lions@epfl

EE-556: MATHEMATICS OF DATA: FROM THEORY TO COMPUTATION
LABORATORY FOR INFORMATION AND INFERENCE SYSTEMS
FALL 2021

EPFL

INSTRUCTOR
PROF. VOLKAN CEVHER

HEAD TAS
FABIAN LATORRE, ALI KAVIS

HOMEWORK 1

In this homework, we consider a binary classification task that we will model using logistic regression. Your goal, should you choose to accept it, will be to find a classifier using first-order methods and accelerated gradient descent methods. You will also explore proximal gradient methods to obtain sparse solutions. The first part will consist of more theoretical questions, and the second one will require you to implement these methods. In the last part, you will use these proximal methods for image reconstruction where you get to fill in the blanks.

1 Logistic Regression - 10 Points

Logistic regression is a classic approach to *binary classification*. Before we dive in, let us first define the standard logistic function σ on which most of what follows is built:

$$\sigma : x \mapsto \frac{1}{1 + \exp(-x)}.$$

In logistic regression, we model the *conditional probability* of observing a class label b given a set of features $\mathbf{a}$. More formally, if we observe n independent samples

$$\{(\mathbf{a}_i, b_i)\}_{i=1}^n,$$

where $\mathbf{a}_i \in \mathbb{R}^p$ and $b_i \in \{-1, +1\}$ is the class label, we *assume* that b_i given $\mathbf{a}_i$ is a symmetric Bernouilli random variable with parameter $\sigma(\mathbf{a}_i^T \mathbf{x}^\natural)$, for some unknown $\mathbf{x}^\natural \in \mathbb{R}^p$. In other words, we assume that there exists an $\mathbf{x}^\natural \in \mathbb{R}^p$ such that

$$\mathbb{P}(b_i = 1 \mid \mathbf{a}_i) = \sigma(\mathbf{a}_i^T \mathbf{x}^\natural) \quad \text{and} \quad \mathbb{P}(b_i = -1 \mid \mathbf{a}_i) = 1 - \sigma(\mathbf{a}_i^T \mathbf{x}^\natural) = \sigma(-\mathbf{a}_i^T \mathbf{x}^\natural).$$

This is our statistical model and it can be written in a more compact form as follows,

$$\mathbb{P}(b_i = j \mid \mathbf{a}_i) = \sigma(j \cdot \mathbf{a}_i^T \mathbf{x}^\natural), \quad j \in \{+1, -1\}.$$

Our goal now is to determine the unknown $\mathbf{x}^\natural$ by constructing an estimator.

We are provided with a set of n independent observations, we can write down the negative log-likelihood f as follows :

$$\begin{aligned} f(\mathbf{x}) = -\log(\mathbb{P}(b_1, \ldots, b_n | a_1, \ldots, a_n)) &= -\log\left(\prod_{i=1}^n \mathbb{P}(b_i | a_i)\right) \qquad \text{(by independence)} \\ &= \sum_{i=1}^n -\log(\sigma(b_i \cdot \mathbf{a}_i^T \mathbf{x})) \\ &= \sum_{i=1}^n \log(1 + \exp(-b_i \cdot \mathbf{a}_i^T \mathbf{x})). \end{aligned}$$

(a) (1 point) Show that the function $u \mapsto \log(1 + \exp(-u))$ is convex. Deduce that f is convex.

You have just established that the negative log-likelihood is a convex function. So in principle, any local minimum of the maximum likelihood estimator, which is defined as

$$\mathbf{x}_{ML}^\star = \arg\min_{\mathbf{x} \in \mathcal{R}^p} f(x),$$

is a global minimum. But, does the minimum always exist? Please explain your reasoning.

(b) (1 point) Explain the difference between infima and minima. Give an example of a convex function that does not attain its infimum.

(c) (3 points) Show that if there exists $\mathbf{x}_0$ such that

$$\forall i \in \{1, \dots, n\}, \qquad b_i \mathbf{a}_i^T \mathbf{x}_0 > 0,$$

then f does not attain its infimum. This is called *complete separation* in the literature. Can you think of a geometric reason why this name is appropriate? Draw an simple 2D example where this can happen and then move on to the proof.

Hint: How does $f(2\mathbf{x}_0)$ compare with $f(\mathbf{x}_0)$, how about $f(\alpha \mathbf{x}_0)$ for $\alpha \to +\infty$?

