Proximal_Identification_SOLUTIONS

December 7, 2022

Guest course Dec. 7, 2022

Structure identification:

from nonsmooth optimization to data science

Franck Iutzeler

The goal of this course/lab is to explore the notion of structure identification in nonsmooth optimization and its applications in data science.

1 1 - Proximity operator and stability

The proximity operator is defined for a function $g : \mathbb{R}^n \to \mathbb{R} \cup +\infty$ and a point $x \in \mathbb{R}^n$ as the mapping

$$\mathbf{prox}_{\gamma g}(x) = \arg\min_{u \in \mathbb{R}^n} \left\{ g(u) + \frac{1}{2\gamma}\|x-u\|^2 \right\}$$

1.1 1.a Stability in 1D

Let us first consider the absolute function $g = |\cdot|$ in $\mathbb{R}$ and plot the function

$$u \mapsto s_g(u;\gamma,x) = |u| + \frac{1}{2\gamma}\|x-u\|^2$$

for different values of γ, x.

[1]:
```
from bokeh.io import output_notebook, show
from bokeh.plotting import figure
output_notebook()
```

[2]:
```
from bokeh.layouts import column,row
from bokeh.models import CustomJS, ColumnDataSource, Slider

# Set up data
import numpy as np
N = 200
u = np.linspace(-4, 4, N)
p = np.abs(u)+0.5*np.power(u,2)

source = ColumnDataSource(data=dict(u=u, p=p))
opt = ColumnDataSource(data=dict(u=[0.0], p=[0.0]))
```

```
# Set up plot
plot = figure(height=400, width=400, title="stability of minimizers",
              x_range=[-4, 4], y_range=[-0.5, 12.5])

plot.line('u', 'p', source=source, line_width=3, line_alpha=0.
 ↪6,legend_label="s_g")
plot.circle('u', 'p', source=opt, size = 5,␣
 ↪color="black",legend_label=r"minimizer = prox_{\gamma g}(x)")
plot.legend.location = "top_left"



slider_gamma = Slider(start=0.01, end=4, value=1, step=.01, title="stepsize␣
 ↪$$\gamma$$")
slider_x = Slider(start=-4.0, end=4.0, value=0.0, step=.1, title="input point␣
 ↪$$x$$")

update_curve = CustomJS(args=dict(source=source, gamma=slider_gamma,␣
 ↪xslider=slider_x ), code="""
    const g = gamma.value
    const x = xslider.value
    const u = source.data.u
    const p = Array.from(u, (u) => Math.abs(u) + 0.5*Math.pow(u-x, 2)/g)
    source.data = { u, p }
""")
slider_gamma.js_on_change('value', update_curve)
slider_x.js_on_change('value', update_curve)


update_opt = CustomJS(args=dict(opt=opt, gamma=slider_gamma, xslider=slider_x␣
 ↪), code="""
    const g = gamma.value
    const x = xslider.value
    const p = opt.data.p
    const u = Array.from(p, (p) => Math.sign(x)*Math.max(0,Math.abs(x)-g))
    opt.data = { u , p }
""")
slider_gamma.js_on_change('value', update_opt)
slider_x.js_on_change('value', update_opt)

show(row(column(slider_gamma,slider_x), plot))
```

1. Show that

$$\mathbf{prox}_{\gamma g}(x) = \begin{cases} x-\gamma & \text{if } x \geq \gamma \\ x+\gamma & \text{if } x \leq -\gamma \\ 0 & \text{if } -\gamma \leq x \leq \gamma \end{cases}$$

Notice how the points of non-differentiabiity of g trap the minimizers of the problem.

This is a typical example of how nonsmoothness can be used to promote certain patterns in the solutions of minimization problems.

1.2 1.b Stability in nD

Let us first consider the ℓ_1 norm, ie. $g : x \mapsto \|x\|_1 = \sum_{i=1}^n |x_i|$ in $\mathbb{R}^n$ (where x_i is the i-th component of vector x).

2. How can one compute the proximity operator of the ℓ_1 norm from the one of the absolute value?

In the first part of the lab, we will consider the classical *lasso* problem:

$$\min_{x\in\mathbb{R}^n} \frac{1}{2}\|Ax-b\|^2 + \lambda\|x\|_1 \tag{lasso}$$

3. Observe the level sets of the *lasso* problem implemented below. Run the code several times to generate different sensing matrices A and vector $b = Ax_0 + \text{noise}$. Notice how the solution of the problem often seem to have a null first coordinate, as x_0.

[3]:
```
import numpy as np

x0 = np.array([0,1,0,0.5,-0.9])
n = x0.size

m = 10

A = np.random.randn(m,n)

b = np.dot(A,x0) + 0.1*np.random.randn(m)

lam = 0.5
```

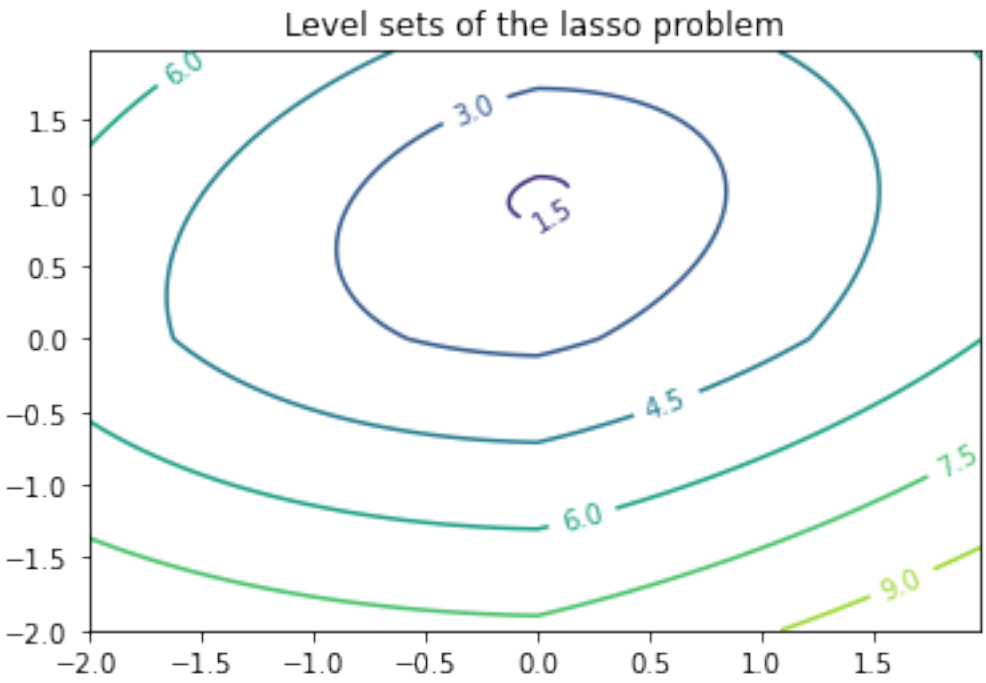
[4]:
```
def lasso(x,A=A,b=b,lam=lam):
    return 0.5*np.linalg.norm(np.dot(A,x)-b) + lam*np.linalg.norm(x,1)
```

