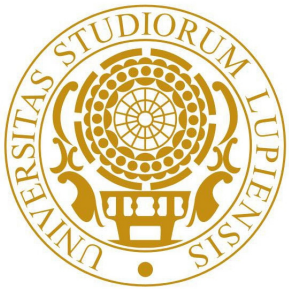
UNIVERSITÀ DEL SALENTO



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

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:

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

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

- The cost function is convex;
- the agents are distributed over a time-varying topology;
- the graph (V, E_∞) is connected, where E_∞ 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 $t_0, t_1, t_2, ...$ 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 comunicate

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:

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

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 $\{(x^{(1)}, y^{(1)}), ..., (x^{(N)}, y^{(N)})\}$. Each $x^{(i)} \in R^{d_x}$ for $i = 1, ..., N$ 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 $\{1, ..., K\}$.

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

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

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

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

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

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

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

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

where the weights $\omega = \theta = (\theta^{(1)}, ..., \theta^{(K)}) \in R^{d_x}$.
Using this function, the solution of the problem is given by finding:

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

with

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

where $\mathbf{1\{\cdot\}}$ being the *indicator function.* [2]

1.1.3 Pseudocode

Algorithm 1

Stop Rules:
$||x_{k+1} - x_k|| \leq \varepsilon$ $\quad$ ε fixed
Number of maximum iterations reached
Start:
Fix initial conditions for each node $x_i(0) = [0 \quad ... \quad 0]^T$
Define the Adjancency Matrix, Weights Matrix, $\alpha^i = \alpha$ constant for each iteration
while No stop rule is true, each node i does: **do**
    calculate ∇f_i
    for each neighbor j **do**
        $x_i(k+1) = x_i(k+1) + a_j^i(k)x^j(k)$
    end for
    $x_i(k+1) = x_i(k+1) - \alpha \nabla f_i$
end while
Result:
Each node i should converge to x^*
The minimum of function is $\sum_{i=1}^{m} f_i(x^*)$

1.2 Code Implementation

Chapter 2

Chapter 2 Results of simulations

The software described in the previous chapter was used to solve a Multinomial Logistic Regression problem. Specifically, we classified the data of the Iris Dataset.

2.1 Dataset, graph description and minimization function

This dataset is composed of 150 instances, 120 used for training, the rest for tests. The instances contain 3 classes, each representing a type of Iris flower. Every instance has 4 features, sepal length, sepal width, petal length, petal width expressed in *cm*. We tried different types of graphs. These graphs are all strongly connected and the weight matrices for the nodes are doubly stochastic, as per the assumptions of convergence of the algorithm described in Chapter 1. The results discussed in this chapter, if not differently noted, refer to cyclic graphs with a variable number of nodes. The program can minimize all kinds of loss functions. As shown in Chapter 1.2, the quadratic and exponential functions can also be used. These last 2 functions don't guarantee useful results and/or convergence. The minimization function used in this chapter is the one described in Chapter 1.1.2, softmax.

2.2 Performance

There are some key factors that influence *the computational time*, *the numbers of iterations necessary* and *the accuracy of the results.*

2.2.1 Number of nodes

The Python program, thanks to the MPI platform, is capable of running on an arbitrary number of nodes. It was tested on as little as 2 nodes to as many as 60 nodes, which means that every node was processing the data of 2 instances (120 instances divided into 60 nodes). The best performances are obtained when the number of nodes corresponds to the number of physical cores of the machine where it runs. When the number of nodes exceeds greatly the number of physical cores, the resources are oversubscribed. In this case, the performances degrade notably as the nodes compete for cache and memory and the processors' schedulers are put in a difficult situation. On a 4-core test machine, a computation with 5000 iterations and 30 nodes is done in 5 minutes. The same machine can do the same number of iterations, but with 60 nodes, in 15 minutes. Therefore, the following tests will be shown on a 4 nodes setup.

2.2.2 Epsilon

This is a small constant used as stop condition. If the result of the current calculation differs less than epsilon from the previous, the algorithm is stopped.

2.2.3 Learning rate

2.2.4 Fixed step-size

The step-size alpha plays a big role in the speed of convergence of the algorithm. There are 3 kinds of step-size. Fixed, diminishing and adaptive (Armijo). For reasons not discussed in this paper, it's not possible to use Armijo rule in a distributed problem. We will first deal with a simpler fixed step-size. With an epsilon equal to $0.001(10^{-3})$:

Value of fixed step-size	**Iteration required**	**Execution time in s**	**Wrong guesses o.30**
0.5	4601	6.8	1
0.1	2109	3.24	1
0.05	2342	3.68	0
0.01	1153	2.05	2
0.005	680	1.14	9
0.001	22	0.05	22

These results show a general truth about the step-size. If it is too little the learning process proceeds in a very slow way and it requires a huge amount of iterations. If the learning rate is too high and the gradient descent most probably will overshoot the minimum and not converges. Through trial and

error, the step-size 0.001 was identified which allow reaching good performance and accuracy. In fact, in only 2.6 seconds, we can make predictions with no errors, using our 30 instances test dataset.

2.2.5 Diminishing step-size

The diminishing step-size implemented in the code is in this form:
Again, several tests were run by tweaking the constant and the exponent.

psi_coeff	**alpha_exp coefficient**	**Iteration required**	**Execution time in s**	**Wrong guesses o. 30**
1	0.01	>20000	>30	13
1	0.1	4033	6.66	1
0.1	0.01	2130	5.19	1
0.1	0.1	2028	3.07	0
0.01	0.01	1067	1.60	2
0.01	0.1	628	1	8

The best result in term of accuracy is done with both parameters set to 0.1. The computation time is similar to the fixed-step size. *Reducing the epsilon to 0.01, allow the program to reach consensus in 604 with parameters 0.01 and 0.1 and zero error.*

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.