Convex functions always have an infimum but do not always attain it. So it is possible for the maximum-likelihood estimator $\mathbf{x}_{ML}^\star$ to not exist. We will resolve this issue by adding a regularizer. (**End of Hint**).

In what follows, we consider the function

$$f_\mu(\mathbf{x}) = f(\mathbf{x}) + \frac{\mu}{2}\|\mathbf{x}\|_2^2$$

with $\mu > 0$.

(d) (1 point) Show that the gradient of f_μ can be expressed as

$$\nabla f_\mu(\mathbf{x}) = \sum_{i=1}^{n} -b_i \sigma(-b_i \cdot \mathbf{a}_i^T \mathbf{x}) \mathbf{a}_i + \mu \mathbf{x}. \tag{1}$$

Hint: Recitation 2 demonstrates examples on how to proceed with this question.

(e) (1 point) Show that the Hessian of f_μ can be expressed as

$$\nabla^2 f_\mu(\mathbf{x}) = \sum_{i=1}^{n} \sigma(-b_i \cdot \mathbf{a}_i^T \mathbf{x})(1 - \sigma(-b_i \cdot \mathbf{a}_i^T \mathbf{x})) \mathbf{a}_i \mathbf{a}_i^T + \mu \boldsymbol{I}. \tag{2}$$

For future use, it is convenient to observe that we can write the Hessian in a more compact form by defining the matrix

$$\mathbf{A} = \begin{bmatrix} \leftarrow & \mathbf{a}_1^T & \rightarrow \\ \leftarrow & \mathbf{a}_2^T & \rightarrow \\ & \cdots & \\ \leftarrow & \mathbf{a}_n^T & \rightarrow \end{bmatrix}.$$

It is easy to see that we have

$$\nabla^2 f_\mu(\mathbf{x}) = \mathbf{A}^T \mathrm{Diag}\left(\sigma(-b_i \cdot \mathbf{a}_i^T \mathbf{x})(1 - \sigma(-b_i \cdot \mathbf{a}_i^T \mathbf{x}))\right) \mathbf{A} + \mu \boldsymbol{I}.$$

Note that you do *not* have to show the correctness of this matrix form for this exercise.

(f) (2 points) Show that f_μ is μ-strongly convex. Is it possible for the minimum to be not attained in this case?[1] Justify your reasoning.

(g) (1 point) Show that f_μ is L-smooth, i.e ∇f_μ is L-Lipschitz with respect to the Euclidean norm, with

$$L = \|A\|_F^2 + \mu, \text{ where } \|\cdot\|_F \text{ denotes the Frobenius norm.}$$

Hint: Decompose the proof into the following three steps:

(1) Show that $\lambda_{max}(\mathbf{a}_i \mathbf{a}_i^T) = \|\mathbf{a}_i\|_2^2$, where $\lambda_{max}(\cdot)$ denotes the largest eigenvalue.
(2) Using (2), show that $\lambda_{max}(\nabla^2 f_\mu(\mathbf{x})) \leq \sum_{i=1}^n \|\mathbf{a}_i\|_2^2 + \mu$.
(3) Given above, explain how f_μ is L-smooth for $L = \|A\|_F^2 + \mu$.

Our estimator for $\mathbf{x}^\natural$ is the solution of the smooth strongly convex problem,

$$\mathbf{x}^\star = \arg\min_{\mathbf{x} \in \mathbb{R}^p} f(\mathbf{x}) + \frac{\mu}{2}\|\mathbf{x}\|_2^2. \tag{3}$$

[1] TA's will give you candy or virtual high-fives, if you provide a complete proof.

2 Numerical methods for Logistic Regression - 75 points

Associated code folder: `exercise2/`

Our aim in the remainder of this homework is to implement different optimization algorithms for solving the regularized Logistic Regression problem (3). We will use the breast-cancer dataset from the UCI Machine Learning Repository [2] consisting of $n = 546$ samples, each with $p = 10$ features. In all experiments, unless otherwise stated, use $\mu = 0.1$. Some Python skeleton code is provided which we highly recommend you use as it helps with uniform grading across the class. Here is a description of the files:

