Convex Optimization - Homework 3

Nicolás Violante

November 2021

1 Derivation of LASSO dual

For $X \in \mathrm{R}^{n\times d}$, $w \in \mathrm{R}^d$, $y \in \mathrm{R}^n$ and $\lambda > 0$, the LASSO minimization problem is defined as

$$\min_w \frac{1}{2}||Xw - y||_2^2 + \lambda||w||_1 \tag{1}$$

We solve LASSO by applying the Barrier Method to its dual, which happens to be a Quadratic Problem.

First, we reformulate (1) by introducing an equality constraint

$$\begin{aligned} &\min_w \frac{1}{2}||t||_2^2 + \lambda||w||_1 \\ &\text{s.t } t = Xw - y \end{aligned} \tag{2}$$

The Lagrangian of this problem is $L(t, w, v) = \frac{1}{2}||t||_2^2 + \lambda||w||_1 + v^T(t - Xw + y)$. Thus, in order to find the dual function $g(v)$ we need to find its infimum over t and w, which can be written as $g(v) = v^T y + \inf_t\{\frac{1}{2}||t||_2^2 + v^T t\} + \inf_w\{\lambda||w||_1 - v^T Xw\}$.

Since $\frac{1}{2}||t||_2^2 + v^T t$ is convex, we can find its minimum by setting its derivative to zero, $t + v = 0$, thus obtaining $t = -v$. For the term $\inf_w\{\lambda||w||_1 - v^T Xw\}$, given that $\lambda > 0$, we can write

$$\inf_w\{\lambda||w||_1 - v^T Xw\} = -\lambda \sup_w\{\frac{v^T Xw}{\lambda} - ||w||_1\} = -\lambda||.||_1^*\left(\frac{X^T v}{\lambda}\right)$$

where $||.||_1^*$ is the conjugate of $||.||_1$ which, as proven in the previous homework, is given by

$$||.||_1^*(y) = \begin{cases} 0 \text{ if } ||y||_\infty \leq 1 \\ \infty \text{ otherwise} \end{cases}$$

.

Therefore, the dual function is

$$g(v) = \begin{cases} v^T y - \frac{1}{2}v^T v \text{ if } ||\frac{X^T v}{\lambda}||_\infty \leq 1 \\ \infty \text{ otherwise} \end{cases}$$

and the dual problem (expressed as a minimization problem) is

$$\begin{aligned} \min_{v} \quad & \frac{1}{2} v^T v - v^T y \\ \text{s.t } & ||X^T v||_\infty \leq \lambda \end{aligned} \tag{3}$$

The constraint $||X^T v||_\infty \leq \lambda$ is equivalent to $|(X^T v)_i| \leq \lambda$ for all $i = 1, \ldots, d$. Therefore, we have $2d$ linear inequality constraints, namely $(X^T v)_i \leq \lambda$ and $-(X^T v)_i \leq \lambda$ for all $i = 1, \ldots, d$. Finally, we can rewrite (3) as a generic Quadratic Problem

$$\begin{aligned} \min_{v} \quad & v^T Q v + p^T v \\ \text{s.t } & Av \leq b \end{aligned} \tag{4}$$

where

- $Q = \frac{1}{2} I_n \in \mathrm{R}^{n \times n}$
- $p = -y \in \mathrm{R}^n$
- $b = \lambda \mathbf{1} \in \mathrm{R}^{2d}$
- $A = \begin{pmatrix} X^T \\ -X^T \end{pmatrix} \in \mathrm{R}^{2d \times n}$

Note that by solving the dual problem we can obtain w^*, since $Xw^* = y - v^*$

2 Experiments

2.1 Number of iterations and μ

We run the Barrier method for different values of μ and observe that the convergence is linear and after $\mu = 15$ there is actually not much difference on the number of iterations required to converge, which is around 60 iterations for the example of Figure 1. We also observe that for low values of μ each centering step requires less Newton iterations and the progress in each step is small and conversely, for large values of μ each centering step requires more Newton iterations and the progress in each step is much bigger. Therefore, μ controls the trade-off between the number of iterations and the progress in each centering step. In this particular experiment, a value of μ between $\mu = 15$ and $\mu = 100$ seems is reasonable choice to have a good balance between number of iterations required and progress made in each centering step.

During the experiments it was observed that $f(v_t)$ increased a little bit after each centering step. Newton's method guarantees that $tf(v) + \phi(v)$ decreases at each iteration, but the guarantees on $f(v)$ are after each centering step is completed, since the gap $f(v_t^*) - f^* \leq m/t$. Therefore, for the purpose of visualization we have only plotted the difference $f(v_t) - f^*$ for the v_t obtained at the end of each centering step in Figure 1. The results confirms that after each centering step the value of f effectively decreases.

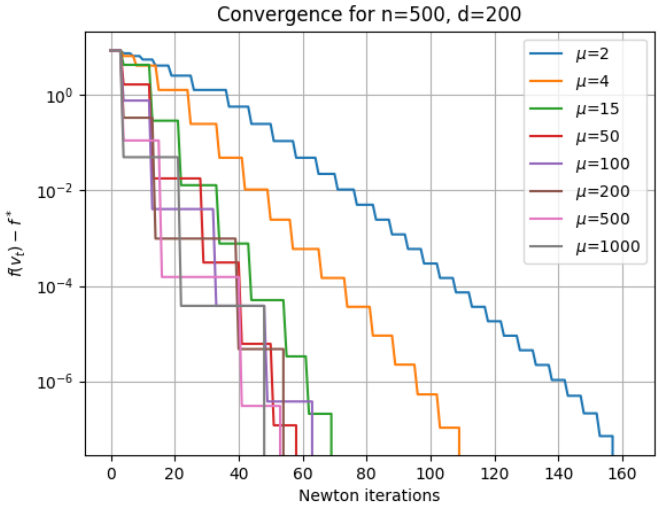

Figure 1: Results of the Barrier Method for $n = 500$, $d = 200$ varying μ.

