Treball final de màster

MÀSTER DE MATEMÀTICA AVANÇADA

Facultat de Matemàtiques
Universitat de Barcelona

Multinomial Logistic Regression and Stochastic Natural Gradient Descent

Autor: Borja Sánchez López

Director: Dr. Jesús Cerquides Bueno
Realitzat a: Departament de Matemàtiques I Informàtica

Barcelona, 11 de setembre de 2018

Abstract

Function optimization is a widely faced problem nowadays. Its interest, in particular, lies in every learning algorithm in AI, whose achievements are measured by a Loss-Function. On one hand, Multinomial Logistic Regression is a commonly applied model to engage and simplify the problem of predicting a categorical distributed variable which depends on a set of distinct categorical distributed variables. On the other hand, Gradient Descent allows us to reach local extrema of a smooth function. Moreover, large datasets force the use of online optimization.

Improving the convergence speed and reducing the computational cost of gradient based online learning algorithms will automatically translate into a significant enhancement on many machine learning processes.

In this text, we present a Stochastic Gradient Descent algorithm variant, specifically designed for Multinomial Logistic Regression learning problems by taking advantage of the geometry and the intrinsic metric of the space. We compare it to current most advanced stochastic algorithms, and we provide the favorable experimental results obtained.

Keywords: Function Optimization, Convex Function, Machine Learning, Linesearch, Online Optimization, Stochastic Gradient Descent, Riemannian Geometry, Fisher Metric, Natural Gradient, Exponential Family, Logistic Regression.

Contents

1 Motivation and Introduction

Function optimization is mainly approached nowadays with two strategies: Linesearch, which we will see in chapter 2, and Region Trust strategies [7]. We focus only on the first one, that tries to decrease iteratively the value of the function by "updating the parameter following a straight line with a chosen direction for a certain distance". Gradient Descent (GD) is an offline Linesearch algorithm that sets the direction to be the opposite of the function gradient. That makes sense, since gradient points to the steepest direction.

It is known that the effectiveness of Gradient Descent algorithm is parametrization dependent, as has been shown for example in [13]. This raises the question on how should we select the parametrization, which leads us to a brief study in Riemannian Manifolds in section 2.3, in order to understand the space metric we work on. Many texts showed that taking into account the metric of the parameter space allows a more direct path to the minimum [9, 11, 1]. They basically make use of the Natural Gradient, which is a variation of the function gradient according to the metric of the manifold. This new direction actually points to the steepest direction in the manifold [9]. We give our proof of this fact in 2.3.2. Setting the Natural Gradient into the Linesearch algorithm speeds up convergence in many problems. This different set up for the Linesearch gives rise to the Natural Gradient Descent (NGD).

Commonly, there is no more than an approximation, after a sample of observations, of the function we want to optimize. In these cases usually an online Linesearch is the best option, especially if the sample is large. Stochastic Gradient Descent (SGD) is an online Linesearch algorithm that iteratively computes the gradient of a piece of the function for a single observation and it updates after the Linesearch equation. Similarly, the Stochastic Natural Gradient Descent (SNGD) computes the Natural Gradient for every observation instead.

We are interested in better understanding the characteristics and qualities of SNGD for machine learning/stochastic applications.

Maximum Likelihood (ML) is a widely used estimation approach for density estimation problem. In chapter 3 the exponential family manifold, the Fisher Information Metric (FIM) and mean parametrization are defined to face the ML problem. Next, we prove that, in the exponential family manifold, SNGD using the mean parametrization together with FIM provides an online method for exactly assessing the ML estimator. Consequently, Natural Gradient arises as an interesting concept to use in harder problems.

Similarly, for classification problems, the Maximum Conditional Likelihood (MCL) is a common approach. An example is the well known Logistic Regression problem. In chapter 4 we formalize the MCL problem and we define the Multinomial Logistic Regression manifold, the metric and β parametrization to use in last chapter.

Chapter 5 starts proposing online estimators for the MCL task: We adapt the SNGD algorithm to the Multinomial Logistic Regression problem by selecting the parametrization and metric explained. This algorithm turns out to be too costly in computational terms due to the need to compute the inverse of the metric matrix at each update of the parameters. Moreover, it exploits the sample many more times than standard algorithms. That's why we define the MOD algorithm, a variation of SNGD which is much more efficient, since it computes the matrix inverse just once per epoch, and a fair competitor, since it scans the sample as many times as other algorithms. Finally, one last Stochastic Natural Gradient and really fast algorithm is defined, MEGD algorithm, that uses the metric of the maximum entropy point of the manifold.

Then in the chapter, we evaluate the error of the SNGD, MOD and MEGD estimators. First, we study how well SNGD, MOD and MEGD compare with standard approaches such as SGD and AdaGrad in terms of conditional likelihood of the training sample. That is, how well they perform as optimizers. We show that SNGD, MOD and MEGD improve over standard methods. Furthermore, we notice that MOD and MEGD are not only more efficient han SNGD, but also more accurate. Hence, we ignore SNGD soon for the remaining experiments.

Second, we compare MOD and MEGD with standard methods in terms of the expected prediction error, showing that it sligthly improves in large data scenarios. This error can be understood as the prediction capability of the estimator.

2 Function optimization and Gradient Descent

This section introduces basic concepts and strategies used to solve optimization problems [7, 2]. We start by giving a natural way to engage this kind of problems in which we can't find the solution analytically. Then, we are going to see what are the clever ways to apply this intuition, also identifying the limitations encountered.

2.1 Linesearch

Let $\mathcal{L}(\theta)$ be a smooth function from $\mathbb{R}^k$ to $\mathbb{R}$. Function optimization problem asks about the extrema of a function. More precisely, we try to find the point $\overset{*}{\theta} \in \mathbb{R}^k$ for which the function reaches a minimum. That is;

$$\overset{*}{\theta} := \arg\min_{\theta \in \mathbb{R}^k} \mathcal{L}(\theta)$$

Due to the complexity of the function, we suppose $\overset{*}{\theta}$ can not be found analytically, and we are forced to seek the solution numerically. The most used strategy to find $\overset{*}{\theta}$ is called Linesearch, which is very intuitive: starting with a guess $\theta^0 \in \mathbb{R}^k$ of the solution we update it into θ^1 in the surroundings such that $\mathcal{L}(\theta^0) \geq \mathcal{L}(\theta^1)$. We then repeat the process iteratively. Formally;

$$\theta^{i+1} = \theta^i - r_i \cdot g_i, \qquad r_i \in \mathbb{R} \quad g_i, \theta_i \in \mathbb{R}^k \tag{1}$$

By looking equation 1, we can say that at iteration i we "update the parameter θ^i into θ^{i+1} by following a straight line with direction g_i for a distance r_i". We call g_i and r_i the direction and the learning rate of iterate i, respectively. Different settings on g_i and r_i provide diferent Linesearch algorithms. More precisely, these are the steps;

Listing 1: Linesearch

```
beta:= beta_init
for i:=0 to max_it do
        r:= learning rate of iteration i
        g:= direction at current position
        beta = beta − r x g
end;
```

It is important to notice that Linesearch may seek a local minimum instead of the global minimum of the function. A study branch we will not discuss about dedicates the effort to solve this hard problem. The issue disappears if the function is convex, and we assume the function has this property from now on.

2.2 Gradient Descent

Assume that $\mathcal{L}(\theta)$ is differentiable. Since our goal is to reach a minimum of the function, it's natural to think about using the steepest direction as g_i in the update, in order to decrease the function value the fastest possible.

The gradient vector $\nabla\mathcal{L}(\theta^i)$ points to the steepest direction. To see this, first realize that steepest direction is achieved by the normalized vector that maximizes the directional derivative. So, normalized steepest direction is the solution to;

$$\underset{v\in\mathbb{R}^k, ||v||^2=1}{\arg\max} \quad d_{\theta^i}\mathcal{L}(v) \tag{2}$$

now observe that the expression we want to maximize is;

$$d_{\theta^i}\mathcal{L}(v) =< \nabla\mathcal{L}(\theta^i), v >= ||\nabla\mathcal{L}(\theta^i)|| \cdot ||v|| \cos(\alpha) = ||\nabla\mathcal{L}(\theta^i)|| \cos(\alpha)$$

where α is the angle between $\nabla\mathcal{L}(\theta^i)$ and v. In the latest expression, the only term that depends on v is α, while $||\nabla\mathcal{L}(\theta^i)||$ is constant. Since $cos(\alpha) \leq 1$, we reach a maximum when $\cos(\alpha) = 1$, that is when $\alpha = 0$, implying that the normalized steepest direction vector and $\nabla\mathcal{L}(\theta^i)$ are proportional as we said.

Choosing the gradient as the direction g_i in equation 1 ensures that Linesearch updates accomplish $\mathcal{L}(\theta^i) \geq \mathcal{L}(\theta^{i+1})$ for a small enought learning rate r_i . There are several options to choose as r_i value – constant, $\frac{a}{1+bi}$, adaptive methods,... –. As suggested by [8], this project assumes the learning rate to be $r_i = \frac{a}{1+bi}$ for some $a, b \in R^+$ which will be decided after a short training – see section 5.

Gradient Descent algorithm (GD) updates the parameter θ^i according to equation 1 using the following setting;

- $g_i = \nabla\mathcal{L}(\theta^i)$
- $r_i = \frac{a}{1+bi}$ for some $a, b \in \mathbb{R}^+$.

2.3 Natural Gradient Descent

In most occasions SGD is run, $\mathcal{L}(\theta)$ is not really a function from $\mathbb{R}^k$. We have instead;

$$\begin{aligned} \mathcal{L} : \mathcal{M} &\longrightarrow \mathbb{R} \\ p &\longmapsto \mathcal{L}(p) \end{aligned}$$

where $\mathcal{M}$ is a k-dimensional Riemannian manifold equipped with a metric G_p for every $p \in \mathcal{M}$.

To work on a manifold, we are forced to use a parametrization $\phi : \mathring{U} \subset \mathbb{R}^k \to \mathcal{M}$ and so work with the composition $\mathcal{L}(\phi(\theta))$;

$$\begin{array}{ccccc} \mathbb{R}^k & \xrightarrow{\phi} & \mathcal{M} & \xrightarrow{\mathcal{L}} & \mathbb{R} \\ \theta & \mapsto & \phi(\theta) & \longmapsto & \mathcal{L}(\phi(\theta)) \end{array}$$

At this point we agree θ still belongs to $\mathbb{R}^k$ and we can still follow the direction given by the gradient of $\mathcal{L}(\phi(\theta))$ by means of the parametrization. However, this doesn't coincide with the steepest direction on the manifold in general. The parametrization doesn't give any information about the metric on $\mathcal{M}$, and metric magnitudes of actual space $\mathcal{M}$ are distorted viewed from $\mathbb{R}^k$. The vector truly pointing the steepest direction in $\mathcal{M}$ is called Natural Gradient and it is written as $\widetilde{\nabla}\mathcal{L}(\phi(\theta))$ in the references [11, 9]. Let's see an example that shows the fact;

Example. Let $S_2 = \{(x, y, z) \in \mathbb{R}^3 : x, y, z > 0, x+y+z = 1\}$ be the open surface simplex with the induced metric from $\mathbb{R}^3$. Consider the orthogonal projection parametrization of S_2 along axis z ;

$$\begin{array}{rcl} \phi : U \subset (0,1)^2 & \longrightarrow & S_2 \\ (a, b) & \longmapsto & (a, b, 1-a-b) \\ (x, y) & \longleftarrow & (x, y, z) \end{array}$$

where $U = \{(a, b) \in (0, 1)^2 : a + b < 1\}$.

Finally, let $\mathcal{L}(x, y, z) = y$ be a function defined on S_2 that we try to maximize. Starting at point $p = (\frac{1}{3}, \frac{1}{3}, \frac{1}{3}) \in S_2$, we proceed to compute the gradient vector with respect to the parametrization coordenates;

$$\nabla\mathcal{L}(p) = \nabla_{(a,b)} b_{|\phi^{-1}(p)} = (0, 1)$$

This vector is drawn in red continuous arrow in Figure 1, in both U and S_2. However, we can see in the actual space S_2, that the best way to increase the value of $\mathcal{L}(x, y, z) = y$ is to move towards y axis, represented by the blue discontinuous vector in Figure 1. Then, the steepest direction actually is ;

$$\widetilde{\nabla}\mathcal{L}(p) = (-0.33, 0.66)$$

Intuitively, we see that Natural Gradient direction actually increases the value of $\mathcal{L}$ faster. We will see a proof of this after we recall some basic definitions on Riemannian Manifolds.

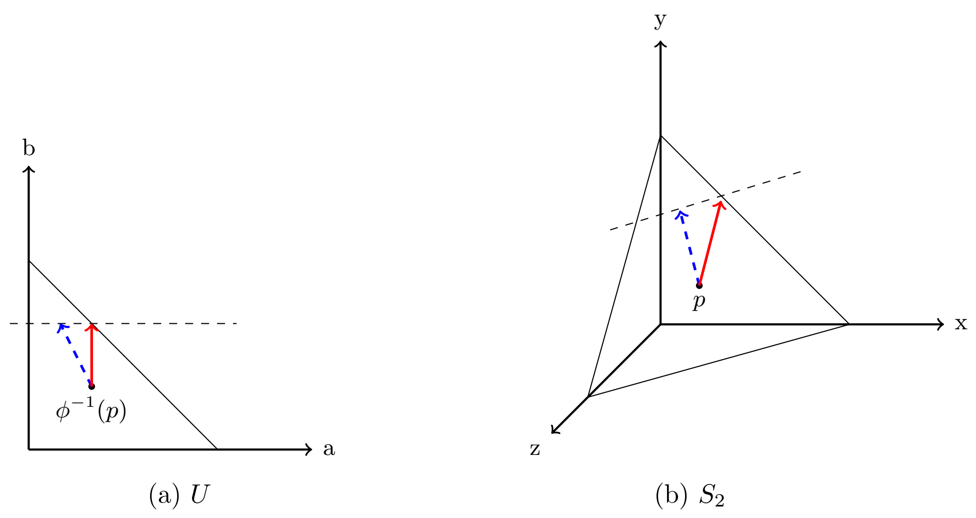


Figure 1: Two directions on the simplex. In red continuous, regular gradient. In blue discontinuous, Natural Gradient.

2.3.1 Brief Introduction to Riemannian Manifolds

Some basic definitions on Riemannian Geometry are given now. We recall only the concepts of [4] that are needed later on in this project, and we strongly recommend to check the reference for a more extended material.

Definition 2.1. A differentiable manifold of dimension k is a set $\mathcal{M}$ and a family of injective mappings $\phi_i : \mathring{U}_i \subset \mathbb{R}^k \to \mathcal{M}$ of open sets $\mathring{U}_i$ such that;

- $\bigcup_i \phi_i(\mathring{U}_i) = \mathcal{M}$
- for any pair ϕ_i, ϕ_j with $\phi_i(\mathring{U}_i) \cap \phi_j(\mathring{U}_j) = V \neq \emptyset$, the sets $\phi_i^{-1}(V)$ and $\phi_j^{-1}(V)$ are open sets in $\mathbb{R}^k$ and the mappings $\phi_j^{-1} \circ \phi_i$ and $\phi_i^{-1} \circ \phi_j$ are differentiable.

The pair $(\mathring{U}, \phi\ )$ – or just ϕ – is called a parametrization of $\mathcal{M}$ at any $p \in \phi(\mathring{U})$. We want now to meet the concept of tangent space $T_p\mathcal{M}$ of $\mathcal{M}$ at a point $p \in \mathcal{M}$. To do so, we first notice that thanks to parametrization existence, we are able to define differentiable functions between two manifolds.

Definition 2.2. Let $\mathcal{M}_1, \mathcal{M}_2$ be differentiable manifolds of dimension n and m. A mapping $f : \mathcal{M}_1 \to \mathcal{M}_2$ is differentiable at $p \in \mathcal{M}_1$ if given a parametrization $\phi_2 : V \subset \mathbb{R}^m \to \mathcal{M}_2$ at $f(p)$ there exists a parametrization $\phi_1 : U \subset \mathbb{R}^m \to \mathcal{M}_1$ at p such that $f(\phi_1(U)) \subset \phi_2(V)$ and the mapping $\phi_2^{-1} \circ f \circ \phi_1 : U \subset \mathbb{R}^n \to \mathbb{R}^m$ is differentiable at $\phi_1^{-1}(p)$. f is differentiable on an open set of $\mathcal{M}_1$ if it is differentiable at all of the points in this open set.