Python	Description
`log_reg.commons.compute_error`	compute errors
`log_reg.commons.Oracles`	return function values, gradients, stochastic gradients
`log_reg.algorithms.GD`	Gradient descent
`log_reg.algorithms.GDstr`	Gradient descent (strongly convex)
`log_reg.algorithms.AGD`	Accelerated gradient descent
`log_reg.algorithms.AGDstr`	Accelerated gradient descent (strongly convex)
`log_reg.algorithms.AGDR`	Accelerated gradient descent with restart
`log_reg.algorithms.LSGD`*	Gradient descent with line search
`log_reg.algorithms.LSAGD`*	Accelerated gradient descent with line search
`log_reg.algorithms.LSAGDR`*	Accelerated gradient descent (line search + restart)
`log_reg.algorithms.AdaGrad`	Adaptive Gradient Method
`log_reg.algorithms.ADAM`*	Adaptive moment estimation algorithm
`log_reg.algorithms.SGD`	Stochastic gradient descent
`log_reg.algorithms.SAG`	Stochastic averaging gradient
`log_reg.algorithms.SVR`	Stochastic gradient descent (variance reduction)
`log_reg.algorithms.SubG`	Subgradient method
`log_reg.algorithms.ista`	Iterative shrinkage-thresholding algorithm
`log_reg.algorithms.fista`	Fast iterative shrinkage-thresholding algorithm (restart)
`log_reg.algorithms.prox_sg`	Stochastic proximal gradient method
`log_reg.py`	Main code

* Algorithms marked with stars are optional.

General guidelines:

- You are required to complete the missing parts in the codes, indicated by markers *YOUR CODE GOES HERE* or *???*.
- Include the resulting figures and record the resulting ratios of incorrectly classified points in the test data set in your report.
- We would like you to us the skeleton code provided.

2.1 First-order methods - 25 Points

The optimality condition of (3) is

$$\nabla f_\mu(\mathbf{x}^\star) = \sum_{i=1}^{n} -b_i \sigma(-b_i \cdot \mathbf{a}_i^T \mathbf{x}^\star)\mathbf{a}_i + \mu \mathbf{x}^\star = 0. \tag{4}$$

Condition (4) is in fact the necessary and sufficient condition for $\mathbf{x}^\star$ to be optimal for (3). We can equivalently write this condition as

$$\mathbf{x}^\star = \mathbf{x}^\star - \mathbf{B}\nabla f_\mu(\mathbf{x}^\star) \tag{5}$$

for any symmetric, positive definite matrix $\mathbf{B}$. This is a fixed-point formulation of (3), which will be used to develop first-order methods.

Notice that choosing $\mathbf{B} = \alpha\mathbb{I}$ with $\alpha > 0$ in the formulation (5) suggests using the gradient descent method to solve (3). In this problem, you will implement different variants of the gradient descent algorithm. We have defined 3 input arguments (see the documentations in `commons.Oracles` to know how to use them in your codes):

1. **fx**: The function that characterizes the objective to be minimized;
2. **gradf**: The function that characterizes the gradient of the objective function **fx**;

3. **parameter**: A Python dictionary that includes the fields `maxit` (i.e. number of iterations), `x0` (i.e. initial estimate), `strcnvx` (i.e. strongly convex constant) and `Lips` (i.e. Lipschitz constant).

(a) (6 points) The main ingredients of the gradient descent algorithm are the descent direction (or search direction) $\mathbf{d}^k$ and step-size α_k. In this part, we consider the gradient descent algorithm with constant step-size $1/L$, i.e., $\alpha_k = 1/L$ for any $k = 0, 1, ...$ where L is the gradient-Lipschitz constant computed from Problem 1.(g).

Implement this algorithm by completing `algorithms.GD`.

Note that the objective function is strongly convex. Therefore, we can select the constant step-size α as $2/(L+\mu)$ to get a faster convergence rate. Implement this variant of the gradient descent method by completing the code in `algorithms.GDstr`.

(b) (7 points) We can accelerate the above gradient descent algorithm as follows ($t_0 = 1$):

$$\begin{cases} \mathbf{x}^{k+1} & := \mathbf{y}^k - \alpha_k \nabla f_\mu(\mathbf{y}^k), \\ t_{k+1} & := \frac{1}{2}(1 + \sqrt{1 + 4t_k^2}) \\ \mathbf{y}^{k+1} & := \mathbf{x}^{k+1} + \frac{t_k - 1}{t_{k+1}}(\mathbf{x}^{k+1} - \mathbf{x}^k). \end{cases}$$

Details of this accelerated gradient algorithm can be found in Lecture 4.