[5]:
```
from scipy.optimize import minimize
x0 = np.zeros(n)
res = minimize(lasso, x0)
```

[6]:
```
import matplotlib.cm as cm
import matplotlib.pyplot as plt

delta = 0.025
x = np.arange(-2.0, 2.0, delta)
y = np.arange(-2.0, 2.0, delta)
X, Y = np.meshgrid(x, y)

Z = np.copy(X)
```

```
p = res.x
for index,x in np.ndenumerate(X):
    p[0] = x
    p[1] = Y[index]
    Z[index] = lasso(p)


fig, ax = plt.subplots()
CS = ax.contour(X, Y, Z)
ax.clabel(CS, inline=True, fontsize=10)
ax.set_title('Level sets of the lasso problem')
```

[6]: `Text(0.5, 1.0, 'Level sets of the lasso problem')`



2 2 - Stucture identification & Optimization

In this section, we will again consider the lasso problem

$$\min_{x\in\mathbb{R}^n} \frac{1}{2}\|Ax-b\|^2 + \lambda\|x\|_1$$

using the data generated below and we will compare the solutions obtained by several optimization methods.

[7]:
```
import numpy as np

n, m = 20, 50
prop_null = 0.5
null_coord = np.random.choice(n, int(prop_null*n), replace=False)

x0 = np.random.randn(n)
x0[null_coord] = 0.0


np.random.seed(42)

A   = np.random.randn(m,n)

b   = np.dot(A,x0) + 0.1*np.random.randn(m)

lam = 0.1*n
```

[8]:
```
x0
```

[8]:
```
array([ 0.        , -0.32051567,  0.16218066,  0.        , -2.53108861,
        0.        , -1.63263694,  0.        ,  1.14547535, -0.88881661,
        1.10114863, -1.00240739,  0.        ,  0.        ,  0.        ,
        0.26347435,  0.        ,  0.92921617,  0.        ,  0.        ])
```

[9]:
```
def lasso(x,A=A,b=b,lam=lam):
    return 0.5*np.linalg.norm(np.dot(A,x)-b)**2 + lam*np.linalg.norm(x,1)
```

In Section 2.1 to 2.5, you will be asked to solve this problem by different manners. Make sure to fill at least one method from 2.1,2.2,2.3 and one from 2.4,2.5. You will be asked to compare the solution in Section 2.6.

2.1 2.1 General purpose minimization

4. Solve the problem with a general purpose solver such as `scipy.optimize.minimize` https://docs.scipy.org/doc/scipy/reference/generated/scipy.optimize.minimize.html#scipy.optimize.minimiz Store the result in a variable `xopt_GP`.

[10]:
```
from scipy.optimize import minimize
```

[11]:
```
res = minimize(lasso, np.zeros(n),  options={'gtol': 1e-5, 'disp': True})
```

```
Warning: Desired error not necessarily achieved due to precision loss.
         Current function value: 19.368645
         Iterations: 82
         Function evaluations: 4690
         Gradient evaluations: 223
```

[12]:
```
xopt_GP = res.x
```

2.2 2.2 Modelling software

5. Solve the problem with a modelling software (eg. `cvxpy` https://www.cvxpy.org/) or a QP solver. Store the result in a variable `xopt_CVXPY`.

[13]:
```
import cvxpy as cp

# Construct the problem.
x = cp.Variable(n)
objective = cp.Minimize(0.5*cp.sum_squares(A @ x - b)+lam*cp.norm(x,1))
prob = cp.Problem(objective)

# The optimal objective value is returned by `prob.solve()`.
result = prob.solve(verbose = True)
# The optimal value for x is stored in `x.value`.
```

```
===============================================================================
                                     CVXPY
                                     v1.2.1
===============================================================================
(CVXPY) Dec 07 10:30:38 AM: Your problem has 20 variables, 0 constraints, and 0
parameters.
(CVXPY) Dec 07 10:30:38 AM: It is compliant with the following grammars: DCP,
DQCP
(CVXPY) Dec 07 10:30:38 AM: (If you need to solve this problem multiple times,
but with different data, consider using parameters.)
(CVXPY) Dec 07 10:30:38 AM: CVXPY will first compile your problem; then, it will
invoke a numerical solver to obtain a solution.
-------------------------------------------------------------------------------
                                  Compilation
-------------------------------------------------------------------------------
(CVXPY) Dec 07 10:30:38 AM: Compiling problem (target solver=OSQP).
(CVXPY) Dec 07 10:30:38 AM: Reduction chain: CvxAttr2Constr -> Qp2SymbolicQp ->
QpMatrixStuffing -> OSQP
(CVXPY) Dec 07 10:30:38 AM: Applying reduction CvxAttr2Constr
(CVXPY) Dec 07 10:30:38 AM: Applying reduction Qp2SymbolicQp
(CVXPY) Dec 07 10:30:38 AM: Applying reduction QpMatrixStuffing
(CVXPY) Dec 07 10:30:38 AM: Applying reduction OSQP
(CVXPY) Dec 07 10:30:38 AM: Finished problem compilation (took 1.259e-02
seconds).
-------------------------------------------------------------------------------
                                Numerical solver
-------------------------------------------------------------------------------
(CVXPY) Dec 07 10:30:38 AM: Invoking solver OSQP  to obtain a solution.
-------------------------------------------------------------------
           OSQP v0.6.2  -  Operator Splitting QP Solver
              (c) Bartolomeo Stellato,  Goran Banjac
        University of Oxford  -  Stanford University 2021
-------------------------------------------------------------------
```

```
problem:  variables n = 90, constraints m = 90
          nnz(P) + nnz(A) = 1180
settings: linear system solver = qdldl,
          eps_abs = 1.0e-05, eps_rel = 1.0e-05,
          eps_prim_inf = 1.0e-04, eps_dual_inf = 1.0e-04,
          rho = 1.00e-01 (adaptive),
          sigma = 1.00e-06, alpha = 1.60, max_iter = 10000
          check_termination: on (interval 25),
          scaling: on, scaled_termination: off
          warm start: on, polish: on, time_limit: off

iter   objective    pri res    dua res    rho        time
   1  -3.2000e+02   8.00e+00   1.04e+04   1.00e-01   3.12e-04s
 100   1.9369e+01   6.18e-05   7.46e-06   1.30e+00   9.79e-04s
plsh   1.9369e+01   1.46e-15   8.38e-15   --------   1.19e-03s

status:               solved
solution polish:      successful
number of iterations: 100
optimal objective:    19.3686
run time:             1.19e-03s
optimal rho estimate: 2.13e+00

------------------------------------------------------------------------------
                                    Summary
------------------------------------------------------------------------------
(CVXPY) Dec 07 10:30:38 AM: Problem status: optimal
(CVXPY) Dec 07 10:30:38 AM: Optimal value: 1.937e+01
(CVXPY) Dec 07 10:30:38 AM: Compilation took 1.259e-02 seconds
(CVXPY) Dec 07 10:30:38 AM: Solver (including time spent in interface) took
2.117e-03 seconds
```

[14]:
```
xopt_CVXPY = x.value
```

2.3 2.3 Dedicated library

6. Solve the problem with a dedicated library as **sklearn** https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html. Store the result in a variable **xopt_sklearn**.