Definition 2.3. Let $\mathcal{M}$ be a differentiable manifold. A differentiable function $\lambda : (-\epsilon, \epsilon) \to \mathcal{M}$ is called a (differentiable) curve in $\mathcal{M}$. Suppose that $\lambda(0) = p \in \mathcal{M}$, and let D be the set of functions on $\mathcal{M}$ that are differentiable at p. The tangent vector to the curve λ at $t = 0$ is a fucntion $\lambda'(0) : D \to \mathbb{R}$ given by;

$$\lambda'(0)f = \frac{d(f \circ \lambda)}{dt}|_{t=0}, \qquad f \in D$$

Definition 2.4. With the same notation as in previous definition, a tangent vector at p is the tangent vector at $t = 0$ of some curve $\lambda : (-\epsilon, \epsilon) \to \mathcal{M}$ with $\lambda(0) = p$. The set of all tangent vectors to $\mathcal{M}$ at p is $T_p\mathcal{M}$.

Note that in the above definitions, if $\phi : U \subset \mathbb{R}^n \to \mathcal{M}$ is a parametrization, then;

- $f \circ \phi(\theta) = f(\theta_1, ..., \theta_n)$, where $\theta = (\theta_1, ..., \theta_n) \in U$
- $\phi^{-1} \circ \lambda(t) = (\lambda_1(t), ..., \lambda_n(t))$

So we can express the tangent vector $\lambda'(0)$ at p in this parametrization;

$$\lambda'(0)f = \frac{d(f \circ \lambda)}{dt}|_{t=0} = \frac{d}{dt} f(\lambda_1(t), ..., \lambda_n(t))|_{t=0} = \left(\sum_{i=1}^{n} \lambda_i'(0) \left(\frac{\partial}{\partial \theta_i} \right)_0 \right) f$$

Definition 2.5. A Riemannian metric (or Riemannian structure) on a differentiable manifold $\mathcal{M}$ is a correspondence which associates to each point $p \in \mathcal{M}$ an inner product $< \cdot, \cdot >_p$ that is, a symmetric, bilinear, positive-definite form, on the tangent space $T_p\mathcal{M}$, which varies differentiably in the following sense: If $\phi : U \in \mathbb{R}^n \to \mathcal{M}$ is a parametrization around p, with $\phi(\theta_1, ..., \theta_n) = q \in \phi(U)$ and $\frac{\partial}{\partial \theta_i}(q) = d\phi_q(0, ..., 1, ..., 0)$ then $< \frac{\partial}{\partial \theta_i}(q), \frac{\partial}{\partial \theta_j}(q) >_q = g_{ij}$ is a differentiable function on U.

Observe that if we define the matrix $G_q = g_{ij}$ then the inner product of two vectors $u, v \in T_q\mathcal{M}$ is $< u, v >_q = (u_1, ..., u_n)^T \cdot G_q \cdot (v_1, ..., v_n)$ where both vectors are expressed in the base $\{\frac{\partial}{\partial \theta_i}(q)\}_i$. Previous definition does not depend on the choice of parametrization, so for all $p \in \mathcal{M}$ we will say G_p even if no parametrization is specified.

The pair $(\mathcal{M}, \{G_p\}_{p \in \mathcal{M}})$ is known as a Riemannian Manifold, where the inner product of two vectors $u, v \in T_p\mathcal{M}$ is $< u, v >_p = u^T G_p v$. Whenever there is no confusion, we will just write $< u, v > = u^T G v$.

Previous definition gives a way to measure vectors of $T_p\mathcal{M}$;

Definition 2.6. The length – or norm – $||v||$ of any vector $v \in T_p\mathcal{M}$ is $||v|| := < v, v >^{1/2} = (v^T G v)^{1/2}$.

Definition 2.7. Let $(\mathcal{M}, \{G_p\}_{p\in\mathcal{M}})$ be a Riemannian Manifold. The Riemannian Manifold with the dual metric – or inverse metric – is $(\mathcal{M}, \{G_p^{-1}\}_{p\in\mathcal{M}})$. Notice this is well defined since G_p^{-1} is also a symmetric, bilineal and positive-definite form. If $u, v \in T_p\mathcal{M}$, we denote the inner product with respect to the inverse metric as $< u, v >_* := u^T G^{-1} v$, and the norm as $||v||_* :=< v, v >_*^{1/2} = (v^T G^{-1} v)^{1/2}$.

2.3.2 Natural Gradient

Let $\mathcal{L} : \mathcal{M} \to \mathbb{R}$ be a differentiable function defined in a Riemannian Manifold $(\mathcal{M}, \{G_p\}_{p\in\mathcal{M}})$. We know that if $p \in \mathcal{M}$ then;

$$< u, v >= u^T \cdot G \cdot v, \qquad u, v \in T_p\mathcal{M}$$

and

$$||u||^2 :=< u, u >, \qquad u \in T_p\mathcal{M}$$

We define the Natural Gradient $\widetilde{\nabla}\mathcal{L}(p) \in T_p\mathcal{M}$ as a vector that points to the direction that increases the fastest the function $\mathcal{L}(p)$ in $\mathcal{M}$. That is, Natural Gradient is achieved by the normalized vector that maximizes the directional derivative;

$$\underset{v\in T_p\mathcal{M}, ||v||^2=1}{\arg\max} \; d_p\mathcal{L}(v) = \underset{v\in T_p\mathcal{M}, ||v||^2=1}{\arg\max} \; \nabla\mathcal{L}(p)^T \cdot v$$

similar to equation 2 but paying attention to the metric of $\mathcal{M}$ when normalizing. In particular, we will see that the Natural Gradient is obtained by the inverse of G;

$$\widetilde{\nabla}\mathcal{L}(p) = G^{-1} \cdot \nabla\mathcal{L}(p)$$

To be entirely convinced of this fact, check the proof in [9] or see our next result.

Proposition 2.1. Let $(\mathcal{M}, \{G_p\}_{p\in\mathcal{M}})$ be a Riemannian Manifold and $\mathcal{L} : \mathcal{M} \to \mathbb{R}$ be a smooth function defined on $\mathcal{M}$. Let $p \in \mathcal{M}$, then

$$\widetilde{\nabla}\mathcal{L}(p) = G^{-1} \cdot \nabla\mathcal{L}(p)$$

Proof. We need to find the steepest normalized vector of $T_p\mathcal{M}$, by solving;

$$\widetilde{v} = \underset{||v||^2=1}{\arg\max} \, \nabla\mathcal{L}(p)^T \cdot v, \qquad v \in T_p\mathcal{M}$$

which can be rewritten as;

$$\widetilde{v} = \underset{v\in T_p\mathcal{M}}{\arg\max} \, \nabla\mathcal{L}(p)^T \cdot \frac{v}{||v||^2} = \underset{v\in T_p\mathcal{M}}{\arg\max} \, \frac{\nabla\mathcal{L}(p)^T \cdot v}{< v, v >_p}$$

Recall that G and G^{-1} are in particular symmetric invertible matrices. Furthermore, G^{-1} can be seen as an automorphism in the vector space $T_p\mathcal{M}$. So equivalently we can solve

$$\widetilde{u} = \arg\max_{u \in T_p\mathcal{M}} \frac{\nabla\mathcal{L}(p)^T G^{-1} u}{< G^{-1}u, G^{-1}u >}$$

where we can recover the solution to our original problem by doing $\widetilde{v} = G^{-1}\widetilde{u}$. We continue

$$\widetilde{u} = \arg\max_{u \in T_p\mathcal{M}} \frac{\nabla\mathcal{L}(p)^T G^{-1} u}{(G^{-1}u)^T G (G^{-1}u)} = \arg\max_{u \in T_p\mathcal{M}} \frac{\nabla\mathcal{L}(p)^T G^{-1} u}{u^T (G^{-1})^T u}$$

Again G^{-1} is symmetric and $(G^{-1})^T = G^{-1}$;

$$\widetilde{u} = \arg\max_{u \in T_p\mathcal{M}} \frac{< \nabla\mathcal{L}(p), u >_*}{< u, u >_*} = \arg\max_{u \in T_p\mathcal{M}} < \nabla\mathcal{L}(p), \frac{u}{||u||_*} >_* =$$
$$= \arg\max_{u \in T_p\mathcal{M}, ||u||_*^2 = 1} < \nabla\mathcal{L}(p), u >_*$$

Then, by the same reasoning we applied to equation 2 in 2.2, this implies the solution of the problem is $\widetilde{u} = \frac{\nabla\mathcal{L}(p)}{||\nabla\mathcal{L}(p)||_*}$, wich finally implies that $\widetilde{v} = G^{-1}\widetilde{u} = G^{-1}\nabla\mathcal{L}(p)\lambda$ and then $\widetilde{\nabla}\mathcal{L}(p) = G^{-1}\nabla\mathcal{L}(p)$ as we wanted to prove. $\square$

2.3.3 Natural Gradient Descent

Now that the natural gradient has been introduced, and considering its nice properties, a new and interesting setting for the Linesearch scheme of listing 1 is available;

- $g_i = \widetilde{\nabla}\mathcal{L}(p) = G^{-1}\nabla\mathcal{L}(p)$
- $r_i = \frac{a}{1+bi}$ for some $a, b \in \mathbb{R}^+$.

2.4 Stochastic Gradient Descent

Previous algorithms are offline, which means they are fed with the function $\mathcal{L}(\theta)$ in every iteration to compute the gradient. Online algorithms instead, they are fed with a piece $\mathcal{L}(s_i, \theta)$ of the whole function that depends on observation s_i of iteration i.

This project focuses on this later case, where the function we want to optimize $\mathcal{L}_S(\theta)$ depends on a sample S. The function is a sum over the sample points, and that allows us to set differently the direction g in the Linesearch strategy. Let's formally state everything.

Let $S := \{s_0, s_1, ..., s_n\}$ with $s_i \in \Omega$ be a sample observed and let $\mathcal{L}_S(\theta)$ be a smooth convex function from $\mathbb{R}^k$ to $\mathbb{R}$ such that;

$$\mathcal{L}_S(\theta) := \sum_{i=0}^{n} \mathcal{L}(s_i, \theta), \qquad \theta \in \mathbb{R}^k$$

Similarly as before, our objective is finding

$$\overset{*}{\theta} := \arg\min_{\theta \in \mathbb{R}^k} \mathcal{L}_S(\theta)$$

but here we are not going to use GD on $\mathcal{L}_S(\theta)$. The idea is updating after every observation using $\nabla\mathcal{L}(s_i, \theta)$ instead of $\nabla\mathcal{L}_S(\theta)$ as g_i, so we read equation 1 sligthly different. With this set up, we are running Stochastic Gradient Descent (SGD) and it is asymptotically equivalent to GD. To see this, we show next result.

Lemma 2.2.

$$\mathbb{E}_\Omega[\nabla\mathcal{L}(s, \theta)] \approx \frac{1}{n+1} \nabla\mathcal{L}_S(\theta)$$

Proof. To start with, notice that;

$$\mathcal{L}_S(\theta) = \sum_{i=0}^{n} \mathcal{L}(s_i, \theta) = \sum_{s \in \Omega} \mathcal{L}(s, \theta) \cdot n_s \Longrightarrow \nabla\mathcal{L}_S(\theta) = \sum_{s \in \Omega} \nabla\mathcal{L}(s, \theta) \cdot n_s$$

where $n_s = \#\{i : 0 \leq i \leq n, s_i = s\}$. Also $\frac{n_s}{n+1} \approx p(s)$ and then;

$$\mathbb{E}_\Omega[\nabla\mathcal{L}(s, \theta)] = \sum_{s \in \Omega} \nabla\mathcal{L}(s, \theta) \cdot p(s) \approx \sum_{s \in \Omega} \nabla\mathcal{L}(s, \theta) \cdot \frac{n_s}{n+1} = \frac{1}{n+1} \nabla\mathcal{L}_S(\theta)$$

□

So SGD is basically following the same direction as GD, nevertheless it's way easier to compute $\nabla\mathcal{L}(s, \theta)$ than $\nabla\mathcal{L}_S(\theta)$. It has been proved that the convergence of SGD is asymptotically as good as GD [8] or Batch Gradient Descent – that is, when every s_i represents a set of observations instead of a single observation – .

We define the setting on equation 1 for SGD;

- $g_i = \nabla\mathcal{L}(s_i, \theta^i)$
- $r_i = \frac{a}{1+bi}$ for some $a, b \in \mathbb{R}^+$.

Observe that in this case, the gradient is only defined when $i \leq n$. Once it is $i = n$, the algorithm has run over an epoch. Then usually a permutation of the indices of elements in S is applied and the algorithm continues to run with the same but reordered set. This is repeated after every epoch. The next schema shows this;

Listing 2: Stochastic Gradient Descent scheme

```
beta:= beta_init
for e:= 0 to max_epochs do
        for i:= 0 to n do
                it = i+n*e
                r:= learning rate of iteration it
                g:= gradient( observation i of S )
                beta = beta - r x g
        permute elements on set S
        end;
end;
```

Many problems fit in better with online algorithms. For example, the Maximum Likelihood problem (ML), that aims to find the maximum of known likelihood function $\mathcal{L}_S(\theta)$ that depends on a sample S observed after an unknown probability distribution determined by the parameter $\bar{\theta}$. In fact, the point $\overset{*}{\theta}$ that maximizes $\mathcal{L}_S(\theta)$ defines the probability distribution that most likely generated the sample S. In these cases, the objective is normally to approach the unknown parameter $\bar{\theta}$ by doing an online optimization of the known function $\mathcal{L}_S(\theta)$ since $\overset{*}{\theta} \overset{|S|\to\infty}{\longrightarrow} \bar{\theta}$

2.5 Stochastic Natural Gradient Descent

If $\mathcal{L} : \mathcal{M} \to \mathbb{R}$ is a differentiable function as in 2.4 defined in a Riemannian Manifold $(\mathcal{M}, \{G_p\}_{p\in\mathcal{M}})$ that we want to optimize, we may redefine the SGD algorithm to obtain the Stochastic Natural Gradient Descent (SNGD). This takes into account the metric of $\mathcal{M}$. Using the same schema as in listing 2, the algorithm is set;

- $g_i = \widetilde{\nabla}\mathcal{L}(s_i, p) = G^{-1}\nabla\mathcal{L}(s_i, p)$
- $r_i = \frac{a}{1+bi}$ for some $a, b \in \mathbb{R}^+$.

Thus, the main difference between SGD and SNGD is that whilst SGD assumes that we are optimizing a function from $\mathbb{R}^k$ to $\mathbb{R}$ and uses the Euclidean metric as default, SNGD assumes we are optimizing a function defined over a specific Riemannian manifold and uses the metric provided by that manifold. Hence SNGD is a generalization of SGD to manifolds different from the classic $\mathbb{R}^k$ with the Euclidean metric.

3 ML problem and Exponential Families

Previous section introduced the SNGD algorithm, a variant of the SGD algorithm that can be run in a Riemannian Manifold. The difference is that SNGD computes the Natural Gradient instead of the gradient, which we have seen that points to the trully steepest direction in the manifold. Is it then a good idea to use SNGD to optimize a differentiable function defined on a manifold? For a particular set of ML problems, we are going to see it's the best idea.

In this section we restrict ML problem to a specific subset; the manifold is a Statistical Manifold containing distributions in the exponential family $\mathcal{M} = \{P(X, \theta) | \theta \in \Theta \subset \mathbb{R}^k\}$, and clearly the function to optimize is the likelihood function obtained from a sample $\{s_1, ..., s_n\}$ where $s_i \in X$. There are a lot of different probability distributions belonging to this set, such as Bernoulli, Normal, Gamma, Poisson, Categorical, Dirichlet and Chi-squared distributions among others.

This is a too easy problem to use numerical algorithms, since in this case the analytic solution exists, and so the ML estimator can be obtained. However, we use this kind of problem to prove that SNGD algorithm together with the FIM imitates ML estimator, making it insuperable, providing to SNGD a solid basis, and showing SNGD as an interesting option to use for some harder problems.

3.1 ML problem

The problem we want to answer is: Given $\mathcal{M}$ a set of probability distributions and a sample S observed from a probability distribution of $\mathcal{M}$, find the member of $\mathcal{M}$ that maximizes the probability of generating the sample. As mentioned in 2.4, this is equivalent to maximize the likelihood function. Or, instead of maximizing the likelihood function $L_S(\theta)$, we equivalently can minimize the log-likelihood function $\mathcal{L}_S(\theta) := -\log(L_S(\theta))$.