Implement this algorithm with constant step-size $\alpha = 1/L$, by completing the missing parts in the function `AGD` in `algorithms.AGD`.

Note that the objective function is strongly convex. Therefore, we can use the accelerated gradient algorithm for strongly convex objectives to converge faster. This variant can be summarized as follows:

$$\begin{cases} \mathbf{x}^{k+1} & := \mathbf{y}^k - \alpha_k \nabla f_\mu(\mathbf{y}^k), \\ \mathbf{y}^{k+1} & := \mathbf{x}^{k+1} + \frac{\sqrt{L} - \sqrt{\mu}}{\sqrt{L} + \sqrt{\mu}}(\mathbf{x}^{k+1} - \mathbf{x}^k). \end{cases}$$

Implement this variant of the accelerated gradient method by completing the code in `algorithms.AGDstr`.

(c) (Self study, 0 points) We can obtain better performance by considering a line search procedure, which adapts the step-size α_k to the local geometry. The line-search strategy to determine the step-size α_k for the standard GD algorithm,

$$\mathbf{x}^{k+1} = \mathbf{x}^k - \alpha_k \nabla f_\mu(\mathbf{x}^k),$$

is the following: At the step *kth* iteration, let $\mathbf{x}^k$ be the current iterate and $\mathbf{d}^k = -\nabla f_\mu(\mathbf{x}^k)$ be a given descent direction, and perform:

- Set $L_0 = L$.
- At each iteration, set $L_{k,0} = \frac{1}{2} L_{k-1}$, where k is the iteration counter.
- Using a for loop, find the minimum integer $i \geq 0$ that satisfies $f_\mu\left(\mathbf{x}^k + \frac{1}{2^i L_{k,0}} \mathbf{d}^k\right) \leq f_\mu(\mathbf{x}^k) - \frac{1}{2^{i+1} L_{k,0}} \|\mathbf{d}^k\|^2$.
- Set $L_k = 2^i L_{k,0}$ and use the step-size $\alpha_k := \frac{1}{L_k}$ (i.e., use the new estimate that you have used in the line-search: $\mathbf{x}^k + \frac{1}{2^i L_{k,0}} \mathbf{d}^k$).

Complete the missing parts in the file `algorithms.LSGD` in order to implement gradient descent with line-search.

We now incorporate a line-search enhancement to accelerated gradient method in (b) as follow: at step k, one has the current iteration $\mathbf{x}^k$ together with an *intermediate variable* $\mathbf{y}^k$ and its corresponding direction $\mathbf{d}^k = -\nabla f_\mu(\mathbf{y}^k)$. **Note that the intermediate variable is then used in the gradient step and hence the line-search will be performed on it to determine the step-size**.

- Perform a line-search strategy as above with respect to $\mathbf{y}^k$ and direction $\mathbf{d}^k$ to determine the step-size α_k as follows:
  - Set $L_0 = L$.
  - At each iteration, set $L_{k,0} = \frac{1}{2} L_{k-1}$, where k is the iteration counter.
  - Using a for loop, find the minimum integer $i \geq 0$ that satisfies $f_\mu\left(\mathbf{y}^k + \frac{1}{2^i L_{k,0}} \mathbf{d}^k\right) \leq f_\mu(\mathbf{y}^k) - \frac{1}{2^{i+1} L_{k,0}} \|\mathbf{d}^k\|^2$.
  - Set $L_k = 2^i L_{k,0}$ and use the step-size $\alpha_k := \frac{1}{L_k}$.

- Update the next iterations:

$$\begin{cases} \mathbf{x}^{k+1} & := \mathbf{y}^k - \alpha_k \nabla f_\mu(\mathbf{y}^k), \\ t_{k+1} & := \frac{1}{2}\left(1 + \sqrt{1 + 4\frac{L_k}{L_{k-1}} t_k^2}\right) \\ \mathbf{y}^{k+1} & := \mathbf{x}^{k+1} + \frac{t_k - 1}{t_{k+1}}(\mathbf{x}^{k+1} - \mathbf{x}^k). \end{cases}$$

Complete the missing parts in the file `algorithms.LSAGD` in order to implement accelerated gradient descent with line-search.

