

# Fouille de données et medias sociaux

## TP3 : régularisation L1

Ludovic Denoyer

2 octobre 2017

### Contexte

Nous allons nous intéresser à l'implémentation d'une méthode qui permet de classifier **tout en sélectionnant** les caractéristiques d'entrée les plus utiles. Nous considérons les notations suivantes :

- $\mathcal{X} = (x^1, \dots, x^\ell)$  l'ensemble des données d'apprentissage dans  $\mathbb{R}^n$  tel que  $x^j = (x_1^j, \dots, x_n^j)$
- $\mathcal{Y} = (y^1, \dots, y^\ell)$  l'ensemble des étiquettes associées tel que  $y^j \in \mathbb{R}$

Notre objectif est de trouver une fonction  $f_\theta : \mathbb{R}^n \rightarrow \mathbb{R}$  qui permet de faire une bonne classification. Pour toute nouvelle donnée  $x$ , on considérera que la classe associée à  $x$  par  $f_\theta$  est la classe positive si  $f_\theta(x) > 0$  et la classe négative sinon. L'*accuracy* sera la mesure finale d'évaluation qui mesurera le pourcentage de bonne classification sur un ensemble de test.

### Modèle linéaire

Nous allons considérer un critère d'apprentissage de type moindres carrés, et une fonction linéaire de décision  $f_\theta(x) = \langle \theta; x \rangle$  avec  $\theta \in \mathbb{R}^n$ . Nous définissons la fonction de coût suivante définie sur l'ensemble d'apprentissage :

$$\mathcal{L}(\theta) = \frac{1}{\ell} \sum_{i=1}^{\ell} (y^i - f_\theta(x^i))^2 + \lambda C(\theta)$$

où  $C(\theta)$  est le nombre de caractéristiques utilisées par  $f_\theta$  pour la prédiction ;  $\lambda$  est un coefficient choisi manuellement qui permet de mesurer le compromis entre la qualité du modèle, et le nombre de features utilisées.

### Descente de gradient

---

**Algorithm 1** Descente de gradient

---

```
1 : procedure GRADIENT( $\theta, \mathcal{X}, \mathcal{Y}, I; \epsilon$ )
2 :   for  $it = 1, I$  do
3 :     for  $i = 1, n$  do
4 :        $idx \leftarrow \text{random}(\ell)$ 
5 :        $\theta \leftarrow \theta - \epsilon \nabla_{\theta} \mathcal{L}(\theta)(idx)$ 
6 :     end for
7 :     Afficher  $\mathcal{L}(\theta)$  et  $\text{accuracy}(\theta)$  pour contrôle
8 :   end for
9 :   return  $\theta$ 
10 : end procedure
```

---

Afin de trouver la valeur  $\theta^*$  qui minimise la fonction  $\mathcal{L}(\theta)$ , nous allons utiliser l'algorithme du gradient. Cette algorithme très simple est un algorithme itératif donc le pseudocode est donnée en Algorithme 1.

## Régularisation $L_1$

Dans l'idéal, pour réaliser la sélection des variables, la fonction de coût  $C(\theta)$  doit être égale au nombre de caractéristiques utilisées en entrée. Remarquons que, étant donné que  $f_\theta(x) = \langle \theta; x \rangle$ , le nombre de caractéristiques utilisées est égale au nombre de valeurs non-nulles du vecteur  $\theta$ . On appelle cela la norme  $L_0$  :

$$L_0(\theta) = \sum_k \mathbb{1}_{\theta_k \neq 0}$$

où  $\mathbb{1}$  est la fonction indicatrice.

La fonction indicatrice n'est pas une fonction dérivable, et nous ne pouvons donc utiliser l'algorithme de descente de gradient. Nous allons utiliser une **relaxation continue** de la norme  $L_0$  qui est la norme  $L_1$ . Le nouveau problème d'apprentissage est alors défini par la fonction de coût suivante :

$$\mathcal{L}(\theta) = \frac{1}{\ell} \sum_{i=1}^{\ell} (y^i - f_\theta(x^i))^2 + \lambda |\theta|$$

La fonction  $L_1$  n'est toujours pas dérivable en tous ses points, et nous allons utiliser une technique de gradient-clipping afin calculer la valeur des poids  $\theta$ . Le principe consiste à annuler les poids  $\theta_i$  quand ceux-ci changent de signe durant la descente de gradient. Le nouvel algorithme est donné en algorithme 2.