[15]:
```
from sklearn import linear_model

clf = linear_model.Lasso(alpha=lam/n,fit_intercept=False)
clf.fit(A,b)
```

[15]:
```
Lasso(alpha=0.1, fit_intercept=False)
```

[16]:
```
xopt_sklearn = clf.coef_
```

2.4 2.4 Proximal Gradient Method

Recall, the iterations of the proximal gradient are of the form

$$x_{k+1} = \mathbf{prox}_{\gamma g}\left(x_k - \gamma \nabla f(x_k)\right)$$

where γ is a stepsize to choose.

7. Solve the problem with a proximal gradient method. Store the result in a variable `xopt_PG`.

[17]:
```
def f(x,A=A,b=b):
    return 0.5*np.linalg.norm(np.dot(A,x)-b)

def grad_f(x,A=A,b=b):
    return np.dot(A.T,(np.dot(A,x)-b))
```

[18]:
```
def g(x,lam=lam):
    return  lam*np.linalg.norm(x,1)

def prox_g(x,gamma,lam=lam):
    n = x.size
    p = np.zeros(n)
    for i in range(n):
        if x[i] > gamma*lam:
            p[i] = x[i] - gamma*lam
        elif x[i] < -gamma*lam:
            p[i] = x[i] + gamma*lam
    return p
```

[19]:
```
x = np.zeros(n)

ITE_MAX = 10000

gamma = 1/np.linalg.norm(A)**2

for i in range(ITE_MAX):
    g = grad_f(x)
    x_prev = x.copy()

    x = prox_g(x-gamma*g,gamma)

    if np.linalg.norm(x-x_prev)<= 1e-5:

        break
```

[20]:
```
xopt_PG = x.copy()
```

2.5 2.5 Proximal Splitting methods: Douglas-Rachford, ADMM, etc.

Use any proximal splitting method you wish, eg. the Douglas-Rachford method which reads

$$x_k = \mathbf{prox}_{\gamma g}(u_k)$$
$$z_k = \mathbf{prox}_{\gamma f}(2x_k - u_k)$$
$$u_{k+1} = u_k + z_k - x_k$$

8. Solve the problem with a splitting method such as Douglas-Rachford, ADMM, etc. Store the result in a variable `xopt_Split`. Several variables may converge to the same value, you can pick one arbitrarily and get back to it later.

[21]:
```
def f(x,A=A,b=b):
    return 0.5*np.linalg.norm(np.dot(A,x)-b)

def prox_f(x,gamma,A=A,b=b):
    return np.linalg.solve(np.dot(A.T,A)+np.eye(n)/gamma, np.dot(A.T,b) + x/
 ↪gamma)
```

[22]:
```
def g(x,lam=lam):
    return  lam*np.linalg.norm(x,1)

def prox_g(x,gamma,lam=lam):
    n = x.size
    p = np.zeros(n)
    for i in range(n):
        if x[i] > gamma*lam:
            p[i] = x[i] - gamma*lam
        elif x[i] < -gamma*lam:
            p[i] = x[i] + gamma*lam
    return p
```

[23]:
```
x = np.zeros(n)
u = np.zeros(n)
z = np.zeros(n)

ITE_MAX = 10000

gamma = 1.0

for i in range(ITE_MAX):
    x = prox_g(u,gamma)
    z = prox_f(2*x-u,gamma)
    u = u + z - x
```

[24]:
```
xopt_Split = x.copy()
```

2.6 2.6 Comparison

9. Compare the objective values for the points obtained above. Compare the sparsity pattern of these points and compare them to the one of x_0.