Definition 3.1. Let $\mathcal{M} = \{P(X, \theta) | \theta \in \Theta \subset \mathbb{R}^k\}$ be a set of probability distributions and $S = \{s_0, ..., s_n\}$ a sample with $s_i \in X$. Then the likelihood function is

$$L_S(\theta) = \prod_{i=0}^{n} P(s_i, \theta)$$

and the log-likelihood function is

$$\mathcal{L}_S(\theta) = -\log(L_S(\theta)) = -\sum_{i=0}^{n} \log P(s_i, \theta)$$

Regarding the notation in 2.4, $\mathcal{L}(s, \theta) = -\log P(s, \theta)$. Minimizing $\mathcal{L}_S(\theta)$ is equivalent to maximize $L_S(\theta)$ because logarithm is a monotonic and increasing function.

3.2 Exponential Family Manifold

When solving the ML problem, each point p in $\mathcal{M}$ describes a probability distribution $P(X, p)$ defined in the same probability space. This is what we call a Statistical Manifold.

We restrict now $\mathcal{M}$ to be in the Exponential Family. The definition already gives a parametrization and introduces $\mathcal{M}$ as a manifold.

Definition 3.2. A set – or family – of probability distributions $\mathcal{M} = \{P(X, \theta) | \theta \in \Theta \subset \mathbb{R}^k\}$ is in the exponential family if there exists;

$$\begin{array}{rcllcl} \phi : \Theta \subset \mathbb{R}^k & \longrightarrow & \mathcal{M} & & & \\ \theta & \longmapsto & \phi(\theta) := P(X, \theta) : & X & \longrightarrow & \mathbb{R} \\ & & & s & \longmapsto & P(s, \theta) = h(s) \cdot e^{<\eta(\theta), T(s)> - \psi(\theta)} \end{array}$$

where $h(s)$ and $\psi(\theta)$ are known real valued functions, $\eta(\theta) \in \mathbb{R}^k$ and $T(s) \in \mathbb{R}^k$ is a sufficient statistic. We will consider $h(s) = 1 \ \forall s \in X$ from now on.

Definition 3.3. A set of probability distributions in the Exponential Family is in canonical form if;

$$P(s, \theta) = e^{<\theta, T(s)> - \psi(\theta)}, \qquad \forall \theta \in \Theta, \forall s \in X$$

We can always assume we are in this later case, by using $\eta(\theta)$ as parameter – we will keep using the notation θ for the parameter –.

Definition 3.4. A statistic T is minimal if it satisfies:

$$< \theta, T(s) > \text{ is constant for all } s \in X \text{ if and only if } \theta = 0.$$

We always can also find a statistic T that is minimal. Asking $\mathcal{M}$ to be a manifold with parametrization ϕ is equivalent to ask for T to be minimal. By looking the definition 2.1 it suffices to check that ϕ is an injective map;

$$\phi(\theta_1) = \phi(\theta_2) \Leftrightarrow P(X, \theta_1) = P(X, \theta_2) \Leftrightarrow < \theta_1 - \theta_2, T(s) > = c \in \mathbb{R} \ \forall s \in X$$

Last expression and the fact that T is a minimal statistic implies that $\theta_1 = \theta_2$ and ϕ is injective. Then ϕ is a parametrization of the manifold and it is known as natural parametrization.

3.3 Mean parametrization

Departing from previous parametrization, it derives a new one that is known as mean parametrization, denoted by $\xi \in \mathbb{R}^k$;

$$\xi(\theta) = \nabla \psi(\theta) = \mathbb{E}[T(s)]$$

Moreover, the ML problem is way easier to solve with this parametrization, since we have the analytic solution of ML estimator. The ML estimate for a single observation $s \in X$ of the mean parametrization is $\overset{*}{\xi} = T(s)$ and the ML estimator for an observed sample $S = \{s_0, ..., s_n\}$ is the mean of the ML estimates for single observations, that is $\overset{*}{\xi} = \frac{m_{n+1}}{n+1}$ with $m_{n+1} = \sum_{j=0}^{n} T(s_j)$.

3.4 Metric for Exponential Family Manifold

The metric that we set for the manifold is the Fisher Information Metric (FIM). We are going to give the expression of the matrix according to this metric afterwards, but first let's go through the definitions.

3.4.1 Fisher Information Metric

Here we present the metric that we will use in the manifolds we will work on. Recall $\mathcal{M}$ is a Statistical Manifold. This kind of manifold can be enriched with the Fisher Information Metric (FIM) to obtain a Riemannian Manifold $(\mathcal{M}, \{G_p\}_{p \in \mathcal{M}})$.

Definition 3.5. Let p be a point of a Statistical Manifold $\mathcal{M}$. The Fisher Information Metric (FIM) is defined as;

$$G_p = g_{ij} = \int_X \frac{\partial \log P(s,p)}{\partial_i} \frac{\partial \log P(s,p)}{\partial_j} P(s,p) ds = \mathbb{E}_X \left[\frac{\partial \log P(s,p)}{\partial_i} \frac{\partial \log P(s,p)}{\partial_j} \right]$$

In that case, $(\mathcal{M}, \{G_p\}_{p \in \mathcal{M}})$ is a Riemannian Manifold.

So if we choose a parametrization $\phi : U \in \mathbb{R}^n \to \mathcal{M}$ around p, with $\phi(\theta) = p$ and $\frac{\partial}{\partial \theta_i} = d\phi_p(0, ..., 1, ..., 0)$, then $G_p = G_\theta = g_{ij} = \int_X \frac{\partial \log P(s,\theta)}{\partial \theta_i} \frac{\partial \log P(s,\theta)}{\partial \theta_j} P(s,\theta) ds$. It can be easily proved that another equivalent definition is;

$$G_\theta = g_{ij} = \int_X \frac{\partial^2(-\log P(s,\theta))}{\partial \theta_i \partial \theta_j} P(s,\theta) ds = \mathbb{E}_X \left[\frac{\partial^2(-\log P(s,\theta))}{\partial \theta_i \partial \theta_j} \right]$$

An alternative way to understand the FIM is to define the Kullback-Leibler divergence and then calculate its Hessian. Basically, a divergence is a relaxed version of distance, in which the divergence from a to b has not to be equal to the one from b to a. Hence, usually the triangle inequality doesn't hold either.

Definition 3.6. Let $\mathcal{M}$ be a Manifold. A divergence is a function $D : \mathcal{M} \times \mathcal{M} \to \mathbb{R}$ such that;

- $D(P(X, \overset{*}{\theta}), P(X, \theta)) \geq 0 \qquad \forall P(X, \overset{*}{\theta}), P(X, \theta) \in \mathcal{M}$.

- $D(P(X,\overset{*}{\theta}), P(X,\theta)) = 0$ if and only if $P(X,\overset{*}{\theta}) = P(X,\theta)$.

Definition 3.7. Let $P(X,\overset{*}{\theta}), P(X,\theta)$ be two probability distributions. Kullback-Leibler divergence D_{KL} is the divergence defined as;

$$D_{KL}(P(X,\overset{*}{\theta}), P(X,\theta)) = \int_X \log \frac{P(s,\overset{*}{\theta})}{P(s,\theta)} P(s,\overset{*}{\theta}) ds$$

Change the integral by a summation if the probability space is discrete. From here, it is clear that $\frac{\partial^2}{\partial\theta_i\partial\theta_j} D_{KL}(P(s,\overset{*}{\theta}), P(s,\theta))_{|\overset{*}{\theta}=\theta} = g_{ij}$ and this gives another intuition about the FIM.

When we work over a set in the exponential family in canonical form, and according to previous definitions, the Fisher Information Metric can be easily computed;

$$g_{ij} = \int_X \frac{\partial^2(-\log P(s,\theta))}{\partial\theta_i\partial\theta_j} P(s,\theta)ds = \int_X \frac{\partial^2(-<\theta, T(s)> + \psi(\theta))}{\partial\theta_i\partial\theta_j} P(s,\theta)ds =$$

$$\frac{\partial^2\psi(\theta)}{\partial\theta_i\partial\theta_j} \int_X P(s,\theta)ds = \frac{\partial^2\psi(\theta)}{\partial\theta_i\partial\theta_j} \Longrightarrow$$

$$G_\theta = H\psi(\theta) = D\nabla\psi(\theta) = D\xi(\theta)$$

If the parametrization used is the mean parametrization, we use the fact that ξ and θ are dually flat parametrizations, which in particular means $G_\theta|^{-1}_{\theta(\xi)} = G_\xi$. Then the FIM with respect to ξ is related to the inverse metric with respect to θ. This directly proves next lemma.

Lemma 3.1. Let $\theta(\xi)$ be the inverse function of $\xi(\theta) = \nabla\psi(\theta)$. Then $D\theta(\xi) = G_\xi$

Proof.

$$D\theta(\xi) = D(\xi(\theta)^{-1}) = D\xi(\theta)|^{-1}_{\theta(\xi)} = D\nabla\psi(\theta)|^{-1}_{\theta(\xi)} = G_\theta|^{-1}_{\theta(\xi)} = G_\xi$$

□

3.5 Solving exactly the ML problem in Exponential Families with SNGD

The following lemmas are tools that will serve us after to prove that SNGD on the manifold of exponential families equipped with the FIM using the mean parametrization ξ solves exactly the ML problem.

Lemma 3.2. In the ξ coordinate system, $\widetilde{\nabla}\mathcal{L}(s,\xi) = \xi - T(s)$

Proof. Note that in the natural parametrization $\mathcal{L}(s,\theta) = \psi(\theta) - < \theta, T(s) >$. Thus, $\nabla\mathcal{L}(s,\theta) = \nabla\psi(\theta) - T(s) = \xi(\theta) - T(s)$

Then, $\nabla\mathcal{L}(s,\theta(\xi)) = D\theta(\xi)\nabla\mathcal{L}(s,\theta)|_{\theta(\xi)} = D\theta(\xi)(\xi - T(s)) = G_\xi(\xi - T(s))$

$$\widetilde{\nabla}\mathcal{L}(s,\xi) \;=\; G_\xi^{-1}\nabla\mathcal{L}(s,\xi) = G_\xi^{-1}\nabla\mathcal{L}(s,\theta(\xi)) = G_\xi^{-1}G_\xi(\xi - T(s)) = \xi - T(s)$$

□

Theorem 3.3. Let $\mathcal{M} = \{p_\theta \mid \theta \in \Theta\}$ be a k-dimensional Manifold in the exponential family and $S = \{s_0, \ldots, s_n\}$ a data sample. Then SNGD on the mean parametrization with learning rate $r_i = \frac{1}{1+i}$ imitates ML estimator. This means that at each step i for $i > 0$, ξ^i is the ML estimate for $\{s_0, \ldots, s_i\}$ (if it exists).

Proof. By induction. Assume ξ^i coincides with the maximum likelihood estimate for $\{s_0, \ldots, s_i\}$. We will prove that the same holds for ξ^{i+1}. By induction hypothesis we have that $\xi^i = \frac{m_i}{i}$ where $m_i = \sum_{j=0}^{i-1} T(s_j)$. From the SNGD update equation and Lemma 3.2, we have that;

$$\xi^{i+1} = \xi^i - r_i\widetilde{\nabla}\mathcal{L}(s_i,\xi^i) \overset{3.2}{=} \xi^i - \frac{\xi^i - T(s_i)}{i+1} = \frac{i\xi^i + T(s_i)}{i+1} \overset{I.H.}{=} \frac{m_i + T(s_i)}{i+1} = \frac{m_{i+1}}{i+1}$$

proving the theorem. □

3.6 Example: Solving ML in Categorical distributions

The categorical distribution is a set of probability distributions belonging to the exponential family. It describes, for instance, the probability space obtained by a die of unknown probabilities. We show all concepts seen in this section for this particular example.

For this example, ML problem objective seeks the most likely probabilities of each face of the die, regarding a set of observations. That means, the goal is to maximize the Likelihood function. That's obtained by the ML estimator. By looking all results seen in the chapter, it should be possible to perform as well as ML estimator using the SNGD algorithm with the mean parametrization and the FIM. To run SNGD, it's necessary to findthe mean parametrization, the function to optimize, the FIM matrix inverse and the Natural Gradient. Each of those are described below.

Mean parametrization

Let $D = \{0, 1, ..., k\}$ be a $(k+1)$-faces die and let $\xi = (\xi_0, ..., \xi_k)$ be the vector where ξ_i is the probability of face i to happen for $i \in D$. As we know, the sum of all face probabilities adds up to 1. This means the manifold described is easily parametrized by the k-simplex S_k. Previously appeared the natural and mean parametrizations of exponential family manifold – θ and ξ parametrizations respectively. For this particular example, these are;

$$\begin{array}{rcl} \phi_1 : \mathbb{R}^k & \longrightarrow & \mathcal{M} \\ \theta & \longmapsto & P(D,\theta) : \begin{array}{rcl} D & \longrightarrow & \mathbb{R} \\ s & \longmapsto & P(s,\theta) = e^{<\theta,T(s)>-\psi(\theta)} \end{array} \end{array}$$

where $T(s) = \left\{ \begin{array}{ll} e_{s+1} \in \mathbb{R}^k & \text{if } s \neq k \\ (0,...,0) & \text{if } s = k \end{array} \right.$ is a sufficient statistic and $\psi(\theta) = \log \sum_{s=0}^k e^{<\theta,T(s)>}$.

$$\begin{array}{rcl} \phi_2 : \overset{\circ}{S}_k & \longrightarrow & \mathcal{M} \\ (\xi_0,...,\xi_{k-1}) & \longmapsto & P(D,\xi) : \begin{array}{rcl} D & \longrightarrow & \mathbb{R} \\ s & \longmapsto & P(s,\xi) = \xi_s \end{array} \end{array}$$

such that, recalling 3.3, $(\xi_0,...,\xi_{k-1}) = \mathbb{E}[T(D)] = \nabla\psi(\theta)$

Loglikelihood function

Consider we throw the die several times and we write down the results in a set of observations $S = \{s_i\}_{i=0}^n$, where $s_i \in D$ is the face occurred in i-th throw. By 3.1, the Likelihood function in this case is;

$$L(\phi_2(\xi)) = \prod_{i=0}^n P(s_i,\xi) = \prod_{i=0}^n \left\{ \begin{array}{ll} \xi_{s_i} & \text{if } s_i \neq k \\ 1-(\xi_0+...+\xi_{k-1}) & \text{if } s_i = k \end{array} \right.$$

and the log-Likelihood function to minimize is;

$$\mathcal{L}(\phi_2(\xi)) = \sum_{i=0}^n \left\{ \begin{array}{ll} -\log(\xi_{s_i}) & \text{if } s_i \neq k \\ -\log(1-(\xi_0+...+\xi_{k-1})) & \text{if } s_i = k \end{array} \right.$$

FIM matrix inverse

The FIM matrix G and its inverse for this example are the following;

$$G = \begin{pmatrix} \frac{1}{\xi_k}+\frac{1}{\xi_0} & \frac{1}{\xi_k} & \cdots & \frac{1}{\xi_k} \\ \frac{1}{\xi_k} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \frac{1}{\xi_k} \\ \frac{1}{\xi_k} & \cdots & \frac{1}{\xi_k} & \frac{1}{\xi_k}+\frac{1}{\xi_{k-1}} \end{pmatrix} \rightarrow$$

$$G^{-1} = \begin{pmatrix} \xi_0(1-\xi_0) & -\xi_0\xi_1 & \cdots & -\xi_0\xi_{k-1} \\ -\xi_0\xi_1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -\xi_{k-2}\xi_{k-1} \\ -\xi_0\xi_{k-1} & \cdots & -\xi_{k-2}\xi_{k-1} & \xi_{k-1}(1-\xi_{k-1}) \end{pmatrix}$$

Natural Gradient

Finally, we compute the Natural Gradient. Notice the gradient is

$$\nabla\mathcal{L}(s_i,\xi) = \left\{ \begin{array}{ll} \frac{-1}{\xi_{s_i}} e_{s_i+1} & \text{if } s_i \neq k \\ \frac{1}{\xi_k}(1,...,1) & \text{if } s_i = k \end{array} \right.$$

Then, the Natural Gradient is

$$\widetilde{\nabla}\mathcal{L}(s_i,\xi) = G^{-1}\nabla\mathcal{L}(s_i,\xi) = \begin{cases} (\xi_0, ..., \xi_{s_i}-1, ..., \xi_{k-1}) & \text{if } s_i \neq k \\ (\xi_0, ..., \xi_{k-1}) & \text{if } s_i = k \end{cases} = \xi - T_k(s_i)$$

as lemma 3.2 states.