2.2 Optimal w

We also analyse the impact of the choice of μ on the output vector w^*. From Figure 2 we conclude that the Barrier Method is robust with respect to μ since we obtain the same results for all the values used. In addition, we also check that the optimal solution w^* found is effectively sparse.

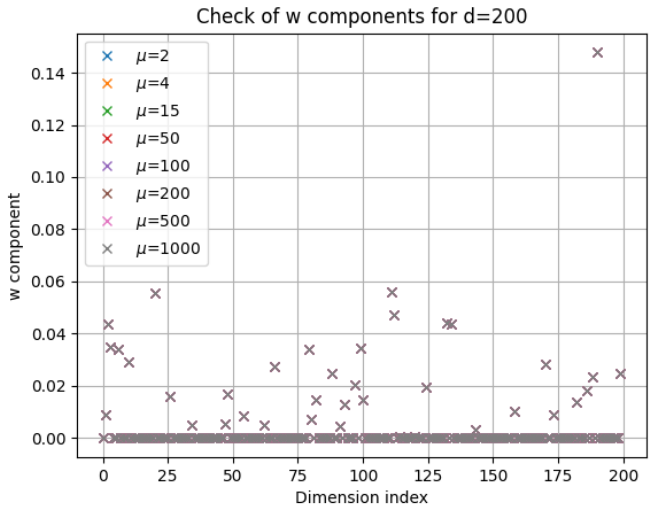

Figure 2: Optimal w^* found with the Barrier Method for $n = 500$, $d = 200$ varying μ

Convex Optimization

November 27, 2021

1 Convex Optimization - Homework 3

MVA 2021-2022

Nicolás Violante nviolante96@gmail.com

[ ]:
```
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt
from time import time
import warnings
warnings.filterwarnings("ignore", category=RuntimeWarning)


def backtracking_line_search(t, Q, p, A, b, v, grad, step, alpha=0.25, beta=0.
 ↪9):
    stop_criterium_met = False
    size = 1.0
    f = lambda x:t*(np.dot(x, Q @ x) + np.dot(p, x)) - np.sum(np.log(b - A@x))

    f_v = f(v)
    while not stop_criterium_met:
        stop_criterium_met = (f(v + size * step) <= (f_v + alpha * size * np.
 ↪dot(grad, step)))
        size = size * beta
    return v + (size/beta) * step


def centering_step(Q, p, A, b, t, v0, eps):

    stop_criterium_met = False
    v = v0
    v_seq = []
    while not stop_criterium_met:
        v_seq.append(v.copy())
        # Netwon step
        h = (1.0/(A@v -b))
        grad = t*(2 * Q @ v + p) - h @ A
```

```
        hessian = t*2*Q.T + (A.T*(h**2))@A
        step = - inv(hessian) @ grad

        # Newton decrement
        decrement = - np.dot(grad, step)
        stop_criterium_met = (0.5 * decrement < eps)

        # Line search
        v = backtracking_line_search(t, Q, p, A, b, v, grad, step)


    return v_seq


def barr_method(Q, p, A, b, v0, eps, mu=2, verbose=False):
    num_contraints = A.shape[0]
    t = 1

    iter = 1
    stop_criterium_met = False
    v = v0
    v_seq = []
    v_seq_to_plot = []
    while not stop_criterium_met:
        v_seq1 = centering_step(Q, p, A, b, t, v, eps)
        v = v_seq1[-1]
        v_seq_to_plot.extend([v]*len(v_seq1))
        v_seq.extend(v_seq1)
        stop_criterium_met = (num_contraints/t < eps)
        t = mu*t
        value = np.dot(v, Q@v) + np.dot(p, v)
        if verbose:
            print(f"Barrier iter {iter}: value {value}")
        iter += 1

    return v_seq, v_seq_to_plot



if __name__ == '__main__':
    # Matrices definition
    n = 500
    d = 200
    lamb = 10
    np.random.seed(42)

    X = np.random.rand(n, d)
```

```
y = np.random.rand(n)

A = np.concatenate((X.T, -X.T), axis=0)
b = lamb * np.ones((2*d,))
v0 = np.zeros(n)
Q = 0.5 * np.eye(n)
p = -y
eps=1e-6

# Plot for different values of mu
f = lambda x:np.dot(x, Q @ x) + np.dot(p, x)
mu_list = [2, 4, 15, 50, 100, 200, 500, 1000]
ws = []
plt.figure()
for mu in mu_list:
    tic = time()
    _, v_seq = barr_method(Q, p, A, b, v0, eps, mu)
    toc = time()
    f_values = np.array([f(v) for v in v_seq])
    plt.semilogy(f_values-f_values[-1], label=f'$\mu$={mu}')
    print(f"  mu={mu}, time: {toc-tic}")
    ws.append(np.linalg.pinv(X) @ (y - v_seq[-1]))

plt.grid()
plt.xlabel('Newton iterations')
plt.ylabel('$f(v_t) - f^*$')
plt.title(f'Convergence for n={n}, d={d}')
plt.legend()
plt.savefig(f'results/mu_plot_for_n_{n}_d_{d}.png')
plt.show()


plt.figure()
for i in range(len(mu_list)):
    plt.plot(ws[i], 'x', label=f'$\mu$={mu_list[i]}')
plt.title(f'Check of w components for d={d}')
plt.ylabel('w component')
plt.xlabel('Dimension index')
plt.grid()
plt.legend()
plt.savefig(f'results/w_for_n_{n}_d_{d}.png')
plt.show()
```