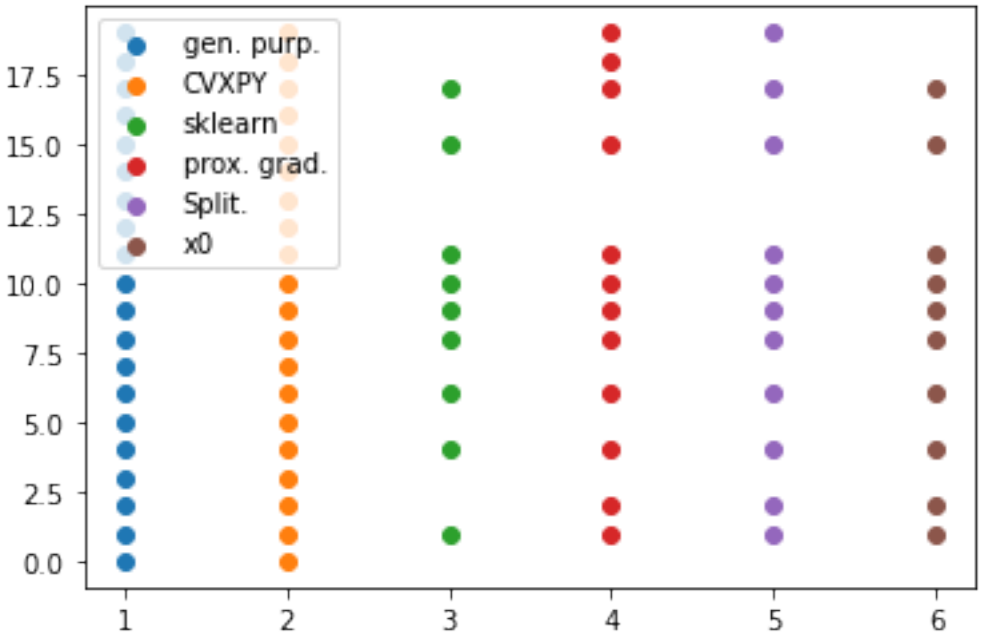
[25]:
```
print("         Gen.Purp. \t\t CVXPY     \t\t sklearn   \t\t Prox. Grad. \t\t␣
 ↪Split.")
for i in range(n):
    print("{:3d} - \t{:10.3e} \t\t{:10.3e} \t\t{:10.3e} \t\t{:10.3e} \t\t{:10.
 ↪3e}".
 ↪format(i+1,xopt_GP[i],xopt_CVXPY[i],xopt_sklearn[i],xopt_PG[i],xopt_Split[i]))

print("")
print("obj.: \t{:7.3f} \t\t{:7.3f} \t\t{:7.3f} \t\t{:7.3f} \t\t{:7.3f}".
 ↪format(lasso(xopt_GP),lasso(xopt_CVXPY),lasso(xopt_sklearn),lasso(xopt_PG),lasso(xopt_Split
```

```
         Gen.Purp.                CVXPY                    sklearn
Prox. Grad.              Split.
  1 -   -3.898e-09                1.459e-21                0.000e+00
0.000e+00                0.000e+00
  2 -   -2.860e-01               -2.860e-01               -2.452e-01
-2.861e-01               -2.860e-01
  3 -    5.114e-02                5.114e-02               -0.000e+00
5.100e-02                5.114e-02
  4 -    3.488e-09                2.204e-21                0.000e+00
0.000e+00                0.000e+00
  5 -   -2.449e+00               -2.449e+00               -2.323e+00
-2.449e+00               -2.449e+00
  6 -   -1.965e-08                2.374e-21               -0.000e+00
0.000e+00                0.000e+00
  7 -   -1.563e+00               -1.563e+00               -1.454e+00
-1.563e+00               -1.563e+00
  8 -   -9.235e-09               -1.785e-22                0.000e+00
0.000e+00                0.000e+00
  9 -    1.055e+00                1.055e+00                9.128e-01
1.055e+00                1.055e+00
 10 -   -7.764e-01               -7.764e-01               -6.727e-01
-7.763e-01               -7.764e-01
 11 -    9.661e-01                9.661e-01                8.403e-01
9.662e-01                9.661e-01
 12 -   -9.750e-01               -9.750e-01               -9.263e-01
-9.749e-01               -9.750e-01
 13 -   -2.482e-09               -1.038e-21               -0.000e+00
0.000e+00                0.000e+00
 14 -   -2.801e-07               -1.201e-21               -0.000e+00
0.000e+00                0.000e+00
 15 -    5.894e-09                1.064e-21                0.000e+00
0.000e+00                0.000e+00
```

```
 16 -    1.928e-01                1.928e-01                1.462e-01
1.927e-01                1.928e-01
 17 -    4.849e-09                2.146e-22               -0.000e+00
0.000e+00                0.000e+00
 18 -    8.375e-01                8.375e-01                6.885e-01
8.370e-01                8.375e-01
 19 -    2.528e-05               -1.338e-21                0.000e+00
5.931e-05                0.000e+00
 20 -    1.568e-02                1.568e-02                0.000e+00
1.597e-02                1.568e-02

obj.:    19.369                   19.369                   20.661
19.369                   19.369
```

[26]:
```
def non_null_coords(x,thres=1e-30):
    non_null = []
    for i,v in enumerate(x):
        if np.abs(v)>thres:
            non_null.append(i)
    return non_null
```

[27]:
```
import matplotlib.pyplot as plt

fig = plt.figure()
ax1 = fig.add_subplot(111)

ax1.scatter(np.ones(np.
 ↪size(non_null_coords(xopt_GP))),non_null_coords(xopt_GP),label="gen. purp.")
ax1.scatter(2*np.ones(np.
 ↪size(non_null_coords(xopt_CVXPY))),non_null_coords(xopt_CVXPY),label="CVXPY")
ax1.scatter(3*np.ones(np.
 ↪size(non_null_coords(xopt_sklearn))),non_null_coords(xopt_sklearn),label="sklearn")
ax1.scatter(4*np.ones(np.
 ↪size(non_null_coords(xopt_PG))),non_null_coords(xopt_PG),label="prox. grad.")
ax1.scatter(5*np.ones(np.
 ↪size(non_null_coords(xopt_Split))),non_null_coords(xopt_Split),label="Split.
 ↪")
ax1.scatter(6*np.ones(np.
 ↪size(non_null_coords(x0))),non_null_coords(x0),label="x0")
plt.legend(loc='upper left')
plt.show()
```



10. Compare the results obtain when the dimensions and the sparsity increases, eg. `n, m = 100, 200; prop_null = 0.6` or `n, m = 400, 200; prop_null = 0.8`

3 3 - a Practical example in Classification

3.0.1 Machine Learning as an Optimization problem

We have some *data* $\mathcal{D}$ consisting of m *examples*; each example consisting of a *feature* vector $a_i \in \mathbb{R}^n$ and an *observation* $b_i \in \mathcal{O}$: $\mathcal{D} = \{[a_i, b_i]\}_{i=1..m}$. In this lab, we will consider the student performance dataset.

The goal of *supervised learning* is to construct a predictor for the observations when given feature vectors.

A popular approach is based on *linear models* which are based on finding a *parameter* x such that the real number $\langle a_i, x\rangle$ is used to predict the value of the observation through a *predictor function* $g : \mathbb{R} \to \mathcal{O}$: $g(\langle a_i, x\rangle)$ is the predicted value from a_i.

In order to find such a parameter, we use the available data and a *loss* ℓ that penalizes the error made between the predicted $g(\langle a_i, x\rangle)$ and observed b_i values. For each example i, the corresponding error function for a parameter x is $f_i(x) = \ell(g(\langle a_i, x\rangle); b_i)$. Using the whole data, the parameter that minimizes the total error is the solution of the minimization problem

$$\min_{x\in\mathbb{R}^n} \frac{1}{m}\sum_{i=1}^{m} f_i(x) = \frac{1}{m}\sum_{i=1}^{m} \ell(g(\langle a_i, x\rangle); b_i).$$

3.0.2 Binary Classification with Logisitic Regression

In our setup, the observations are binary: $\mathcal{O} = \{-1, +1\}$, and the *Logistic loss* is used to form the following optimization problem

$$\min_{x \in \mathbb{R}^n} f(x) := \frac{1}{m} \sum_{i=1}^{m} \log(1 + \exp(-b_i \langle a_i, x \rangle)) + \frac{\lambda}{2} \|x\|_2^2.$$

where the last term is added as a regularization (of type ℓ_2, aka Tikhonov) to prevent overfitting.

Under some statistical hypotheses, $x^\star = \arg\min f(x)$ maximizes the likelihood of the labels knowing the features vector. Then, for a new point d with features vector a,

$$p_1(a) = \mathbb{P}[a \in \text{ class } +1] = \frac{1}{1 + \exp(-\langle a; x^\star \rangle)}$$

Thus, from a, if $p_1(a)$ is close to 1, one can decide that d belongs to class 1; and the opposite decision if $p(a)$ is close to 0.

