxation ($k = 1, I_1 = \{1, 2, \ldots, n\}$), $S_{1 \times n} = [1, 1, ..., 1]^T$.

Chapter 1

Introduction

We consider learning problems of the form

$$min_w \sum_{i=1}^{N} l(y_i w^T x_i) + \gamma R(w)$$

where l is the loss function, x_i a training example and y_i is its label ($+1$ or -1 for binary classification), and w is the model, but in a distributed setting, where data is distributed across the nodes. R is the regularizer. For instance, $R(w) = w^T w/2$.

1.1 Iterative Parameter Mixing

1.1.1 Problem Statement

The effort is to augment the objective function at a node with a linear or quadratic approximating the objectives at every other node. The local models so obtained are combined into a global model. The approximations are updated, models evaluated and the process is continued iteratively.

1.1.2 Approach

Our approach was to initially use gradient descent on hinge loss (sub-gradient descent, actually) and logistic loss. Later, experiments we also conducted with a Trust Region Newton Method Parallelism was simulated. Theoretical results were also developed.

1.2 ADMM variant with reduced communication

1.2.1 Problem Statement

General algorithms in the distributed optimization framework require communication of a vector of length equal to the feature dimension. We try to reduce this, and ADMM is best suited method for this purpose. The original distributed objective can be recast as a constrained optimization problem, with constraints enforcing models on different machines to be equal. To reduce communication costs, the constraints are relaxed slightly- we enforce sums over arbitrary sets of features to be equal, but every feature need not be equal across nodes. Theoretical results have also been presented.

1.2.2 Approach

Experiments were run with (sub-) gradient descent as the inner optimization algorithm with hinge and logistic loss functions. Paralleslism was simulated and experiments were conducted with different datasets for different number of iterations.

1.3 Organization of the Report

The rest of the report is organized as follows: In Chapter 2, we review literature in the field. In Chapter 3, we propose the algorithm. In Chapter 4, we analyze convergence and rate of convergence theoretically. In Chapter 5, we explore explore experimental results. In Chapter 6, we discuss possible improvements and conclude the report.

Chapter 2

Literature Review and Previous Work

2.1 Parameter Mixing

The most popular way to solve such a problem in a distributed manner is to independently learn a model on each machine in a cluster using a partition of the dataset [1]. The global model is then computed as a convex combination of the local models, usually, a simple average [2]. It does not have strong performance guarantees but works well in practice.

In [3], the authors try to incorporate some gradient information in obtaining coefficients of the above mentioned convex combination. This weight matrix also appears in the update rule in the stochastic gradient descent that each machine locally runs (in algorithm 1 of [3]). It appears similar to the Hessian matrix in a second order optimization, suggesting that the Hessian could be used to obtain weights for the convex combination.

It was precisely this that I worked on during my summer internship, and we obtained an expression for mixing that did better than simple averaging. This method worked the quadratic approximation (Taylor series expansion) of the individual objective functions about their local minima and solving the resulting quadratic in closed form for the global solution. What if the quadratic approximation was not good enough at the point of mimimum? We look to Iterative Parameter Mixing.

2.2 Iterative Parameter Mixing(IPM)

The first literature on Iterative Parameter Mixing so far is [4]: The authors opine that parameter mixing is empirically powerful with no strong theoret-

ical guarantees. Further, it does not perform well for distributed perceptron training, the subject of that paper. They explore an iterative approach to parameter mixing, where the inner optimization is repeated several times with the previous global result as a new initial guess. It works well in practice and provides theoretical guarantees for the case of the perceptron. IPM is the logical solution to cover for the inadequacies of parameter mixing in our case too.

[5] is the most recent work on IPM (published in January 2014, after Stage-1 of this project). [5] proposes a fairly general scheme using functional approximations and also provides strong theoretical guarantees. A big drawback, however, is that [5] requires a global line search step, which could prove to be very costly. We try to produce some results without the line search, at the cost of some generality.

2.3 Alternating Direction Method of Multiplers (ADMM)

A separable problem of the form $min_w f(w) + g(w)$ is recast as $min_{x,y} f(x) + g(y)$ subject to $x = y$. Because of the separable nature, x is updated keeping y fixed and then, y is updated keeping x fixed (similar in spirit to the Jacobi method or the Gauss Seidel method used to solve diagonally dominant systems of linear equations). Dual variables are updated and the process is repeated. This method can be trivially parallelized: see [6].

The same idea can be generalized to $min_w \sum_{i=1}^{m} f_i(w)$ ([7]). We rewrite the problem either as

$$\begin{aligned} \min_{w_i,\ldots,w_m} \quad & \sum_{i=1}^{m} f_i(w_i) \\ \text{subject to} \quad & w_i = w_{i+1}; i = 1, \ldots, m-1. \end{aligned}$$

or as

$$\begin{aligned} \min_{z,w_i,\ldots,w_m} \quad & \sum_{i=1}^{m} f_i(w_i) \\ \text{subject to} \quad & w_i = z; i = 1, \ldots, m. \end{aligned}$$

If λ_j represents the Lagrangian dual variable correspoing to the j^{th} constraint and c is the augmented lagrangian parameter, the first formulation can be solved by the iterations (obtained by appropriate application of equa-

tion (4.75) of [7]):

$$w_i^{(t+1)} = \underset{w}{\operatorname{argmin}}\{f_i(w) + c\|w\|^2 + w^T(\lambda_i^{(t)} - \lambda_{i-1}^{(t)} - c(w_i^{(t)} + \frac{w_{i-1}^{(t)} + w_{i+1}^{(t)}}{2}))\} \tag{2.3.1}$$

$$\lambda_i^{(t+1)} = \lambda_i^{(t)} + \frac{c}{2}(w_i^{(t+1)} - w_{i+1}^{(t+1)}) \tag{2.3.2}$$

On the other hand, if we use the second formulation (equations (4.72-4.74) of [7]), we have,

$$z^{(t+1)} = \frac{\sum_{i=1}^m w_i^{(t)}}{m} - \frac{\sum_{i=1}^m \lambda_i^{(t)}}{mc} \tag{2.3.3}$$

$$w_i^{(t+1)} = \underset{w}{\operatorname{argmin}}\{f_i(w) + \frac{c}{2}\|w\|^2 - w^T(\lambda_i^{(t)} + cz^{(t+1)})\} \tag{2.3.4}$$

$$\lambda_i^{(t+1)} = \lambda_i^{(t)} + c(z^{(t+1)} - w_i^{(t+1)}) \tag{2.3.5}$$

2.4 A general framework for distributed optimization

Most distributed optimization algorithms add a linear or a quadratic correction $C(w)$ to the objective function minimised at each node: the correction is iteratively improved along with the solution ([5], [6], [8]).

In the functional approximations in [5], at node i in iteration t, a functional approximation (or an upper bound) to the objective function at every other node is chosen: as examples, we have $C^{(t)}(w) = (\hat{f}_{-i}^{w_t}(w))$, or $C^{(t)}(w) = f(w_t) + \nabla f(w_t)^T w + \frac{1}{2} w^T \nabla^2 f(w_t) w$ (the quadratic Taylor approximation) where $\nabla^2 f(w_t)$ is the Hessian matrix evaluated at $w = w_t$.

In each global iteration of the Alternating Direction Method of Multipliers, node i optimizes $f_i(w) + C_i(w)$ where $C_i(w) = c\|w\|^2 + w^T(\lambda_i^{(t)} - \lambda_{i-1}^{(t)} - c(w_i^{(t)} + \frac{w_{i-1}^{(t)} + w_{i+1}^{(t)}}{2}))$ depends on the augmented lagragian parameter c, the values of the dual variables $\lambda^{(t)}$ and the current solutions in neighbouring nodes $w_i^{(t)}$. Similarly, when Fenchel duality is used, we have a linear term ([8], equation (2)) used to tie together solutions from various nodes. Also, it is updated iteratively, along with the solutions from other nodes.

Convergence is shown, under suitable conditions, to be $log(1/\epsilon)$ in all three method in their respective papers.

Chapter 3

Algorithm

3.1 General Algorithm for Distributed Optimization

This is the pseudo-code of the algorithm to be run on node i. As we can see, this is fairly general and can capture all the variants described earlier.

Algorithm 1 General Algorithm

1: Initialise $w^{(0)}$ and other quantities required arbitrarily
2: **for** $t = 1, 2, ..$ (outer iterations) **do**
3: Compute $C_i^{(t)}(w)$, the correction
4: $w_i^{(t)} = \underset{w}{\text{argmin}}(f_i(w) + C_i^{(t)}(w))$ by some method
5: Communication: communicate the required vectors (such as dual vector, or gradients)
6: **end for**
7: **return** $w^{(t)} = \texttt{ParameterMixing}(w_i^t)$

Here, the function `ParameterMixing` is the convex combination

$$w^{(t+1)} = \sum_{i=1}^{\texttt{NumNodes}} \alpha_i w_i^{(t)}$$

.

3.2 Algorithm for IPM

The algorithm presented can use any inner optimization technique and in particular, we try out Gradient Descent and a Trust Region Newton Method.

