EE-556: Mathematics of data: from theory to computation
Homework Exercise 1

Natalie Bolón Brun

10/2018

1 Geometric properties of the objective function

Given $\boldsymbol{x} \in \mathrm{R}^p$ denote by $\mathbb{I}_L$, $\mathbb{I}_Q$ and $\mathbb{I}_0$ nxn such that $\mathbb{I}_L(i,i) = 1$ if $b_i(\boldsymbol{a_i^T x} + \mu) < 1 - h$, $\mathbb{I}_Q(i,i) = 1$ if $\left|1 - b_i(\boldsymbol{a_i^T x} + \mu)\right| \leq h$, $\mathbb{I}_0(i,i) = 1$ if $b_i(\boldsymbol{a_i^T x} + \mu) > 1 - h$ and 0 otherwise. $\boldsymbol{A} := [\boldsymbol{a_1}, ..., \boldsymbol{a_n}]^T$ and $\boldsymbol{b} := [b_1, ..., b_n]^T$. For convenience, it has also been set $\widetilde{\boldsymbol{A}} := [b_1\boldsymbol{a_1}, ..., b_n\boldsymbol{a_n}]^T$

a) Do the necessary calculations to show that:

$$\nabla f(\boldsymbol{x}) = \lambda \boldsymbol{x} + \frac{1}{4nh}\widetilde{\boldsymbol{A}}^T \mathbb{I}_Q[\widetilde{\boldsymbol{A}}\boldsymbol{x} - (1+h)\boldsymbol{1}] - \frac{1}{2n}\widetilde{\boldsymbol{A}}^T \mathbb{I}_L \boldsymbol{1} \tag{1}$$

The function can be rewritten as $f(\boldsymbol{x}, 0) = \frac{1}{2n}\sum_{i=0}^{n} g_i(\boldsymbol{x}, 0) + \frac{\lambda}{2}\left\|\boldsymbol{x}\right\|^2$ as

$$f(\boldsymbol{x}, 0) = \frac{\lambda}{2}\left\|\boldsymbol{x}\right\|^2 + \mathbb{I}_0 \boldsymbol{0} + \frac{1}{4h}\mathbb{I}_Q[(1+h)\boldsymbol{1} - \widetilde{\boldsymbol{A}}\boldsymbol{x}]^2 + \mathbb{I}_L[\boldsymbol{1} - \widetilde{\boldsymbol{A}}\boldsymbol{x}] \tag{2}$$

Given the previous function, its gradient is given by:

$$\nabla f(\boldsymbol{x}) = \lambda \boldsymbol{x} + \frac{1}{2n}[-\frac{2}{4h}\widetilde{\boldsymbol{A}}^T \mathbb{I}_Q[(1+h)\boldsymbol{1} - \widetilde{\boldsymbol{A}}\boldsymbol{x}] - \widetilde{\boldsymbol{A}}^T \mathbb{I}_L \boldsymbol{1}] = \tag{3}$$

$$= \lambda \boldsymbol{x} + \frac{1}{4nh}\widetilde{\boldsymbol{A}}^T \mathbb{I}_Q[\widetilde{\boldsymbol{A}}\boldsymbol{x} - (1+h)\boldsymbol{1}] - \frac{1}{2n}\widetilde{\boldsymbol{A}}^T \mathbb{I}_L \boldsymbol{1} \tag{4}$$

a2) Explain that ∇f is L-Lipschitz continuous:

L - Lipschitz continuous

▨ f is L-Lipschitz continuous $\Leftrightarrow$ $\|\nabla f(x) - \nabla f(y)\| \leqslant L \| f(x) - f(y)\|$

▨ $\| \lambda x + \frac{1}{4nh}\tilde{A}^T [\tilde{A}x - (1+h)\mathbb{1}] - \lambda y - \frac{1}{4nh}\tilde{A}^T[\tilde{A}y - (1-h)\mathbb{1}] \| =$

$$= \| \lambda(x-y) + \frac{1}{4nh}\tilde{A}^T\tilde{A}(x-y)\| \underset{(1)}{\leqslant} \| \lambda + \frac{1}{4nh}\tilde{A}^T\tilde{A}\| \|x-y\| \leqslant$$

$$\underbrace{\left(\lambda + \frac{1}{4nh}\|\tilde{A}^T\|\|\tilde{A}\|\right)}_{L}\|x-y\|$$

$$L = \lambda + \frac{1}{4nh}\|\tilde{A}^T\|\|\tilde{A}\| \overset{(2)}{=} \lambda + \frac{1}{4nh}\|A^T\|\|A\|$$

(1) Triangle Inequality

(2) $\|\tilde{A}^T\| = \|A^T\|$

$\|\tilde{A}\| = \|A\|$

b) Explain why f is twice-differentiable in x. Show that:

$$\nabla^2 f(\boldsymbol{x}) = \lambda * \boldsymbol{I} + \frac{1}{4nh}\boldsymbol{A^T A} \tag{5}$$

- f twice differentiable:

▨ If we consider $\mathbb{I}_Q = \mathbb{I} \Rightarrow \mathbb{I}_L = \mathbb{I}_0 = \mathbf{0}$

▨ Given $\nabla f(x) = \lambda x + \frac{1}{4nh}\tilde{A}^T \mathbb{I}\left[\tilde{A}x - (1+h)\mathbb{1}\right]$ we want to show

$\exists\, \nabla^2 f(x) \quad \forall\, x \in \text{dom}(f)$.

$\nabla^2 f(x) = \lambda \mathbb{I} + \frac{1}{4nh}\tilde{A}^T\tilde{A}$ which is finite $\forall\, x \in \text{dom}(f)$.

▨ $\left[\nabla^2 f(x)\right]_{ij} = \frac{\partial f}{\partial x_i \partial x_j} = \lambda + \frac{1}{4nh}\left[\sum_{k=1}^{n} b_k^2 (a_{ki}\cdot a_{kj})\right] = \text{constant}$

▨ $\Rightarrow \left[\nabla^2 f(x)\right]_{ij}$
- ▨ constant $\forall x \Rightarrow$ continuous $\frac{\partial f}{\partial x_i \partial x_j}$ $\forall\, x \in \text{dom}(f)$
- ▨ constant $\forall x \Rightarrow$ finite value $\Rightarrow \exists\, \frac{\partial f}{\partial x_i \partial x_j}$ $\forall\, x \in \text{dom}(f)$

▨ finally $\left[\nabla^2 f(x)\right]_{ij} = \lambda + \frac{1}{4nh}\left[\sum_{k=1}^{n} b_k^2 (a_{ki}\cdot a_{kj})\right]$

results in $\nabla^2 f(x) = \lambda \mathbb{I} + \frac{1}{4nh}\tilde{A}^T\tilde{A}$

c) Show that f is λ-strongly convex:

- f λ- strongly convex.

▨ f is said to be λ- strongly convex if

$h(x) = f(x) - \frac{\lambda}{2}\|x\|_2^2$ is convex.

▨ $h(x) = \frac{\lambda}{2}\|x\|^2 + \frac{1}{2n}\left[\mathbb{I}_0 \cdot \mathbf{0} + \frac{1}{4h}\mathbb{I}_Q\left((1+h)\mathbb{1} - \tilde{A}x\right) + \mathbb{I}_L\left(\mathbb{1} - \tilde{A}x\right)\right] - \frac{\lambda}{2}\|x\|^2 =$

$= \frac{1}{8nh}\mathbb{I}_Q\left((1+h)\mathbb{1} - \tilde{A}x\right) + \frac{1}{2n}\mathbb{I}_L\left(\mathbb{1} - \tilde{A}x\right)$

▨ convexity of $h(x)$

$h(x)$ is convex on its domain if

$\nabla^2 h(x) \succeq 0 \quad \forall\, x \in \text{dom}(h)$

From the results obtained in exercise 3 we can say:

$$\nabla^2 h(x) = \begin{cases} \frac{1}{4nh}\tilde{A}^T\tilde{A} & \text{if } h \text{ is differentiable in } x \\ \mathbf{0} & \text{otherwise} \end{cases}$$

▨ As $\tilde{A}^T\tilde{A} \succeq 0 \longrightarrow \nabla^2 h(x) \succeq 0 \Rightarrow h$ is convex $\Rightarrow f$ is λ- strongly convex.

2 Stochastic Gradient Methods for SVM

a)Show that $\nabla f_{ik}(\boldsymbol{x})$ is an unbiased estimator of $\nabla f(\boldsymbol{x})$. Explain why ∇f_{ik} is Lipschitz continuous.

1.

$$\mathbb{E}[\nabla f_{ik}(x)] = \frac{1}{n}\sum_{i=1}^{n} \nabla f_{ik}(x) = \nabla f_k(x) \tag{6}$$

2.

– L- lipschitz continuous

$\nabla f_{ik}(x)$ is Lipschitz continuous iff

$$\| \nabla f_{ik}(x) - \nabla f_{ik}(y) \| \leq L \|x-y\|$$

$$\| \nabla f_{ik}(x) - \nabla f_{ik}(y) \| \overset{(0)}{=}$$

(0) For readibility, the subindex k has been omited as is common for all elements.

$$= \| \lambda x + 1_{1f} \frac{1}{4h} a_i (a_i^T x - b_i(1+h)) - 1_{1f} \cdot \frac{1}{2} b_i a_i - \lambda y - 1_{1f} \frac{1}{4h} a_i (a_i^T y - b_i(1+h)) + 1_{1f} \frac{1}{2} b_i a_i \| \overset{(1)}{=}$$

$$= \| \lambda (x-y) + \frac{1}{4h} a_i a_i^T (x-y) \| = \| (\lambda + \frac{1}{4h} a_i a_i^T)(x-y) \| \overset{(2)}{\leq}$$

$$\leq \| \lambda + \frac{1}{4h} a_i a_i^T \| \|x-y\| \leq \underbrace{(\lambda + \frac{1}{4h} \|a_i a_i^T\|)}_{L} \|x-y\|$$

$$L(f_{ik}) = \lambda + \frac{1}{4h} \|a_{ik} a_{ik}^T\| = \lambda + \frac{1}{4h} \|a_{ik}\|^2$$

(1) we consider x & y are to close points in the domain of f.
As a consequence either both belong to the quadratic region or to the linear one.
In this case we consider both belong to the quadratic region as $\nabla f_{ik}(x)$ is constant in the linear part $(L=0)$.

(2) Triangle inequality.

(0): For readibility the subindex k has been omited as it is common for all elements.

(1): Given x and y two close points in the domain of f, it can be considered that either both belong to the quadratic region or to the linear one. In this case, it has been considered both belong to the quadratic region as $\nabla f_{ik}(x)$ is constant in the linear part $(L = 0)$.

(2): Triangle inequality.

