Homework 3

Vincent HERFELD

Question 1 : We formulate the LASSO as :

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \quad \frac{1}{2} \| Xw - y \|_2^2 + \lambda \|w\|_1$$

where $X = (x_1^T, \dots, x_n^T) \in \mathbb{R}^{n \times d}$, $y = (y_1, \dots, y_n) \in \mathbb{R}^n$ and $\lambda > 0$.

Let's derive it's dual problem :

First we want to add constraints that do not affect the problem but make the Lagrangian more interesting :

$$\min_{w, z \in \mathbb{R}^d} \quad \frac{1}{2} \|z\|_2^2 + \lambda \|w\|_1$$

$$\text{such that} \quad z - Xw + y = 0$$

We can write the Lagrangian : (with $\phi \in \mathbb{R}^n$)

$$\mathcal{L}(w, z, \phi) = \frac{1}{2} \|z\|_2^2 + \lambda \|w\|_1 + \phi^T (z - Xw + y)$$

We want the dual so we minimize according to w and z this comes from the gradients of the Lagrangian since it is convex (sum of convex functions) :

$$\nabla_z (w, z, \phi) = z + \phi = 0 \iff z = -\phi$$

- The problem is seperable so :

$$\inf_{z, w} \mathcal{L}(w, z, \phi) = \inf_z \left(\frac{1}{2} \|z\|_2^2 + \phi^T z \right) + \inf_w \left(\lambda \|w\|_1 - \phi^T X w \right) + \phi^T y$$

and as we have done in Home work 2: (and using $\lambda > 0$)

$$\inf_{\omega} \lambda \|\omega\|_1 + \phi^T(y - X\omega) = -\sup_{\omega}\left(\frac{1}{\lambda}\phi^T X\omega - \|\omega\|_1\right)$$

$$= -f^*\left(\frac{1}{\lambda}\phi X^T\right)$$

where f^* is the conjugate of $f : \omega \mapsto \|\omega\|_1$

defined as $f^* : x \longmapsto \sup_{\omega}\left(\omega^T x - \|\omega\|_1\right)$

and $f^*(x) = \begin{cases} 0 \text{ if } \|x\|_* \leq 1 \\ +\infty \text{ otherwise} \end{cases}$ $\left(\text{with } \|x\|_* = \sup_{\|u\|_1 \leq 1} u^T x\right)$

In fact $\|x\|_* = \|x\|_\infty$, we can show this by double inequality:

$\rightarrow$ $\|x\|_* = \sup_{\|u\|_1 \leq 1} u^T x = \sup_{\|u\|_1 \leq 1} \sum u_i x_i \leq \sup_{\|u\|_1 \leq 1} \sum |u_i x_i| \leq \sup_{\|u\|_1 \leq 1} \sum |u_i| \|x\|_\infty \leq \|x\|_\infty$

$\rightarrow$ taking $u_0 = (0, \ldots, 0, \underset{\uparrow}{\text{sign}(x_i)}, 0, \ldots 0)$ we have $\|u\|_1 = 1 \leq 1$.

i such that $|x_i| = \|x\|_\infty$

so $u_0^T x \leq \sup_{\|u\| \leq 1} u^T x \iff \|x\|_\infty \leq \|x\|_*$

So $\|x\|_* = \|x\|_\infty$.

So the dual function is:

$$g(\phi) = \frac{1}{2}\|-\phi\|_2^2 - \|\phi\|_2^2 - f_*\left(\frac{1}{\lambda}\phi X^T\right) + \phi^T y$$

And since we want to maximise g we need f_* to be 0

and so we want $\left\|\frac{1}{\lambda}\phi X^T\right\|_\infty \leq 1$ (ie $\forall i, |\phi^T X|_i \leq \lambda$)

And finally the dual problem is:

$$\underset{\phi \in \mathbb{R}^n}{\text{maximise}} \quad -\frac{1}{2}\phi^T\phi + \phi^T y$$
$$\text{such that} \quad \begin{cases} \phi^T X_i \geqslant -\lambda \\ \phi^T X_i \leq \lambda \end{cases} \qquad \forall i \in \{1, \dots, n\}$$

And in another form: (we minimize the opposit)

$$\underset{\phi \in \mathbb{R}^n}{\text{minimize}} \quad \frac{1}{2}\phi^T\phi - y^T\phi$$
$$\text{such that} \quad \forall i \in \{1, \dots, n\} \quad \begin{cases} -X^T\phi_i - \lambda \leq 0 \\ X^T\phi_i - \lambda \leq 0 \end{cases}$$