3.0.3 L1-regularization

We will consider an ℓ_1 regularization of this problem to promote sparsity of the iterates. A sparse final solution would select the most important features. The new function (below) is non-smooth but it has a smooth part, f, and a non-smooth part, g.

$$\min_{x \in \mathbb{R}^n} F(x) := \underbrace{\frac{1}{m} \sum_{i=1}^{m} \log(1 + \exp(-b_i \langle a_i, x \rangle)) + \frac{\lambda_2}{2} \|x\|_2^2}_{f(x)} + \underbrace{\lambda_1 \|x\|_1}_{g(x)}.$$

3.0.4 Features signification

The dataset is comprised of 27 features described below and the goal is to predict if the student may pass its year or not. It is thus of importance to investigate which features are the most significant for the student success. We will see how the ℓ_1 regularization can help to this goal. 1 sex - student's sex (binary: "F" - female or "M" - male) 2 age - student's age (numeric: from 15 to 22) 3 address - student's home address type (binary: "U" - urban or "R" - rural) 4 famsize - family size (binary: "LE3" - less or equal to 3 or "GT3" - greater than 3) 5 Pstatus - parent's cohabitation status (binary: "T" - living together or "A" - apart) 6 Medu - mother's education (numeric: 0 - none, 1 - primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education) 7 Fedu - father's education (numeric: 0 - none, 1 - primary education (4th grade), 2 – 5th to 9th grade, 3 – secondary education or 4 – higher education) 8 traveltime - home to school travel time (numeric: 1 - <15 min., 2 - 15 to 30 min., 3 - 30 min. to 1 hour, or 4 - >1 hour) 9 studytime - weekly study time (numeric: 1 - <2 hours, 2 - 2 to 5 hours, 3 - 5 to 10 hours, or 4 - >10 hours) 10 failures - number of past class failures (numeric: n if 1<=n<3, else 4) 11 schoolsup - extra educational support (binary: yes or no) 12 famsup - family educational support (binary: yes or no) 13 paid - extra paid classes within the course subject (Math or Portuguese) (binary: yes or no) 14 activities - extra-curricular activities (binary: yes or no) 15 nursery - attended nursery school (binary: yes or no) 16 higher - wants to take higher education (binary: yes or no) 17 internet - Internet access at home (binary: yes or no) 18 romantic - with a romantic relationship (binary:

yes or no) 19 famrel - quality of family relationships (numeric: from 1 - very bad to 5 - excellent) 20 freetime - free time after school (numeric: from 1 - very low to 5 - very high) 21 goout - going out with friends (numeric: from 1 - very low to 5 - very high) 22 Dalc - workday alcohol consumption (numeric: from 1 - very low to 5 - very high) 23 Walc - weekend alcohol consumption (numeric: from 1 - very low to 5 - very high) 24 health - current health status (numeric: from 1 - very bad to 5 - very good) 25 absences - number of school absences (numeric: from 0 to 93) 26 G1 - first period grade (numeric: from 0 to 20) 27 G2 - second period grade (numeric: from 0 to 20)

3.1 3.1 Objective definition

[28]:
```
import numpy as np
import csv

#### File reading
dat_file = np.load('student.npz')
A = dat_file['A_learn']
final_grades = dat_file['b_learn']
m = final_grades.size
b = np.zeros(m)
for i in range(m):
    if final_grades[i]>11:
        b[i] = 1.0
    else:
        b[i] = -1.0

A_test = dat_file['A_test']
final_grades_test = dat_file['b_test']
m_test = final_grades_test.size
b_test = np.zeros(m_test)
for i in range(m_test):
    if final_grades_test[i]>11:
        b_test[i] = 1.0
    else:
        b_test[i] = -1.0


d = 27 # features
n = d+1 # with the intercept




lam2 = 0.1 # for the 2-norm regularization best:0.1
lam1 = 0.03 # for the 1-norm regularization best:0.03


L = 0.25*max(np.linalg.norm(A,2,axis=1))**2 + lam2
```

The size of x is given as n, and L represents the Lipschitz constant of the gradient of f.

3.1.1 Oracles

Below are functions returning the gradient of f and proximity operator of g.

[29]:
```
def f(x,A=A,b=b):
    l = 0.0
    for i in range(A.shape[0]):
        if b[i] > 0 :
            l += np.log( 1 + np.exp(-np.dot( A[i] , x ) ) )
        else:
            l += np.log( 1 + np.exp(np.dot( A[i] , x ) ) )
    return l/m + lam2/2.0*np.dot(x,x)

def grad_f(x,lam2=lam2):
    g = np.zeros(n)
    for i in range(A.shape[0]):
        if b[i] > 0:
            g += -A[i]/( 1 + np.exp(np.dot( A[i] , x ) ) )
        else:
            g += A[i]/( 1 + np.exp(-np.dot( A[i] , x ) ) )
    return g/m + lam2*x
```

[30]:
```
def g(x,lam1=lam1):
    return lam1*np.linalg.norm(x,1)

def prox_g(x,gamma,lam1=lam1):
    p = np.zeros(n)
    for i in range(n):
        if x[i] < - lam1*gamma:
            p[i] = x[i] + lam1*gamma
        if x[i] > lam1*gamma:
            p[i] = x[i] - lam1*gamma
    return p
```

[31]:
```
def F(x):
    return f(x) + g(x)
```

3.1.2 Prediction Function

Below are two functions that compote the training accuracy (accuracy on the training set), and testing accuracy, given a parameter x.

[32]:
```
def prediction_train(w,PRINT=True):
    pred = np.zeros(A.shape[0])
    perf = 0
    for i in range(A.shape[0]):
```

```
        p = 1.0/( 1 + np.exp(-np.dot( A[i] , w ) ) )
        if p>0.5:
            pred[i] = 1.0
            if b[i]>0:
                correct = "True"
                perf += 1
            else:
                correct = "False"
            if PRINT:
                print("True class: {:d} \t-- Predicted: {} \t(confidence: {:.
 ↪1f}%)\t{}".format(int(b[i]),1,(p-0.5)*200,correct))
        else:
            pred[i] = -1.0
            if b[i]<0:
                correct = "True"
                perf += 1
            else:
                correct = "False"
            if PRINT:
                print("True class: {:d} \t-- Predicted: {} \t(confidence: {:.
 ↪1f}%)\t{}".format(int(b[i]),-1,100-(0.5-p)*200,correct))
    return pred,float(perf)/A.shape[0]

def prediction_test(w,PRINT=True):
    pred = np.zeros(A_test.shape[0])
    perf = 0
    for i in range(A_test.shape[0]):
        p = 1.0/( 1 + np.exp(-np.dot( A_test[i] , w ) ) )
        if p>0.5:
            pred[i] = 1.0
            if b_test[i]>0:
                correct = "True"
                perf += 1
            else:
                correct = "False"
            if PRINT:
                print("True class: {:d} \t-- Predicted: {} \t(confidence: {:.
 ↪1f}%)\t{}".format(int(b[i]),1,(p-0.5)*200,correct))
        else:
            pred[i] = -1.0
            if b_test[i]<0:
                correct = "True"
                perf += 1
            else:
                correct = "False"
            if PRINT:
```

```
                print("True class: {:d} \t-- Predicted: {} \t(confidence: {:.
↪1f}%)\t{}".format(int(b[i]),-1,100-(0.5-p)*200,correct))
    return pred,float(perf)/A_test.shape[0]
```