(d) (6 points) The accelerated gradient method is non-monotonic, so it can be oscillatory, i.e. $f_\mu(\mathbf{x}^{k+1}) \nleq f_\mu(\mathbf{x}^k)$ for all $k \geq 0$. To prevent such behavior, we can use the so-called adaptive restart strategy. Briefly, one such strategy can be explained as follows: At each iteration, whenever $\mathbf{x}^{k+1}$ is computed, we evaluate $f_\mu(\mathbf{x}^{k+1})$ and compare it with $f_\mu(\mathbf{x}^k)$:

- If $f_\mu(\mathbf{x}^k) < f_\mu(\mathbf{x}^{k+1})$, restart the iteration, i.e., recompute $\mathbf{x}^{k+1}$ by setting $\mathbf{y}^k := \mathbf{x}^k$ and $t_k := 1$;
- Otherwise, let the algorithm continue.

This strategy requires the evaluation of the function value at each iteration, which increases the computational complexity of the overall algorithm.

Implement the adaptive restart strategy which uses the function values for the accelerated gradient algorithm with constant step-size $\alpha_k = 1/L$ by completing the function `AGDR` in `algorithms.AGDR`.

(e) (Self study, 0 points)w

Incorporate the line-search, acceleration and function values restart by completing the function `LSAGDR` in `algorithms.LSAGDR`.

Hint: Note that while the restart is executed on $\mathbf{x}^k$, the line-search strategy is executed on $\mathbf{y}^k$ and the direction $\mathbf{d}^k = -\nabla f(\mathbf{y}^k)$ to determine the step-size and hence, use line-search each time you encounter an intermediate variable $\mathbf{y}^k$.

(f) (6 points) We can also apply an optimization technique that does not exploit the knowledge of the Lipschitz constant, and instead adapts to the local geometry by making use of past gradient information. AdaGrad adapts the step-size using the inverse square ℓ_2-norm of past gradients. Starting with $Q_0 = 0$ it iterates as follows:

$$\begin{cases} Q_k & = Q_{k-1} + \|\nabla f_\mu(\mathbf{x}^k)\|^2 \\ \mathbf{H}_k & = (\sqrt{Q_k} + \delta) I \\ \mathbf{x}^{k+1} & = \mathbf{x}^k - \alpha \mathbf{H}_k^{-1} \nabla f_\mu(\mathbf{x}^k) \end{cases}$$

Complete the missing parts in the file `algorithms.AdaGrad` in order to implement the above adaptive gradient method using $\alpha = 1$, $\delta = 10^{-5}$.

(g) (Self study, 0 points) Another famous adaptive optimization method is called the ADAptive Moment estimation algorithm, also known as ADAM,

$$\begin{cases} \mathbf{g}_k & = \nabla f_\mu(\mathbf{x}^{k-1}) \\ \mathbf{m}_k & = \beta_1 \mathbf{m}_{k-1} + (1-\beta_1)\mathbf{g}_k \leftarrow \text{Momentum} \\ \mathbf{v}_k & = \beta_2 \mathbf{v}_{k-1} + (1-\beta_2)\mathbf{g}_k^2 \leftarrow \text{Adaptive term} \\ \hat{\mathbf{m}}_k & = \mathbf{m}_k / (1-\beta_1^k) \\ \hat{\mathbf{v}}_k & = \mathbf{v}_k / (1-\beta_2^k) \leftarrow \text{Scaling for removing bias} \\ \mathbf{H}_k & = \sqrt{\hat{\mathbf{v}}_k} + \epsilon \\ \mathbf{x}^{k+1} & = \mathbf{x}^k - \alpha \hat{\mathbf{m}}_k / \mathbf{H}_k \end{cases}$$

Note that all operations shown above, when applied to vectors, are applied element-wise. In particular, $\mathbf{g}_k^2$ is a vector of the same size as $\mathbf{g}_k$ where each element is squared.

Complete the missing parts in the file `algorithms.ADAM` in order to implement the above adaptive gradient method using $\alpha = 0.1, \beta_1 = 0.9, \beta_2 = 0.999$ and $\epsilon = 10^{-8}$.

It has been shown that ADAM can fail to converge to the global minimum of a convex problem [6]. The authors provided a variant of ADAM, called AMSgrad in order to fix this convergence issue. However, in practice it is not clear which method performs best. (You are not required to implement this method, but advised to have a look at it for personal interest).