Tools to run SNGD are found. The answer we are looking for is $\overset{*}{\xi} := \arg\min_{\xi \in S_k} \mathcal{L}(\phi_2(\xi))$ and it's the ML estimator. Section 3.3 specifies for the mean parametrization that $\overset{*}{\xi} = \frac{m_{n+1}}{n+1}$, where $m_{n+1} = \sum_{j=0}^{n} T(s_j)$. By induction, suppose that we have run the algorithm until iteration i and SNGD has imitated ML estimator, that is $\xi^i = \frac{m_i}{i}$. The idea now is to run one step of SNGD and check that it actually updates identically as ML estimator does. This is a particular example of theorem 3.3.

$\xi^{i+1} = \xi^i - r_i \cdot \widetilde{\nabla}\mathcal{L}_{s_i}(\xi^i) = \xi^i - r_i \cdot (\xi^i - T_k(s_i)) = \frac{m_i}{i} - \frac{1}{1+i} \cdot (\frac{m_i}{i} - T_k(s_i)) = \frac{m_{i+1}}{i+1}.$

4 MCL problem and Logistic Regression

In previous section we have shown that, for the ML problem, running SNGD algorithm in a specific Riemannian Manifold carrying the FIM is an unsurpassable method. We wonder about SNGD yield in harder problems. In this section we will introduce another Riemannian Manifold $\mathcal{M}$ and the MCL problem in it, as well as a parametrization and the matrix computation of the metric of $\mathcal{M}$.

4.1 MCL problem

Similar to section 3.1, the problem we want to solve is: Given $\mathcal{M}$ a manifold of conditional probability distributions and a sample S obtained from a conditional probability distribution of $\mathcal{M}$, find the member of $\mathcal{M}$ that maximizes the probability of generating the sample. This is equivalent to maximize the conditional likelihood function $CL_S(\beta)$. Or, instead, we equivalently can minimize the conditional log-likelihood function $\mathcal{L}_S(\beta) := -\log(CL_S(\beta))$.

Definition 4.1. Let $\mathcal{M} = \{P(y|X,\theta)|\theta \in \Theta \subset \mathbb{R}^k\}$ a set of conditional probability distributions and $S = \{(y_1,x_1),...,(y_n,x_n)\}$ with $(y_i,x_i) \in y \times X$. Then the conditional likelihood function is

$$CL_S(\theta) = \prod_{i=0}^{n} P(y_i|x_i,\theta)$$

and the conditional log-likelihood function is

$$\mathcal{L}_S(\theta) = -\log(CL_S(\theta)) = -\sum_{i=0}^{n} \log P(y_i|x_i,\theta)$$

Following the notation in 2.4, it is $\mathcal{L}((y_i,x_i),\theta) = -\log P(y_i|x_i,\theta)$.

4.2 Multinomial Logistic Regression Manifold

We want to describe the Manifold in which we will work from now on. First of all, some definitions needed are presented before the Manifold is specified.

Definition 4.2. Let (Ω, F, P) be a probability space, $X = (x^1,...,x^k)$ be a random variables vector and y a random variable. We define the conditional probability of Y given X as follows;

$$P(y|X) := \frac{P(y,X)}{P(X)}$$

In our case, the variables y and X will be discrete random variables.

Notation. Let y be a discrete random variable. If y has $s+1$ different values then we write $|y| = s+1$.

Observation. If $|x^l|, |y| = 2 \ \forall l$, conditional probability space has dimension 2^k. If $|x^l| = t+1 \ \forall l$, $|y| = s+1$, conditional probability space has dimension $s \cdot (t+1)^k$

Observation. For us, it is $|x^l| = |x^j|$ for all $1 \leq l, j \leq k$, however everything in the text can easily be extended to the more general case.

Definition 4.3. We say X is component wise conditionally independent random variable given y, denoted X ccig. y, if;

$$\forall x_i, x_j \subset X, P(x_i, x_j|y) = P(x_i|y)P(x_j|y) \tag{3}$$

Definition 4.4. Let $X = (x^1, ..., x^k)$ be a random variables vector, $|x^l| = t+1$, and y a random variable, $|y| = s+1$. The Multinomial Logistic Regression manifold $\mathcal{M}$ is the manifold of conditional probability spaces $P(y|X)$ such that X is component wise conditionally independent random variable given y, that is

$$\mathcal{M} = \{P(y|X) : X \text{ ccig. } y\}$$

We have not given yet a correct parametrization for this manifold. That will be the goal of next section.

4.3 β Parametrization

Now it's time to find a parametrization for the manifold described in 4.2. We start recalling the well known Bayes Theorem;

Theorem 4.1 (Bayes).

$$P(y|X) = \frac{P(y) \cdot P(X|y)}{\sum_{y' \in y} P(y') \cdot P(X|y')} \tag{4}$$

By using equations 3 and 4, if we work with the Manifold $\mathcal{M}$, we can rewrite the conditional probability of y given X as:

$$P(y|X) := \frac{P(y,X)}{P(X)} = \frac{P(y) \cdot P(x^1, ..., x^k|Y)}{\sum_{y' \in y} P(y') \cdot P(x^1, ..., x^k|y')} = \frac{P(y) \cdot P(x^1|y) \cdot ... \cdot P(x^k|y)}{\sum_{y' \in y} P(y') \cdot P(x^1|y') \cdot ... \cdot P(x^k|y')} \tag{5}$$

In 5, we can describe our space using $P(x^i|y)$ and $P(y)$ as parameters. However, that would not be a correct parametrization of the Manifold $\mathcal{M}$, since we are overparametrizing $\mathcal{M}$, that is, injectivity is not fullfilled. The parametrization we describe below can be found in [6]. To simplify, we give the parametrization separating the Bivalued Logistic Regression case (BLG) – that is, when $|x^l|, |y| = 2$ – from the Multivaliued Logistic Regression one (MLG) – when $|x^l| = t+1$, $|y| = s+1$.

Case 1: BLG

$$\begin{cases} \beta_0 = \log(\frac{P(y=1)}{P(y=0)}) + \log(C) \\ \beta_l = \log(\frac{P(x^l=1|y=1)}{P(x^l=0|y=1)}) - \log(\frac{P(x^l=1|y=0)}{P(x^l=0|y=0)}) \end{cases}$$

with $1 \leq l \leq k$ and

$$\log C = \sum_i \log(P(x^i = 0|y = 1) - \log P(x^i = 0|y = 0))$$

Case 2: MLG

$$\begin{cases} \beta_0^{y'} = \log(\frac{P(y=y')}{P(y=0)}) + \log(C_{y'}) \\ \beta_{x^l,x'}^{y'} = \log(\frac{P(x^l=x'|y=y')}{P(x^l=0|y=y')}) - \log(\frac{P(x^l=x'|y=0)}{P(x^l=0|y=0)}) \end{cases}$$

with $1 \leq y' \leq s, \quad 1 \leq x' \leq t, \quad 1 \leq l \leq k$ and

$$\log C_{y'} = \sum_i \log(P(x^i = 0|y = y') - \log P(x^i = 0|y = 0))$$

That is the parametrization used further on for the Multinomial Logistic Regression manifold $\mathcal{M}$ of conditional probabiliy spaces.

Observation. If $|x^l|, |y| = 2$, then the manifold $\mathcal{M}$ has dimension $k + 1$. If $|x^l| = t + 1$, $|y| = s + 1$, then $\mathcal{M}$ has dimension $(k \cdot t + 1) \cdot s$

Using the β's, we are able to write the parametrization of the manifold $\mathcal{M}$ as;

$$\begin{array}{rcl} \phi : \mathbb{R}^{(k\cdot t+1)\cdot s} & \longrightarrow & \mathcal{M} \\ \beta & \longmapsto & \phi(\beta) := P(y|X,\beta) : \quad \begin{array}{rcl} y \times X & \longrightarrow & \mathbb{R} \\ (y', x') & \longmapsto & P(y'|x', \beta) \end{array} \end{array}$$

with

$$P(y'|x', \beta) = \frac{\delta_{y'=0} + e^{\beta^{y'} \cdot T(x')} \cdot \delta_{y' \neq 0}}{1 + \sum\limits_{0 \neq y'' \in y} e^{\beta^{y''} \cdot T(x')}} \tag{6}$$

where

$$\beta^{y'} = \begin{pmatrix} \beta^{y'}_{0} \\ \beta^{y'}_{x^1,1} \\ \beta^{y'}_{x^1,2} \\ \vdots \\ \beta^{y'}_{x^1,t} \\ \beta^{y'}_{x^2,1} \\ \vdots \\ \beta^{y'}_{x^2,t} \\ \vdots \\ \beta^{y'}_{x^k,t} \end{pmatrix} \quad \text{and} \qquad T(x') = \begin{pmatrix} 1 \\ - \\ T(x'^1) \\ - \\ \vdots \\ - \\ T(x'^k) \end{pmatrix} \qquad \text{with } T(x'^i) = \begin{pmatrix} \delta_{x'^i=1} \\ \vdots \\ \delta_{x'^i=t} \end{pmatrix}$$

4.4 Metric for Multinomial Logistic Regression Manifold

The task right now is to enrich the manifold $\mathcal{M}$ with a metric to have a Riemannian Manifold. Considering that FIM worked perfectly for the ML problem in the exponential family manifold, it seems a good idea to keep on that track. However, FIM is not actually defined in our manifold, which contains conditional probability distributions instead of probability distributions. We decide to use a close metric to that one [12], and we will still name it FIM. Just as we explained in 3.4.1, we have two different points of view to understand this metric. Nevertheless the computations are the same. Let's see a variation of definitions in 3.4.1 to use them in the current manifold.

Definition 4.5. Let p be a point of $\mathcal{M}$. The Fisher Information Metric (FIM) is defined as;

$$G_p = g_{ij} = \mathbb{E}_{x\in X}\left[\int_{y'\in y} \frac{\partial \log P(y'|x,p)}{\partial_i}\frac{\partial \log P(y'|x,p)}{\partial_j} P(y'|x,p)dy'\right] =$$
$$\int_{x\in X} P(x)\left[\int_{y'\in y} \frac{\partial \log P(y'|x,p)}{\partial_i}\frac{\partial \log P(y'|x,p)}{\partial_j} P(y'|x,p)dy'\right] dx$$

This definition works not only for the discret case. We now can consider the Riemannian Manifold $(\mathcal{M}, \{G_p\}_{p\in\mathcal{M}})$. An alternative and equivalent reasoning to obtain the same Riemannian Manifold starts by defining the following divergence in $\mathcal{M}$.

Definition 4.6. Kullback-Leibler divergence D_{KL} in $\mathcal{M}$ is the divergence defined as;

$$D_{KL}(P(y|X,\overset{*}{\beta}), P(y|X,\beta)) = \int_{x\in X} P(x)\left[\int_{y'\in y} P(y'|x,\overset{*}{\beta})\cdot \log\frac{P(y'|x,\overset{*}{\beta})}{P(y'|x,\beta)}dy'\right] dx$$

Again, as we mentioned in 3.4.1, the Hessian Matrix of the KL divergence coincides with the matrix of the Fisher Information Metric (FIM). Then, finding the FIM matrix can be done by finding the Hessian of D_{KL} with respect to β and evaluating $\overset{*}{\beta} = \beta$ – remember the parametrization taken with equation 6 in section 4.3 –.

4.4.1 Matrix G_β computation

Below there is the proposition that expresses the metric matrix G_β with the FIM and the β parametrization defined previously. The proof of the result can be found in the annex, section 6.1.

Proposition 4.2. Let $\beta_1 := \beta^{y^1}_{x^i,u}$ and $\beta_2 := \beta^{y^2}_{x^j,v}$. Let $\beta_3 := \beta^{y^3}$ and $\beta_4 := \beta^{y^4}$. Then the metric matrix G_β according to FIM is such that

if $y^2 = y^1$:

$$g_{\beta_1,\beta_2} = \sum_{x\in X|x^i=u,x^j=v} P(x)\cdot P(y^1|x)\cdot(1-P(y^2|x))$$

if $y^2 \neq y^1$:

$$g_{\beta_1,\beta_2} = -\sum_{x\in X|x^i=u,x^j=v} P(x)\cdot P(y^1|x)\cdot P(y^2|x)$$

if $y^2 = y^3$:

$$g_{\beta_3,\beta_2} = \sum_{x\in X|x^j=v} P(x)\cdot P(y^3|x)\cdot(1-P(y^2|x))$$

if $y^2 \neq y^3$:

$$g_{\beta_3,\beta_2} = -\sum_{x\in X|x^j=v} P(x)\cdot P(y^3|x)\cdot P(y^2|x)$$

if $y^3 = y^4$:

$$g_{\beta_3,\beta_4} = \sum_{x\in X} P(x)\cdot P(y^3|x)\cdot(1-P(y^4|x))$$

if $y^3 \neq y^4$:

$$g_{\beta_3,\beta_4} = -\sum_{x\in X} P(x)\cdot P(y^3|x)\cdot P(y^4|x)$$

Observation. Proposition 4.2 uses $\beta_1, \beta_2, \beta_3$ and β_4 to express matrix G_β because they serve as coordinates. More precisely, G_β matrix rows and columns are sorted following the vector

$$(\beta^1, \beta^2, \cdots, \beta^s) \text{ with } \beta^{y'} = \left(\beta^{y'}_0\, \beta^{y'}_{x^1,1}\beta^{y'}_{x^1,2}\cdots\beta^{y'}_{x^1,t}\beta^{y'}_{x^2,1}\cdots\beta^{y'}_{x^2,t}\cdots\beta^{y'}_{x^k,t}\right)$$

4.4.2 G_β matrix vectorization

Regarding last work, we do some remarkable considerations. Since the definition of the metric G_β involves $\mathbb{E}_{x\in X}$, then the metric is not defined until the probability distribution $P(X)$ is stablished, which at first remained unknown and irrelevant before. Suppose then, that $P(X)$ is fixed. The goal here is to provide a fast and vectorized way to compute matrix

G_β. For that purpose, define A the matrix having $T(x) \in \mathbb{R}^{kt+1}$ as columns $\forall x \in X$ and B the diagonal matrix having $P(x)$ in the diagonal $\forall x \in X$;

$$A = \begin{pmatrix} T(x_1) & | & T(x_2) & | & \cdots & | & T(x_i) & | & \cdots \end{pmatrix}$$
$$B = \{P(x) | \forall x \in X\}$$

Now let the diagonal matrices $q_j, \forall j \in \{0, 1, ..., s\}$ defined by its diagonal terms

$$q_j = \{P(y = j|x, \beta)|\forall x \in X\}$$

and let M be the diagonal blocks matrix;

$$M = \begin{pmatrix} q_1 & 0 & \cdots & 0 \\ 0 & q_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & q_s \end{pmatrix} - \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_s \end{pmatrix} \cdot \begin{pmatrix} q_1 & q_2 & \cdots & q_s \end{pmatrix}$$

Notation. We will write for example AM the product of the matrix A and M. Notice that the matrices dimensions don't match to multiply them. The notation in fact expresses the product of A from the left by every block of M. Similarly, the product from the right MA^t.

Then next result has a straightforward proof, annexed in 6.2.

Proposition 4.3.

$$G_\beta = AMBA^t,$$

4.4.3 Maximum entropy distribution metric

Following lines present the metric of a specific and special point of the manifold. Before any of the stochastic algorithms has started running, and before any sample has been observed, the uncertainty is the highest possible. The best guess we can do about the probability distribution that is about to generate the samples is that every event $(y', x) \in y \times X$ has the same probability to happen.

This corresponds to the parametrized point of $\mathcal{M}$ as $\beta = 0 \in \mathbb{R}^{(kt+1)s}$, because then $P(y'|x, \beta) = \frac{1}{s+1}$ is constant. This point describes the maximum entropy distribution.

Moreover, consider the maximum entropy distribution of $P(X)$, that is, $P(x) = \frac{1}{n+1}$ $\forall x \in X$ with $|X| = n + 1$. Then it is possible to compute the metric matrix inverse G_0^{-1} really fast, by saving the cost of doing any matrix inversion. Proposition below shows how, but before see a needed technical lemma with its proof in section 6.3.