Algorithm 2 Our Algorithm for IPM

1: Initialise $w^{(0)}$
2: **for** $t = 1, 2, ..$ (outer iterations) **do**
3: $\tilde{f}_i^{(t)}(w) = f_i(w) + f_{-i}(w^{(t)}) + \nabla f_{-i}(w^{(t)})^T(w - w^{(t)}) + \frac{L_{-i}}{2}\|w - w^{(t)}\|^2$
4: $w_i^{(t)} = \underset{w}{\operatorname{argmin}}(\tilde{f}_{i,t}(w))$ by some method with initial estimate $w^{(t)}$
5: $w^{(t+1)} = \texttt{ParameterMixing}(w_i^{(t)})$
6: Obtain $f(w^{(t+1)})$ and $\nabla f(w^{(t+1)})$ by communication
7: **end for**
8: **return** $w^{(t)}$

Algorithm 3 IPM Algorithm proposed in [5]

1: Initialise $w^{(0)}$
2: **for** $t = 1, 2, ..$ (outer iterations) **do**
3: Choose a descent direction $d^{(t)}$, for instance, $\texttt{ParameterMixing}(w_i^{(t)}) - w^{(t)}$ where $w_i^{(t)}$ is as defined in step 4 of of algorithm 3.2.
4: $w^{(t+1)} = w^{(t)} + \tau d^{(t)}$ where τ is a step length satisfying Armijo-Wolfe conditions.
5: Communicate the required quantities.
6: **end for**
7: **return** $w^{(t)}$

3.2.1 Discussion

Clearly, algorithm 3.2 is a special case of the algorithm **??** proposed in [5] with a specific form for the descent direction $d^{(t)}$ and **no line search**. The distributed step length computations across various machines could potentially be slow and very expensive. But, line search is needed in the proof for linear convergence presented in [5]. We try to prove, in the next chapter, linear convergence without line search, that is for `Algorithm 2`.

3.3 Reducing Communication Cost in ADMM

3.3.1 Discussion

Each iteration of ADMM requires node to communicate dual vectors $\lambda_i^{(t)}$ and model vectors, $w_i^{(t)}$ to neighbours $i-1, i+1$ (if they exist) in approach 1 (equations 2.3.1 and 2.3.2). Approach 2 (equations 2.3.3, 2.3.4 and 2.3.5) requires sum of these vectors over all nodes. If λ is the dual variable corresponding to the vector equality $x = y$, we have a scalar $\lambda[i]$ for the scalar inequality $x[i] = y[i]$. Hence, we communicate vectors of $\mathcal{O}(f)$ where f is the number of features. This much communication is also required for IPM and the Fenchel duality related approach discussed in [8]. Can we reduce the communication cost further?

3.3.2 Approach

Relaxing the vector equalities of the form $w_i = w_{i+1}$ reduces the communication cost. In particular, if we group elements in to $k \leq n$ sets (not necessarily disjoint) and enforce equalities of the form $\sum_{j \in I} w_i[j] = \sum_{j \in I} w_{(i+1)}[j]$ for each set I. Using the coefficient matrix notation introduced earlier, with $S \in \mathbb{R}^{n \times k}$, we can succintly represent these k equalities as a vector single equality: $S^T w_i = S^T w_{i+1}$. That is, we solve the following optimization problem:

$$\begin{aligned} \min_{w_i, \ldots, w_m} \quad & \sum_{i=1}^{m} f_i(w_i) \\ \text{subject to} \quad & S^T w_i = S^T w_{i+1}; i = 1, \ldots, m-1. \end{aligned} \tag{3.3.1}$$

The corresponding update rules change as follows (obtained by appropri-

ate application of equation (4.75) of [7]):

$$w_i^{(t+1)} = \underset{w}{\operatorname{argmin}}\{f_i(w)+c\|S^T w\|^2+w^T S(\lambda_i^{(t)}-\lambda_{i-1}^{(t)}-c(S^T w_i^{(t)}+\frac{S^T w_{i-1}^{(t)}+S^T w_{i+1}^{(t)}}{2}))\} \tag{3.3.2}$$

$$\lambda_i^{(t+1)} = \lambda_i^{(t)} + \frac{c}{2}(S^T w_i^{(t+1)} - S^T w_{i+1}^{(t+1)}) \tag{3.3.3}$$

Communication required is $S^T w$ with neighbours, each of which is a k-dimensional vector. Each node maintains λ_i and λ_{i-1} and hence communication is not necessary.

On the other hand, if we use approach 2, we have,

$$z^{(t+1)} = \frac{\sum_{i=1}^m S^T w_i^{(t)}}{m} - \frac{\sum_{i=1}^m \lambda_i^{(t)}}{mc} \tag{3.3.4}$$

$$w_i^{(t+1)} = \underset{w}{\operatorname{argmin}}\{f_i(w) + \frac{c}{2}\|w\|^2 - w^T S(\lambda_i^{(t)} + cz^{(t+1)})\} \tag{3.3.5}$$

$$\lambda_i^{(t+1)} = \lambda_i^{(t)} + c(z^{(t+1)} - S^T w_i^{(t+1)}) \tag{3.3.6}$$

In approach 2, communication required is sum of $S^T w$ and λ, both k-dimensional vectors across all nodes. By controlling the value of k, we can control the amount of communication.

Both approaches are similar computationally, but the way the communication is done is different. In approach 1, we used a protocol where all each node first passes messages to the node on the left, accepts messages from the node on the right, passes messages to the node on the right and accepts messages from the node on the left. To efficiently communicate in approach two, the AllReduce method, described in [3] could be used: a rooted tree structure is induced on the nodes. Messages are passed leaf upwards. Each intermediate node sums up (or performs any aggregation operation as necessary) the messages recieved with its own quantity and passes this aggregate as the message to its parent. The root then broadcasts the global aggregate to every node in the tree. Approach 1 requires synchronization between pairs of neighbouring nodes alone, while approach two requires global synchronization, and hence, a larger overhead. Throughout the experiments, we use approach 1.

It must noted that the approximation presented in the equation 3.3.1 can be solved by a variety of techniques. ADMM is just one such a technique an

d is the technique that led to the conceptualization of this approximation. We have solved problem from equation 3.3.1 using `cvx` as a blackbox solver to compare results as well. However, this was done with synthetic datasets alone as `cvx` is a general purpose solver and does not scale even for small real-life datasets.

3.3.3 Algorithm

Another huge saving in communication cost can be obtained by skipping the final parameter mixing step described in Algorithm 1. The final model in any one of the nodes can be used. Experiments show this is close to the values obtained from parameter mixing in the final step of the process.

Algorithm 4 ADMM with reduced communication

1: Initialise $w^{(0)}, w_i^{(0)}, \lambda_i = \mathbf{0}_k = \lambda_{i-1}$
2: **for** $t = 0, 1, 2, ..$ **do**
3: Solve for $w_i^{(t+1)}$ as per equation 3.3.2
4: Communicate models with neighbours and obtain $w_{i-1}^{(t+1)}$ and $w_{i+1}^{(t+1)}$
5: Update λ_i and λ_{i-1} by equation 3.3.3
6: **end for**
7: **return** $w_i^{(t+1)}$ or `ParameterMixing`(w_i^{t+1})

Chapter 4

Theoretical Results

4.1 Convergence

Theorem 1. *If f is convex and differentiable, ∇f is Lipschitz continuous with constant L, a strict decrease in global objective function f can be guaranteed in every outer iteration, that is, $f(w^{(k+1)}) < f(w^{(k)})$. provided the optimization algorithm used in step 4 of algorithm3.2 guarantees a strict decrease in the objective function solved by it and under some additional conditions.*

In particular, for gradient descent with a constant step size, the condition is that the step size h satisfies

$$0 < h \leq 1/L$$

Proof. The inner optimization algorithm should guarantee a strict decrease in the objective functions. This is true, since we have a convex objective. Let c_i be the inner iteration number for the i^{th} node.

That is,

$$\tilde{f}_{i,k}(w_i^{(k,c_i)}) < \tilde{f}_{i,k}(w_{i,k}^{c_i-1}) < \cdots < \tilde{f}_{i,k}(w^{(k)}) = f(w^{(k)})$$

Now,

$$
\begin{aligned}
f(w_{k+1}) &= f(\sum_{i=1}^{p} \alpha_i w_i^{(k,c_i)}) && \text{by definition} \\
&\leq \sum_{i=1}^{p} \alpha_i f(w_i^{(k,c_i)}) && \text{Jensen's inequality} \\
&\leq \sum_{i=1}^{p} \alpha_i \tilde{f}_{i,k}(w_i^{(k,c_i)}) && \text{since } \tilde{f}_{i,k}(w) \geq f(w) \\
&< \sum_{i=1}^{p} \alpha_i (\tilde{f}_{i,k}(w_k)) && \text{strict decrease} \\
&\leq \sum_{i=1}^{p} \alpha_i (f(w_k)) && \tilde{f}_{i,k}(w^{(k)}) = f(w^{(k)}) \\
&= f(w_k) && \text{since } \sum_i \alpha_i = 1
\end{aligned}
$$

Since the objective is convex, it has a unique global minimiser and the algorithm converges.

□

4.2 Order of Convergence

The order of convergence results hold for Gradient Descent. For a convex objective function whose gradient is Lipschitz continous, with a constant number of inner iterations, when sufficiently close to the global optimum, the convergence is locally $\mathcal{O}(1/k)$. For a strongly convex objective, convergence is linear. We have almost proved stronger versions of the above result for global convergence.

Local Convergence

Theorem 2. *If f if convex and differentiable, ∇f is Lipschitz continuous with constant L, the inner optimisation is solved with gradient descent i.e. $w_i^{(k,j+1)} = w_i^{(k,j)} - h\nabla \tilde{f}_{i,k}(w_i^{(k,j)})$ with a fixed number of steps c and fixed step size $h = 1/L(c^2 + 2c - 2)$ and the initial guess $w^{(0)}$ is sufficiently close to the global optimum w^* then algorithm 3.2 converges as*

$$
f(w^{(k)}) - f(w^*) \leq \frac{2L\|w^{(0)} - w^*\|^2}{k+4}\beta^2 \tag{4.2.1}
$$

where $\beta > 0$ is a constant i.e., convergence is $\mathcal{O}(1/k)$, for k outer iterations.