We have managed to write the dual of the LASSO as a QP :

$$\underset{x \in \mathbb{R}^n}{\text{minimize}} \quad x^T Q x + p^T x$$
$$\text{such that} \quad Ax - b \leq 0$$

where here $Q = \frac{1}{2} Id$ $\quad p = -y$, $A = \begin{pmatrix} X^T \\ -X^T \end{pmatrix}$ and $b = \lambda \mathbb{1}_{2d}$

2) We will need the following :

$$f_0(x) = x^T Q x + p^T x$$
$$f_0(x+h) = x^T Q x + h^T Q x + x^T Q h + h^T Q h + p^T x + p^T h$$
$$= f_0(x) + \langle h, \underbrace{2Qx + p}_{\nabla f_0(x)} \rangle + o(\|h\|)$$

$$df_{0\,x}(h) = \langle h, 2Qx + p \rangle$$

$$df_{0\,x+k}(h) = \langle h, 2Qx + p \rangle + \langle h, \underbrace{2Qk}_{\mathrm{Hess}(f)(x)k} \rangle$$

Finally : $\nabla f_0(x) = 2Qx + p$ and $\mathrm{Hess}(f_0)(x) = 2Q \succcurlyeq 0$

Since for a given t we will need $\nabla F(x)$ and $\mathrm{Hess}(F)(x)$

where $F(x) = t f_0(x) + \varphi(x) = t f_0(x) - \sum_{i=1}^{2d} \log(-f_i(x))$

We also need $\nabla \varphi(x)$ and $\mathrm{Hess}(\varphi)(x)$:

$$\nabla \varphi(x) = - \sum_{i=1}^{2d} \nabla\big(\log(-f_i(\cdot))\big)(x)$$
$$= - \sum_{i=1}^{2d} \nabla f_i(x) \cdot \frac{1}{f_i(x)}$$

We recall that

$$\forall i \in \{1, \dots, d\} \quad f_i(x) = (X^T x)_i - \lambda \Rightarrow \nabla f_i(x) = X_i = A_i^T$$

and $\forall i \in \{d+1, \dots, 2d\}$, $f_i(x) = -(X^T x)_i - \lambda \Rightarrow \nabla f_i(x) = -X_i = A_i^T$

So $$\nabla \varphi(x) = - \sum_{i=1}^{2d} \frac{A_i}{(Ax - b)_i}$$

And $$\nabla^2 \varphi(x) = -\sum_{i=1}^{2d} \nabla^2 f_i(x) \cdot \frac{1}{f_i(x)} - \sum_{i=1}^{2d} \nabla f_i(x) \cdot \left(\nabla f_i(x)^T \cdot \frac{-1}{f_i(x)^2} \right)$$

$$= \sum_{i=1}^{2d} \frac{1}{f_i(x)^2} \nabla f_i(x) \nabla f_i(x)^T - \frac{1}{f_i(x)} \nabla^2 f_i(x)$$

where $\nabla^2 f_i(x) = O_{\mathbb{R}^{n\times n}}$

So $$\nabla^2 \varphi(x) = \sum_{i=1}^{2d} \frac{A_i A_i^T}{(Ax-b)_i^2}$$

And finally we will have:

$$\nabla F(x) = t \nabla f_0(x) + \nabla \varphi(x)$$
$$= t(2Qx+p) - \sum_{i=1}^{2d} \frac{A_i}{(Ax+b)_i}$$

and $$\nabla^2 f(x) = t \nabla^2 f_0(x) + \nabla^2 \varphi(x)$$
$$= 2tQ + \sum_{i=1}^{2d} \frac{A_i A_i^T}{(Ax-b)_i^2}$$

Let's use all this to build a computational solver :

Homework3

November 20, 2023

1 Convex Optimization : Homework 3

1.1 Author : Vincent HERFELD

1.1.1 Question 2:

[286]:
```python
import numpy as np
import matplotlib.pyplot as plt
```

Given the answer to question 1) we would like to solve the following Quadratic Program problem which results in solving LASSO thanks to strong duality (convexity of the primal objective and constraints):

[

$$\begin{aligned} \underset{x\in\mathbb{R}^n}{\text{minimize}} \quad & x^TQx + p^Tx \\ \text{such that} \quad & Ax - b \preceq 0 \end{aligned}$$

]

where in our case $Q = I_n/2$, $p = -y$, $A = \begin{pmatrix} X^T \\ -X^t \end{pmatrix} \in \mathbb{R}^{2d\times n}$ and $b = \lambda \mathbf{1}_{2d}$. We note that $Q \succeq 0$.