Lemma 4.4. Suppose the maximum entropy distribution of $P(X)$ is selected. Let K_t be a $t \times t$ matrix with ones at every entry and $|X| = n+1$. So then, if $L_t = K_t + Id_t$,

$$\left(\frac{AA^t}{n+1}\right)^{-1} = (t+1)\begin{pmatrix} \frac{1+tk}{t+1} & -1 & \cdots & \cdots & -1 \\ -1 & L_t & 0 & \cdots & 0 \\ \vdots & 0 & L_t & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ -1 & 0 & \cdots & 0 & L_t \end{pmatrix}$$

Corolary 4.5. In the more general case where $|x_l| = t_l + 1 \; \forall l \in \{1, ..., k\}$, it is

$$\left(\frac{AA^t}{n+1}\right)^{-1} = \begin{pmatrix} 1+\sum t_i & -t_1-1 & \cdots & \cdots & -t_k-1 \\ -t_1-1 & (t_1+1)L_{t_1} & 0 & \cdots & 0 \\ \vdots & 0 & (t_2+1)L_{t_2} & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ -t_k-1 & 0 & \cdots & 0 & (t_k+1)L_{t_k} \end{pmatrix}$$

Next result shows how to compute the metric matrix inverse saving the cost of computing the matrix G_0 and also saving the matrix inversion cost, using the lemma 4.4. The reader can find the proof annexed in section 6.4.

Proposition 4.6. Suppose the maximum entropy distribution of $P(X)$ and $P(y|X)$ are selected and $|X| = n+1$. Then

$$G_0^{-1} = (s+1)\left(\frac{AA^t}{n+1}\right)^{-1} \begin{pmatrix} 2Id & Id & \cdots & Id \\ Id & 2Id & \ddots & \vdots \\ \vdots & \ddots & \ddots & Id \\ Id & \cdots & Id & 2Id \end{pmatrix}$$

with $Id = Id_{kt+1}$

4.4.4 Selecting $P(X)$

Proposition 4.3 shows a vectorized way to compute G_β. However, as we said, the distribution $P(X)$ must be fixed previously. Furthermore, notice that if the dimension of the manifold is huge, then the products AM and BA^t may become also huge in computation cost.

However we don't have any hint about what's the best distribution to choose, since the manifold ignores completly that space – $P(X)$ is irrelevant from the manifold viewpoint –. So when deciding what distribution to pick, a good choice can be, for example, the empirical distribution oberved from the sample. That way, an estimation of the actual underlying $P(X)$ distribution comes available. Furthermore, it may decrease a lot the amount of operations done since $P(x_i) = 0$ if x_i has not ocurred in the sample $S = \{(y_0, x_0), ..., (y_n, x_n)\}$, allowing the saving of many trivial operations. To see how, redefine matrix A of section 4.3 as;

$$A^* = \big(T(x_0) \quad | \quad T(x_1) \quad | \quad \cdots \quad | \quad T(x_n)\big)$$

Now let the diagonal matrices $p_j, \forall j \in \{0, 1, ..., s\}$ defined by its diagonal terms

$$p_j = \{P(y = j|x_i, \beta)|\forall i \in \{0, ..., n\}\}$$

and define M^* identically, but using these new matrix instead;

$$M^* = \begin{pmatrix} p_1 & 0 & \cdots & 0 \\ 0 & p_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & p_s \end{pmatrix} - \begin{pmatrix} p_1 \\ p_2 \\ \vdots \\ p_s \end{pmatrix} \cdot \begin{pmatrix} p_1 & p_2 & \cdots & p_s \end{pmatrix}$$

Notice that A^* and M^* are way smaller matrices now. Finally rewrite proposition 4.3 as follows and see its proof in section 6.5;

Proposition 4.7 (AMAT)**.** Let $P(X)$ be the empirical distribution given a sample $S = \{(y_0, x_0), ..., (y_n, x_n)\}$. Then

$$G_\beta = \frac{\overset{*}{A}\overset{*}{M}\overset{*}{A}^t}{n+1}$$

4.5 Gradient computation

Since this project describes stochastic gradient algorithms – see section 2.4 –, the computation of the gradient of $\mathcal{L}((y_i, x_i), \beta) = -\log P(y_i|x_i, \beta)$ is needed. It's enough to compute the partial derivatives of the function. If $\beta_1 := \beta^{y^1}_{x^j,u}$, then the derivative with respect to β_1 is

$$-\tfrac{\partial}{\partial \beta_1} \log P(y_i|x_i, \beta) = \begin{cases} 0 & \text{if } (x_i)^j \neq u \\ P(y^1|x_i, \beta) & \text{if } (x_i)^j = u,\ y_i \neq y^1 \\ -1 + P(y^1|x_i, \beta) & \text{if } (x_i)^j = u,\ y_i = y^1 \end{cases}$$

5 Experiments

The objective of this chapter is implementing previous chapter 4, and using Natural Gradient algorithms to solve the Maximum Conditional Likelihood problem (MCL) with the Logistic Regression model. The intention is to test 3 algorithms that use the Natural Gradient against standard SGD and AdaGrad algorithms that we will explain later.

Two problems are faced. The first one, is obviously to compare the behavior of Natural Gradient algorithms with the results given by SGD and AdaGrad when solving the MCL problem. The second one, is assessing the estimates and evaluate the error they provide in terms of the expected prediction error that will be defined. This value helps us to judge the estimates as predictors. Our programming can be found in [3].

We are going to run increasingly complex problems – increasing the dimension of $\mathcal{M}$ – in the following subsections. Table 1 shows the different experiment settings used;

Table 1

Experiments	Dimension	n	Epochs	Instances
Bivalued	$\lvert y\rvert = 2, X = (x^1, x^2, x^3, x^4), \lvert x_i\rvert = 2$	20	100	100
Small	$\lvert y\rvert = 3, \lvert x^1\rvert = 2, \lvert x^2\rvert = 3$	100	100	100
Medium	$\lvert y\rvert = 4, \lvert x^1\rvert = 2, \lvert x^2\rvert = 4, \lvert x^3\rvert = 7$	100	100	100
Large	$\lvert y\rvert = 7, \lvert x^1\rvert = 2, \lvert x^2\rvert = 3, \lvert x^3\rvert = 4, \lvert x^4\rvert = 5$	100	100	100

We proceed like this: We set the dimension of $\mathcal{M}$ – that is $|X|$ and $|y|$ values – and then pick up $\bar{\beta}$ a coordinate point of $\mathcal{M}$. Each coordenate $\bar{\beta}_j$ of $\bar{\beta}$ will be randomly drawn from a Normal Distribution $N(0, 0.5)$. We then obtain a sample S out of $P(y|X, \bar{\beta}) \in \mathcal{M}$ and our goal is to find the parameter $\overset{*}{\beta}$ so that $P(y|X, \overset{*}{\beta}) \in \mathcal{M}$ is the most likely conditional probability distribution to generate S. That's equivalent to minimize function $\mathcal{L}_S(\beta)$ specified in 4.1.

The learning rate $r_i = \frac{a}{1+bi}$ depends on two positive real numbers a, b. We decide the constants $a, b \in \mathbb{R}^+$ by doing a fast test in a small sample. We run every algorithm over a few epochs and during about 10 seconds we pick the constants that worked better when optimizing the function. AdaGrad learning rate works differently, we specify it later in this section.

Every algorithm starts with the initial point $\beta_0 = 0 \in \mathbb{R}^{(kt+1)s}$ since it describes the maximum entropy conditional distribution and therefore the maximum uncertainty point of $\mathcal{M}$, as reasonable.

5.1 Algorithms

This section provides the description of the algorithms that will be tested, all of them using the Linesearch strategy to optimize a function. There are 5 algorithms in total. First 2 algorithms are standard and commonly used, SGD and AdaGrad. They apply the gradient in its updates and they are chosen very often to optimize because of its low computational complexity. 3 next algorithms make use of the Natural Gradient, or a variation of it, instead of the regular gradient. For every algorithm it appears its pseudocode and its computational complexity assessment. To do so, let d be the dimension of the manifold, e the total number of epochs and n the sample length. Moreover, $M(a, b, c)$ denotes the number of operations needed in a 2 matrices product of dimensions $a \times b$ and $b \times c$. The inversion of a squared matrix of dimension d is represented with $I(d)$. If standard algorithms to multiply and invert matrices are applied, then it is $O(M(a, b, c)) = O(abc)$ and $O(I(d)) = O(d^3)$.

SGD

This algorithm is the usual SGD described in section 2.4. Its pseudocode appears in Listing 2, which allows the analysis of its complexity below.

The complexity of SGD is the lowest in this project, having to compute the gradient, a vector-scalar product and a vectors addition per iteration. The vector operations have linear complexity with respect to d. To compute the gradient in iteration i, it is enough to calculate $\beta^{y'} \cdot T(x_i)$ for every $y' \in y$ – see 4.5 and equation 6 of section 4.3. The complexity of computing $\beta^{y'} \cdot T(x_i)$ is $(1 + kt)$ so gradient computational complexity is also $O(s(1 + kt)) = O(d)$. Since the total iterations is $e \cdot n$, the complexity of SGD is

$$O(en(d + d + d) = O(end)$$

AdaGrad

Here there is the description of AdaGrad algorithm. Its structure is exactly the same as SGD – see scheme 2 –, except the fact that the learning rate r is different. To begin with, it works every coordenate separately. So by just looking the coordenate j of β, at iteration i, this is how r_i for coordenate j looks like;

$$r_{i,j} = \frac{a}{\sqrt{c_{i,j} + f}}$$

where $c_{i,j} = \sum_{t=0}^{i} (\nabla \mathcal{L}(s_t, \beta)^j)^2$ and f is a fudge factor. This final algorithm is an adaptive learning rate algorithm very used nowadays due to its simplicity and good results.

We are not going to study this kind of algorithms, but we will include it in the experiments to compare. Read [5, 14] for a more detailed information about adaptive learning rate algorithms .

The computational complexity of AdaGrad is the same as SGD, since the complexity of above calculations is linear with respect to d. So the total complexity of AdaGrad is

$$O(end)$$

SNGD

This algorithm is the same SNGD defined in 2.5 but fixing the metric to be the FIM and $P(X)$ to be the empirical one. It uses $AMAT$ result in 4.7 to compute matrix G_β. Next there is SNGD scheme.

Listing 3: SNGD scheme

```
n:= sample length
beta:= 0
for e:= 0 to max_epochs do
        for i:=0 to n do
                it = i+n*e
                r:= learning rate of iteration it
                g:= gradient(observation i of S)
                G:= AMAT / n w.r.t beta
                inverseG:= inverse of G
                ng:= inverseG X g
                beta:= beta - r x ng
        permute elements on set S
        end;
end;
```

This is computationally expensive since it requieres, in addition to SGD operations, the assessment of the G_β matrix, its inverse and a matrix-vector product after each observation – after each iteration – in the dataset.

Starting with G_β matrix computation using the vectorized form AMA^t in 4.7, the task asks for matrix M computation and clearly the matrix product AMA^t. Section 4.4.4 describes M and it can be computed with complexity $O(n(d+s^2))$. Next, AMA^t matrix product complexity is of order $O(s^2M(1+tk, n, 1+tk))$. Since $O(n(d+s^2)) \leq O(s^2M(1+tk, n, 1+tk))$, the complexity of the whole task is just $O(s^2M(1+tk, n, 1+tk))$. Notice that many matrix operations needed involve diagonal matrices, and this fact has been taken into account for the computational complexity analysis.

Secondly, the dimension of the matrix the algorithm inverts is d, so it needs $I(d)$ operations to be inverted at every iteration.

Lastly, the product of the squared d dimensional inversed matrix and the gradient vector has computational complexity of order $O(d^2)$.

Then, the computational complexity of SNGD is

$$O(en(3d + s^2M(1 + tk, n, 1 + tk) + I(d) + d^2) = O(en(s^2M(1 + tk, n, 1 + tk) + I(d))$$

because $O(3d) < O(d^2) \leq O(I(d))$.

For example, with standard algorithms to multiply and invert matrices, the computational complexity of SNGD is

$$O(end^2(n + d))$$

.

MOD

SNGD algorithm becomes terribly slow when the problem complexity and the number of iterations is large. The reason is the inverse matrix computation needed to compute the Natural Gradient. That's why we build a variation of SNGD algorithm that we call MOD – Manifold Optimized Descent – . This is similar to SNGD but it inverts a matrix just once per epoch, and it uses the same inverse matrix for the whole epoch.

Observe that once $P(X)$ is fixed to be the empirical distribution, when the algorithm MOD updates G_β once per epoch, it needs to run again over the sample S to compute it. If we want to compare algorithms, it doesn't seem fair that MOD scans the sample twice in comparison to the standard algorithms.

To fix it, the update of the matrix G_β will be computed with respect to first β of the epoch, and moreover, we do an average of past epochs G_β matrices. MOD algorithm visits every observation of S as much times as other algorithms, making it a fair competitor. This is represented in the next schema;

Listing 4: MOD scheme

```
n:= sample length
beta:= 0
inverseOfG:= inverseG_0
previousG:= 0
for e:= 0 to max_epochs do
        betaG := beta copy
        for i:=0 to n do
                it = i+n*e
                r:= learning rate of iteration it
                g:= gradient( observation i of S )
                ng := inverseOfG X gradient
                beta = beta - r x ng
        G := AMAT/n w.r.t betaG
        alpha:= 1 - ( 1/( e+1 ) )
        G = previousG x alpha +  G x ( 1-alpha )
        previousG:= G
        inverseOfG := inverse of G
        permute elements on set S
        end;
end;
```

Recall the complexity analysis of algorithm SNGD, since the computations done are the same. The difference, as said previously, is that the matrix computation of G_β using $AMAT$ and its inversion is done just once per epoch. That is, the computational complexity of MOD is

$$O(e(s^2M(1+tk,n,1+tk))+I(d)+n(3d+d^2)))=O(e(I(d)+nd^2)),$$

because $O(s^2M(1+tk,n,1+tk)) \leq O(s^2(1+tk)^2n) = O(nd^2)$. Observe that for MOD, the cost of computing G_β doesn't affect to the final computational complexity of the algorithm.

If schoolbook matrix inversion algorithm is used, then it is

$$O(e(d^3+nd^2)=O(ed^2(n+d))$$

MEGD

One last algorithm is described, computationally close to the regular SGD algorithm. We call it MEGD – standing for Maximum Entropy Gradient Descent – and it uses $g = G_0^{-1}\nabla\mathcal{L}(s_i,\beta)$ in every iteration so it is set;

- $g_i = G_0^{-1}\nabla\mathcal{L}(s_i,\beta)$

- $r_i = \frac{a}{1+bi}$ for some $a, b \in \mathbb{R}^+$.

The reason we thought about this is the following: SGD doesn't do any modification to the gradient. That can be understood as using the identity matrix Id as metric matrix. However it doesn't really describe the metric of any point of the manifold. The manifold is never as flat as that, but it reaches its most flattened state when $\beta = 0$, when the entropy is the highest. We can apply instead this metric information. Also, the inverse of the metric matrix at $\beta = 0$ can be done without paying the cost of any matrix inversion by result 4.6.

MEGD follows schema below;

Listing 5: Maximum Entropy Gradient Descent scheme

```
beta:= 0
inverseG0:= inverse of maximum entropy G_0 matrix
for e:= 0 to max_epochs do
        for i:=0 to n do
                it = i+n*e
                r:= learning rate of iteration it
                g:= gradient(observation i of S)
                g= inverseG0 X g
                beta:= beta - r x g
        permute elements on set S
        end;
end;
```

Recall the computational complexity study of SGD. Observe that additionally a matrix-vector product is computed at every iteration. So then, the computational complexity of MEGD is

$$O(en(3d + d^2)) = O(end^2))$$

5.2 Multinomial Logistic Regression problem

We want to answer two questions in this section;

- Does natural gradient lead to higher quality solutions than those provided by SGD and AdaGrad?, and
- At which percentage of the data does the winning algorithm reach the quality of the losing ones?

5.2.1 Normalized Loglikelihood Difference

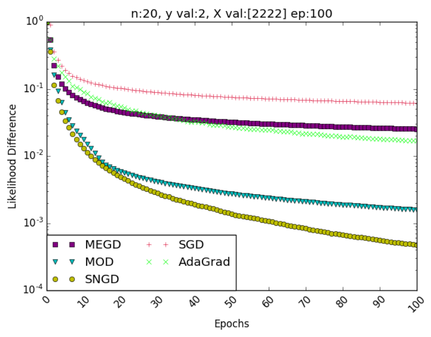
If β_m is the best approximation we got when minimizing $\mathcal{L}_S(\beta)$, let define the Normalized Loglikelihood Difference, $N\mathcal{L}_S D(\beta)$, as;

Definition 5.1.