3.2 3.2 Proximal Gradient algorithm

For minimizing a function $F : \mathbb{R}^n \to \mathbb{R}$ equal to $f + g$ where f is differentiable and the **prox** of g is known, given: * the function to minimize `F` * a 1st order oracle for f `f_grad` * a proximity operator for g `g_prox` * an initialization point `x0` * the sought precision `PREC` * a maximal number of iterations `ITE_MAX` * a display boolean variable `PRINT`

this algorithm perform iterations of the form

$$x_{k+1} = \mathbf{prox}_{\gamma g}(x_k - \gamma \nabla f(x_k))$$

where γ is a stepsize to choose.

11. Fill the function below with the proximal gradient algorithm.

[33]:
```
import numpy as np
import timeit

def proximal_gradient_algorithm(F , grad_f , prox_g , x0 , step , PREC ,␣
↪ITE_MAX , PRINT ):
    x = np.copy(x0)
    x_tab = np.copy(x)
    if PRINT:
        print("-----------------------------------\n Proximal gradient␣
↪algorithm\n-----------------------------------\nSTART    -- stepsize = {:
↪0}".format(step))
    t_s =  timeit.default_timer()
    for k in range(ITE_MAX):
        g = grad_f(x)
        x = prox_g(x - step*g , step, lam1=lam1)  #######  ITERATION

        x_tab = np.vstack((x_tab,x))


    t_e =  timeit.default_timer()
    if PRINT:
        print("FINISHED -- {:d} iterations / {:.6f}s -- final value: {:f}\n\n".
↪format(k,t_e-t_s,F(x)))
    return x,x_tab
```

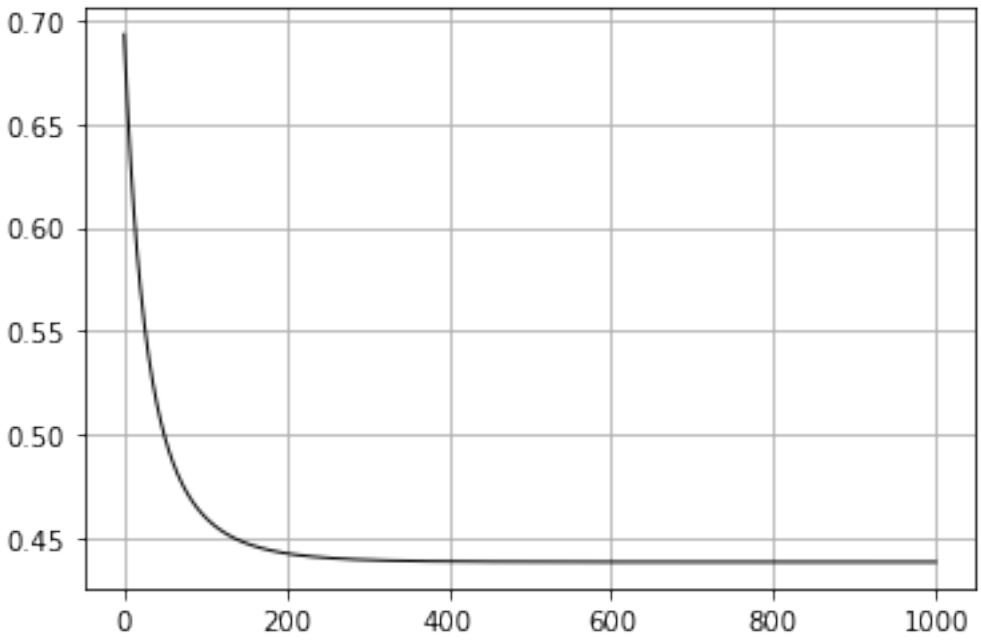
12. Run the algorithm and investigate the decrease of the algorithm.

[34]:
```
#### Parameter we give at our algorithm
PREC    = 1e-5                      # Sought precision
ITE_MAX = 1000                       # Max number of iterations
x0      = np.zeros(n)               # Initial point
```

```
step    = 1.0/L

##### gradient algorithm
x,x_tab = proximal_gradient_algorithm(F , grad_f , prox_g , x0 , step , PREC,␣
 ↪ITE_MAX , True)
```

```
-------------------------------------
 Proximal gradient algorithm
-------------------------------------
START    -- stepsize = 0.030334772813507393
FINISHED -- 999 iterations / 1.577216s -- final value: 0.438712
```

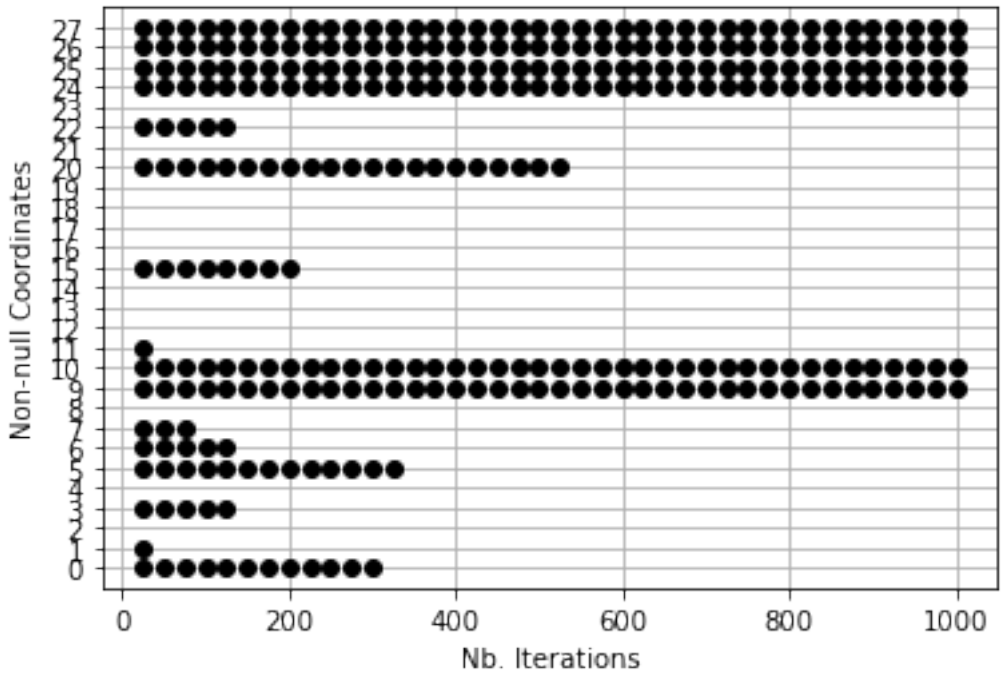
[35]:
```
import matplotlib.pyplot as plt


Fval = []
for i in range(x_tab.shape[0]):
    Fval.append( F(x_tab[i]))

plt.figure()
plt.plot( Fval, color="black", linewidth=1.0, linestyle="-")
plt.grid(True)
plt.show()
```



13. Plot the support of the vector x_k (i.e. one point for every non-null coordinate of

x_k) versus the iteration number. What do you notice? Was it expected?