If f is strongly convex with constant μ, convergence is linear as

$$\|w^{(k)} - w^*\| \leq (\frac{L-\mu}{L+\mu})^k \|w^{(0)} - w^*\| \tag{4.2.2}$$

Proof.

$$w_i^{(k,j+1)} = w_i^{(k,j)} - h\nabla \tilde{f}_{i,k}(w_i^{(k,j)})$$

Claim1:

$$\nabla \tilde{f}_{i,k}(w_i^{(k,r)})^T \nabla \tilde{f}_{i,k}(w_i^{(k,r-1)}) \leq \|\nabla \tilde{f}_{i,k}(w_i^{(k,r-1)})\|^2 \tag{4.2.3}$$

This follows from $(\nabla \tilde{f}_{i,k}(w_i^{(k,r)}) - \nabla \tilde{f}_{i,k}(w_i^{(k,r-1)}))^T (w_i^{(k,r)} - w_i^{(k,r-1)}) \geq 0$, for convex $\tilde{f}_{i,k}$.

Claim2:

$$\|\nabla \tilde{f}_{i,k}(w_i^{(k,c)})\| \leq \|\nabla \tilde{f}_{i,k}(w_i^{(k,c-1)})\| \leq ... \leq \|\nabla \tilde{f}_{i,k}(w_i^{(k,0)})\| = \|\nabla f(w_k)\| \tag{4.2.4}$$

Proof: Using,

$$(\nabla \tilde{f}_{i,k}(w_i^{(k,r)}) - \nabla \tilde{f}_{i,k}(w_i^{(k,r-1)}))^T (w_i^{(k,r)} - w_i^{(k,r-1)}) \geq \frac{1}{L}\|\nabla \tilde{f}_{i,k}(w_i^{(k,r)}) - \nabla \tilde{f}_{i,k}(w_i^{(k,r-1)})\|^2$$

We have,

$$\frac{1}{L}\|\nabla \tilde{f}_{i,k}(w_i^{(k,r)})\|^2 \leq (\frac{2}{L} - h)(\nabla \tilde{f}_{i,k}(w_i^{(k,r)})^T \nabla \tilde{f}_{i,k}(w_i^{(k,r-1)})) + (h - \frac{1}{L})\|\nabla \tilde{f}_{i,k}(w_i^{(k,2-1)})\|^2$$

For $h \leq \frac{2}{L}$, using Claim1, we get the required result, that seems very intuitive for the inner gradient descent.

Now, consider $\|w_i^{(k,c)} - w^*\|^2$:

$$\begin{aligned}
\|w_i^{(k,c)} - w^*\|^2 =& \|w^{(k)} - w^* - h\sum_{r=0}^{c-1} \|\nabla \tilde{f}_{i,k}(w_i^{(k,r)})\|^2 \\
=& \|w^{(k)} - w^*\|^2 - 2h\nabla f(w^{(k)})^T (w^{(k)} - w^*) \\
& - 2h(w^{(k)} - w^*)^T (\sum_{r=1}^{c-1} \nabla \tilde{f}_{i,k}(w_i^{(k,r)})) + h^2 \|\sum_{r=0}^{c-1} \nabla \tilde{f}_{i,k}(w_i^{(k,r)})\|^2
\end{aligned}$$

We reduce each of the above terms as follows.

- Following [9], using $(\nabla f(w^{(k)}) - \nabla f(w^*))^T (w^{(k)} - w^*) \geq \frac{1}{L}\|\nabla f(w^{(k)}) - \nabla f(w^*)\|^2$ and $\nabla f(w^*) = 0$, we get,

$$-\nabla f(w^{(k)})^T (w^{(k)} - w^*) \leq -\frac{1}{L}\|\nabla f(w^{(k)})\|^2$$

- To reduce $-(w^{(k)}-w^*)^T(\nabla\tilde{f}_{i,k}(w_i^{(k,r)}))$: Using $f(y)-f(w)\geq\nabla f(w)^T(y-w)$, we get:

$$\tilde{f}_{i,k}(w^*)-\tilde{f}_{i,k}(w_i^{(k,r)})\geq\nabla\tilde{f}_{i,k}(w_i^{(k,r)})^T(w^*-w_i^{(k,r)})$$

$$\nabla\tilde{f}_{i,k}(w_i^{(k,r)})^T(w_i^{(k,r)}-w^*)\geq\tilde{f}_{i,k}(w_i^{(k,r)})-\tilde{f}_{i,k}(w^*)$$

$$-\nabla\tilde{f}_{i,k}(w_i^{(k,r)})^T(w_i^{(k,r)}-w^*)\leq\tilde{f}_{i,k}(w^*)-\tilde{f}_{i,k}(w_i^{(k,r)})$$

  If $w^{(k)}$, $w_i^{(k,r)}$ and w^* are sufficiently close to each other, we have $\tilde{f}_{i,k}(w)\approx f(w)$. Using the fact that w^* is the global minimum of f, we get,

$$-\nabla\tilde{f}_{i,k}(w_i^{(k,r)})^T(w_i^{(k,r)}-w^*)\leq 0$$

  .

- The third term reduces using Cauchy-Schwartz Inequality and claim2:

$$\begin{aligned}\|\sum_{r=0}^{c-1}\nabla\tilde{f}_{i,k}(w_i^{(k,r)})\|^2&=\sum_{r=0}^{c-1}\|\nabla\tilde{f}_{i,k}(w_i^{(k,r)})\|^2+2\sum_{r=1}^{c-1}\sum_{j=0}^{r}\nabla\tilde{f}_{i,k}(w_i^{(k,r)})^T\nabla\tilde{f}_{i,k}(w_i^{(k,j)})\\&\leq\|\nabla f(w^{(k)})\|^2(c+2(\frac{c^2+c}{2}-1))\\&=(c^2+2c-2)\|\nabla f(w^{(k)})\|^2\end{aligned}$$

  So we have,

$$\|w_i^{(k,c)}-w^*\|^2\leq(1+2(c-1)L)\|w^{(k)}-w^*\|^2-h(\frac{2}{L}-(c^2+2c-2)h)\|\nabla f(w^{(k)})\|^2$$

That is,

$$\|w_i^{(k,c)}-w^*\|\leq\beta\|w^{(k)}-w^*\| \tag{4.2.5}$$

where $\beta^2=(1+2(c-1)L)$ when $h\leq\frac{2}{L(c^2+2c-2)}$ The best step-size is obtained by maximising the quadratic $q(h)=h(\frac{2}{L}-(c^2+2c-2)h)$. That is when,

$$h^*=\frac{1}{(c^2+2c-2)}.\frac{1}{L} \tag{4.2.6}$$

Finally,

$$\begin{aligned}
\|w_{k+1} - w^*\| &= \|\sum_{i=0}^{p} \alpha_i (w_i^{(k,c)} - w^*)\| && \text{by definition} \\
&\leq \sum_{i=0}^{p} \alpha_i \|w_i^{(k,c)} - w^*\| && \text{triangle inequality} \\
&\leq \sum_{i=0}^{p} \alpha_i \beta \|w^{(k)} - w^*\| && \text{from equation 4.2.5}
\end{aligned}$$

Since the α_i add up to one, we have,

$$\|w_i^{(k+1)} - w^*\| \leq \beta \|w^{(k)} - w^*\| \tag{4.2.7}$$

$$\tilde{f}_{i,k}(w_i^{(k,j)}) \leq \tilde{f}_{i,k}(w_i^{k,j-1}) + \nabla \tilde{f}_{i,k}(w_i^{k,j-1})^T (w_i^{(k,j)} - w_i^{k,j-1}) + \tfrac{L}{2} \|(w_i^{(k,j)} - w_i^{k,j-1})\|^2$$

$$= \tilde{f}_{i,k}(w_i^{k,j-1}) - \frac{1}{2L} \|\nabla \tilde{f}_{i,k}(w_i^{k,j-1})\|^2$$

since $h = 1/L$. Using the above inequality repeatedly gives,

$$\tilde{f}_{i,k}(w_i^{k,c}) \leq \tilde{f}_{i,k}(w_k) - \frac{1}{2L} \sum_{r=1}^{c} \|\nabla \tilde{f}_{i,k}(w_{i,k}^{r-1})\|^2 \tag{4.2.8}$$

Now,

$$\begin{aligned}
f(w^{(k+1)}) &= f(\sum_{i=1}^{p} \alpha_i w_i^{(k,c)}) && \text{by definition} \\
&\leq \sum_{i=1}^{p} \alpha_i f(w_i^{(k,c)}) && \text{Jensen's inequality} \\
&\leq \sum_{i=1}^{p} \alpha_i \tilde{f}_{i,k}(w_i^{(k,c)}) && \text{since } \tilde{f}_{i,k}(w) \leq f(w) \\
&\leq \sum_{i=1}^{p} \alpha_i (\tilde{f}_{i,k}(w^{(k)}) - \frac{1}{2L} \sum_{r=1}^{c} \|\nabla \tilde{f}_{i,k}(w_{i,k}^{r-1})\|^2) && \text{shown above} \\
&\leq \sum_{i=1}^{p} \alpha_i (\tilde{f}_{i,k}(w^{(k)}) - \frac{1}{2L} \|\nabla \tilde{f}_{i,k}(w^{(k)})\|^2) && \text{first term from each inner sum} \\
&= \tilde{f}_{i,k}(w^{(k)}) - \frac{1}{2L} \|\nabla \tilde{f}_{i,k}(w^{(k)})\|^2 && \text{since } \sum_i \alpha_i = 1 \\
&= f(w^{(k)}) - \frac{1}{2L} \|\nabla f(w^{(k)})\|^2
\end{aligned}$$

