NLP project SD211 12/01/23

Hippolyte Giraud et Clément Dardare

In [6]:
```
import numpy as np
import scipy.sparse as sp
from scipy.linalg import norm, solve
from scipy.optimize import check_grad
import matplotlib.pyplot as plt
import time
```

Question 3.1

GRADIANT

$$\nabla f_1(w_0, w) = \begin{pmatrix} \frac{\partial f_1}{\partial w_0}(w_0, w) \\ \frac{\partial f_1}{\partial w}(w_0, w) \end{pmatrix} \qquad \begin{matrix} w_0 \in \mathbb{R} \\ w \in \mathbb{R}^p \end{matrix}$$

$$\frac{\partial f_1}{\partial w_0}(w_0, w) = -\frac{1}{n}\sum_{i=1}^{n} y_i \frac{e^{-y_i(x_i^T w + w_0)}}{1 + e^{-y_i(x_i^T w + w_0)}}$$

i.e. $\boxed{\frac{\partial f_1}{\partial w_0}(w_0, w) = -\frac{1}{n}\sum_{i=1}^{n} y_i \frac{1}{1 + e^{+y_i(x_i^T w + w_0)}}}$

$\forall w = \begin{pmatrix} w_1 \\ \vdots \\ w_p \end{pmatrix} \in \mathbb{R}^p, \forall i \in [\![1, p]\!]$

$$\frac{\partial \|.\|_2^2}{\partial i}(w) = 2 w_i$$

$$\underbrace{x_i^T w}_{g_i(w)} = \sum_{j=1}^{n} x_{ij} w_j \Rightarrow \frac{\partial g_i(w)}{\partial j} = x_{ij}$$

d'où $\boxed{\forall j \geq 1 \quad \frac{\partial f_1}{\partial w_j} = -\frac{1}{n}\sum_{i=1}^{n} y_i x_{ij} \frac{1}{1 + e^{y_i(x_i^T w + w_0)}} + 2 w_j}$

d'où, $\forall (w_0, w) \in \mathbb{R} \times \mathbb{R}^p$

$$\nabla f_1(w_0, w) = \begin{pmatrix} \frac{\partial f_1}{\partial w_0}(w_0, w) \\ \frac{\partial f_1}{\partial w_1}(w_0, w) \\ \vdots \\ \frac{\partial f_1}{\partial w_p}(w_0, w) \end{pmatrix}$$

→ Les expressions de chaque dérivé partielle sont écrites au dessus

HESSIAN MATRIX

D'après le TH de Schwartz :

$f_1 \in \mathcal{C}^2 \Rightarrow \forall i, j \quad \frac{\partial^2 f_1}{\partial i \partial j} = \frac{\partial^2 f_1}{\partial j \partial i}$

En posant $\forall i \in [\![1, n]\!]$ $\boxed{\varphi_i(w_0, w) = \frac{e^{-y_i(x_i^T w + w_0)}}{(1 + e^{-y_i(x_i^T w + w_0)})^2}}$

On a : (avec $\forall i \; y_i^2 = 1$)

- $\forall \, 1 \leq j \leq p \quad \frac{\partial^2 f_1}{\partial w_0 \partial w_j}(w_0, w) = \frac{1}{n} \sum_{i=1}^{n} x_{ij} \varphi_i(w_0, w)$
- $\frac{\partial^2 f_1}{\partial w_0^2}(w_0, w) = \frac{1}{n} \sum_{i=1}^{n} \varphi_i(w_0, w)$
- $\forall \, 1 \leq j \leq p \quad \frac{\partial^2 f_1}{\partial w_j^2}(w_0, w) = \frac{1}{n} \sum_{i=1}^{n} x_{ij}^2 \varphi_i(w_0, w) + \rho$
- $\forall \, 1 \leq k, j \leq p$, $k \neq j$ $\quad \frac{\partial^2 f_1}{\partial w_k \partial w_j} = \frac{1}{n} \sum_{i=1}^{n} x_{ij} x_{ik} \varphi_i(w_0, w)$

En posant, $\forall i \in [\![1, n]\!]$ $\hat{x}_i = (1, x_i)$
et $\hat{w} = (w_0, w)$

$$H_{f_1}(\hat{w}) = \sum_{i=1}^{n} \hat{x}_i \hat{x}_i^T \frac{\varphi_i(\hat{w})}{n} + \rho \begin{pmatrix} 0 & & 0 \\ & 1 & \\ 0 & & 1 \end{pmatrix}$$

Or $\forall i$ $\hat{x}_i \hat{x}_i^T \frac{\varphi_i(w)}{n}$ est positive :

En effet, $\forall u \in \mathbb{R}^p$ $u^T \hat{x}_i \hat{x}_i^T \frac{\varphi_i(w)}{n} u = \| \hat{x}_i \sqrt{\frac{\varphi_i(w)}{n}} u \|$