[36]:
```
plt.figure()

for i in np.arange(0,x_tab.shape[0],int(x_tab.shape[0]/40)):
    for j in range(n):
        if np.abs(x_tab[i,j])>1e-14:
            plt.plot( i , j  , 'ko')

plt.grid(True)
plt.ylabel('Non-null Coordinates')
plt.xlabel('Nb. Iterations')
plt.ylim(-1,d+1)
plt.yticks(np.arange(0,d+1))
plt.show()
```



3.3 3.3 Regularization path.

We saw above that the algorithm *selected* some coordinates as the other get to zero. Considering our machine learning task, this translates into the algorithm selecting a subset of the features that will be used for the prediction step.

14. Change the parameter λ_1 of the problem (`lam1`) in the code above and investigate how it influences the number of selected features.

In order to quantify the influence of this feature selection, let us consider the *regularization path*

that is the support of the final points obtained by our minimization method versus the value of λ_1.

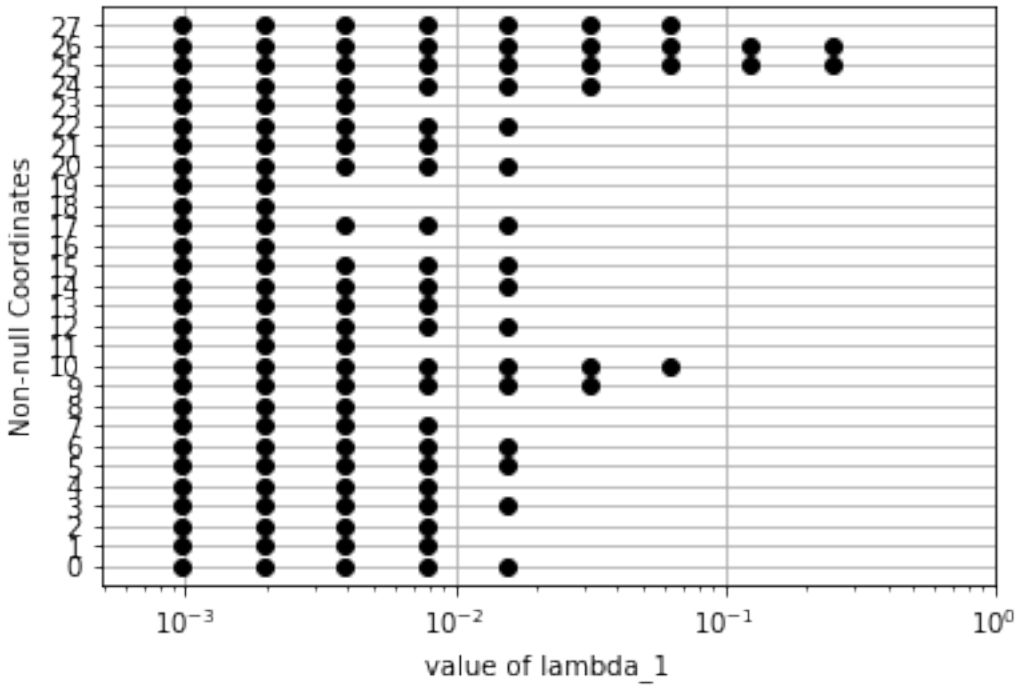
15. For $\lambda_1 = 2^{-10}, 2^{-9}, .., 2^1$, run the proximal gradient algorithm on the obtained problem and store the support of the final point, the prediction performance on the *training set* (`prediction_train`) and on the *testing set* (`prediction_test`). What is the best regularization choice and which are the most important features of the dataset?

[37]:
```
import matplotlib.pyplot as plt

import numpy as np


#### Parameter we give at our algorithm (see algoGradient.ipynb)
PREC    = 1e-5                      # Sought precision
ITE_MAX = 500                       # Max number of iterations
x0      = np.zeros(n)               # Initial point
step    = 1.0/L

reg_l1_tab = np.power( 2.0, np.arange(-10,1,1) )
lam2 = 1e-1

Perf_train = []
Perf_test = []
Pts = []
for reg_l1 in reg_l1_tab:
    lam1 = reg_l1
    x,x_tab = proximal_gradient_algorithm(F , grad_f , prox_g , x0 , step , ␣
 ↪PREC , ITE_MAX, False )
    for i in range(n):
        if np.abs(x[i])>1e-14:
            Pts.append([reg_l1,i])

    pred_train,perf_train = prediction_train(x,False)
    pred_test,perf_test = prediction_test(x,False)

    Perf_train.append(perf_train*100.0)
    Perf_test.append(perf_test*100.0)
    print("lambda1 = {:.2e} , lambda2 = {:.2e}, performance training: {:3.1f}%,␣
 ↪performance test: {:3.1f}%".format(lam1,lam2,perf_train*100,perf_test*100))
```

