LASSO Optimisation Project

Arkadiy Aliev, Vsevolod Ivanov

17.12.2024

1 Problem statement

The LASSO problem in its standard (primal) form is given as:

$$\min_{x \in \mathbb{R}^d} f(x) = \min_{x \in \mathbb{R}^d} \|Ax - b\|^2 \quad \text{subject to} \quad \|x\|_1 \leq 1$$

where $A \in \mathbb{R}^{n \times d}, b \in \mathbb{R}^d$.

Here we observe, that the constraint set $X = \{x \in \mathbb{R}^d : \|x\|_1 \leq 1\}$ is the unit l_1-ball, the convex hull of the unit basis vectors: $X = conv(\{\pm e_1, \dots, \pm e_d\})$

As a solution to this problem, we will consider two methods: Nesterov Accelerated Gradient and the Frank-Wolfe algorithm. We will describe each method, examine their theoretical convergence rates, and compare their execution times for two types of matrices A, b, x - dense and sparse. In the end, we will summarize the comparison, highlighting which method proved to be more effective for which tasks and why.

2 Nesterov Gradient Method

Nesterov method for LASSO problem works as follows:

1. Start at random point $x^{(0)}$, initialize $y^{(0)} = x^{(0)}$
2. Set $n = 100$, $\beta^{(k)} = \frac{\sqrt{L}-\sqrt{\mu}}{\sqrt{L}+\sqrt{\mu}}$ for strongly convex or $\frac{k}{k+3}$ for convex + L-smooth case
3. Repeat steps $4-5$ n times:
4. set $x^{(k+1)} = \Pi_{\|\cdot\|_1 \leq 1}(y^{(k)} - \frac{1}{L}\nabla f(y^{(k)}))$
5. set $y^{(k+1)} = x_k + \beta^{(k)}(x^{(k+1)} - x^{(k)})$
6. After n steps, return $x^{(n)}$

2.1 Convergence rate

For convergence rate explanation, we need to do some reformulations. First of all, our problem can be reformulated in the following way:

$$\min_{x\in\mathbb{R}^d, \|x\|_1\leq 1} f(x) = \min_{x\in\mathbb{R}^d,} (g(x) + h(x))$$

Where $g(x) = \|Ax - b\|$ - convex and differentiable, $h(x) = I_{\|x\|\leq 1}$ - convex function.

So, one step of algorithm now looks as follows:

$$y = x^{(k-1)} + \frac{k-2}{k-1}(x^{(k-1)} - x^{(k-2)})$$

$$x^{(k)} = prox_{t_k}(y - t_k\nabla g(y))$$

Where the prox function is defined as

$$prox_t(x) = \arg\min_{z\in R^n} \frac{1}{2t}\|x - z\|^2 + h(z)$$

Theorem: The Accelerated Generalized Gradient Method with fixed step size $t \leq \frac{1}{L}$ satisfies:

$$f(x^{(k)}) - f(x^\star) \leq \frac{2\|x^{(0)} - x^\star\|^2}{t(k+1)^2}$$

Achieves the optimal $O\left(\frac{1}{k^2}\right)$ rate for first-order methods!

To achieve $f(x^{(k)}) - f(x^\star) \leq \epsilon$, need $O\left(\frac{1}{\sqrt{\epsilon}}\right)$ iterations. We will use:

$$1 - \frac{\theta_k}{\theta_k^2} \leq \frac{1}{\theta_{k-1}^2}, \quad k = 1, 2, 3, \ldots$$

We will also use:

$$h(v) \leq h(z) + \frac{1}{t}(v - w)^T(z - v), \quad \forall z, w, v = \text{prox}_t(w)$$

Why is this true?

By definition of the prox operator, v minimizes:

$$\frac{1}{2t}\|w - v\|^2 + h(v) \iff 0 \in \frac{1}{t}(v - w) + \partial h(v)$$

This implies:

$$-\frac{1}{t}(v - w) \in \partial h(v)$$

Now apply the definition of the subgradient.

