

UNIVERSITÀ DEL SALENTO



Facoltà di Ingegneria  
Corso di Laurea Magistrale in Computer Engineering

Advanced Control Techniques Project  
**Multinomial Logistic Regression for a  
Supervised Learning problem**

Professor: **Giuseppe Notarstefano**

Students: **Alfarano Gianluca,  
Basile Davide,  
Capoccia Leonardo,  
Larini Ludovico**

Academic year 2017/2018



# Abstract

In this report, it will be shown how to solve a Multinomial Logistic Regression (also known as *Softmax Regression*) using a distributed method. The sub-gradient method has been used to distribute calculations among a configurable number of agents. A portion of the dataset is given to each agent and used to minimize a cost function related to the portion of the dataset in its possession.

# Contents

<b>Introduction</b>	<b>6</b>
<b>1 Chapter 1 Problem and its implementation</b>	<b>7</b>
1.1 Theory of the problem . . . . .	7
1.1.1 Distributed Subgradient Methods for Multi-Agent Op- timization . . . . .	7
1.1.2 Multinomial Logistic Regression . . . . .	8
1.1.3 Pseudocode . . . . .	9
1.2 Code Implementation . . . . .	9
<b>2 Chapter 2 Results of simulations</b>	<b>10</b>
<b>Conclusions</b>	<b>11</b>
<b>Bibliography</b>	<b>12</b>

# List of Figures

# Introduction

In the past there was a single *Mainframe* that executed all digital operations. Years after, with the creation of the Personal Computer, more people could execute the same operations in private. Today's *Microcontrollers* allow to make smart an in finity of devices. More algorithms have been created to connect these devices to distribute.

This work implements a scenario in which there are some agents that estimate a cost function using their own information and those of the other agents; they use a *Distributed Sub-gradient Method* to update their own estimate and, in particular, they resolve a *Multinomial Logistic Regression*. After some test in *MATLAB*, it is used *MPI* implemented with *Python*.

The present work is divided into two chapters. In Chapter 1, it is introduced the theory behind the problem and it is visualized and commented the implementation code. In Chapter 2 there are the results of simulations with some considerations.

# Chapter 1

## Chapter 1 Problem and its implementation

First-chapter for problem set-up and description of the implemented solution.

### 1.1 Theory of the problem

#### 1.1.1 Distributed Subgradient Methods for Multi-Agent Optimization

In this problem there are  $m$  agents that cooperatively minimize a common additive cost. The optimization general problem is:

$$\underset{x}{\text{minimize}} \quad \sum_{i=1}^m f_i(x) \quad \text{subject to} \quad x \in \mathbb{R}^n, \quad (1.1)$$

where  $f_i : \mathbb{R}^n \rightarrow \mathbb{R}$  is the cost function of agent  $i$ , known by this agent only, and  $x \in \mathbb{R}$  is a decision vector. It is assumed that:

- The cost function is convex;
- the agents are distributed over a time-varying topology;
- the graph  $(V, E_\infty)$  is connected, where  $E_\infty$  is the set of edges  $(j, i)$  representing agent pairs communicating directly infinitely many times;
- there isn't communication delay.

Every agent  $i$  generates and maintains estimates of the optimal decision vector based on information concerning its own cost function and exchanges this estimate with its directly neighbors at discrete times  $t_0, t_1, t_2, \dots$ . Moreover, each agent  $i$  has a vector of weights  $a^i(k) \in \mathbb{R}^m$  at any time  $t_k$ ; for each time, the scalar  $a_i^j(k)$  is zero if the agent  $i$  doesn't directly communicate

with  $j$ , else it is the weight assigned from the agent  $i$  to the information  $x^j$  obtained from  $j$  during the time interval  $(t_k, t_{k+1})$ . The estimates are updated according to the update rule:

$$x^i(k+1) = \sum_{j=1}^m a_j^i(k) x^j(k) - a^i(k) d_i(k) \quad (1.2)$$

where  $\alpha^i(k) > 0$  is the (diminishing) stepsize used by agent  $i$  and the vector  $d_i(k)$  is a subgradient of agent  $i$  objective function  $f_i(x)$  at  $x = x^i(k)$ . [1]

### 1.1.2 Multinomial Logistic Regression

The problem to be solved is a Supervised Learning problem called Multinomial Logistic Regression, also known as Softmax Regression, and it generalizes the more common Logistic Regression. The difference between them is that in the latter there are several classes to be considered, in the matter, there are only two classes (or equivalently a binary class).