---

### Algorithm 2 Descente de gradient $L_1$

---

```

1 : procedure GRADIENTL1( $\theta, \mathcal{X}, \mathcal{Y}, I; \epsilon$ )
2 :   for  $it = 1, I$  do
3 :     for  $i = 1, n$  do
4 :        $idx \leftarrow \text{random}(\ell)$ 
5 :        $\theta' \leftarrow \theta - \epsilon \nabla_{\theta} \mathcal{L}(\theta)(idx)$ 
6 :       for  $j=1, n$  do ▷ Gradient Clipping
7 :         if  $\theta'_j * \theta_j < 0$  then
8 :            $\theta_j \leftarrow 0$ 
9 :         else
10 :           $\theta_j \leftarrow \theta'_j$ 
11 :        end if
12 :      end for
13 :    end for
14 :    Afficher  $\mathcal{L}(\theta)$  et  $\text{accuracy}(\theta)$  pour contrôle
15 :  end for
16 :  return  $\theta$ 
17 : end procedure

```

---

## Implantation

### Question 1

Implémenter le modèle linéaire régularisé  $L_1$  sous la forme d'une classe Python (voir annexe).

## Question 2

Tester le modèle pour une valeur de  $\lambda = 0$  afin de vérifier son bon fonctionnement.

## Question 3

Utiliser une méthode de type cross-validation afin de calculer la performance moyenne de ce modèle pour tout un ensemble de valeurs de  $\lambda$ .

## Question 4

Dessiner la courbe de performance (sur un ensemble de test) en fonction de la **parcimonie** (ou sparsité) du modèle.

## Question 5

Quelle est la meilleure valeur de  $\lambda$  ?

## Question 6 *Extension*

Remplacez le terme  $L_1(\theta)$  par un terme  $L_2(\theta) = |\theta|^2$ . Que constatez vous ?

## Question 7 *Extension*

Remplacez le terme  $L_1(\theta)$  par un terme  $\lambda_1 L_1(\theta) + \lambda_2 L_2(\theta)$ . Que constatez vous ?

## Question 8 *Extension*

Comparez au modèle implémenté dans `sklearn.linear_model.Lasso`

# Aide Python

## Chargements de données "classiques"

Le chargement de données "classiques" s'effectue en *sklearn* par l'utilisation de la méthode *sklearn.datasets.fetch\_mldata*. Vous trouverez sur mldata tout un ensemble de datasets de **classification binaire**. Vous trouverez aussi des données à l'adresse <https://www.csie.ntu.edu.tw/~cjlin/libsvmtools/datasets/>

## Exemple de développement d'un nouveau modèle

L'API standard de *sklearn* pour les classifieurs de *sklearn* est relativement simple, l'essentiel du travail est fait dans les méthodes `fit` et `predict`.

```
import numpy as np
from sklearn.base import BaseEstimator, ClassifierMixin
from sklearn.svm import SVC
from sklearn import cross_validation

class RandomOuSvmClassifier(BaseEstimator, ClassifierMixin):
```

```

""" Si la nature du classifieur est "random" il prédit au hasard des 0 et
des 1 sinon il utilise un SVM pour prédire"""

def __init__(self, nature="random"):
    self.nature = nature
    self.svm = SVC()

def fit(self, X, y):
    if self.nature=="svm":
        self.svm.fit(X, y)
    return self

def predict(self, X):
    if self.nature=="random":
        return np.random.randint(0,2,len(X))
    else:
        return self.svm.predict(X)

##### Utilisation #####

from sklearn import datasets
from sklearn import metrics
iris = datasets.load_iris()
X = iris.data
y = iris.target
classifieurRandom = RandomOuSvmClassifier(nature="random")
classifieurSvm = RandomOuSvmClassifier(nature="svm")

scoresRandom = cross_validation.cross_val_score(classifieurRandom, X, y, cv=5,
                                                scoring="accuracy")

scoresSvm = cross_validation.cross_val_score(classifieurSvm, X, y, cv=5,
                                             scoring="accuracy")

```