Proof of Theorem 1 We focus first on one iteration and drop k notation (so x^+, u^+ are updated versions of x, u). Since g is Lipschitz with constant $L > 0$ and $t \leq \frac{1}{L}$:

$$g(x^+) \leq g(y) + \nabla g(y)^T(x^+ - y) + \frac{1}{2t}\|x^+ - y\|^2$$

From our bound using the prox operator:

$$h(x^+) \leq h(z) + \frac{1}{t}(x^+ - y)^T(z - x^+) + \nabla g(y)^T(z - x^+), \quad \forall z$$

Adding these together and using convexity of g:

$$f(x^+) \leq f(z) + \frac{1}{t}(x^+ - y)^T(z - x^+) + \frac{1}{2t}\|x^+ - y\|^2, \quad \forall z$$

Using this bound at $z = x$ and $z = x^\star$:

$$f(x^+) - f(x^\star) - (1-\theta)(f(x) - f(x^\star)) \leq \frac{1}{t}(x^+ - y)^T(\theta x^\star + (1-\theta)x - x^+) + \frac{1}{2t}\|x^+ - y\|^2 =$$

$$= \theta^2 \frac{1}{2t}(\|u - x^\star\|^2 - \|u^+ - x^\star\|^2)$$

At iteration k:

$$t\theta_k^2(f(x^{(k)}) - f(x^\star)) + \frac{1}{2}\|u^{(k)} - x^\star\|^2 \leq (1-\theta_k)t\theta_k^2(f(x^{(k-1)}) - f(x^\star)) + \frac{1}{2}\|u^{(k-1)} - x^\star\|^2 \leq \frac{1}{2}\|x^{(0)} - x^\star\|^2$$

Therefore,

$$f(x^{(k)}) - f(x^\star) \leq \frac{\theta_k^2}{2t}\|x^{(0)} - x^\star\|^2 = \frac{2}{t(k+1)^2}\|x^{(0)} - x^\star\|^2$$

This completes the proof.

3 Frank-Wolfe algorithm

The Frank-Wolfe algorithm for LASSO problem works as follows:

1. Start at random point $x^{(0)}$
2. Set $n = 100$, $\gamma_k = \frac{2}{k+1}$
3. Repeat steps $4 - 5$ n times:
4. set $s^{(k+1)} = argmin_{z \in L_1} \langle \nabla f(x^{(k)}), z \rangle$
5. set $x^{(k+1)} = \gamma_k s + (1 - \gamma_k)x^{(k)}$
6. After n steps, return $x^{(n)}$

3.1 Convergence rate

Following Jaggi (2011), define the curvature constant of f over C:

$$M = \max_{\gamma \in [0,1], x, s, y \in C, y=(1-\gamma)x+\gamma s} \frac{1}{2\gamma^2} \left(f(y) - f(x) - \nabla f(x)^T (y-x) \right)$$

Note that $M = 0$ for linear f, and $f(y) - f(x) - \nabla f(x)^T (y-x)$ is called the Bregman divergence, defined by f.

Theorem: The Frank-Wolfe method using standard step sizes $\gamma_k = \frac{2}{k+1}$, $k = 1, 2, 3, \dots$ satisfies:

$$f(x^{(k)}) - f^* \leq \frac{2M}{k+2}$$

Thus, the number of iterations needed for $f(x^{(k)}) - f^* \leq \epsilon$ is $O(1/\epsilon)$.

This matches the sublinear rate for projected gradient descent for Lipschitz ∇f, but how do the assumptions compare?

For Lipschitz ∇f with constant L, recall:

$$f(y) - f(x) - \nabla f(x)^T (y-x) \leq \frac{L}{2} ||y-x||^2$$

Maximizing over all $y = (1-\gamma)x + \gamma s$, and multiplying by $2/\gamma^2$:

$$M \leq L ||x-s||^2$$

Hence, assuming a bounded curvature is basically no stronger than what we assumed for projected gradient. **The key inequality** used to prove the Frank-Wolfe convergence rate:

$$f(x^{(k)}) \leq f(x^{(k-1)}) - \gamma_k g(x^{(k-1)}) + \frac{\gamma_k^2}{2} M$$

Here, $g(x) = \max_{s \in C} \nabla f(x)^T (x-s)$ is the duality gap.