To do so we will use the log-barrier method which relies on a log-barrier to approximate the inequality constraints and the Newton method to solve the centering problem and find the optimal point iteratively.

So finally we will solve the problem :

[

$$\underset{x\in\mathbb{R}^n}{\text{minimize}} \quad x^TQx + p^Tx - \frac{1}{t}\sum_{i=1}^{2d}\log(-f_i(x))$$

]

where $\forall i \in \{1, ..., d\} \quad f_i(x) = (X^tx)_i - \lambda = (Ax - b)_i$

and $\forall i \in \{d+1, ..., 2d\} \quad f_i(x) = -(X^tx)_i - \lambda = (Ax - b)_i$

So $\forall i \in \{1, ..., 2d\} \quad f_i(x) = (Ax - b)_i$

For the centering step we will use Newton's method that relies on the gradient and the Hessian of $F : x \mapsto t(x^T Qx + p^T x) - \phi(x)$.

We have (the proofs are before the notebook) :

$\nabla F(x) = t(2Qx + p) - \sum_{i=1}^{2d} \frac{A_i^T}{(Ax-b)_i}$

$Hess(F)(x) = 2tQ + \sum_{i=1}^{2d} \frac{A_i A_i^T}{(Ax-b)_i^2}$, where A_i is a row of the matrix A.

[492]:
```python
def F(x, t, Q, p, A, b):
    return t * (x.T @ Q @ x + p.T @ x) - np.sum(np.log(-(A@x - b)))

def gradF(x, t, Q, p, A, b):
    return t * (2 * Q @ x + p) - np.einsum("ij, ik->jk", A, 1 / (A @ x - b))

def HessF(x, t, Q, A, b):
    B = np.zeros((A.shape[1],A.shape[1]))
    for i in range(A.shape[0]):
        B += np.einsum("i, j -> ij", A[i], A[i]) / (A @ x - b)[i]**2
    return 2 * t * Q + B

def line_search(x, alpha, beta, lmbda2, Dx, A, b, t, Q, p):
    nu = 1
    while F(x + nu * Dx, t, Q, p, A, b) >= F(x, t, Q, p, A, b) - alpha * nu *
 ↪lmbda2:
        nu *= beta
    return nu

def centering_step(Q, p, t, A, b, v0, eps): #minimize over x, t*f_0(x) + phi(x)
 ↪with no constraints
    x_hat_list = [v0]
    x = v0
    while(True):
        #1. Compute the Newton step and decrement.
        Dx = -np.linalg.inv(HessF(x, t, Q, A, b)) @ gradF(x, t, Q, p, A, b)
        lmbda2 = gradF(x, t, Q, p, A, b).T @ -Dx

        #2. Stopping criterion. quit if  2/2    .
        if(lmbda2 / 2 <= eps):
            break

        #3. Line search. Choose step size nu by backtracking line search.
        nu = line_search(x, alpha, beta, lmbda2, Dx, A, b, t, Q, p)
        #4. Update. x := x + nu  xnt.
        x = x + nu * Dx
        x_hat_list.append(x)
    return x_hat_list
```

```
def barr_method(Q, p, t, A, b, v0, eps):
    v = [[v0]]
    m = A.shape[0]
    x_hat = v0
    while(True):
        #1. Centering step. Compute x (t) by minimizing tf0 +  , subject to Ax =␣
 ↪b.
        #2. Update. x := x (t).
        x_hat_list = centering_step(Q, p, t, A, b, x_hat, eps)
        #3. Stopping criterion. quit if m/t <  .

        if (m / t < eps):
            break
        #4. Increase t. t :=  t.
        t = mu * t
        x_hat = x_hat_list[-1]
        v.append(x_hat_list)
    return v
```

1.1.2 Question 3:

Let's apply our code, we recall that we need to start the algorithm at a feasible point, in our case it is simple to find $v_0 = 0_{\mathbb{R}^n}$ is a feasible point.

Thanks to strong duality (since the primal objective function is convex) once we obtain an optimal value for the dual we have the optimal value for the primal (they are equal).

We also note that thanks to our equations in question 1 we have an expression for the dual optimal point :

$z = -v$ where $z = Xw - y$ and v is the lagrangian multiplier (and the optimization variable for the dual), so considering the pseudo-inverse of X we have : $w = X^{\dagger}(y - v)$ so once we obtain an optimal point for the dual we can use it to find the optimal point for the primal.