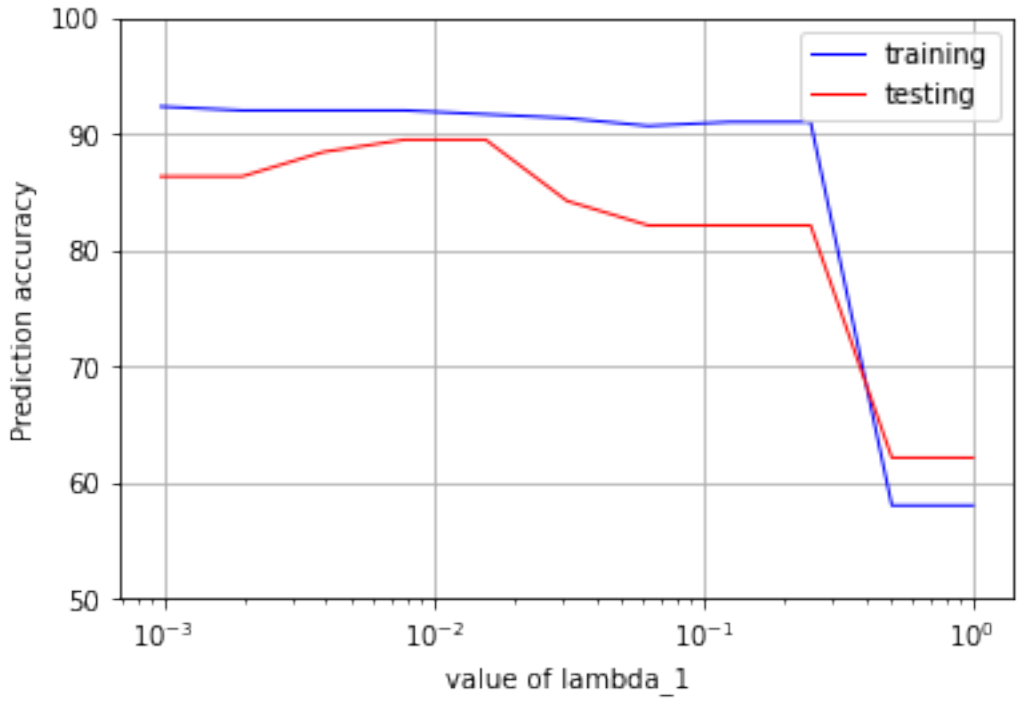
```
lambda1 = 9.77e-04 , lambda2 = 1.00e-01, performance training: 92.3%,
performance test: 86.3%
lambda1 = 1.95e-03 , lambda2 = 1.00e-01, performance training: 92.0%,
performance test: 86.3%
lambda1 = 3.91e-03 , lambda2 = 1.00e-01, performance training: 92.0%,
performance test: 88.4%
```

```
lambda1 = 7.81e-03 , lambda2 = 1.00e-01, performance training: 92.0%,
performance test: 89.5%
lambda1 = 1.56e-02 , lambda2 = 1.00e-01, performance training: 91.7%,
performance test: 89.5%
lambda1 = 3.12e-02 , lambda2 = 1.00e-01, performance training: 91.3%,
performance test: 84.2%
lambda1 = 6.25e-02 , lambda2 = 1.00e-01, performance training: 90.7%,
performance test: 82.1%
lambda1 = 1.25e-01 , lambda2 = 1.00e-01, performance training: 91.0%,
performance test: 82.1%
lambda1 = 2.50e-01 , lambda2 = 1.00e-01, performance training: 91.0%,
performance test: 82.1%
lambda1 = 5.00e-01 , lambda2 = 1.00e-01, performance training: 58.0%,
performance test: 62.1%
lambda1 = 1.00e+00 , lambda2 = 1.00e-01, performance training: 58.0%,
performance test: 62.1%
```

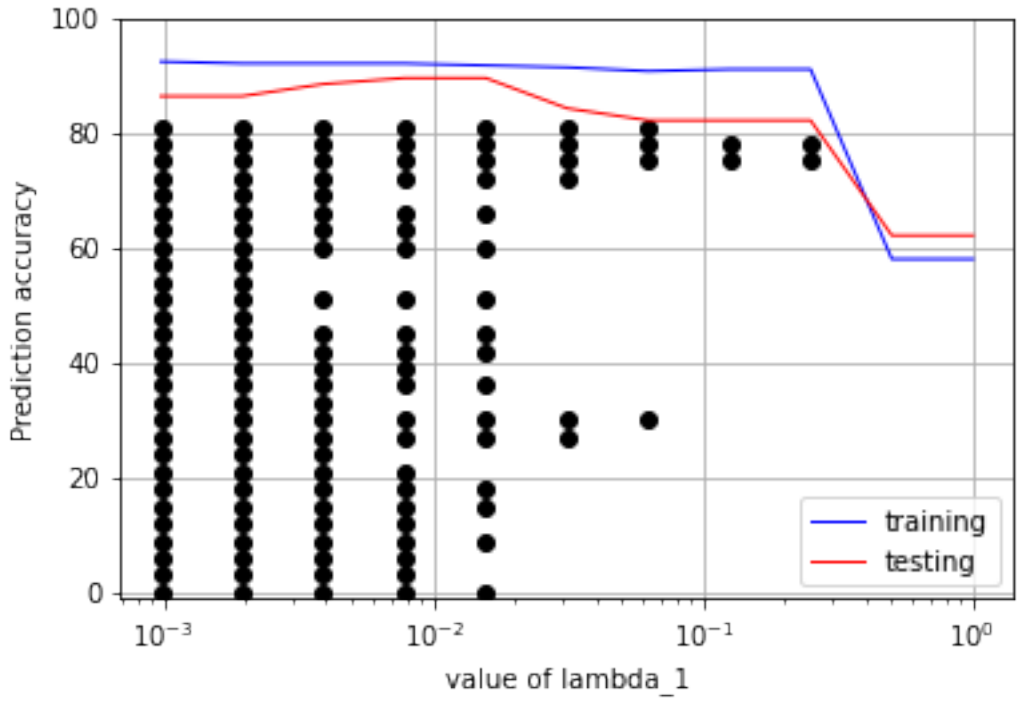
[38]:
```
plt.figure()
for point in Pts:
    plt.plot( point[0], point[1] , 'ko')
plt.xscale('log')
plt.grid(True)
plt.xlim(min(reg_l1_tab/2.0),max(reg_l1_tab))
plt.ylim(-1,d+1)
plt.yticks(np.arange(0,d+1))
plt.ylabel('Non-null Coordinates')
plt.xlabel('value of lambda_1')
plt.show()
```



[39]:
```
import matplotlib.pyplot as plt


plt.figure()
plt.plot( reg_l1_tab , Perf_train , color="blue", linewidth=1.0, linestyle="-",␣
 ↪label="training")
plt.plot( reg_l1_tab , Perf_test , color="red", linewidth=1.0, linestyle="-",␣
 ↪label="testing")
plt.grid(True)
plt.xscale('log')
plt.ylim([50,100])
plt.ylabel('Prediction accuracy')
plt.xlabel('value of lambda_1')
plt.legend()
plt.show()
```



[40]:
```
plt.figure()
for point in Pts:
    plt.plot( point[0], point[1]*3 , 'ko')
plt.xscale('log')
plt.grid(True)
plt.ylabel('Non-null Coordinates')
plt.xlabel('value of lambda_1')
plt.plot( reg_l1_tab , Perf_train , color="blue", linewidth=1.0, linestyle="-",␣
 ↪label="training")
plt.plot( reg_l1_tab , Perf_test , color="red", linewidth=1.0, linestyle="-",␣
 ↪label="testing")
plt.ylim([-1,100])
plt.ylabel('Prediction accuracy')
plt.xlabel('value of lambda_1')
plt.legend()
plt.show()
```



3.4 3.4 Bonus questions

16. What happens if you replace the ℓ_1 norm by the ℓ_∞ norm $\|x\|_\infty = \max_i |x_i|$?

[ ]:

17. What happens if you replace the proximal gradient by SAGA (see https://arxiv.org/pdf/1407.0202.pdf)

[ ]: