TD-ADABOOST

Exercice 1 – Retour sur Adaboost

Cet exercice permet de retrouver les résultats proposés dans :
Robert E. Shapire. *Theoritical Views of Boosting*. EuroCOLT'99, 1999

1. Décrire succinctement l'algorithme d'Adaboost vu en cours.
2. Quel rôle joue la distribution D_t ?
3. Comment est obtenue l'hypothèse finale H ? A quoi correspondent les différents coefficients dans la formulation de H ?
4. Montrer que :
$$D_{final}(i) = \frac{1}{\ell \Pi_t Z_t} \exp(-y_i f(x_i)), \quad f(x_i) = \sum_t \alpha_t h_t(x_i)$$
5. Trouver un critère simple C_i permettant de détecter une erreur de classification sur x_i. Donner l'expression de l'erreur empirique moyenne en utilisant la fonction échelon. Tracer la valeur de la fonction échelon en fonction de ce critère.
6. Montrer ensuite que l'erreur empirique moyenne de H est bornée par :
$$\frac{1}{\ell} \sum_i \exp(-y_i f(x_i)) = \Pi_t Z_t$$
Vous pourrez dessiner cette fonction afin de trouver plus facilement la réponse.
7. $\varepsilon_t = P_{\tilde{D}_t}(h_t(x_i) \neq y_i)$. Dans le cas où on utilise l'estimation suivante : $\varepsilon_t = \sum_{i:h_t(x_i)\neq y_i} D_t(i)$, montrer que :
$$Z_t = 2\sqrt{\varepsilon_t(1-\varepsilon_t)}$$
8. Montrer que l'erreur empirique tend vers 0 lorsque t tend vers l'infini à condition que $\varepsilon < 0.5$.
9. Coder sur feuille une première version de l'algorithme AdaBoost.

Exercice 2 – Apprentissage du perceptron

On dispose d'une base de N exemples (observations), $\{\mathbf{x}_i\}_{i=1,\ldots,N}$, dont les classes sont connues ; la classe de $\mathbf{x}_i$ est notée y_i. On utilise l'algorithme suivant (une des variantes de l'algorithme du perceptron) pour apprendre automatiquement la valeur des paramètres, c-à-d du vecteur w : (ε est un nombre positif ; $\mathbf{x}^T$ est le transposé de $\mathbf{x}$)

Algorithme 1 : Algorithme d'apprentissage du perceptron

Entrée : $\{\mathbf{x}_i, y_i\}_{i=1,\ldots,N}$;
Initialisation de w : $w(1)$;
$t = 1$;
repeat
  Tirer aléatoirement un exemple : $\mathbf{x}_i$;
  if $y_i \times \mathbf{x}_i w(t) \leq 0$ **then**
    $w(t+1) \leftarrow w(t) + \varepsilon y_i \mathbf{x}_i^T$;
    $t = t + 1$;
until *(critère d'arrêt satisfait)* ;

Le critère d'arrêt peut être, par exemple qu'il n'y a pas eu d'erreur de classification pendant un certain nombre d'itérations successives.
Une alternative consiste à minimiser l'erreur au sens des moindre carré en utilisant par exemple l'algorithme ADALINE (suivant la règle dite de Widrow-Hoff)

Algorithme 2 : Algorithme ADALINE

Entrée : $\{\mathbf{x}_i, y_i\}_{i=1,\dots,N}$;
Initialisation de w : $w(1)$;
$t = 1$;
repeat
  for $i = 1, \dots, N$ **do**
    $w(t+1) \leftarrow w(t) - \varepsilon \left(\mathbf{x}_i w(t) - y_i\right) \mathbf{x}_i^T$;
    $t = t + 1$;
until *(critère d'arrêt satisfait)* ;

Exercice 3 – TP : Application du boosting sur des exemples jouets

On propose dans un premier temps d'appliquer le boosting sur des données 2D dans un environnement octave, afin de bien maîtriser l'algorithme avant de s'attaquer à des données plus volumineuses.
Télécharger les fichiers utiles (génération des données, squelette de programme principal...) sur le site de l'UE.

Q 3.1 Génération / affichage des données
Tous les éléments sont disponibles dans le squelette qui vous est fourni sur le site de l'UE :

```
%%%%%%%%%%%%%%%%%%%%%%%%%%
% CONSTRUCTION DES DONNEES
N     = 200; % nombre de points
Ntest = 0; % pas utilise ici
sigma = 1; % bruit dans les donnees
[X,Y] = dataset("clowns",N, Ntest, sigma);
```

Q 3.2 Perceptron
Dans le fichier `perceptron.m`, écrire la fonction qui optimise un classifieur linéaire selon l'algorithme du perceptron. Vous vous conformerez à la signature suivante :

```
function [W]=perceptron(X,Y,eps,maxIter)
% X: donnees
% Y: etiquettes
% eps : pas de mise à jour
% maxIter: nombre d'itérations
```

Rappels : l'initialisation est aléatoire (`rand`).
Afin de construire un perceptron plus efficace, vous n'oublierez pas d'ajouter un biais, c'est à dire une colonne de 1 dans la matrice X.

Q 3.3 Affichage d'une frontière de décision
Vous utiliserez les fonctions `generateGrid` et `plotFrontiere` en vous inspirant de l'exemple suivant pour tracer la frontière de décision. La première fonction permet de générer une grille de point couvrant tout l'espace, il vous est ensuite demander d'évaluer votre modèle sur tous les points de la grille. Une

fois cette opération effectuée, la méthode plotFrontiere permet d'interpoler et de tracer la position de la frontière de décision.
NB : afin de ne pas avoir de problème, il faut utiliser la version 4.2 de gnuplot (ou ultérieure). Cette version n'est pas la version par défaut sur les machines de l'ARI. Il faut ajouter la ligne de code suivante au début de votre programme :

```
gnuplot_binary("/usr/local/bin/gnuplot"); % bonne version de gnuplot
```

```
% xapp, k existent...
ngrid = 30; % resolution de la grille
xgrid = generateGrid(xapp, ngrid);
ygrid = modele(xgrid); % determiner la classe des xgrid en utilisant un modele
plotFrontiere(xgrid, ygrid);
```

Q 3.4 Boosting
Une fois que le perceptron est opérationnel, nous allons introduire le boosting. Vous développerez la fonction `boosting.m` de signature suivante (paramètres du perceptron et du boosting) :

```
function [TabAlpha, TabW, ErrEps]=boosting(X,Y,eps,maxIter, T)
```