The last step is because $\tilde{f}_{i,k}(w^{(k)}) = f(w^{(k)})$ and $\nabla \tilde{f}_{i,k}(w^{(k)}) = \nabla f(w^{(k)})$.

$$f(w^{(k+1)}) \leq f(w^{(k)}) - \frac{1}{2L}\|\nabla f(w^{(k)})\|^2 \tag{4.2.9}$$

Using equations 4.2.7 and 4.2.9 the rest of the proof is a direct application of Theorem 2.1.14, 2.1.15 and Corollary 2.1.2 of [9]

□

Global Convergence

In [5], global convergence was proved for IPM, but a global linear seach step was required (alogrithm 3.3). We tried to prove a similar result for our algorithm but were unable to prove the following theorem:

Theorem 3. *If f if convex and differentiable, ∇f is Lipschitz continuous with constant L, the inner optimisation is solved with gradient descent i.e. $w_i^{(k,j+1)} = w_i^{(k,j)} - h\nabla \tilde{f}_{i,k}(w_i^{(k,j)})$ with a fixed number of steps c and fixed step size $h = 1/L(c^2 + 2c - 2)$, then algorithm 3.2 converges as*

$$f(w^{(k)}) - f(w^*) \leq \frac{2L\|w^{(0)} - w^*\|^2}{k+4}\beta^2$$

where $\beta > 0$ is a constant i.e., convergence is $\mathcal{O}(1/k)$, for k outer iterations.

If f is strongly convex with constant μ, convergence is linear as

$$\|w^{(k)} - w^*\| \leq (\frac{L-\mu}{L+\mu})^k \|w^{(0)} - w^*\|$$

We were only able to prove a weaker version global rate of convergence:

Theorem 4. *If f if convex and differentiable, ∇f is Lipschitz continuous with constant L, the inner optimization is solved with gradient descent i.e. $w_i^{(k,j+1)} = w_i^{(k,j)} - h\nabla \tilde{f}_{i,k}(w_i^{(k,j)})$ with a fixed number of steps c, and a fixed step size of $h = 1/L$, we have,*

$$\|\nabla f(w^{(k)}\| \leq \sqrt{\frac{2L(f(w^{(0)} - f(w^*))}{k+1}} \tag{4.2.10}$$

In other words, convergence is $\mathcal{O}(1/\sqrt{k})$, for k outer iterations.

Proof. Equations 4.2.3, 4.2.4, 4.2.9 can be proved as above. 4.2.7 does not hold globally and hence we have to settle for a poorer rate of convergence. Equation (4.2.9) states that:

$$f(w^{(k+1)}) \leq f(w^{(k)}) - \frac{1}{2L}\|\nabla f(w^{(k)}\|^2$$

This is a special case of equation (1.2.13) of [9] with $\omega = 1/2$. It follows from equation (1.2.15) of [9] that

$$\|\nabla f(w^{(k)}\| \leq \frac{1}{\sqrt{k+1}}(2L(f(w^{(0)} - f(w^*)))^{1/2}$$

Further, this theorem can also be extended to cover the case of line search. □

Chapter 5

Experimental Results

5.1 IPM

Experiments were run using gradient descent and tron as inner optimisation techniques. Distributed system of 2, 4 and 8 nodes were simulated on a single machine- we intend to run similar experiments on a hadoop cluster scaling up to much larger numbers soon. Experiments were run primarily on two datasets obtained from libsvm data (`http://www.csie.ntu.edu.tw/ cjlin/ libsvmtools/datasets/`). `a9a`- a binary classification problem, and `usps`, a multilabel classification problem was transformed into a 3 vs 6 binary classification problem (called `usps36` henceforth) and used for the experiments.

The objective optimsed at each node was:

$$\tilde{f}_{i,k}(w) = f_i(w) + t\hat{f}_{-i,k}(w)$$

where t is a scalar between 0 and 1.

The parameters under control were number of inner iterations, outer iterations and the scalar t above. Every update passed was a linear update, i.e. we took $L = 0$ for the sake of experiments.

Inner optimisation was done with Gradient Descent (GD) and with a trust region Newton Method (TRON), implemented from [10].

5.1.1 Results

Figures 5.1.1, 5.1.1 contains four curves: red to represent the accuracy obtained on the entire dataset, green to represent the accuracy obtained by one node, and blue to obtain the accuracy obtained by IPM- one for gradient descent, and one for TRON. The black line represents accuracy achieved by PM (equivalent to single iteration of IPM). The results shown are for