2.2 Stochastic gradient methods - 20 Points

In this problem, you will implement three different versions of stochastic gradient descent to solve the logistic regression problem. We have defined 4 input arguments (see the documentations in `commons.Oracles` to know how to use them in your codes):

1. **fx**: The function that characterizes the objective to be minimized;
2. **gradf**: The function that characterizes the gradient of the objective function **fx**;
3. **gradfsto**: The function that characterizes the stochastic gradient of the objective function **fx**;
4. **parameter**: A Python dictionary that includes the fields `maxit` (i.e. number of iterations), `x0` (i.e. initial estimate), `strcnvx` (i.e. strongly convex constant), `Lmax` (i.e. see definition below) and `no0functions` (i.e. number of functions).

To apply the stochastic gradient descent, we recast (3) as follows,

$$f(\mathbf{x}) = \frac{1}{n}\sum_{i=1}^{n}\Big\{\underbrace{\log(1+\exp(-b_i\cdot\mathbf{a}_i^T\mathbf{x}))+\frac{\mu}{2}\|\mathbf{x}\|^2}_{f_i(\mathbf{x})}\Big\}, \tag{6}$$

where we for notational convenience suppress the dependency on μ. As we saw in Problem 1, we have that

$$\nabla f_i(\mathbf{x}) = -b_i\sigma(-b_i\cdot\mathbf{a}_i^T\mathbf{x})\mathbf{a}_i + \mu\mathbf{x}$$

Consider the following stochastic gradient update: At the iteration k, pick $i_k \in \{1,\dots,n\}$ uniformly at random and define

$$\mathbf{x}^{k+1} := \mathbf{x}^k - \alpha_k\nabla f_{i_k}(\mathbf{x}^k). \tag{SGD}$$

(a) (2 points) Show that $\nabla f_{i_k}(\mathbf{x})$ is an unbiased estimation of $\nabla f(\mathbf{x})$. Explain why ∇f_{i_k} is Lipschitz continuous with $L(f_{i_k}) = \|\mathbf{a}_{i_k}\|^2 + \mu$.
Hint: Recall how we upper bounded L in Problem 1. Set $L_{\max} = \max_{i\in\{1,\dots,n\}} L(f_i)$.

(b) (4 points) We can use the standard stochastic gradient descent (SGD) to solve (6). Complete the following codes in `algorithms.SGD` using the following step-size rule $\alpha_k = \frac{1}{k}$.

(c) (6 points) Consider the following stochastic averaging gradient method (SAG) to solve (6):

$$\begin{cases}\text{pick } i_k \in \{1,\dots,n\} \text{ uniformly at random}\\ \mathbf{x}^{k+1} := \mathbf{x}^k - \frac{\alpha_k}{n}\sum_{i=1}^n \mathbf{v}_i^k,\end{cases}$$

where

$$\mathbf{v}_i^k = \begin{cases}\nabla f_i(\mathbf{x}^k) & \text{if } i = i_k,\\ \mathbf{v}_i^{k-1} & \text{otherwise.}\end{cases}$$

Complete the following codes in `algorithms.SAG` using the step-size $\alpha_k = \frac{1}{16L_{\max}}$ and $\mathbf{v}^0 = \mathbf{0}$.

(d) (8 points) We can converge faster with SGD by using the following version of SGD with variance reduction:

$$\begin{cases}\tilde{\mathbf{x}} = \mathbf{x}^k, \mathbf{v}^k = \nabla f(\tilde{\mathbf{x}}), \tilde{\mathbf{x}}^0 = \tilde{x}\\ \text{For } l = 0,\dots,q-1:\\ \quad \text{Pick } i_l \in \{1,\dots,n\} \text{ uniformly at random}\\ \quad \mathbf{v}^l = \nabla f_{i_l}(\tilde{\mathbf{x}}^l) - \nabla f_{i_l}(\tilde{\mathbf{x}}) + \mathbf{v}^k\\ \quad \tilde{\mathbf{x}}^{l+1} := \tilde{\mathbf{x}}^l - \gamma\mathbf{v}^l\\ \mathbf{x}^{k+1} = \frac{1}{q}\sum_{l=0}^{q-1}\tilde{\mathbf{x}}^{l+1}.\end{cases}$$

Complete the following codes in Python `algorithms.SVR` with the following rules: $\gamma = 0.01/L_{\max}$ and $q = [1000L_{\max}]$, i.e. q is the integer part of $1000L_{\max}$.