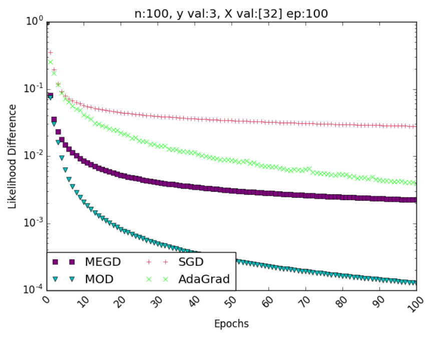
$$N\mathcal{L}_S D(\beta) = \frac{\mathcal{L}_S(\beta) - \mathcal{L}_S(\beta_m)}{\mathcal{L}_S(0) - \mathcal{L}_S(\beta_m)}$$

This function reflects how far a point of the manifold is from the best approximation in terms of its Loglikelihood function value, normalizing by the error made with the starting point $\beta_0 = 0$. Since the goal is to compare algorithms and their convergence to the Loglikelihood function minimum, we will run 100 instances of an experiment and compute the mean of the $N\mathcal{L}_S D(\beta_i)$ values, to approximate $\mathbb{E}_{\bar{\beta}}[N\mathcal{L}_S D(\beta_i)]$.

Figure 2



(a) Bivalued

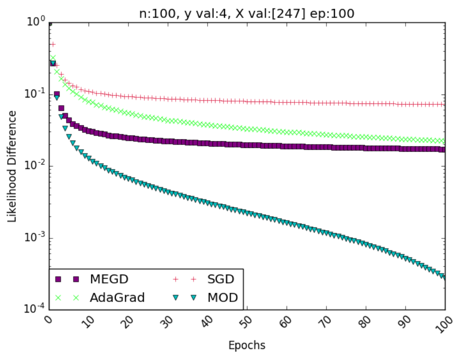


(b) Small

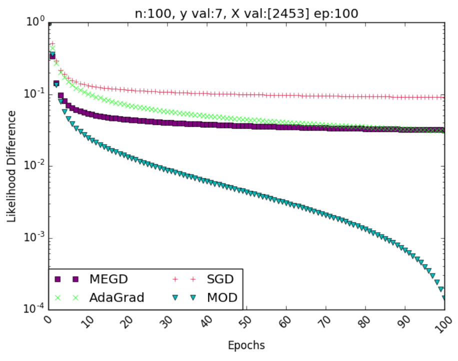
As we can see in figure 2a, Natural Gradient algorithms – SNGD and MOD – optimize the function much faster. However, just for this chart, we had to remove 6 experiments out of the total 100, because SNGD failed to give any result, falling into numerical errors. The lack of robustness shows up because in every stochastic gradient descent algorithm, first steps are wider and erratic, due to the lack of data. So, since SNGD updates the metric at every iteration, computing and using the metric matrix at some bizarre point β_i in fact slows convergence at the beginning, and it may even cause numerical incorrections. In addition, we find that SNGD is terribly slow and, as we know, it actually checks the sample S many more times, which obstructs comparison. Then, we forget about SNGD from now on, and we just test MOD as a Natural Gradient algorithm.

In the same figure, it's important to notice that the curves and their tendency are favorable for SNGD, MOD and AdaGrad, because they descend faster. That means, these algorithms will keep optimizing faster for next epochs. The slopes of SGD and MEGD end to be close to 0, which translates into a really slow convergence towards the minimum. The fact that y axis is log scaled only intensificates this property. We can deduce the same in Figure 2b since the charts are similar, except that SNGD is discarded.

Figure 3



(a) Medium



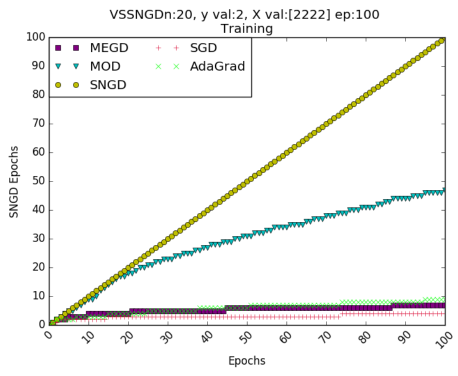
(b) Large

The charts of figure 3 really look alike. Natural gradient algorithm MOD surpasses extendedly the rest of the options. We emphasize the good results of MEGD: the curve it draws behaves just like SGD curve, imitating it, but it is situated in a lower position reaching much faster a nice answer but then stopping rapidly its convergence, as mentioned for Experiments Bivalued and Small.

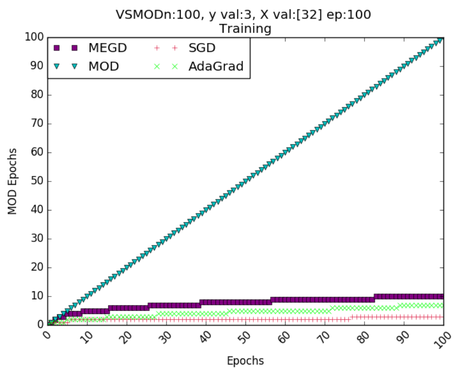
Results convince about the convergence improvement of Natural Gradient algorithms. Nevertheless, the complexity is higher. That's why it's interesting to ask about the data exploitation, which is treated next.

The second question is about the percent of data that the winning algorithm needs to surpass the best achievement of the rest. In figure 4 we compare the best algorithm (y axis) against the others (x axis). In both axis the total epochs is represented, and we draw the points where algorithms reach the same optimization successes. It is considered that an algorithm has secured an error level ϵ at iteration i if its error level at any iteration $j > i$ is smaller than ϵ. That way we make sure achievements aren't reached by luck.

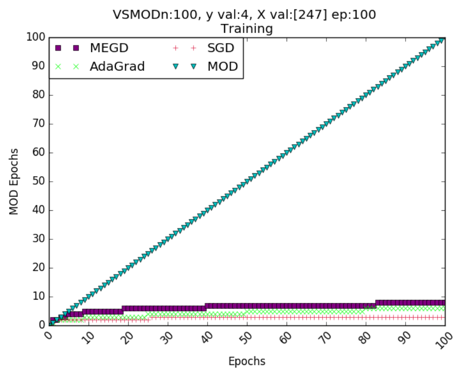
Figure 4



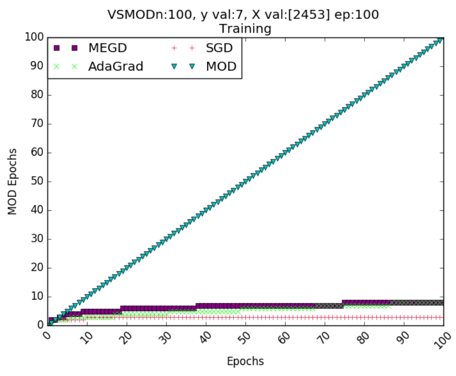
(a) Bivalued



(b) Small



(c) Medium



(d) Large

Start by looking at Figure 4a. In the vertical axis there is SNGD, which means SNGD minimizes better the function, and it secures smaller error levels completing less epochs of the process. The diagonal line standardizes SNGD achievements. This means, for example, that algorithm MOD, because of its flatter slope than diagonal, it secures error levels with the need for scanning more epochs in comparison with SNGD. Also, MOD's curve finishes with a value around 48 when the total 100 epochs have been run. This means that the error level MOD secures after 100 epochs is achieved by SNGD with just scanning 48 epochs. A strong improvement is made if we compare now the rest of the algorithms. SNGD and MOD only need about 10 epochs to surpass the best achievements that AdaGrad, SGD and MEGD ever get to get. Moreover, the slopes of AdaGrad, SGD and MEGD never seem to rise, which translates into a slower convergence to the optimum, even when the achievements request for them is softer.

Look now all charts in Figure 4. The savings of data are huge. About 10% of the whole process is what Natural Gradient algorithm MOD needs to obtain the best result other algorithms eventually reach in their whole process.

5.3 Natural Gradient algorithms as estimators

The questions we want to answer are:

- Do Natural Gradient estimates lead also to better optima for the conditional likelihood of a new sample?, and
- At which percentage of the data does the winning algorithm reach the quality of the losing one?

5.3.1 Expected Prediction Error

We have to make the point that chasing the MCL solution for a given sample doesn't work in general when we want good estimates for the MCL of a new sample drawn from the same distribution – or if we want estimates for the actual parameter $\bar{\beta}$ that generated both samples –. This fact becomes more accentuated if the sample is small in relation with the manifold dimension. To understand this, think about a 6 faces die rolled just 3 times. The ML estimator will set atleast 3 faces as not possible to occur, which in turn translates into a fatal error whenever a new sample contains any observation out of the 3 impossible faces.

Once we realize that solving MCL problem is a totally different task from the one we try to solve now, we modify a bit some settings. In most occasions that we use this kind of optimization tool, the dimension of the parameter space is really huge. However, we commonly have a lot of data.

As mentioned in [2], the problem of minimizing this error can be adressed by running an online gradient descent algorithm without the use of a training set. That is, the observations used don't belong to a fixed sample, but they are each time drawn from the distribution defined by $\bar{\beta}$ instead. So, the most important change is getting new samples once we observed it once, instead of reordering the sample S at the end of every epoch as we have done untill now. That way we take profit of the large amount of data that we have. We change the name *epoch* by *era* to make the difference between both situations – epoch setting repeats samples, era setting draws new samples –.

Definition 5.2 (Expected Prediction Error)**.**

$$Err(\beta) = \mathbb{E}_{\bar{\beta}}\left[\mathbb{E}_S\left[D_{KL}\bar{\beta}, \beta(S))\right]\right] = \int \left[\int_{\mathcal{P}(\Omega)} D_{KL}(\bar{\beta}, \beta(S)) \cdot P(S|\bar{\beta})dS\right] P(\bar{\beta})d\bar{\beta}$$

The expected prediction error can be understood as the average divergence that the estimator would take against the real parameter if a new sample is drawn.

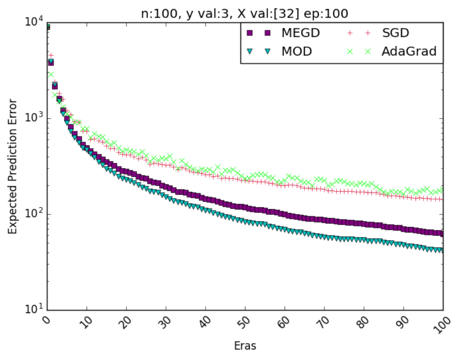
We approximate $\mathbb{E}_S$ by Montecarlo and as before, we run many instances and compute the mean to approximate $\mathbb{E}_{\bar{\beta}}$. So more precisely we do:

$$Err(\beta) \approx \frac{\sum_{i=0}^{m} D_{KL}(\bar{\beta}_i, \beta(S))}{m+1}$$

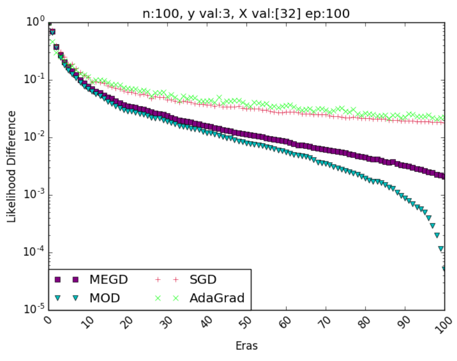
where m is the number of instances ran. We need to also approximate $D_{KL}(\bar{\beta}_i, \beta(S))$ since the probabilities of $P(X)$ are unknown. We do so by taking a 10 times larger sample from $P(y|X, \bar{\beta})$ to empirically estimate the distribution $P(X)$.

In addition, the charts that compute $N\mathcal{L}_S D(\beta)$ are shown to see how well the algorithms respond to the optimization question of past section – remember that now algorithms run over *eras* instead of *epochs*–.

Figure 5: Small

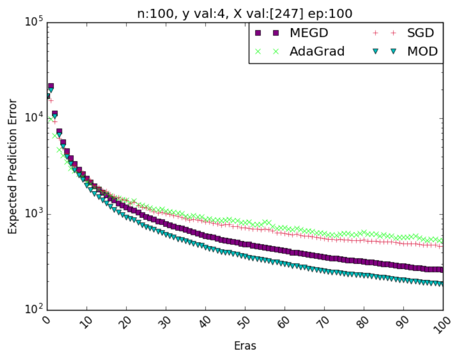


(a) $Err(\beta)$

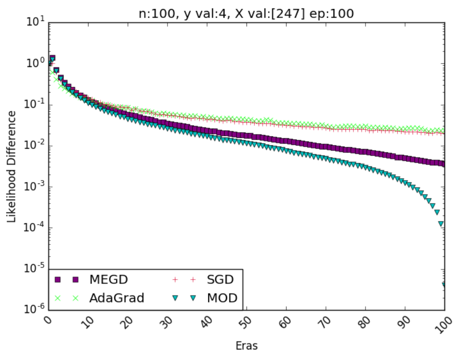


(b) $N\mathcal{L}D(\beta)$

Figure 6: Medium

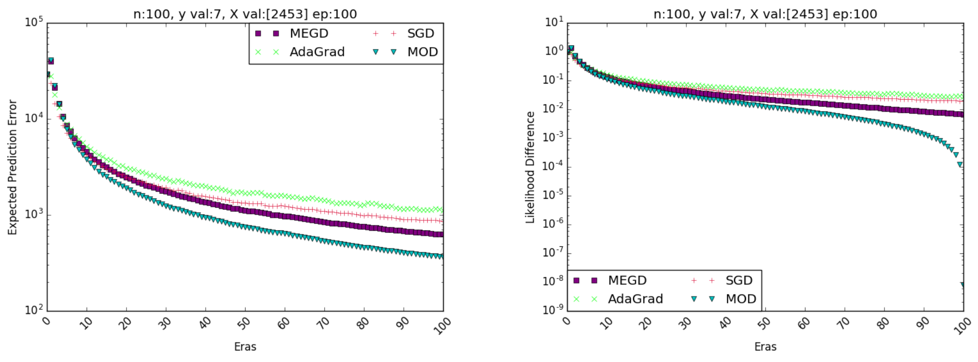


(a) $Err(\beta)$



(b) $N\mathcal{L}D(\beta)$

Figure 7: Large



(a) $Err(\beta)$ (b) $N\mathcal{L}D(\beta)$

Table 2

$Err(\beta)$	AdaGrad	SGD	MEGD	MOD
Experiment Small	181.3684	143.4095	62.5183	41.6386
Experiment Medium	535.9469	466.0853	262.0726	185.0822
Experiment Large	1112.7022	856.8210	626.9184	365.6101

No matter the Experiment we run, the resulting graphs show more or less the same. When testing $Err(\beta)$ in figures 5,6 and 7 it is clear that AdaGrad fails to reduce the error against regular SGD. Notice also that the curve drawn by AdaGrad is not as smooth as other ones, which implies a higher variance in its response. We can see a big improvement with MEGD with respect to standard algorithms. But again, MOD algorithm arises as the best option.

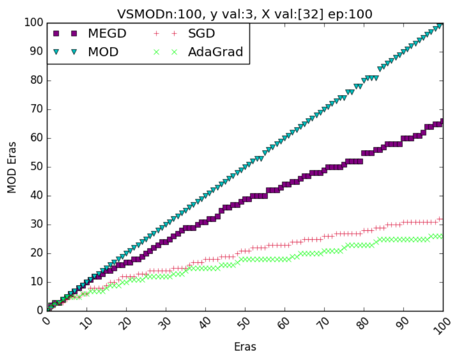
Table 2 lists the errors of final estimators for every algorithm. MEGD decreases significantly the $Err(\beta)$ compared to SGD and AdaGrad. In particular, MOD error is especially lower than MEGD in Experiment Large. This advises to use MOD algorithm over *eras* if we want to find a good predictor parameter.

Moreover, we added the graphs reflecting convergence when solving the MCL problem. We conclude that if we run over *eras*, Natural Gradient algorithms provide good results for $N\mathcal{L}D(\beta)$ and for $Err(\beta)$.