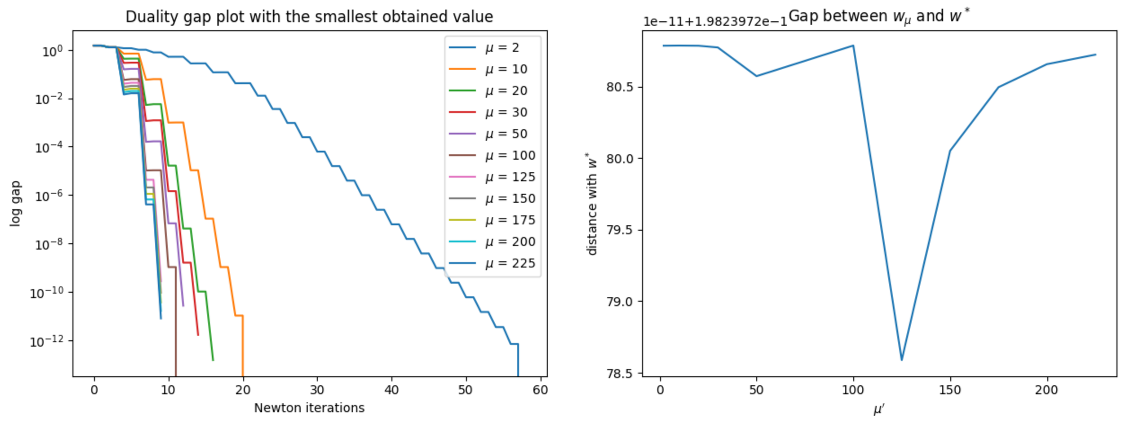
[545]:
```
n = 10
d = 500

alpha = 0.2
beta = 0.1
Mu = [2,10,20,30,50, 100, 125, 150, 175, 200, 225]
lmbda = 10

np.random.seed(20100)

X = np.random.uniform(size = (n,d))
y = np.random.uniform(size = (n,1))

A = np.concatenate((X.T, - X.T), axis = 0)
b = lmbda * np.ones((2*d, 1))
```

```python
Q = np.eye(n,n)/2
p = -y

v0 = np.zeros((n,1))
t = 1
eps = 1e-5
v=[]
for i in range(len(Mu)):
    mu = Mu[i]
    v.append(barr_method(Q, p, t, A, b, v0, eps))
```

We can now plot our results, in semilog scale :

[551]:
```python
def g(x, Q, p):
    return x.T @ Q @ x + p.T @ x

fig,(ax1,ax2) = plt.subplots(1,2, figsize=(15,5))

m = np.inf
w = []
for i,vm in enumerate(v):
    G = []
    for vi in vm:
        for x in vi:
            gv = g(x, Q, p)[0][0]
            G.append(gv)
            if gv < m:
                m = gv
                x_opt = x

    w.append(np.linalg.pinv(X) @ (y - G[-1]))
    ax1.semilogy(np.arange(len(G)),G - m, label="$\mu$ = {}".format(Mu[i]))

ax1.set_title("Duality gap plot with the smallest obtained value")
ax1.set_xlabel("Newton iterations")
ax1.set_ylabel("log gap")
ax1.legend()

w_opt = np.linalg.pinv(X) @ (y - x_opt)

d = []
for i in range(len(Mu)):
    d.append(np.linalg.norm(w[i] - w_opt))

ax2.plot(Mu, d)
ax2.set_title("Gap between $w_\mu$ and $w^*$")
```

```
ax2.set_xlabel("$\mu'$")
ax2.set_ylabel("distance with $w^*$")

plt.show()
```



We recall that when μ is larger we do more inner iterations per outer iteration, but we do less outer iterations, looking at the above graph this corresponds to doing less steps but steeper ones (on the left graph). And for $\mu \geq 10$ when increasing μ we gain a lot less in total Newton iterations : 58 iterations for $\mu = 2$ vs. 20 for $\mu = 10$ (so we devide by 3 the number of iterations when multiplying μ by 5), but for $\mu = 50$ $(= 10 \times 5)$ we only devide by 2. This continues decreasing untill a certain cap : for $\mu \geq 125$.

From my understanding the importance is having the less total Newton iterations in our algorithm whilst considering accuracy also as an important factor. So looking at the graph taking $\mu = 100$ we have a low number of iterations (around 10) and a high accuracy.

For the right graph I have plotted the distances between the obtained primal optimal points and the best point obtained over all μ. We see that modifying μ modifyies very poorly the precision for the primal problem resolution. There doesn't seem to have a rule except that when μ is different from μ^* there exists a small difference between the obtained optimal points.