Aide à la décision - R5.A.11

Méthodes d'optimisation pour l'aide à la décision

Un perceptron linéaire pour classifier

Apprentissage par descente de gradient

Michel Salomon

IUT Nord Franche-Comté
Département d'informatique

Un perceptron (ou neurone) linéaire pour classifier

- Considérons un problème défini par :
  - deux entrées (x_1 et x_2)
  - une sortie (y^t)
- La fonction P_W calculée par le perceptron vérifie :
  - $P_W : \mathbb{R}^2 \to \mathbb{R}$
  - $P_W\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = f\left(w_0 \cdot x_0 + w_1 \cdot x_1 + w_2 \cdot x_2\right) = y$

  avec $x_0 = +1$ la valeur de biais
- Quand f est la fonction identité $f(x) = x$ (neurone linéaire)
  - $P_W\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = w_0 \cdot 1 + w_1 \cdot x_1 + w_2 \cdot x_2 = w_0 + \sum_{i=1}^{2} w_i \cdot x_i$

Apprentissage ou entraînement du perceptron linéaire

- Cas supervisé $\rightarrow$ un jeu de données d'entraînement S
  - n observations qui sont des paires d'entrées-sortie
  - $S = \{(X_1, Y_1^t), \ldots, (X_n, Y_n^t)\}$, avec $\left(X_j, Y_j^t\right)$ qui est une paire
- Apprentissage supervisé $\rightarrow$ un problème d'optimisation
  - Erreur / fonction de perte L (*loss*) $\rightarrow$ guide l'apprentissage

$$\min_W \quad \left[\frac{1}{n}\sum_{j=1}^{n} L\left(P_W(X_j), Y_j^t\right)\right]$$

$$\min_w \quad \left[\frac{1}{n}\sum_{j=1}^{n} L\left(y_j, y_j^t\right)\right]$$

  - Erreur quadratique moyenne $\rightarrow L(y, y^t) = \frac{1}{2}(y - y^t)^2$
  - Pour un problème avec deux entrées et une sortie $\left(X_j, Y_j^t\right) = \left(\left(x_{1j}, x_{2j}\right), y_j^t\right)$ la fonction objectif à minimiser est

$$\min_w \left[\frac{1}{n}\sum_{j=1}^{n}\frac{1}{2}\left(\left(w_0 + \sum_{i=1}^{2} w_i \cdot x_{ij}\right) - y_j^t\right)^2\right]$$

Apprentissage ou entraînement du perceptron linéaire - 1/2

- Règles de mise à jour / correction des poids synaptiques

$$w_i = w_i - \gamma \cdot \frac{\partial L}{\partial w_i}$$

où γ est le pas d'apprentissage (*learning rate*)

- Fonction de perte (ou de coût - *loss function*) composée
  - $L(y_j, y_j^t) = \frac{1}{2} {l_j}^2$
  - $l_j = y_j - y_j^t$
  - $y_j = f(v_j) = v_j$ (f la fonction d'activation $\rightarrow$ identité)
  - $v_j = w_0 + \sum_{i=1}^{2} w_i \cdot x_{ij}$

Apprentissage ou entraînement du perceptron linéaire - 2/2

- Calcul du gradient $\nabla L(y_j, y_j^t) = \left(\frac{\partial L}{\partial w_0}, \frac{\partial L}{\partial w_1}, \frac{\partial L}{\partial w_2}\right)^T$
- Calcul de chaque composante $\frac{\partial L(y_j, y_j^t)}{\partial w_i}$ en utilisant la règle de dérivation en chaîne

$$\frac{\partial L(y_j, y_j^t)}{\partial w_i} = \frac{\partial L}{\partial l_j} \cdot \frac{\partial l_j}{\partial y_j} \cdot \frac{\partial y_j}{\partial v_j} \cdot \frac{\partial v_j}{\partial w_i}$$

où

- $\frac{\partial L}{\partial l_j} = l_j$; $\frac{\partial l_j}{\partial y_j} = 1$; $\frac{\partial y_j}{\partial v_j} = 1$
- $v_j = w_0 + \sum_{i=1}^{2} w_i \cdot x_{ij}$

Mise en œuvre pratique - 1/2

- Considérons un problème de classification binaire
  - Deux classes (labels) $\rightarrow y_j^t \in \{0, 1\}$
  - Générer un jeu de données (`make_blobs` ; `make_moons`)
    - Utiliser le module `datasets` de `sklearn`
    - Consulter `Sample generators`
  - Afficher le jeu de données avec `mathplotlib`
- Écrire une fonction `predict` $\rightarrow$ calcule la sortie du percep.
  - *Entrée* $\rightarrow$ une observation du jeu de données ; les poids
  - *Sortie* $\rightarrow$ la classe de l'entrée (label) y_j
    - 0 si $P_W(X_j)$ est négatif
    - 1 si $P_W(X_j)$ est positif ou nul
- Écrire une fonction `training` $\rightarrow$ entraîne le perceptron
  - *Entrées* $\rightarrow$ jeu de données ; pas γ ; nombre d'époques
    (2 boucles $\rightarrow$ *nb époques* et *données* ; mise à jour w_i pour $\left(X_j, Y_j^t\right)$)
  - *Sortie* $\rightarrow$ poids obtenus après l'apprentissage

Mise en œuvre pratique - 2/2

- Écrire une fonction `accuracy` $\rightarrow$ précision des prédictions
  - *Entrées* $\rightarrow$ jeu de données ; poids synaptiques
  - *Sortie* $\rightarrow$ pourcentage d'observations bien classifiées
- Écrire une fonction `crossValid` $\rightarrow$ cross-validation
  - *Entrées* $\rightarrow$ jeu de données ; nombre de blocs (folds)
    (Partitionner données en n sous-ensembles disjoints - *k-fold* with $k = n$)
    - Utiliser le module `model_selection` de `sklearn`

      pas d'apprentissage γ ; nombre d'époques
  - *Sortie* $\rightarrow$ vecteur contenant les poids du meilleur perceptron
- Combiner les fonctions pour évaluer les performances d'un perceptron via la validation croisée sur le jeu de données
- Il existe différentes variantes de la descente de gradient
  - *Batch GD*, *SGD*, *Mini-batch GD*

Erreur quadratique et neurone linéaire $\rightarrow$ problème de régression