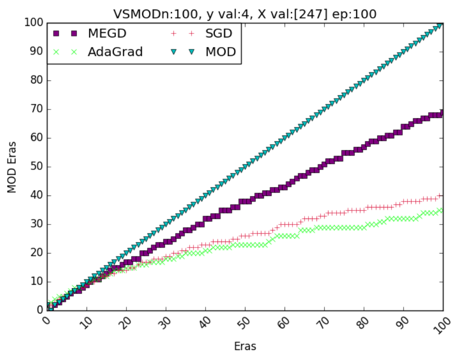
Let's try to answer last question of this section. We build the same comparison charts explained previously. This tells for example how much sample percent could be saved in order to reach the same answers regular algorithms find exploiting the whole sample.

Curves in figure 8 are quite straight lines, so the convergence of all algorithms is proporcional to the one of MOD. That means, other algorithms don't seem to ever get better than MOD.

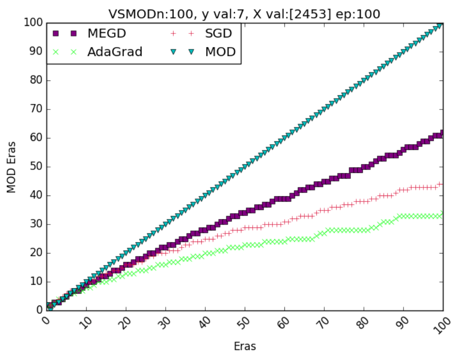
Figure 8



(a) Small



(b) Medium



(c) Large

Now we comment the data savings that MOD can afford . MOD needs between 60% and 70% of the sample to improve MEGD best result. MEGD and MOD need about the 35%-45% of the sample for beeing able to surpase the best results standard algorithms reach scanning the whole sample.

MEGD is interesting, since the sample percent reduction made by MOD is not large, however it has a much lower computational complexity.

6 Annex

6.1 Proof of Proposition 4.2

Proof.

Notation. To alleviate the notation, we let $P(y|x)$ denote $P(y|x,\overset{*}{\beta})$ and $Q(y|x)$ denote $P(y|x,\beta)$.

Recall section 4.4. A possible strategy to obtain G_β is to differentiate twice the divergence;

$$D_{KL}(P(y|X),Q(y|X)) = \sum_{x\in X} P(x) \sum_{y'\in y} P(y'|x)\cdot \log \frac{P(y'|x)}{Q(y'|x)}$$

with respect to β, so only $Q(y'|x)$ term is actually affected by the differentiation. As always, $|x^l| = t+1$, $|y| = s+1$

Ignore the term $P(y'|x)\cdot\log(P(y'|x))$ since it is not afected by differentiation. Derivate with respect to $\beta_1 := \beta^{y^1}_{x^i,u}$ and $\beta_2 := \beta^{y^2}_{x^j,v}$ – check equation 6 in section 4.3 –. First, calculate the 2 next derivatives;

- $\frac{\partial}{\partial\beta_1}\log Q(y'|x) = \frac{1}{Q(y'|x)}\cdot\frac{\partial}{\partial\beta_1}Q(y'|x) = \begin{cases} 0 & \text{if } x^i \neq u \\ -Q(y^1|x) & \text{if } x^i = u,\ y' \neq y^1 \\ 1-Q(y^1|x) & \text{if } x^i = u,\ y' = y^1 \end{cases}$

- $\frac{\partial}{\partial\beta_2}Q(y'|x) = \begin{cases} 0 & \text{if } x^j \neq v \\ -Q(y'|x)\cdot Q(y^2|x) & \text{if } x^j = v,\ y' \neq y^2 \\ Q(y'|x)\cdot(1-Q(y^2|x)) & \text{if } x^j = v,\ y' = y^2 \end{cases}$

Now proceed to compute the derivatives of $-P(y'|x)\cdot\log(Q(y'|x))$. We will end up having to distinguish cases when $y^1 \neq y^2$ and when $y^1 = y^2$;

$$\frac{\partial^2}{\partial\beta_1\partial\beta_2} - P(y'|x)\cdot\log(Q(y'|x)) = -P(y'|x)\cdot\frac{\partial}{\partial\beta_2}(\frac{\partial}{\partial\beta_1}\log Q(y'|x)) =$$

$$\begin{cases} 0 & \text{if } x^i \neq u \\ -P(y'|x)\cdot\frac{\partial}{\partial\beta_2}(-Q(y^1|x) & \text{if } x^i = u,\ y' \neq y^1 \\ -P(y'|x)\cdot\frac{\partial}{\partial\beta_2}(1-Q(y^1|x) & \text{if } x^i = u,\ y' = y^1 \end{cases} = \begin{cases} 0 & \text{if } x^i \neq u \\ P(y'|x)\cdot\frac{\partial}{\partial\beta_2}(Q(y^1|x) & \text{if } x^i = u \end{cases} =$$

$$\begin{cases} 0 & \text{if } x^i \neq u \text{ or } x^j \neq v \\ P(y'|x)\cdot(-Q(y^1|x)\cdot Q(y^2|x) & \text{if } x^i = u,\ x^j = v,\ y^2 \neq y^1 \\ P(y'|x)\cdot Q(y^1|x)\cdot(1-Q(y^2|x) & \text{if } x^i = u,\ x^j = v,\ y^2 = y^1 \end{cases}$$

As we can see, once we bring back the sum for $y' \in y$ of the KL-divergence, the only term inside the sum that depends on y' is $P(y'|x)$, wich means that we can factor out the other terms and we can do the simplification $\sum_{y' \in y} P(y'|x) = 1$.

Then substitute $\overset{*}{\beta} = \beta$ and conclude that the FIM is given by;

if $y^2 = y^1$:

$$g_{\beta_1,\beta_2} = \sum_{x \in X | x^i = u, x^j = v} P(x) \cdot P(y^1|x) \cdot (1 - P(y^2|x))$$

if $y^2 \neq y^1$:

$$g_{\beta_1,\beta_2} = - \sum_{x \in X | x^i = u, x^j = v} P(x) \cdot P(y^1|x) \cdot P(y^2|x)$$

As wanted. The parts involving $\beta_3 := \beta^{y^3}$ and $\beta_4 := \beta^{y^4}$ of the proposition follow analogously. □

6.2 Proof of Proposition 4.3

Proof. Let $\beta_1 := \beta^{y^1}_{x^i,u}$ and $\beta_2 := \beta^{y^2}_{x^j,v}$. Recall 2 equations for the expression of matrix G_β of proposition 4.2;

if $y^2 = y^1 = r$:

$$g_{\beta_1,\beta_2} = \sum_{x_p \in X | x^i_p = u, x^j_p = v} d_{r,p} \tag{7}$$

if $y^2 \neq y^1$:

$$g_{\beta_1,\beta_2} = - \sum_{x_p \in X | x^i_p = u, x^j_p = v} e_{y^2,y^1,p} \tag{8}$$

with

$$d_{r,p} = P(x_p)P(y = r|x_p)(1 - P(y = r|x_p))$$

$$e_{r_1,r_2,p} = P(x_p)P(y = r_1|x_p)P(y = r_2|x_p)$$

Observation. Proposition 4.2 uses $\beta_1, \beta_2, \beta_3$ and β_4 to express matrix G_β because they serve as coordinates. More precisely, G_β matrix rows and columns are sorted following the vector

$$(\beta^1, \beta^2, \cdots, \beta^s) \text{ with } \beta^{y'} = \left(\beta^{y'}_0 \, \beta^{y'}_{x^1,1} \beta^{y'}_{x^1,2} \cdots \beta^{y'}_{x^1,t} \beta^{y'}_{x^2,1} \cdots \beta^{y'}_{x^2,t} \cdots \beta^{y'}_{x^k,t}\right)$$

Let the matrices A, M and B of section 4.4.2. Supose B has dimension $n + 1 \times n + 1$. Let's develop the product $AMBA^t$;

$$
\begin{aligned}
AMBA^t = &\quad A\left(\begin{pmatrix} q_1 & 0 & \cdots & 0 \\ 0 & q_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & q_s \end{pmatrix} - \begin{pmatrix} q_1 \\ q_2 \\ \vdots \\ q_s \end{pmatrix} \cdot \begin{pmatrix} q_1 & q_2 & \cdots & q_s \end{pmatrix}\right) BA^t \\
= &\begin{pmatrix} A(q_1 - q_1^2)BA^t & -Aq_1q_2BA^t & \cdots & -Aq_1q_sBA^t \\ -Aq_1q_2BA^t & A(q_2 - q_2^2)BA^t & \ddots & \vdots \\ \vdots & \ddots & \ddots & -Aq_{s-1}q_sBA^t \\ -Aq_1q_sBA^t & \cdots & -Aq_{s-1}q_sBA^t & A(q_s - q_s^2)BA^t \end{pmatrix}
\end{aligned}
$$

Distinguish between blocks in the diagonal and the other blocks.

Case 1: blocks in the diagonal

The r-th block in the diagonal should correspond with the equation 7 where $y^2 = y^1 = r$. Let's check the matching by developing the block expression;

$$Aq_r(Id - q_r)BA^t = AD_rA^t$$

with

$$D_r = \begin{pmatrix} d_{r,0} & & 0 \\ & \ddots & \\ 0 & & d_{r,n} \end{pmatrix}$$

Notice that $AA^t = \left(\begin{array}{c|c} \sum_{x \in X} 1 & N \\ \hline N^t & M_* \end{array}\right)$ with $M_* = \begin{pmatrix} \sum_{x^1=1,x^1=1}^{x \in X} 1 & \cdots & \sum_{x^1=1,x^k=t}^{x \in X} 1 \\ \vdots & \ddots & \vdots \\ \sum_{x^k=t,x^1=1}^{x \in X} 1 & \cdots & \sum_{x^k=t,x^k=t}^{x \in X} 1 \end{pmatrix}$,

$N = \left(\sum_{x_p^1=1}^{x_p \in X} 1 \cdots \sum_{x_p^1=t}^{x_p \in X} 1 \;\; \sum_{x_p^2=1}^{x_p \in X} 1 \cdots\cdots \sum_{x_p^k=t}^{x_p \in X} 1\right)$

This implies $AD_rA^t = \left(\begin{array}{c|ccc} \sum_{x_p \in X} d_{r,p} & & N_d & \\ \hline & \sum_{x_p^1=1,x_p^1=1}^{x_p \in X} d_{r,p} & \cdots & \sum_{x_p^1=1,x_p^k=t}^{x_p \in X} d_{r,p} \\ N_d^t & \vdots & \ddots & \vdots \\ & \sum_{x_p^k=t,x_p^1=1}^{x_p \in X} d_{r,p} & \cdots & \sum_{x_p^k=t,x_p^k=t}^{x_p \in X} d_{r,p} \end{array}\right)$

with $N_d = \left(\sum_{x_p^1=1}^{x_p \in X} d_{r,p} \cdots \sum_{x_p^1=t}^{x_p \in X} d_{r,p} \;\; \sum_{x_p^2=1}^{x_p \in X} d_{r,p} \cdots\cdots \sum_{x_p^k=t}^{x_p \in X} d_{r,p}\right)$

which is exactly the same as equation 7 and this part is checked. The equations involving $\beta_3 := \beta^{y^3}$ and $\beta_4 := \beta^{y^4}$ with $y^2 = y^3$ and $y^3 = y^4$ of proposition 4.2 correspond to the first row and/or first column of the blocks which are also matching perfectly.

Case 2: blocks outside the diagonal

The r_1 row r_2 column block, with $r_1 \neq r_2$, is outside the diagonal. Similarly, it perfectly matches with equation 8 where $r_1 = y^1$ and $r_2 = y^2$. Proceeding equivalently, develop the block $-Aq_{r_1}q_{r_2}BA^t$ to check the matching with equation 8. This finishes the proof.

$\square$

6.3 Proof of Lemma 4.4

Proof. Proof begins showing that;

$$ABA^t = \frac{AA^t}{n+1} = \frac{1}{(t+1)^2}\begin{pmatrix} (t+1)^2 & t+1 & \cdots & \cdots & t+1 \\ t+1 & (t+1)Id_t & K_t & \cdots & K_t \\ \vdots & K_t & (t+1)Id_t & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & K_t \\ t+1 & K_t & \cdots & K_t & (t+1)Id_t \end{pmatrix}$$

where K_t is a $t \times t$ matrix with ones at every entry and $|X| = n+1$. Notice that if the maximum entropy distribution of space $P(X)$ is selected, then $P(x) = \frac{1}{n+1}\ \forall x \in X$. This implies that $B = \frac{1}{n+1}Id$, which proves first equality $ABA^t = \frac{AA^t}{n+1}$.

For the second equality, recall that proof 6.2 says

$$AA^t = \left(\begin{array}{c|c} \sum_{x\in X} 1 & N \\ \hline N^t & M_* \end{array}\right) \text{ with } M_* = \begin{pmatrix} \sum_{x^1=1,x^1=1}^{x\in X} 1 & \cdots & \sum_{x^1=1,x^k=t}^{x\in X} 1 \\ \vdots & \ddots & \vdots \\ \sum_{x^k=t,x^1=1}^{x\in X} 1 & \cdots & \sum_{x^k=t,x^k=t}^{x\in X} 1 \end{pmatrix},$$

$N = \left(\sum_{x_p^1=1}^{x_p\in X} 1 \cdots \sum_{x_p^1=t}^{x_p\in X} 1 \;\; \sum_{x_p^2=1}^{x_p\in X} 1 \cdots\cdots \sum_{x_p^k=t}^{x_p\in X} 1\right)$

Since $|x^i| = t+1\ \forall i \in \{1, ..., k\}$, then;

$$\sum_{x^i=u,x^j=v}^{x\in X} = \begin{cases} 0 & \text{if } i=j \text{ and } u \neq v \\ \frac{n+1}{t+1} & \text{if } i=j \text{ and } u=v \text{ and therefore} \\ \frac{n+1}{(t+1)^2} & \text{if } i \neq j \end{cases}$$

$$AA^t = \left(\begin{array}{c|cccc} n+1 & & \frac{n+1}{t+1} & \cdots & \frac{n+1}{t+1} \\ \hline & \frac{n+1}{t+1}Id_t & \frac{n+1}{(t+1)^2}K_t & \cdots & \frac{n+1}{(t+1)^2}K_t \\ \frac{n+1}{t+1} & \frac{n+1}{(t+1)^2}K_t & \frac{n+1}{t+1}Id_t & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & \frac{n+1}{(t+1)^2}K_t \\ \frac{n+1}{t+1} & \frac{n+1}{(t+1)^2}K_t & \cdots & \frac{n+1}{(t+1)^2}K_t & \frac{n+1}{t+1}Id_t \end{array}\right) \text{ which finally implies}$$

$$\frac{AA^t}{n+1} = \frac{1}{(t+1)^2}\begin{pmatrix} (t+1)^2 & t+1 & \cdots & \cdots & t+1 \\ t+1 & (t+1)Id_t & K_t & \cdots & K_t \\ \vdots & K_t & (t+1)Id_t & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & K_t \\ t+1 & K_t & \cdots & K_t & (t+1)Id_t \end{pmatrix}$$

as wanted.

Notice that AA^t is a square matrix of dimension $1 + tk$. The lema gives the expression of the inverse of $\frac{AA^t}{n+1}$. Last step of the proof just checks that it actually is the inverse matrix;

$\frac{AA^t}{n+1}\left(\frac{AA^t}{n+1}\right)^{-1} =$

$$\frac{1}{t+1}\left(\begin{array}{c|ccccc} (t+1)^2 & t+1 & \cdots & \cdots & t+1 \\ \hline t+1 & (t+1)Id_t & K_t & \cdots & K_t \\ \vdots & K_t & (t+1)Id_t & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & K_t \\ t+1 & K_t & \cdots & K_t & (t+1)Id_t \end{array}\right)\left(\begin{array}{c|cccc} \frac{1+tk}{t+1} & -1 & \cdots & \cdots & -1 \\ \hline -1 & L_t & 0 & \cdots & 0 \\ \vdots & 0 & L_t & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & 0 \\ -1 & 0 & \cdots & 0 & L_t \end{array}\right)$$

$= \left(\begin{array}{c|c} B_1 & B_2 \\ \hline B_3 & B_4 \end{array}\right)$ such that;

$B_1 = \frac{(t+1)^2}{t+1}\frac{1+tk}{t+1} - \frac{t+1}{t+1}tk = 1$

$B_2 = \begin{pmatrix} b_2 & \cdots & b_2 \end{pmatrix}$ with $b_2 = -\frac{(t+1)^2}{t+1} + 2 + \sum_{i=1}^{t-1} 1 = -(t+1) + (t+1) = 0$

$B_3 = \begin{pmatrix} b_3 \\ \vdots \\ b_3 \end{pmatrix}$ with $b_3 = \frac{1+tk-(t+1)-(k-1)t}{t+1} = \frac{1+tk-t-1-tk+t}{t+1} = 0$

$$B_4 = \frac{1}{t+1}\left(\begin{array}{c|cccc} t+1 & (t+1)Id_t & K_t & \cdots & K_t \\ \vdots & K_t & (t+1)Id_t & \ddots & \vdots \\ \vdots & \vdots & \ddots & \ddots & K_t \\ t+1 & K_t & \cdots & K_t & (t+1)Id_t \end{array}\right)\left(\begin{array}{cccc} -1 & \cdots & \cdots & -1 \\ \hline L_t & 0 & \cdots & 0 \\ 0 & L_t & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & L_t \end{array}\right)$$

$$= -K_{tk} + \begin{pmatrix} L_t & \frac{K_tL_t}{t+1} & \cdots & \frac{K_tL_t}{t+1} \\ \frac{K_tL_t}{t+1} & L_t & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ \frac{K_tL_t}{t+1} & \cdots & \frac{K_tL_t}{t+1} & L_t \end{pmatrix} = -K_{tk} + \begin{pmatrix} L_t & K_t & \cdots & K_t \\ K_t & L_t & \ddots & \vdots \\ \vdots & \ddots & \ddots & \vdots \\ K_t & \cdots & K_t & L_t \end{pmatrix} = -K_{tk} + L_{tk} = Id_{tk}$$

Then, finally

$\frac{AA^t}{n+1}\left(\frac{AA^t}{n+1}\right)^{-1} = \left(\begin{array}{c|c} B_1 & B_2 \\ \hline B_3 & B_4 \end{array}\right) = \left(\begin{array}{c|c} 1 & 0 \\ \hline 0 & Id_{tk} \end{array}\right) = Id_{1+tk}$ and the result is proved.

□

6.4 Proof of Proposition 4.6

Proof. Since $\beta = 0$ and $P(y'|x,\beta) = \frac{1}{1+s}$ and then $p_j = \frac{Id_{n+1}}{s+1}$. That means matrix M takes the shape;

$$M = \frac{1}{(s+1)^2}\begin{pmatrix} sId_{n+1} & -Id_{n+1} & \cdots & -Id_{n+1} \\ -Id_{n+1} & sId_{n+1} & \ddots & \vdots \\ \vdots & \ddots & \ddots & -Id_{n+1} \\ -Id_{n+1} & \cdots & -Id_{n+1} & sId_{n+1} \end{pmatrix}$$

As a particular case of proposition 4.3, it is $G_0 = AMBA^t$. In fact

$$G_0 = AMBA^t = \frac{1}{(s+1)^2}\begin{pmatrix} sABA^t & -ABA^t & \cdots & -ABA^t \\ -ABA^t & sABA^t & \ddots & \vdots \\ \vdots & \ddots & \ddots & -ABA^t \\ -ABA^t & \cdots & -ABA^t & sABA^t \end{pmatrix} =$$

$$= \frac{1}{(s+1)^2}\begin{pmatrix} sId & -Id & \cdots & -Id \\ -Id & sId & \ddots & \vdots \\ \vdots & \ddots & \ddots & -Id \\ -Id & \cdots & -Id & sId \end{pmatrix} ABA^t$$

with $Id = Id_{kt+1}$. As indicated in proof 6.3, notice that if the maximum entropy distribution of space $P(X)$ is selected, then $P(x) = \frac{1}{n+1}\ \forall x \in X$. This implies that $B = \frac{1}{n+1}Id$, which proves the equality $ABA^t = \frac{AA^t}{n+1}$. Finally, verify that the matrix of the proposition is actually the inverse of G_0;

$$G_0 \cdot G_0^{-1} = \frac{1}{s+1}\begin{pmatrix} 2Id & Id & \cdots & Id \\ Id & 2Id & \ddots & \vdots \\ \vdots & \ddots & \ddots & Id \\ Id & \cdots & Id & 2Id \end{pmatrix} \cdot \begin{pmatrix} sId & -Id & \cdots & -Id \\ -Id & sId & \ddots & \vdots \\ \vdots & \ddots & \ddots & -Id \\ -Id & \cdots & -Id & sId \end{pmatrix} = Id$$

□

6.5 Proof of Proposition 4.7

Proof. The goal is to prove that $AMBA^t = \frac{\overset{*}{A}\overset{*}{M}\overset{*}{A}^t}{n+1}$. It's enough to prove the equality for a block of matrix M and $\overset{*}{M}$. That is, the proof is based on checking that

$$Aq_i(Id - q_i)BA^t = \frac{\overset{*}{A}p_i(Id - p_i)\overset{*}{A}^t}{n+1} \tag{9}$$

On the one hand, we develop left hand side of equation 9. Notice that if $P(X)$ is the empirical distribution observed given the sample $S = \{(y_0, x_0), ..., (y_n, x_n)\}$, then $P(x_i) = \frac{n_i}{n+1}$ with n_i the number of apparitions of x_i in S. Therefore

$$B = \tfrac{1}{n+1}\begin{pmatrix} n_0 & & 0 \\ & \ddots & \\ 0 & & n_s \end{pmatrix} \rightarrow$$

$$Aq_i(Id-q_i)BA^t = \left(\begin{array}{c|ccc} \sum_{x_p\in X} d_{r,p} & & N_d & \\ \hline & \sum_{x_p^1=1,x_p^1=1}^{x_p\in X} d_{r,p} & \cdots & \sum_{x_p^1=1,x_p^k=t}^{x_p\in X} d_{r,p} \\ N_d^t & \vdots & \ddots & \vdots \\ & \sum_{x_p^k=t,x_p^1=1}^{x_p\in X} d_{r,p} & \cdots & \sum_{x_p^k=t,x_p^k=t}^{x_p\in X} d_{r,p} \end{array}\right)$$

with $N_d = \left(\sum_{x_p^1=1}^{x_p\in X} d_{r,p} \cdots \sum_{x_p^1=t}^{x_p\in X} d_{r,p}\ \ \sum_{x_p^2=1}^{x_p\in X} d_{r,p} \cdots\cdots \sum_{x_p^k=t}^{x_p\in X} d_{r,p}\right)$ and

$d_{r,p} = P(x_p)P(y=r|x_p)(1-P(y=r|x_p)) = \frac{n_p}{n+1}P(y=r|x_p)(1-P(y=r|x_p))$. Let $d^*_{r,p} = P(y=r|x_p)(1-P(y=r|x_p))$ then $d_{r,p} = \frac{n_p}{n+1}d^*_{r,p}$. Moreover, observe that if x_i has not appeared in S, then $P(x_i)=0$ and then all summations can be simplified. Let $S_X = \{x \in X | x \text{ appears in } S\}$. Let's apply these modifications to last expression;

$$Aq_i(Id-q_i)BA^t = \frac{1}{n+1}\left(\begin{array}{c|ccc} \sum_{x_p\in S_X} n_p d^*_{r,p} & & N_d & \\ \hline & \sum_{x_p^1=1,x_p^1=1}^{x_p\in S_X} n_p d^*_{r,p} & \cdots & \sum_{x_p^1=1,x_p^k=t}^{x_p\in S_X} n_p d^*_{r,p} \\ N_d^t & \vdots & \ddots & \vdots \\ & \sum_{x_p^k=t,x_p^1=1}^{x_p\in S_X} n_p d^*_{r,p} & \cdots & \sum_{x_p^k=t,x_p^k=t}^{x_p\in S_X} n_p d^*_{r,p} \end{array}\right)$$

with $N_d = \left(\sum_{x_p^1=1}^{x_p\in S_X} n_p d^*_{r,p} \cdots \sum_{x_p^1=t}^{x_p\in S_X} n_p d^*_{r,p}\ \ \sum_{x_p^2=1}^{x_p\in S_X} n_p d^*_{r,p} \cdots\cdots \sum_{x_p^k=t}^{x_p\in S_X} n_p d^*_{r,p}\right)$.

On the other hand, consider right hand side of equation 9. It is

$$\frac{\overset{*}{A}p_i(Id-p_i)\overset{*}{A}^t}{n+1} = \frac{1}{n+1}\left(\begin{array}{c|ccc} \sum_{0\le p\le n} d^*_{r,p} & & N_d & \\ \hline & \sum_{x_p^1=1,x_p^1=1}^{0\le p\le n} d^*_{r,p} & \cdots & \sum_{x_p^1=1,x_p^k=t}^{0\le p\le n} d^*_{r,p} \\ N_d^t & \vdots & \ddots & \vdots \\ & \sum_{x_p^k=t,x_p^1=1}^{0\le p\le n} d^*_{r,p} & \cdots & \sum_{x_p^k=t,x_p^k=t}^{0\le p\le n} d^*_{r,p} \end{array}\right)$$

with $N_d = \left(\sum_{x_p^1=1}^{0\le p\le n} d^*_{r,p} \cdots \sum_{x_p^1=t}^{0\le p\le n} d^*_{r,p}\ \ \sum_{x_p^2=1}^{0\le p\le n} d^*_{r,p} \cdots\cdots \sum_{x_p^k=t}^{0\le p\le n} d^*_{r,p}\right)$. Remember that n_i is the number of apparitions of x_i in S, so it is possible to simplify the summations.

$$\frac{\overset{*}{A}p_i(Id-p_i)\overset{*}{A}^t}{n+1} = \frac{1}{n+1}\left(\begin{array}{c|ccc} \sum_{x_p\in S_X} n_p d^*_{r,p} & & N_d & \\ \hline & \sum_{x_p^1=1,x_p^1=1}^{x_p\in S_X} n_p d^*_{r,p} & \cdots & \sum_{x_p^1=1,x_p^k=t}^{x_p\in S_X} n_p d^*_{r,p} \\ N_d^t & \vdots & \ddots & \vdots \\ & \sum_{x_p^k=t,x_p^1=1}^{x_p\in S_X} n_p d^*_{r,p} & \cdots & \sum_{x_p^k=t,x_p^k=t}^{x_p\in S_X} n_p d^*_{r,p} \end{array}\right)$$

with $N_d = \left(\sum_{x_p^1=1}^{x_p \in S_X} n_p d^*_{r,p} \cdots \sum_{x_p^1=t}^{x_p \in S_X} n_p d^*_{r,p} \;\; \sum_{x_p^2=1}^{x_p \in S_X} n_p d^*_{r,p} \cdots \cdots \sum_{x_p^k=t}^{x_p \in S_X} n_p d^*_{r,p} \right)$ and equation 9 holds. This directly implies

$$Aq_i q_j B A^t = \frac{\overset{*}{A} p_i p_j \overset{*}{A}^t}{n+1}$$

and so the result follows. $\square$

7 Conclusions

Natural gradient may improve a lot the convergence speed of Linesearch strategy to optimize a smooth function. We have even seen how ML problem can be solved perfectly, imitating the ML estimator, by the SNGD algorithm over a specific Riemannian Manifold with the Fisher Information Metric (FIM). The results have been really positive when we tried to solve the MCL problem too, which is a way harder and more interesting problem. We compared Natural Gradient algorithms against the classic SGD algorithm and AdaGrad, which is an adaptive learning rate algorithm very used nowadays. We concluded that Natural Gradient algorithms, together with the Riemannian Manifold equipped with the FIM, solves with a faster convergence the ML problem and the MCL problem.

When the data is scarce compared to the manifold dimension, approaching MCL doesn't translate into finding the underlying conditional probability distribution generating of the sample observations. That's why, we proceed to run the algorithms over a slighlty different setting. We fed the algorithms with abundant data, never replicating the sample. At this point, Natural Gradient algorithms showed to be better optimizers and provide better estimates of new samples as well.

A deeper study of the relation between Riemannian manifolds and Linesearch optimization appears as a promising future work research line. Recall the Linesearch strategy is "update the parameter following a straight line with direction g_i for a distance r_i". Let us highlight the concepts "straight line", "direction g_i" and "distance r_i", which are concepts directly related to the metric and connection of a Riemannian Manifold. This strategy, even though it is reasonable, it is parametrization dependent. We can say that in this project, as well as in most literatures in the references, the concept "direction g_i" becomes independent of the parametrization thanks to the Natural Gradient. However, every Natural Gradient algorithm still depends on the parametrization. That means, we could probably find a parametrization where the Natural Gradient performs worse, still using the FIM. A wider study of connections in Riemannian Manifolds is needed to have an algorithm that doesn't deppend at all on the parametrization taken [10, 1].

This project focused on Riemannian Manifolds enriched with the FIM, since the function we want to optimize is the likelihood or the conditional likelihood function. And as we have seen, these Riemannian Manifolds seem to be really appropriate options. We are interested on an abstraction of this. If we have some other kind of smooth function to optimize, we wonder about a generic way to select which Riemannian Manifold and parametrization are the most advantageous to use, and why.

References

[1] Bonnabel, S, Stochastic gradient descent on riemannian manifolds, in: *IEEE Transactions on Automatic Control*, vol. 58, no. 9:pp. 2217–2229, Sept 2013, ISSN 0018-9286, doi:10.1109/TAC.2013.2254619.

[2] Bottou, Léon, On-line learning and stochastic approximations, in: David Saad, ed., *On-line Learning in Neural Networks*, pp. 9–42, New York, NY, USA: Cambridge University Press, 1998, ISBN 0-521-65263-4.

[3] Cerquides, Jesús and Sánchez, Borja, *SNGD and MLR problem*, [GitLab], sep 2018, URL `https://gitlab.iiia.csic.es/borja/TFM_SGA`.

[4] do Carmo, Manfredo Perdigão, *Riemannian geometry; 2nd ed.*, Mathematics : theory and applications, Boston, MA: Birkhäuser, 1992, ISBN 978-0-8176-3490-2.

[5] Duchi, John; Hazan, Elad and Singer, Yoram, Adaptive subgradient methods for online learning and stochastic optimization, in: *Journal of Machine Learning Research*, vol. 12, no. Jul:pp. 2121–2159, 2011, ISSN 1532-4435.

[6] Feelders, Ad and Ivanovs, Jevgenijs, Discriminative scoring of bayesian network classifiers: a comparative study, in: *Proceedings of the 3rd European Workshop on Probabilistic Graphical Models, PGM 2006*, pp. 75–82, 01 2006, URL `https://www.researchgate.net/publication/221135934_Discriminative_Scoring_of_Bayesian_Network_Classifiers_a_Comparative_Study`.

[7] Gould, Nicholas, *An introduction to algorithms for continuous optimization*, Oxford University Computing Laboratory Notes, 2006, URL `http://www.numerical.rl.ac.uk/people/nimg/oupartc/lectures/paper/paper.pdf`.

[8] Murata, Noboru, A statistical study of on-line learning, in: David Saad, ed., *On-line Learning in Neural Networks*, pp. 63–92, New York, NY, USA: Cambridge University Press, 1998, ISBN 0-521-65263-4.

[9] Shun-ichi, Amari, Natural gradient works efficiently in learning, in: *Neural Computation*, vol. 10, no. 2:pp. 251–276, Feb. 1998, ISSN 0899-7667, doi:10.1162/089976698300017746.

[10] Shun-ichi, Amari, *Information Geometry and Its Applications: Convex Function and Dually Flat Manifold*, pp. 75–102, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, ISBN 978-3-642-00826-9, doi:10.1007/978-3-642-00826-9_4.

[11] Shun-ichi, Amari, *Information geometry and its applications*, Springer, 2016, ISBN 978-4-431-55977-1.

[12] Sun, Ke and Nielsen, Frank, Relative natural gradient for learning large complex models, in: *ArXiv*, Jun. 2016, URL `http://arxiv.org/abs/1606.06069`.

[13] Zaidi, N. A.; Carman, M. J.; Cerquides, J. and Webb, G. I., Naive-bayes inspired effective pre-conditioner for speeding-up logistic regression, in: *2014 IEEE International Conference on Data Mining*, pp. 1097–1102, Dec 2014, ISSN 1550-4786, doi:10.1109/ICDM.2014.53.

[14] Zeiler, Matthew D., ADADELTA: an adaptive learning rate method, in: *ArXiv*, dec 2012, URL `http://arxiv.org/abs/1212.5701`.