2.3 Proximal methods - 20 Points

In this problem, you will implement three different versions of proximal gradient methods to solve the logistic regression problem. We have defined 4 input arguments (see the documentations in `commons.Oracles` to know how to use them in your codes):

1. **fx** : The function that characterizes the unregularized objective to be minimized;
2. **gradf**: The function that characterizes the gradient of the unregularized objective function **fx**;
3. **gradfsto**: The function that characterizes the stochastic gradient of the unregularized objective function **fx**;
4. **gx**: The added regularizing function;
5. **proxg**: The proximal operator of **gx**;
6. **parameter**: A Python dictionary that includes the fields `maxit` (i.e. number of iterations), `x0` (i.e. initial estimate), `lambda` (i.e. regularization factor), `prox_Lips` (i.e. gradient Lipschitz constant), `stoch_rate_regime` (i.e. step-size for stochastic variant) and `no0functions` (i.e. number of functions).

We saw in Problem 1 that adding a squared ℓ_2-norm regularizer allowed us to guarantee the existence of a solution to problem (3). There are however other regularizers g that we could have added to the negative log-likelihood f in order to enforce some desired properties (like sparsity) on the solution of (3):

$$f_\mu(\mathbf{x}) = f(\mathbf{x}) + \mu g(\mathbf{x}),$$

a) (1 point) One possibility is to define $g : \mathbb{R}^d \to \mathbb{R},\ g(\mathbf{x}) := \|\mathbf{x}\|_1$. To solve this problem, a first approach can be to turn to subgradients. Recalling Lecture 4, fill in the method `SubG` in the file `algorithms.py`. Note that the constants G, R are given to you in the code.

Another, more efficient approach, as we saw in Lecture 5, is to minimize such a function by using proximal gradient algorithms, provided that g is 'proximable' (i.e., its proximal operator is efficient to evaluate). We recall the proximal operator of g as the solution to the following convex problem:

$$\text{prox}_g(\mathbf{z}) := \arg\min_{\mathbf{y}\in\mathbb{R}^d}\{g(\mathbf{y}) + \frac{1}{2}\|\mathbf{y}-\mathbf{z}\|_2^2\}.$$

In the file named `algorithms.py` you can find the incomplete methods, `ista`, `fista` and `prox_sg`, whose code you need to fill in.

b) (2 points) As in (a), given $g : \mathbb{R}^d \to \mathbb{R},\ g(\mathbf{x}) := \|\mathbf{x}\|_1$, show that its proximal function can be written as

$$\text{prox}_{\lambda g}(\mathbf{z}) = \max(|\mathbf{z}| - \lambda, 0) \circ \text{sign}(\mathbf{z}),\ \ \mathbf{z} \in \mathbb{R}^d \tag{7}$$

where the operators max, sign and $|\cdot|$ are applied coordinate-wise to the vector $\mathbf{z}$ and $\circ$ stands for $(\mathbf{x} \circ \mathbf{y})_i = x_i y_i$. Such a regularizer imposes sparsity on the solutions.

c) (2 point) In the file `operators.py`, fill in the method `l1_prox` with the proximal operator expressions derived previously.

d) (7 points) Using the information in Lecture 5 fill in the codes of methods `ista`, `fista`.

e) (3 points) Adapt `fista` to also handle restart and fill in `gradient_scheme_restart_condition` from the file `algorithms.py`. In the latter method, you need to implement the so-called 'gradient restart condition', described in depth in papers, such as [3, 5]. Formally, the criterion is given by,

$$\langle \mathbf{y}^k - \mathbf{x}^{k+1}, \mathbf{x}^{k+1} - \mathbf{x}^k \rangle > 0,$$

where t is the iteration index. The term $\mathbf{y}^k - \mathbf{x}^{k+1}$ can be seen as a gradient, while $\mathbf{x}^{k+1} - \mathbf{x}^k$ is the descent direction of the momentum term. Overall this criterion resets the momentum of FISTA to 0 when it goes in a bad descent direction.

f) (5 points) Using Lecture 5 as a guideline, fill in the method `prox_sg` from the file `algorithm.py`.

Note: For the `prox_sg` method you need to record convergence with respect to the ergodic iterate $\bar{\mathbf{x}}^k = \frac{\sum_{j=1}^{k} \gamma_j \mathbf{x}^j}{\sum_{j=1}^{k} \gamma_j}$, where γ_j is the learning rate at step j.

2.4 Convergence rates - 10 Points

In this problem, you will study the convergence of gradient descent as well as gradient descent with strong convexity. Recall that gradient descent is expected to converge in the following way under various assumptions.