The problem to be solved is to find a set of coefficients based on a given dataset to predict the belonging class for an unseen set of features, while minimizing a cost function. The dataset is composed of  $N$  labelled examples  $\{(x^{(1)}, y^{(1)}), \dots, (x^{(N)}, y^{(N)})\}$ . Each  $x^{(i)} \in R^{d_x}$  for  $i = 1, \dots, N$  is composed of some features which represent the value upon which we base the estimation of the belonging class, while  $y^{(i)} \in R^{d_y}$  is the belonging class for the  $i$ -th example, and can be a values in  $\{1, \dots, K\}$ .

Given a single training example  $(x^{(i)}, y^{(i)})$ , the definition of the cost function is:

$$f_i(\omega) := \left\| h_\omega(x^{(i)}) - y^{(i)} \right\|^2 \quad (1.3)$$

where the  $\omega \in R^{d_x}$  are the weights of the hypothesis function  $h_\omega$ . The overall cost function can be defined as:

$$f(\omega) := \sum_{i=1}^N f_i(\omega) \quad (1.4)$$

We solve the problem by finding the solution of the following optimization problem:

$$\omega^* := \arg \min_{\omega} f(\omega) \quad (1.5)$$

In the Multinomial Logistic Regression, a common choice for the hypothesis function is the following:

$$h_\theta = \frac{1}{\sum_{j=1}^K \exp(\theta^{(j)\top} x)} \begin{bmatrix} \exp(\theta^{(1)\top} x) \\ \vdots \\ \exp(\theta^{(K)\top} x) \end{bmatrix} \quad (1.6)$$



where the weights  $\omega = \theta = (\theta^{(1)}, \dots, \theta^{(K)}) \in R^{d_x}$ .

Using this function, the solution of the problem is given by finding:

$$\theta^* = \arg \min_{\theta} - \sum_{i=1}^N g_i(\theta) \quad (1.7)$$

with

$$g_i(\theta) := \sum_{k=1}^K \mathbf{1}\{y^{(i)} = e_k\} \log \left( \frac{\exp(\theta^{(k)\top} x^{(i)})}{\sum_{k=1}^K \exp(\theta^{(j)\top} x)} \right) \quad (1.8)$$

where  $\mathbf{1}\{\cdot\}$  being the *indicator function*.

### 1.1.3 Pseudocode

---

#### Algorithm 1

---

- 1: *Stop Rules:*
  - 2:  $\|x_{k+1} - x_k\| \leq \varepsilon$      $\varepsilon$  fixed
  - 3: Number of maximum iterations reached
  - 4: **Start:**
  - 5: Fix initial conditions for each node  $x_i(0) = [0 \quad \dots \quad 0]^T$
  - 6: Define the Adjacency Matrix, Weights Matrix,  $\alpha^i = \alpha$  constant for each iteration
  - 7: **while** No stop rule is true, each node  $i$  does: **do**
  - 8:    calculate  $\nabla f_i$
  - 9:    **for** each neighbor  $j$  **do**
  - 10:      $x_i(k+1) = x_i(k+1) + a_j^i(k)x^j(k)$
  - 11:    **end for**
  - 12:     $x_i(k+1) = x_i(k+1) - \alpha \nabla f_i$
  - 13: **end while**
  - 14: **Result:**
  - 15: Each node  $i$  should converge to  $x^*$
  - 16: The minimum of function is  $\sum_{i=1}^m f_i(x^*)$
- 

## 1.2 Code Implementation

## Chapter 2

# Chapter 2 Results of simulations

Second chapter for description of the results (simulations and experiments where applicable).

Citation [2]

# Conclusions

In this work it has been resolved a Multinomial Logistic Regression problem using MPI in Python. Each agent used a Distributed Sub-gradient method to update its own estimate of optimal solution.

# Bibliography

- [1] A. Nedic and A. Ozdaglar. Distributed subgradient methods for multi-agent optimization. *IEEE Transactions on Automatic Control*, 2009.
- [2] Stanford. Softmax regression.