FDMS - TP2 - Regularisation L_1

Ludovic Denoyer

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 :

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

(1)

Notre objectif est de trouver une fonction $f_\theta : \mathbb{R}^n \to \mathbb{R}$ qui permet de faire une bonne classification. Pour toute nouvelle donnée x, on considerera que la classe associée à x par f_θ est la classe positive si $f_\theta(x) > 0$ et la classe negative 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 :

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

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

Descente de gradient

Afin de trouver la valeur θ^* 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

Algorithm 1 Descente de gradient

procedure GRADIENT($\theta, \mathcal{X}, \mathcal{Y}, I; \epsilon$)
  for $it = 1, I$ **do**
    for $i = 1, n$ **do**
      $idx \leftarrow random(\ell)$
      $\theta \leftarrow \theta - \epsilon \nabla_\theta \mathcal{L}(\theta)(idx)$
    end for
    Afficher $\mathcal{L}(\theta)$ et $accuracy(\theta)$ pour contrôle
  end for
  return θ
end procedure

Régularisation L_1

Dans l'idéal, 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 θ. On appelle cela la nomre L_0 :

$$L_0(\theta) = \sum_k \mathbb{1}_{\theta_k \neq 0} \tag{3}$$

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

La fonction indicatrice n'est pas une fonction dérivable, et nous ne pôuvons 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 :

$$\mathcal{L}(\theta) = \frac{1}{\ell} \sum_{i=1}^{\ell} (y^i - f_\theta(x^i))^2 + \lambda |\theta| \tag{4}$$

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

Travail demandé - individuel - deadline = 5 octobre

Le travail demandé **sera réalisé dans un IPython Notebook qui sera envoyé par email, ou directement accessible sur le Web**. Le notebook sera "joliment" commenté et executable 'tel quel'.

- Implémenter le modèle linéaire régularisé L_1 sous la forme d'une classe python (voir aide python)

Algorithm 2 Descente de gradient L_1

procedure GRADIENTL1($\theta, \mathcal{X}, \mathcal{Y}, I; \epsilon$)
  for $it = 1, I$ **do**
    for $i = 1, n$ **do**
      $idx \leftarrow random(\ell)$
      $\theta' \leftarrow \theta - \epsilon \nabla_\theta \mathcal{L}(\theta)(idx)$
      for j=1,n **do** ▷ Gradient Clipping
        if $\theta'_j * \theta_j < 0$ **then**
          $\theta_j \leftarrow 0$
        else
          $\theta_j \leftarrow \theta'_j$
        end if
      end for
    end for
    Afficher $\mathcal{L}(\theta)$ et $accuracy(\theta)$ pour contrôle
  end for
  return θ
end procedure

- Tester le modèle pour une valeur de $\lambda = 0$ afin de vérifier son bon fonctionnement
- Utiliser une méthode de type cross-validation afin de calculer la performance moyenne de ce modèle pour tout un ensemble de valeurs de λ
- Dessiner la courbe de performance (sur un ensemble de test) en fonction de **la sparsité** du modèle.
- Quelle est la meilleure valeur de λ ?
- **Extension :** Remplacez le terme $L_1(\theta)$ par un terme $L_2(\theta) = |-\theta||^2$ - que constatez vous ?
- **Extension :** Remplacez le terme $L_1(\theta)$ par un terme $\lambda_1 L_1(\theta) + \lambda_2 L_2(\theta)$ - que constatez vous ?
- **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

```
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 classifier est random il predit au hasard des 0 et
     """ des 1 sinon il utilise un svm pour predire"""

 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)



####################Main####

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")
```