3 Simulations Results

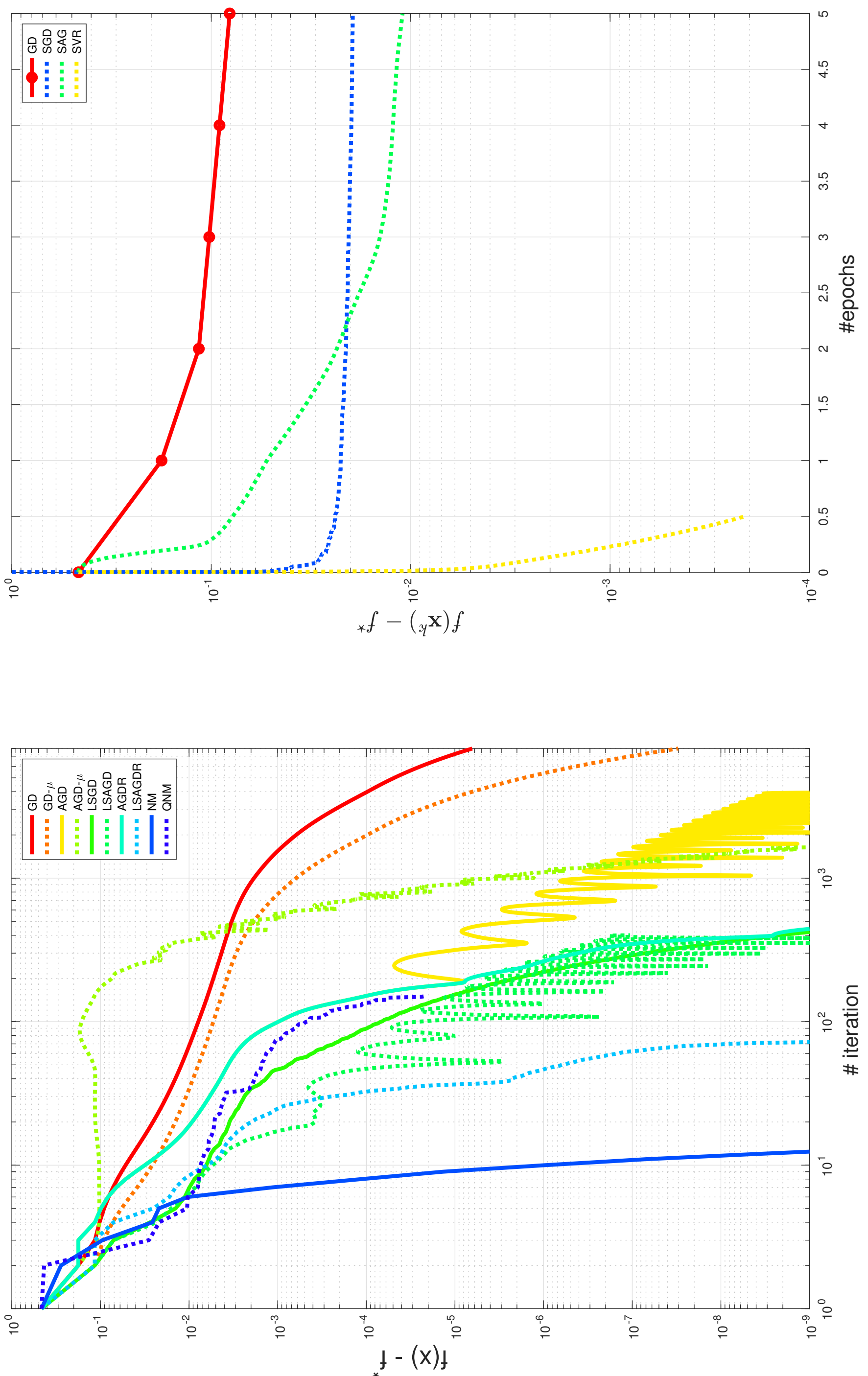

Figure 1: Result of the implementation of the different numerical methods for linear support vector machine implemented in Matlab 2018b.

4 Comment on Simulations

The curves shown in Figure 1 show the evolution of the error vs. the number of iteration for different numerical methods implemented to solve a linear support vector machine problem.

For the different methods:

1. Gradient Descent:
   Implemented with constant step size of $\frac{1}{L}$ where L represents the Lipchitz constant.
   After 8000 iterations the percentage of misclassified points is 2.92%

2. Gradient Descent (Strong convexity):
   Implemented with constant step size of $\frac{2}{L+\lambda}$ where L represents the Lipchitz constant and λ represents the strong convexity parameter of the given function
   After 8000 iterations the percentage of misclassified points is 2.92%

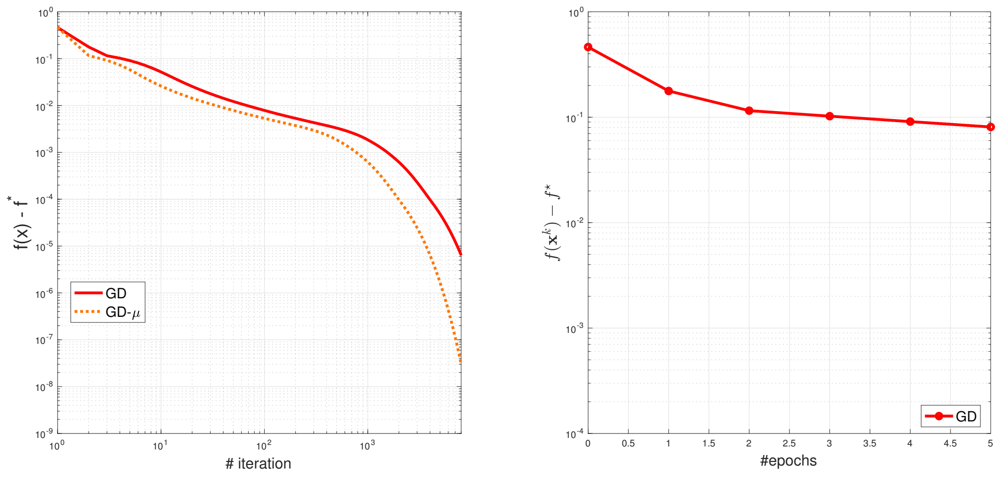

Figure 2: Result of the implementation of Gradient Descent with step size $\alpha_k = \frac{1}{L}$ and Gradient Descent with step size $\alpha_k = \frac{2}{L+\lambda}$. Implementation in Matlab 2018b.

   Comparing these two methods, it can be seen that by introducing a more aggressive step size taking into advantage the strong convexity property of the function, smaller error can be achieved in the same number of iterations.

3. Accelerated Gradient Descent:
   Implemented with constant step size of $\frac{1}{L}$ where L represents the Lipchitz constant.
   After 4000 iterations the percentage of misclassified points is 2.92%

4. Accelerated Gradient Descent (strong convexity):
   Implemented with constant step size of $\frac{2}{L+\lambda}$ where L represents the Lipchitz constant and λ represents the strong convexity parameter of the given function
   After 2000 iterations the percentage of misclassified points is 2.92%

   The Accelerated Gradient methods introduce a momentum term. This method takes into account not only local information about the gradient but also information about its evolution thanks to the introduction of the auxiliary sequence y_k. This results in the acceleration of the convergence of the method.
   Nevertheless, the introduction of the momentum allows oscillations. Opposite to GD, now the sequence towards the optimal value can follow non-decreasing directions. This oscillations are reduced if the step size is chosen taking into account the strong convexity property of the function.

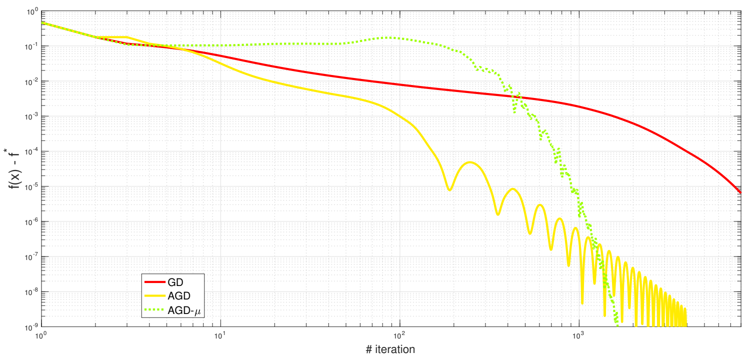

Figure 3: Result of the implementation of Accelerated Gradient Descent with step size $\alpha_k = \frac{1}{L}$ and Accelerated Gradient Descent with step size $\alpha_k = \frac{2}{L+\lambda}$. Implementation in Matlab 2018b.

In Figure 3, it is shown how both accelerated implementations achieve lower error in half the number of iterations than the first implementation of Gradient Descent. It also reflects the increase of the oscillations in AGD with step size chosen to be $\frac{1}{L}$ as the objective approaches the optimal value. These oscillations are reduced when taking into account the strong convexity.
AGD has a more progressive and constant (although the oscillations) approach towards the optimal value while AGD_μ needs a higher number of iterations to initiate the descent towards the optimal value but performs its descent more aggressively and less noisy.

5. Line Search Gradient Descent: This methods introduces a variable step size taking into account at each step the local geometry of the function. The step size is chosen to be $\frac{1}{L}$ where L represents the estimated Lipschitz constant at each iteration
After 450 iterations the percentage of misclassified points is 2.92%

6. Line Search Accelerated Gradient Descent: This method combines line search for the estimation of the step size at each iteration and acceleration through the introduction of a momentum term.
After 400 iterations the percentage of misclassified points is 2.92%

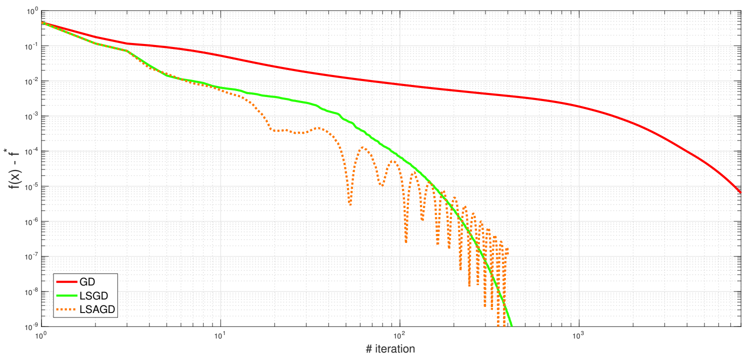

Figure 4: Result of the implementation of Line Search Gradient Descent and Line Search Accelerated Gradient Descent. Implementation in Matlab 2018b.

Line Search methods adapt the step size to the local geometry, reducing the number of iterations to achieve higher accuracy in one order of magnitude.
LSGD decreases monotonically towards the optimal value while LSAGD introduces oscillations in the sequence due to the introduction of the momentum. Results can be seen in Figure 4

7. Accelerated Gradient Descent with Adaptive Restart:
   AGD implemented in such a way that the gradient is restarted when the computed following step will not lead towards an improvement in the error function.
   After 500 iterations the percentage of misclassified points is 2.92%

8. Line Search Accelerated Gradient Descent with Adaptive Restart:
   AGD implemented in such a way that the gradient is restarted when the computed following step will not lead towards an improvement in the error function.
   After 100 iterations the percentage of misclassified points is 2.92%

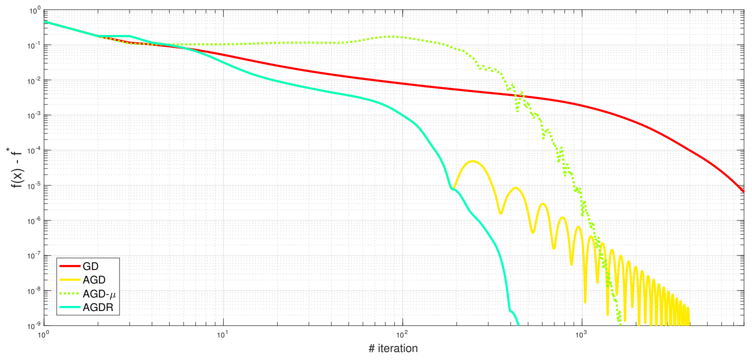

Figure 5: Result of the implementation of Accelerated Gradient Descent with step size $\alpha_k = \frac{1}{L}$ and Accelerated Gradient Descent with step size $\alpha_k = \frac{2}{L+\lambda}$ and Accelerated Gradient Descent with Restart and step size $\alpha_k = \frac{1}{L}$.
Implementation in Matlab 2018b.

Shown in Figure 5 it can be seen that the behaviour of AGD and AGDR is the same until the appearance of oscillations when the restart strategy allows its suppression. In terms of iterations, the AGDR can converge faster as it avoids taking steps in directions that do not lead towards an improvement of the objective function.

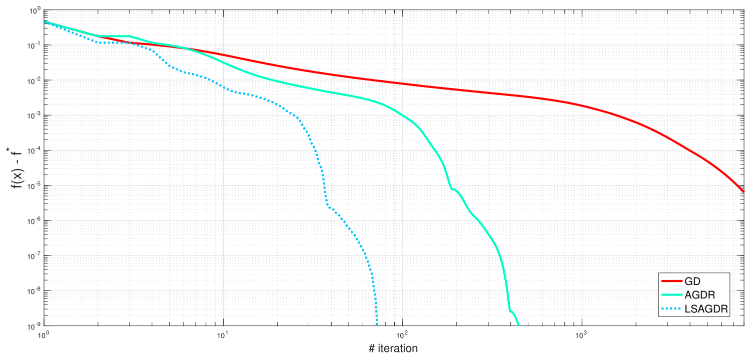

Figure 6: Result of the implementation of Accelerated Gradient Descent with Restart and step size $\alpha_k = \frac{1}{L}$ and Line Search Accelerated Gradient Descent with adaptive step size.
Implementation in Matlab 2018b.

Shown in Figure 6 it can be seen that the introduction of line search allows to reduce the number of iterations to achieve the same error by 4. Within the **First Order Methods** that have been shown up to now, Line Search Accelerated Gradient Descent is the one requiring less number of iterations to achieve an ϵ-optimal value with $\epsilon = 10^{-9}$

9. Newton Method:
   Newton method with adaptative step size computed via line search. The inverse of the Hessian is computed using pcg built in function of Matlab.
   After 20 iterations the percentage of misclassified points is 2.92%

10. Quasi-newton Method:
    Quasi-newton method with adaptative step size computed via line search. The inverse of the Hessian is approximated by a positive definite matrix computed via the BFGS update.
    After 150 iterations the percentage of misclassified points is 2.92%

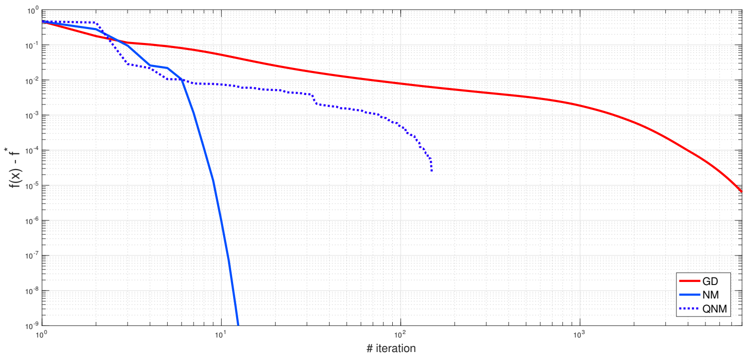

Figure 7: Result of the Newton Method and Quasi-newton Method with adaptive step size computed via line search.
Implementation in Matlab 2018b.

Results in 7 show how Second Order Methods allow the reduction of the number of iterations required to achieve an ϵ-optimal solution given a fixed ϵ by introducing information about the evolution of the gradient through the iterations through the Hessian. Nevertheless, the computational cost of each iteration is much higher than in First Order Methods.

11. Stochastic Gradient Descent:
    SGD with decreasing step size $\alpha_k = \frac{1}{k}$.
    After 5 epochs the percentage of misclassified points is 3.65%

12. Stochastic Averaging Gradient Descent:
    SGD with step size $\alpha_k = \frac{1}{16 L_{max}}$. The direction of descent is given by an average of stochastic gradients.
    After 5 epochs the percentage of misclassified points is 1.46%

13. Stochastic Gradient Descent with Variance Reduction:
    SGD with step size $\gamma = \frac{0.01}{L_{max}}$. Reduction of the variance in achieved through the update component-wise of the full gradient with stochastic gradient values
    After 0.5 epochs the percentage of misclassified points is 2.92%

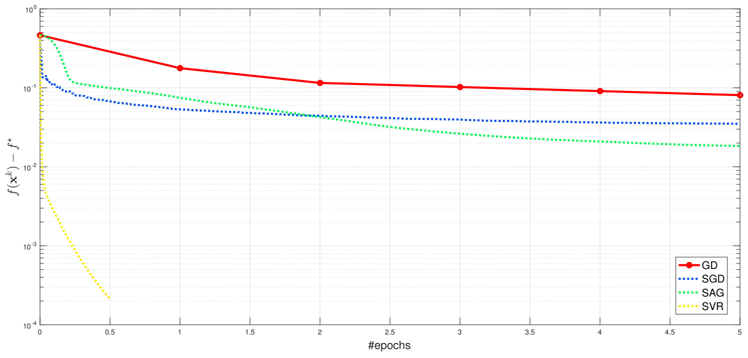

Figure 8: Result of Stochastic Gradient Descent, Stochastic Averaging GD and SGD with Variance Reduction.
Implementation in Matlab 2018b.

Results in 8 show the performance of stochastic methods vs. non stochastic GD. For SGD and SAGD, the error achieved given the same number of epochs is lower than the one achieved by GD. In the case of SVR, the improvement is much higher as it does not even require the completion of an epoch to obtain an error two orders of magnitude lower.