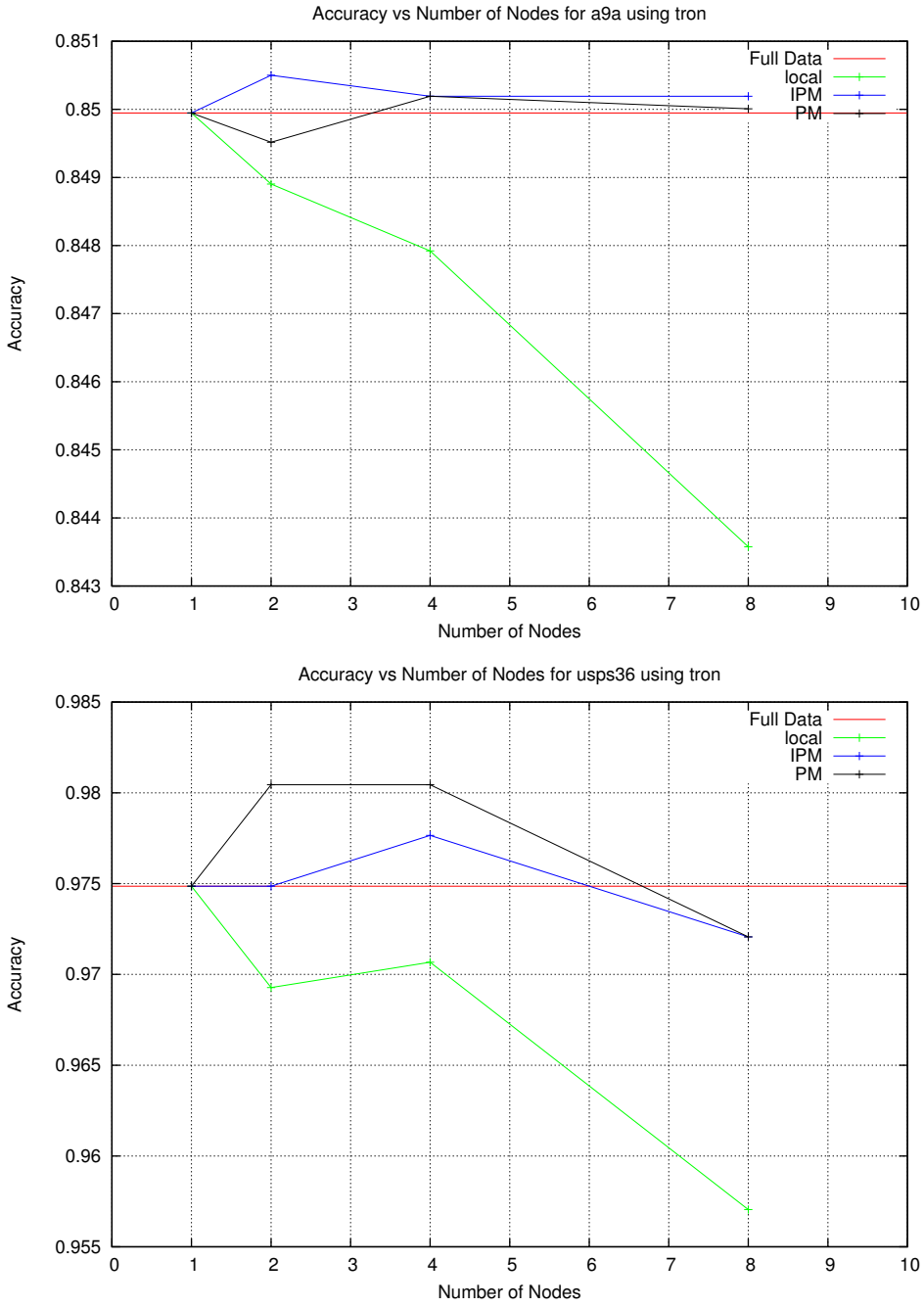

Figure 5.1: IPM with TRON

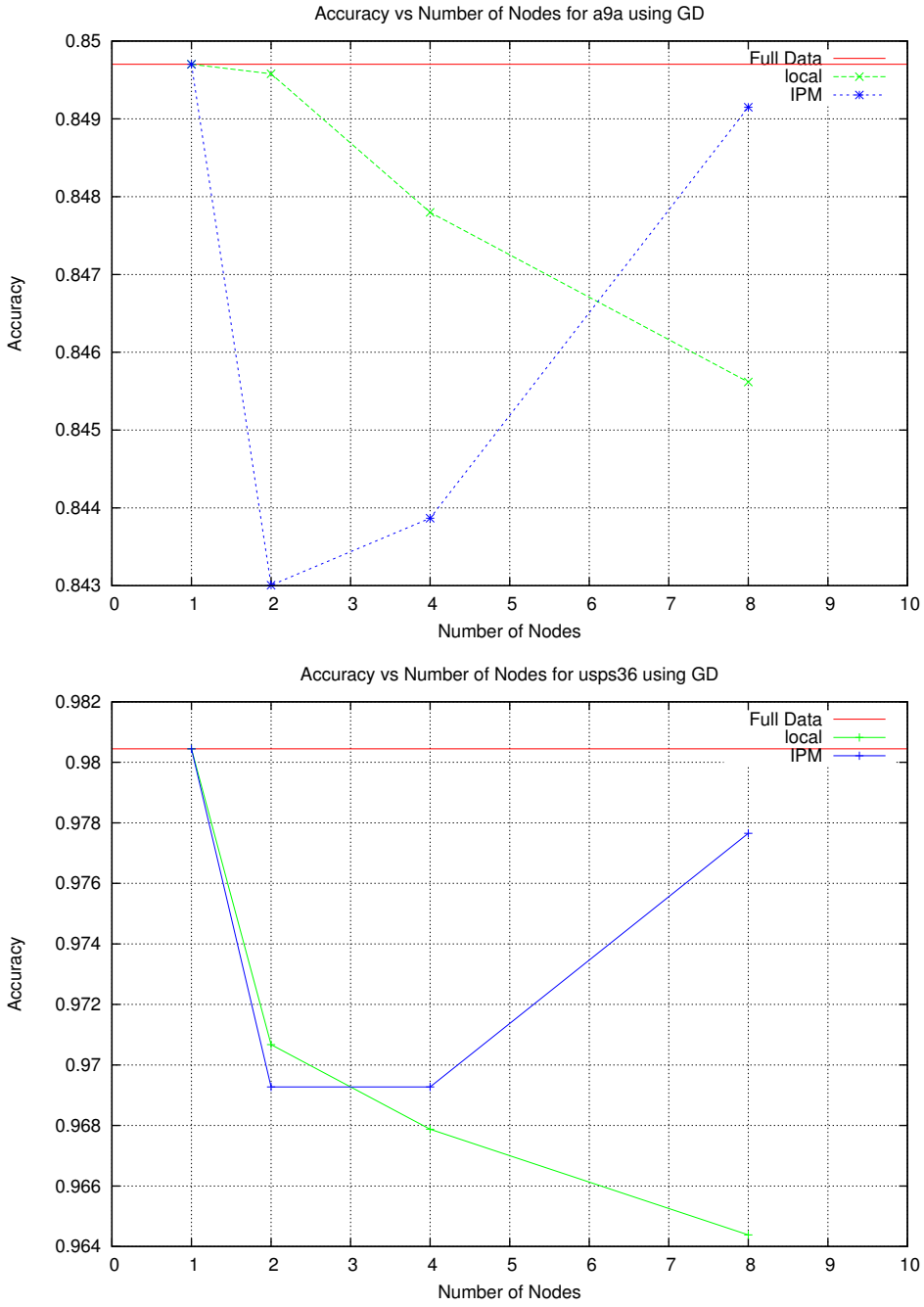

Figure 5.2: IPM with Gradient Descent

Note that larger the number of nodes, the better IPM does.