Proof of inequality: Write $x^+ = x^{(k)}$, $x = x^{(k-1)}$, $s = s^{(k-1)}$, $\gamma = \gamma_k$. Then:

$$f(x^+) = f\left(x + \gamma(s-x)\right) \leq f(x) + \gamma \nabla f(x)^T (s-x) + \frac{\gamma^2}{2} M = f(x) - \gamma g(x) + \frac{\gamma^2}{2} M$$

The second line used the definition of M, and the third line the definition of g.

Now, we **continue the proof** of the convergence result.

Denote by: $h(x) = f(x) - f^*$, the suboptimality gap at x.

Basic inequality:

$$h(x^{(k)}) \leq h(x^{(k-1)}) - \gamma_k g(x^{(k-1)}) + \frac{\gamma_k^2}{2} M$$

Using that $g(x^{(k-1)}) \geq h(x^{(k-1)})$:

$$h(x^{(k)}) \leq h(x^{(k-1)}) - \gamma_k h(x^{(k-1)}) + \gamma_k^2/2M = (1-\gamma_k) h(x^{(k-1)}) + \gamma_k^2/2M$$

To get the desired result we use induction:

$$h(x^{(k)}) \leq \left(1 - \frac{2}{k+1}\right)^2 M(k+1) + \left(\frac{2}{k+1}\right)^2 M \leq \frac{2M}{k+2}$$

This completes our analysis.

4 Theoretical comparison

In general, the convergence rate of the Nesterov Algorithm will be faster than that of Frank-wolfe, however, when we take sparse matrices, then in Frank-Wolfe all multiplications are also performed for sparse matrices, that is, each iteration of Frank-Wolfe works faster. In the Nesterov method, at each step we calculate the projection of the vector onto the ball (projection of a sparse vector can be not sparse), which is why the matrices multiplied in the gradient cease to be sparse, namely:

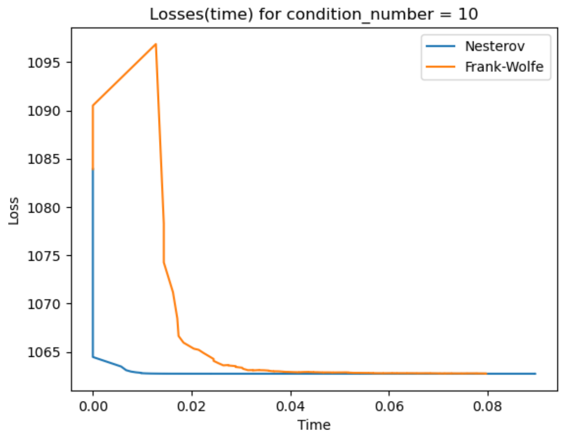
$$\nabla f(x) = A^T(Ax - b)$$

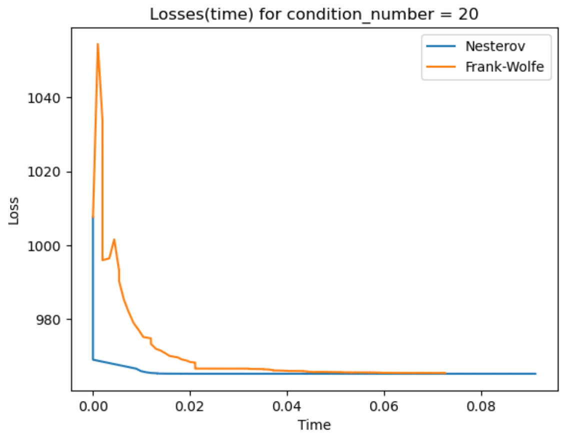
Here, vector x in Nesterov is dense.

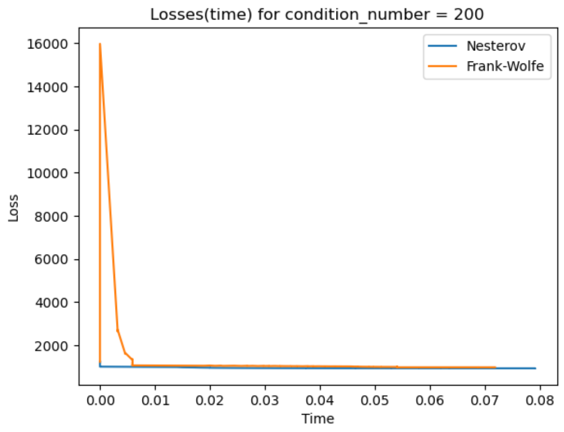
5 Numerical experiments

5.1 Dense matrices

Here is the translation of your text into English: For dense matrices, we experimented with various condition numbers and observed the relationship between loss and computation time. The results are displayed below.