Chargement des données

In [14]:
```
def load_data(file_name_matrix='tfidf_matrix.npz', file_name_feature_
          file_name_labels='train_labels.npy', samples_in_train_set=
         samples_in_test_set=137562):

    # Recuperation des donnees
    TF_IDF_matrix = sp.load_npz(file_name_matrix)
    TF_IDF_feature_names = np.load(file_name_feature_names)
    train_labels = np.load(file_name_labels, allow_pickle=True)
    train_labels_numeric = ((train_labels == 'Oui') + 0)

    X = TF_IDF_matrix[:samples_in_train_set].toarray()
    y = train_labels_numeric[:samples_in_train_set] * 2 - 1

    X_test = TF_IDF_matrix[samples_in_train_set:samples_in_train_set
    y_test = train_labels_numeric[samples_in_train_set:samples_in_tra


    # Standardisation des données
    std_X = np.maximum(np.std(X, axis=0), 1e-7)
    X = X / std_X
    X_test = X_test / std_X

    n = X.shape[0]
```

```python
    n_test = X_test.shape[0]
    m = X.shape[1]

    # Ajout d'une colonne de uns
    eX = np.hstack((np.ones((n,1)), X))
    eX_test = np.hstack((np.ones((n_test,1)), X_test))

    return eX, y, eX_test, y_test

eX, eY, eX_test, y_test = load_data()
```

Initialisation des variables globales

In [15]:
```python
global_rho = 1/len(eX)
dim_p = 3
```

Question 3.2

In [41]:
```python
def norme2(w):
    n = len(w)
    a = 0
    for i in range(n):
        a = w[i]**2
    return np.sqrt(a)

def exp_part(w0, w, xi, yi):
    prod = np.dot(xi.T, w)
    return np.exp(-yi*(prod + w0))

def f1(w0, w, x, y, rho):
    p = len(w)
    n = len(y)
    a = 0
    for i in rang(n):
        log = np.log(1 + exp_part(w0, w, x[i], y[i]))
        a += log
    a = 1/n * a + (rho/2) * norme2(w)
    return a

def grad_f1(w0, w, x, y, rho):
    p = len(w)
    n = len(y)
    t = np.zeros(p+1)
    a = 0
    for i in range(n):
        a += y[i] * (1/(1 + exp_part(w0, w, x[i], -y[i]))) # utiliser
    t[0] = -1/n * a
    for j in range(p):
        a = 0
        for i in range(n):
            a += y[i] * x[i][j] * (1/(1 + exp_part(w0, w, x[i], -y[i]
        a = (-1/n) * a + rho * w[j]
        t[j] = a
    return t

def hess_f1(w0, w, x, y , rho):
    p = len(w)
    n = len(y)
```

```
    t = np.zeros((p+1,p+1))
    a = 0

    for i in range(n): #calcul w0, w0
        a += y[i]**2 * (exp_part(w0, w, x[i], y[i])/ (1 + exp_part(w(
    a = (1/n) * a
    t[0][0] = a

    for j in range(p): #calcul wj, wj j >= 1
        a = 0
        for i in range(n):
            a += (y[i]*x[i][j])**2 * (exp_part(w0, w, x[i], y[i])/ (1
        a = rho + (1/n) * a
        t[j][j] = a

    for j in range(p): #calcul w0, wj j >= 1
        a = 0
        for i in range(n):
            a += y[i]**2 * x[i][j] * (exp_part(w0, w, x[i], y[i])/ (1
        a = (1/n) * a
        t[0][j] = a
        t[j][0] = a

    for j in range(p): #calcul wj, wk j, k >= 1
        for k in range(j, p):
            a = 0
            for i in range(n):
                a += y[i]**2 * x[i][j] * x[i][k] * (exp_part(w0, w, x
            a = (1/n) * a
            t[j][k] = a
            t[k][j] = a

    return t
```

Question 3.3

In [50]:
```
def newton_method (f, grad_f, hess_f, w0, w, eps, x, y, rho):
    grad = grad_f(w0, w, x, y, rho)
    Les_X=[]
    Les_Y=[]
    i=0
    W=np.concatenate((w0,w))
    while (np.linalg.norm(grad)**2 > eps):
        hess = hess_f(w0, w, x, y , rho)
        W = W - np.linalg.inv(hess).dot(grad)
        w0=W[0]
        W=W[1:]
        grad = grad_f(w0, w, x, y, rho)
        Les_X.append(i)
        Les_Y.append(np.log(np.linalg.norm(grad)))
    return w, Les_X, Les_Y
```

Execution de la méthode de Newton avec initialisation (w0,w)=0

In [*]:
```
w = np.zeros(576)
w0= np.zeros(1)
print(len(eX[0]))
```

```
w_min, Les_X, Les_Y = newton_method(f1, grad_f1, hess_f1, w0, w, 10*
print(Les_X,Les_Y)
plt.figure()
plt.plot(Les_Y,Les_Y)
plt.show()
```

Execution de la méthode de Newton avec initialisation (w0,w)=1

In [ ]:

```
w= np.zeros(576)
w0=np.zeros(1)
print(w0)
w_min, Les_X, Les_Y = newton_method(f1, grad_f1, hess_f1, w0, w, 10*
print(Les_X,Les_Y)
plt.figure()
plt.plot(Les_Y,Les_Y)
plt.show()
```