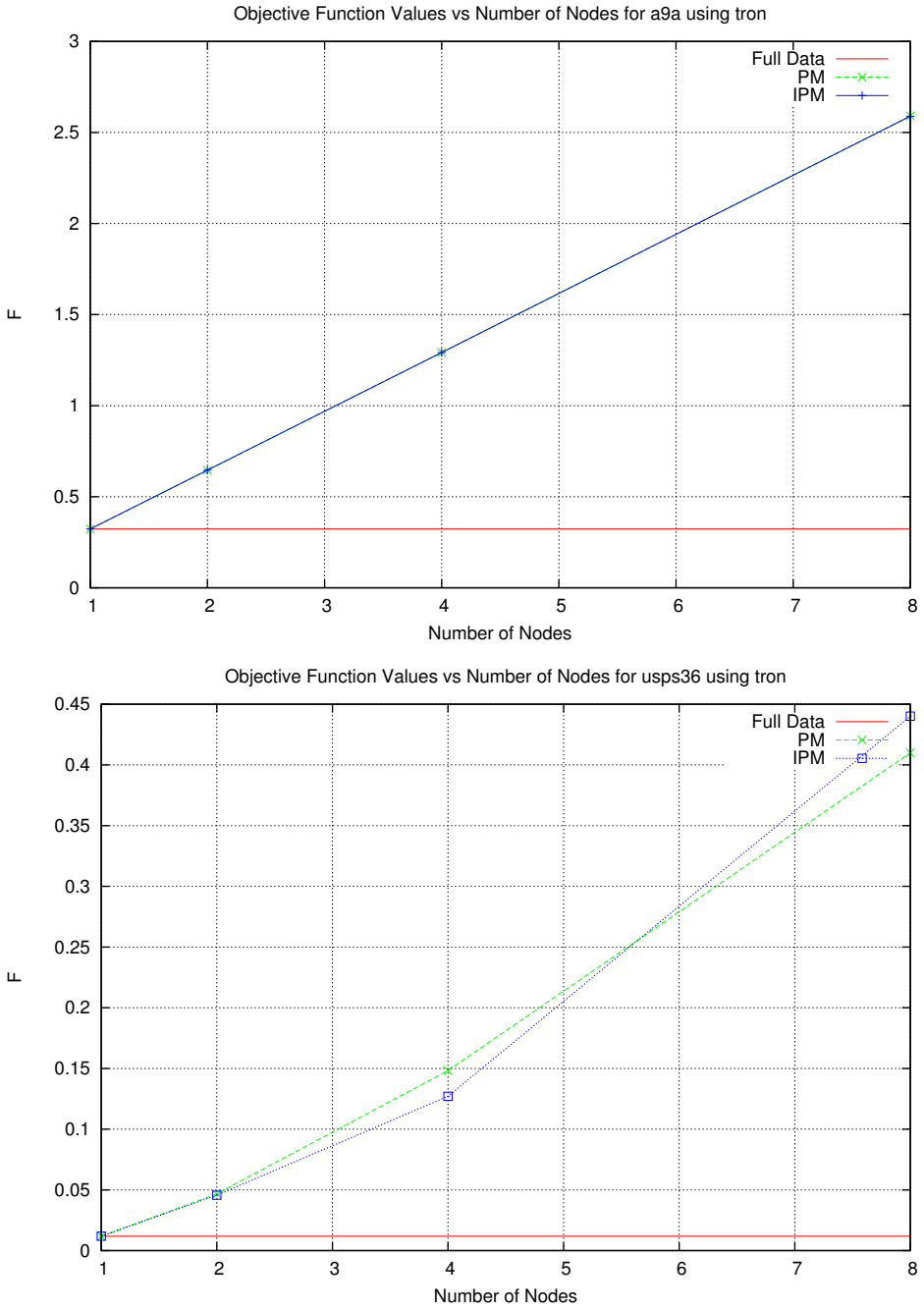

Figure 5.3: IPM with TRON: Objective Function Value

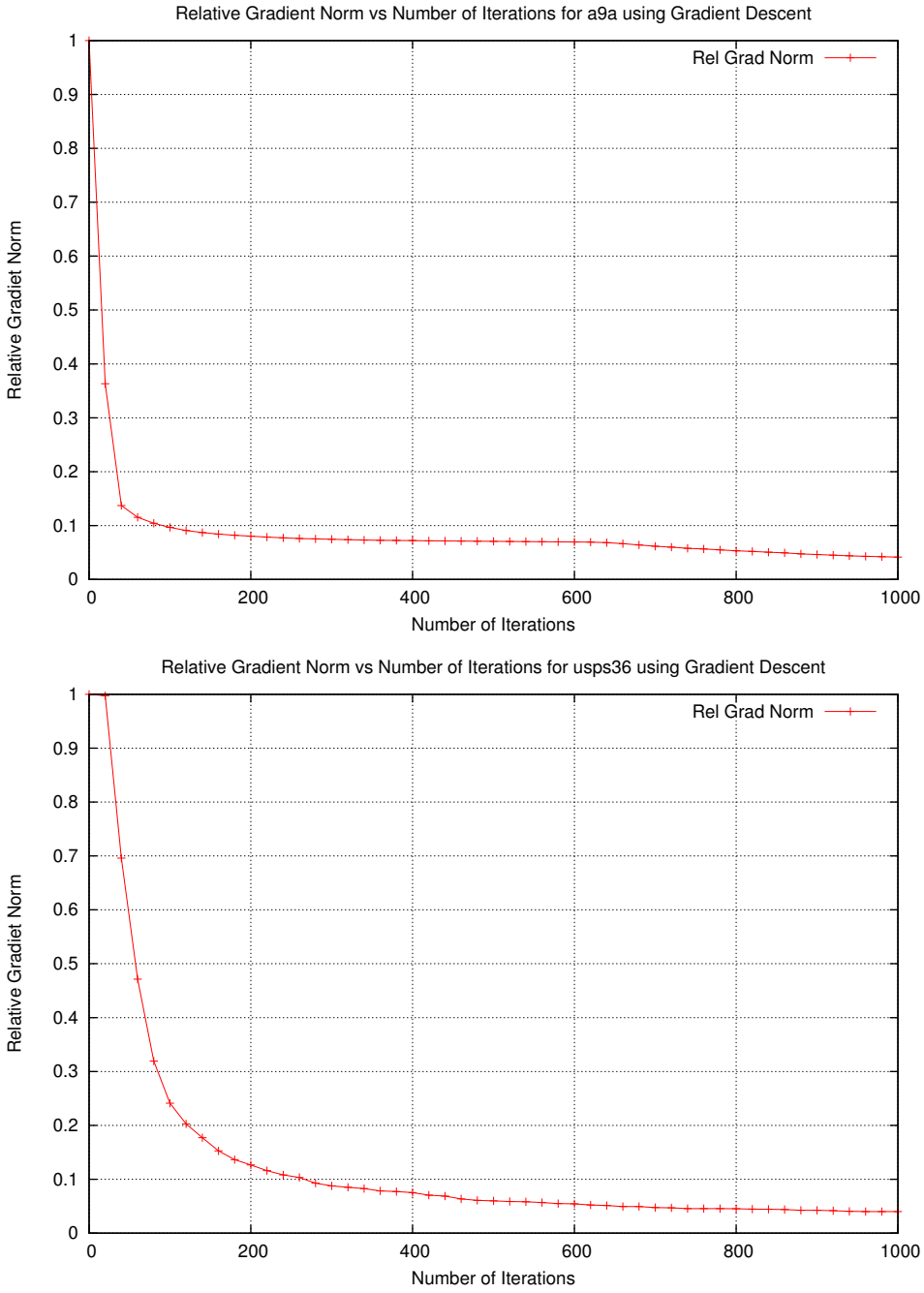

Figure 5.4: Relative Gradient Norm

Gradient approaching zero is a sign of convergence

10 inner iterations (GD or TRON), 1000 outer iterations for GD, 10 outer iterations for TRON. Also $t = 1$ for GD. For TRON, with 2, 4, and 8 nodes, t was taken to be 0.1, 0.01, 0.01 respectively for a9a and 0.08, 2e-03, 2e-03 for usps36. This was necessary because the linear terms add a constant to the gradient which causes trouble with the stopping criterion for the inner conjugate gradient iterations in TRON.

The next set of plots, figure 5.4 is the relative gradient norm, $g = \|\nabla f(w_k)\|/\|\nabla f(w_0)\|$. Since $\nabla f(w^*) = 0$, $g = \|\nabla f(w_k) - \nabla f(w^*)\|/\|\nabla f(w_0) - \nabla f(w^*)\|$ gives a measure of the rate of convergence, here, for the hinge-loss function with sub-gradient descent (since the hinge-loss function is not differentiable). The curve does look like $\mathcal{O}(1/k)$ where k is the number of iterations.

As can be seen from the graphs attached, GD does a poor job with a small number of outer iterations- the larger the number of outer iterations, the better it does. TRON, on the other hand, does reasonably well with as small 2-3 outer iterations. It does as good as running TRON on the entire dataset with around 10 outer iterations. To compare, the original datasets take around 20 iterations for TRON to converge to the optimum.

Figure 5.3 suggests that there is no clear winner amongst PM and IPM. It also seems to be dependent on the inherent qualities of the data whether one iteration will suffice (PM), or whether multiple iterations are required (IPM). The dataset `usps36` is a linear dataset. A single iteration (PM) is sufficient for convergence, and hence, PM does as good as IPM.

Table 5.1: Performance with Synthetic Dataset TestFinal4.mat with 4 nodes

Method	**Objective function value**	**Accuracy on test set**
Full problem	0.4891	0.8000
Parameter Mixing	0.6009	0.7350
Our method	0.5128	0.7900

5.2 ADMM with reduced communication

5.2.1 Synthetic Datasets

We generated synthetic datasets to work with `cvx`. Four to ten dimensional datasets were created with 400- 4000 examples and were randomly divided in to two parts- for training and testing. Problem 3.3.1 was solved directly with `cvx` as a black box solver, with duality gap, $d < \epsilon$ being the stopping criterion. $k = 3$ was taken. That is, $S \in \mathbb{R}^{n \times 3}$ was taken. In some cases, our method achieved as much as 6% more accuracy than simple parameter mixing, and very close to the maximum possible accuracy on that dataset. Our method did a great job in minimising the loss function as well, something that parameter mixing failed at. Table 5.1 presents one such an instance:

5.2.2 Real-life Datasets

Experiments were run on datasets `a9a`, `usps36` and `covtype` (also know as `cov`), a difficult dataset (that is, non-linear) using gradient descent as the inner optimization method. Experiments were run and results compared against the fraction $\frac{k}{n}$ for various values of k. We fixed the augmented lagrangian parameter (c in equations 3.3.2 and 3.3.3) to be 1×10^{-3}. In particular, we look at the objective function value and accuracy on the test set with special interest. Experiments were run simulating 2, 4 and 8 nodes. `cov` is a large dataset and hence, with it, experiments with 25, 50, 75 and 100 nodes have been performed as well. The values obtained are compared against various baseline measures: values for optimising the entire dataset on a single machine and values for parameter mixing.Termination was based on number of iterations- it is easier to compare communication costs this way. Suppose we have T_{inner} iterations of gradient descent in each outer iteration and T_{outer} outer iterations in our ADMM variant, Parameter Mixing was run with T_{inner} iterations of gradient descent on each machine.

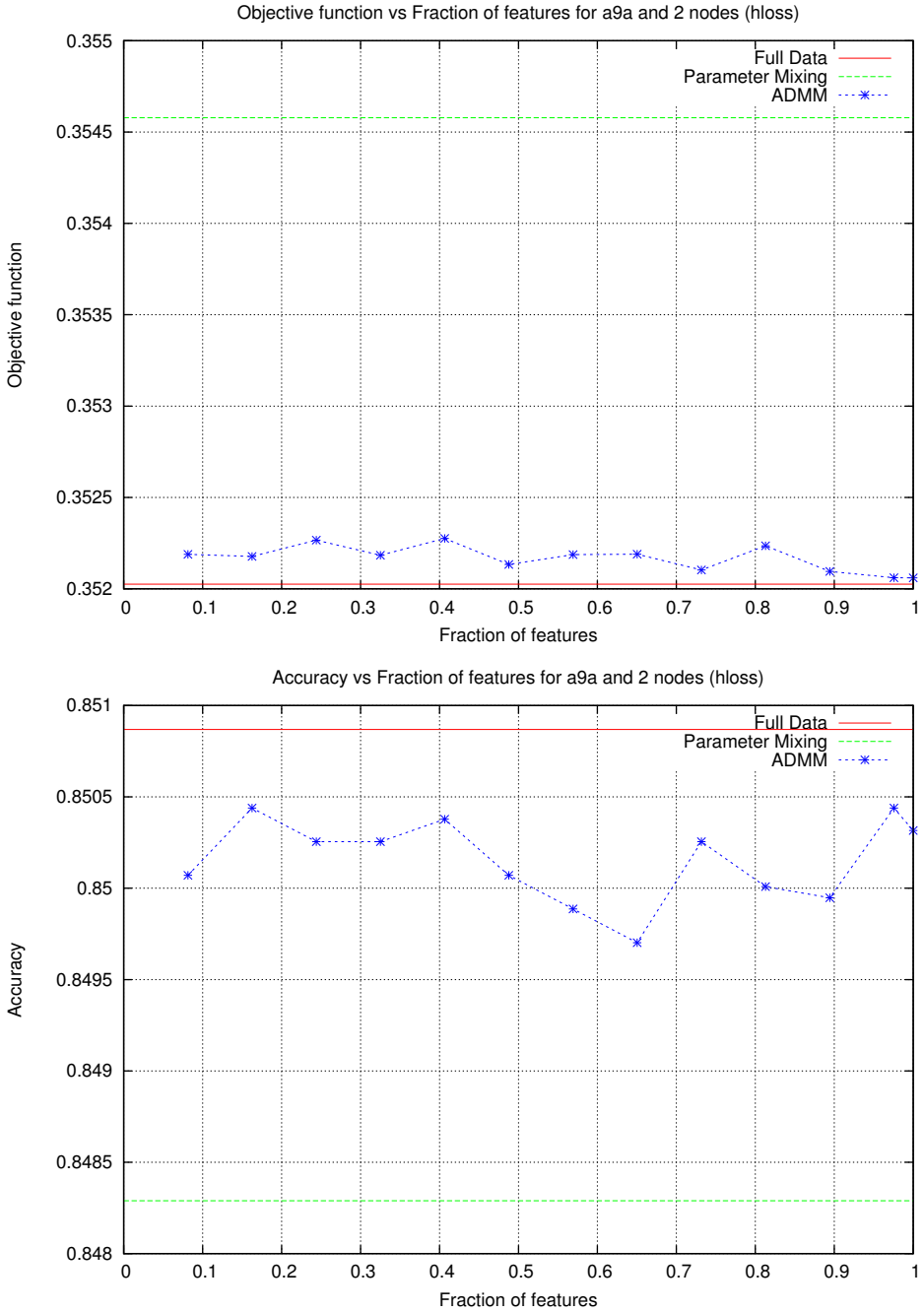

Figure 5.5: a9a split into two nodes: objective function value and accuracy

For some datasets, performance is roughly same for all values of k. For such datasets, we can save a lot on communication cost without greatly affecting performance