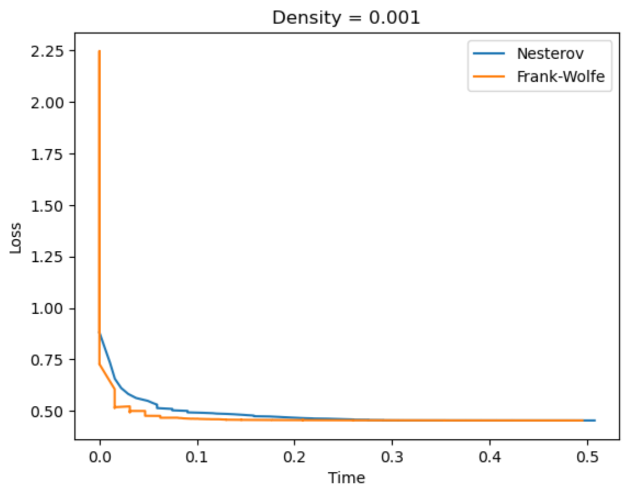


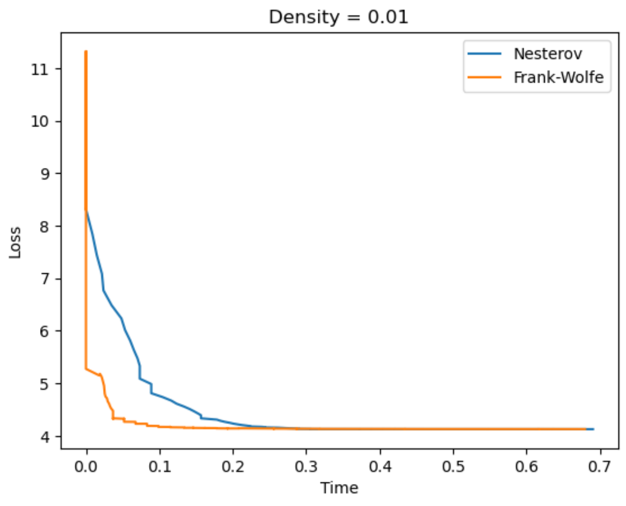


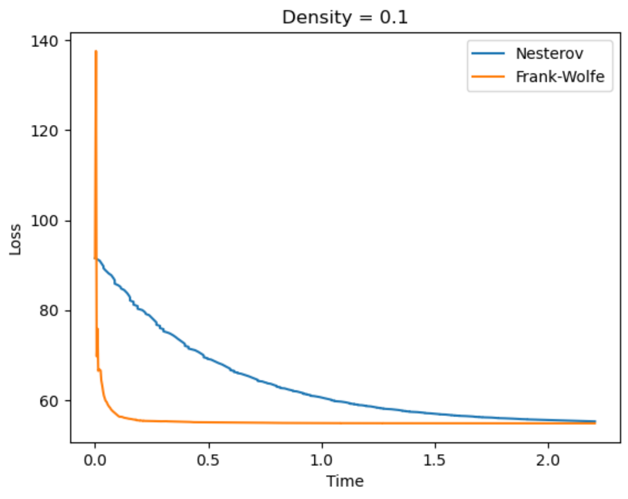
As we can see, in all cases, Nesterov performs more efficiently.

5.2 Sparse matrices

For sparse matrices, we experimented with different densities of the matrix and vectors. For each density, we also examined the relationship between loss and computation time.







In this case, as expected, Frank-Wolfe demonstrated greater efficiency than Nesterov.

6 Conclusion

The Nesterov algorithm will be preferable if you are solving the LASSO problem for dense matrices. However, if you are looking for a solution for sparse matrices, then the Frank-Wolfe algorithm will be more efficient.