Theorem 1. *Convergence rate of gradient descent (Lecture 2, slide 33)*
Let f be a twice-differentiable convex function, if

f is L-smooth,	$\alpha = \frac{1}{L}$:	$f(\mathbf{x}^k) - f(\mathbf{x}^\star)$	$\leq$	$\frac{2L}{k+4}$	$\|\mathbf{x}^0 - \mathbf{x}^\star\|_2^2$
f is L-smooth and μ-strongly convex,	$\alpha = \frac{2}{L+\mu}$:	$\|\mathbf{x}^k - \mathbf{x}^\star\|_2$	$\leq$	$\left(\frac{L-\mu}{L+\mu}\right)^k$	$\|\mathbf{x}^0 - \mathbf{x}^\star\|_2$
f is L-smooth and μ-strongly convex,	$\alpha = \frac{1}{L}$:	$\|\mathbf{x}^k - \mathbf{x}^\star\|_2$	$\leq$	$\left(\frac{L-\mu}{L+\mu}\right)^{\frac{k}{2}}$	$\|\mathbf{x}^0 - \mathbf{x}^\star\|_2$

In the file `exercise_2_4.py`, you can find the incomplete code that you need to fill.

a) (2 points) Fill the lines `theoretical_rate_gd` and `theoretical_rate_gd_str` with their corresponding theoretical convergence rate. Fill the blanks in the code to have the rates plotted as a straight line. Are the observed convergence rates consistent with the theoretical ones? Why?

Remark. Pay particular attention to the hypothesis on the function that holds in this problem.

b) (1 point) Are the observed convergence rates of GD and GDstr linear, sublinear or quadratic? Explain how you can come to this conclusion.

c) (5 points) We will try to estimate the rate of convergence of the GD and GDstr in the specific instance our problem. We assume that the observed rate will be of the form $\|\mathbf{x}^k - \mathbf{x}^\star\|_2 = a \cdot b^k$. Show that a and b can be estimated by doing a linear regression on the graph $(k, \log(\|\mathbf{x}^k - \mathbf{x}^\star\|_2)$. Fill the corresponding code.

Remark. To closely observe the algorithms reaching the theoretical convergence rates, it would require running them for $\sim 100'000$ iterations in this problem. However, you are only be asked to run them for 4000 iterations in consideration of your time. While you will see a trend of convergence, you will not observe the convergence at the asymptotic rate.

d) (2 points) Having obtained the linear regression coefficients $(a_{\text{GD}}, b_{\text{GD}})$ and $(a_{\text{GD\_str}}, b_{\text{GD\_str}})$, fill the blanks in the code to plot the result. Compare the empirical speed of convergence $a_{\text{GD(\_str)}}$ with the theoretical one.

Attach the linear regression coefficients, their transformed form as well as the resulting plot to your report.

3 Image reconstruction - 15 Points

Associated code folder: `exercise3/`

In this exercise an *image* should be understood as the matrix equivalent of a digital grayscale image, where an entry represents the intensity of the corresponding pixel.

3.1 Wavelets

A widely used multi-scale localized representation in signal and image processing is the wavelet transform.[2] This representation is constructed so that piecewise polynomial signals have sparse wavelet expansions [4]. Since many real-world signals can be composed by piecewise smooth parts, it follows that they have sparse or compressible wavelet expansions.

Scale. In wavelet analysis, we frequently refer to a particular scale of analysis for a signal. In particular, we consider dyadic scaling, such that the supports from one scale to the next are halved along each dimension. For images, which can be represented with matrices, we can imagine the highest scale as consisting of each pixel. At other scales, each region correspond to the union of four neighboring regions at the next higher scale, as illustrated in Figure 1.
The Wavelet transform. The wavelet transform offers a multi-scale decomposition of an image into coefficients related to different dyadic regions and orientations. In essence, each wavelet coefficient is given by the scalar product of the image with a wavelet function which is concentrated approximately on some dyadic square and has a given orientation (vertical, horizontal or diagonal). Wavelets are essentially bandpass functions that detect abrupt changes in a signal. The scale of a wavelet, which controls its support both in time and in frequency, also controls its sensitivity to changes in the signal.

[2] This section is an adaption from *Signal Dictionaries and Representations* by M. Watkin (see `http://bit.ly/1wXDDbG`).