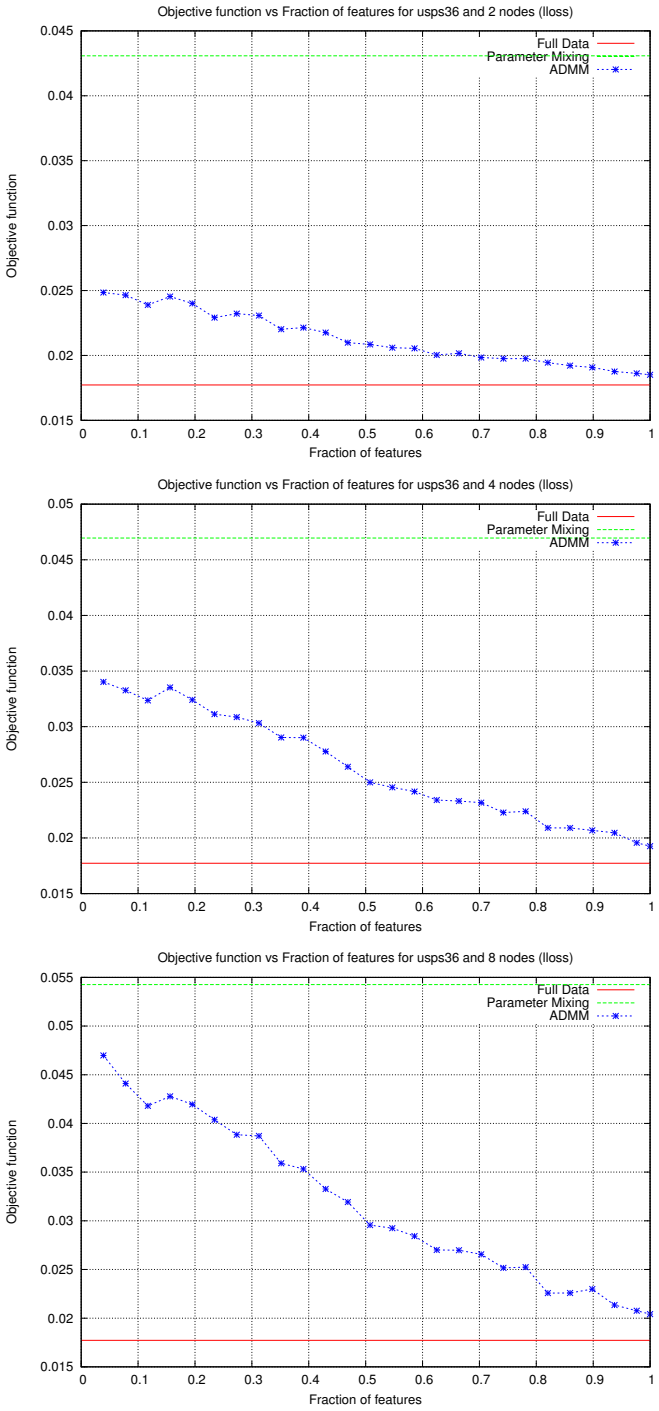

Figure 5.6: usps36: 2, 4 and 8 nodes respectively

For most datasets, however, we get better solutions on increasing k. We still do better than parameter mixing

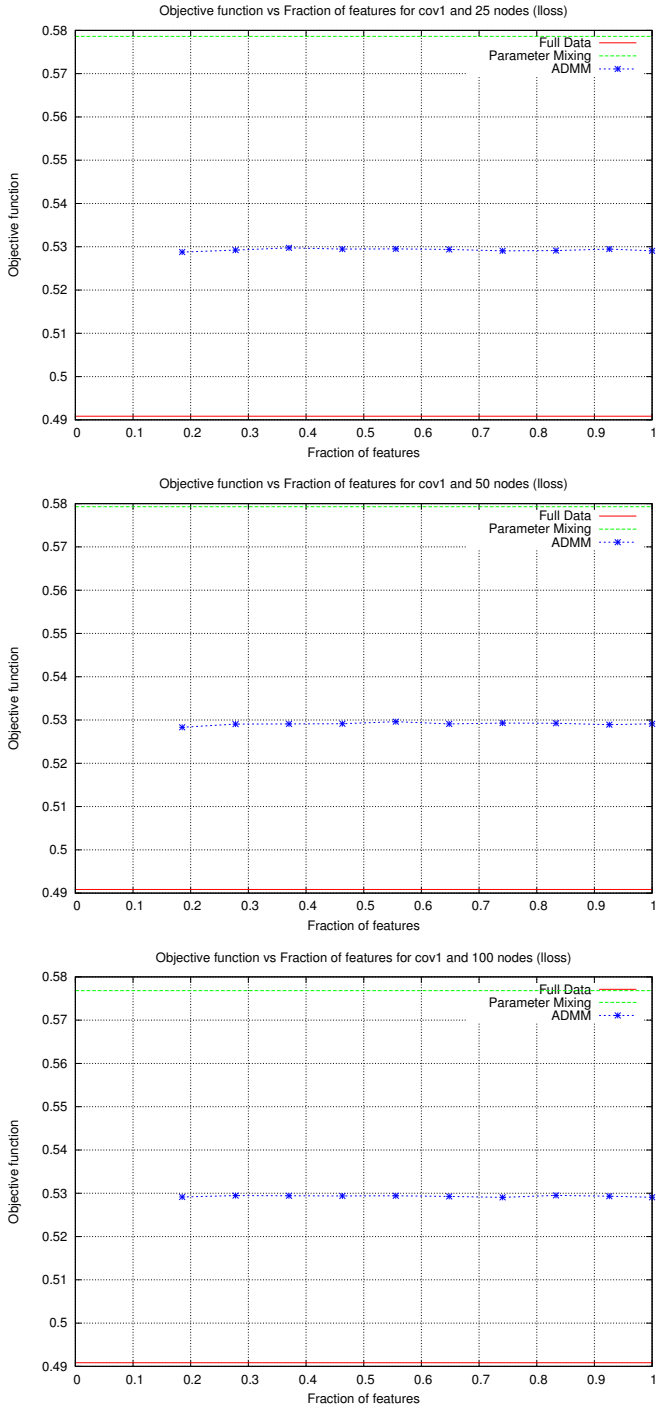

Figure 5.7: cov: 25, 50 and 100 nodes respectively: objective function value

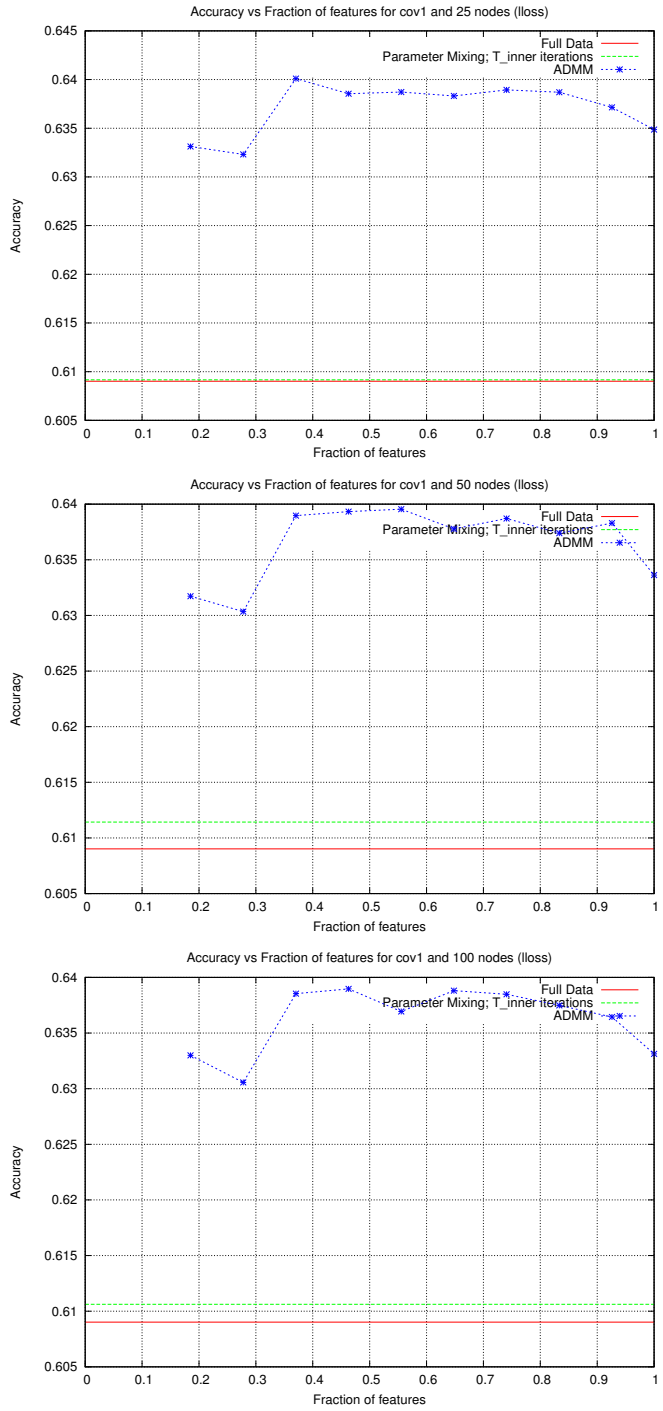

Figure 5.8: cov: 25, 50 and 100 nodes respectively: accuracy

Note the poor accuracy of the red line despite achieving best minimization of the function value. This depends on the dataset

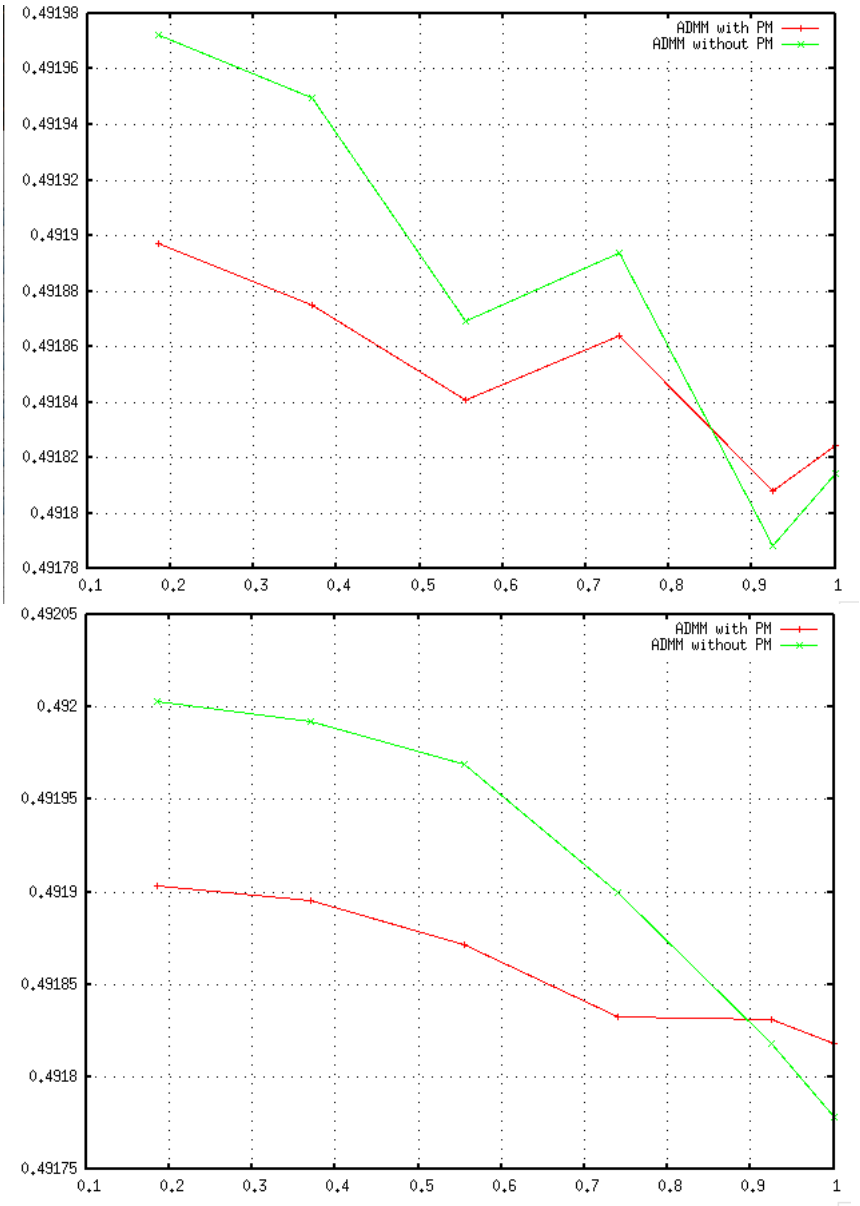

Figure 5.9: cov: 4 and 8 nodes respectively: Comparision of objective function values ADMM with PM and ADMM without PM

Notice that the difference is very small (same up to second decimal place). This means that we can skip the last parameter mixing step to have a smaller communication cost than simple parameter mixing too

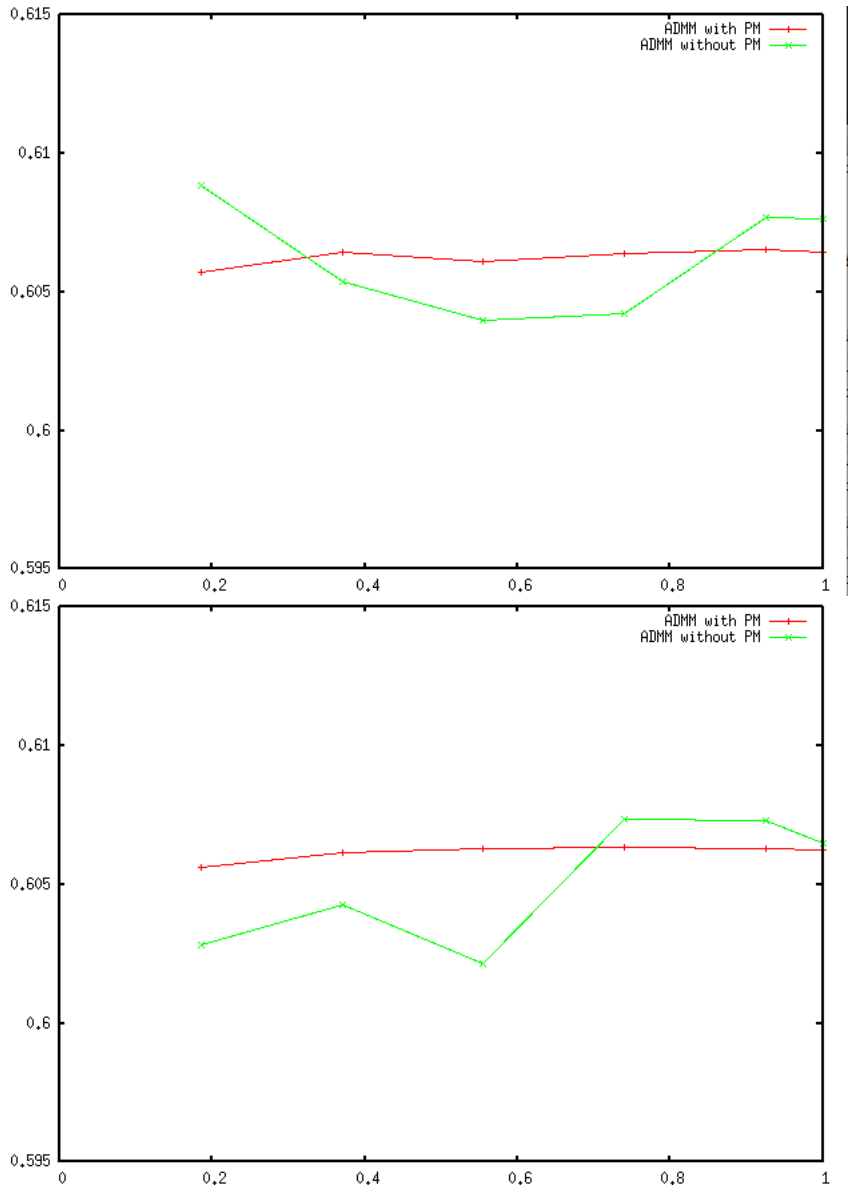

Figure 5.10: cov: 4 and 8 nodes respectively: Comparision of test accuracy values ADMM with PM and ADMM without PM

Notice that the difference is, again, very small.

Chapter 6

Conclusion

IPM is promising. Since it is compatible with Hadoop, it can be scaled. Part of the future work to be done is to run experiments on hadoop with bigger datasets. Work has to be done to tune some of the parameters of the inner optimization. We were unable to prove linear convergence for our version of IPM. We wish to explore some other means to see if linear convergence can be guaranteed without line search.

Our variant of ADMM can compete with simple parameter mixing and beat it too, in certain circumstances, in both accuracy and communication overhead. While it lacks (at the moment) formal theoretical guarantees, it does a good job empirically. A hadoop implementation is necessary to work with huge datasets that are otherwise intractable on a single machine. Further, we wish to come up with a **theoretical basis** for this method. It is driven by the intuition that, for data sampled from the same distribution, the models that fit different samples should be close in expectation.

Another aspect we wish to work upon is **grouping of features** into sets I_j as a pre-processing step on the data. For synthetic datasets, elements were grouped into sets by hand. For the real-world datasets, grouping was arbitrary (based on order in which they appeared). Intuition says that grouping similar features will give a better approximation than arbitrarily grouping features. Two methods of grouping similar features comes to mind: using clustering algortihms or optimizing on a small, random sample of the data and group features based on the value of the model obtained. We wish to theoretically and experimentally validate these observations.

Bibliography

[1] O. L. Mangasarian. *Parallel Gradient Distribution*.

[2] Gideon Mann et al. "Efficient large-scale distributed training of conditional maximum entropy models". In: *In Advances in Neural Information Processing Systems*. 2009.

[3] Alekh Agarwal et al. "A Reliable Effective Terascale Linear Learning System". In: *CoRR* abs/1110.4198 (2011).

[4] Ryan Mcdonald, Keith Hall, and Gideon Mann. *Distributed Training Strategies for the Structured Perceptron*.

[5] Dhruv Mahajan et al. "A Functional Approximation Based Distributed Learning Algorithm". In: *CoRR* abs/1310.8418 (2013).

[6] C S. Boyd et al. *Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers*. 2011.

[7] D.P. Bertsekas and J.N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods*. Athena scientific optimization and computation series. Athena Scientific, 1997. ISBN: 9781886529014. URL: `http://books.google.co.in/books?id=Brd4QgAACAAJ`.

[8] Tamir Hazan, Amit Man, and Amnon Shashua. "A Parallel Decomposition Solver for SVM: Distributed Dual Ascend using Fenchel Duality". In: (2008).

[9] Y. Nesterov and I.U.E. Nesterov. *Introductory Lectures on Convex Optimization: A Basic Course*. Applied Optimization. Springer, 2004. ISBN: 9781402075537. URL: `http://books.google.co.in/books?id=VyYLem-l3CgC`.

[10] Chih-Jen Lin, Ruby C. Weng, and S. Sathiya Keerthi. "Trust Region Newton Method for Logistic Regression". In: *J. Mach. Learn. Res.* 9 (June 2008), pp. 627–650. ISSN: 1532-4435. URL: `http://dl.acm.org/citation.cfm?id=1390681.1390703`.