Classification supervisée

Rémy Degenne

12 janvier 2022

Ces notes de cours contiennent uniquement un bref résumé de ce qui a été dit en cours. Reportez vous à vos notes pour plus de détails.

1 Cours 2: le neurone formel et le perceptron

1.1 Le neurone formel

Un neurone biologique possède des dendrites, un noyau et un axone. Les dendrites reçoivent des excitations électriques. Ces nombreuses sources d'excitation sont agrégées, et un potentiel électrique est créé dans l'axone, ou pas. L'axone est connecté aux dendrites d'autres neurones et transmet ainsi une excitation électrique à son tour.

Attention: ceci est une présentation très (trop) simplifiée du fonctionnement d'un neurone biologique.

Le neurone formel est une approximation très grossière du neurone biologique.

- Il possède $P+1$ entrées $(e_i)_{i\in\{0,\dots,P\}}$. L'entrée e_0 est appelée le biais et vaut toujours 1.
- Il possède une sortie $s \in \mathbb{R}$.
- Il est paramétré par $P+1$ poids $(w_i)_{i\in\{1,\dots,P\}} \in \mathbb{R}^{P+1}$.
- Une fonction, dite d'*activation* et notée ϕ, détermine la valeur de s en fonction des poids et entrées:

$$s = \phi\Big(\sum_{i=1}^{P+1} w_i e_i\Big)\,. \tag{1}$$

Fonctions possibles pour ϕ:

- identité (neurone linéaire): $\phi(x) = x$.
- fonction de Heavyside: 1 si $x \geq 0$, 0 (or -1) sinon.
- logistique: $\phi(x) = \frac{1}{1+e^{-ax}}$.
- tangente hyperbolique: $\phi(x) = \tanh(x) \in [-1, 1]$.
- ReLU: $\phi(x) = \max(0, x)$.

Les fonctions autres que Heavyside sont continues et dérivable, ce qui sera important pour l'entrainement des classeurs utilisant des neurones.

Un classeur: le perceptron Le perceptron est un classeur constitué d'un seul neurone. Sa sortie s est utilisée pour prédire l'étiquette. Ses P entrées autres que le biais prennent la valeur des attributs de la donnée.

Pour que s puisse être une classe, il faut que $s \in \{0, 1\}$ or $s \in \{-1, 1\}$. On peut choisir n'importe quel fonction d'activation ϕ, mais il faudra la composer avec un seuil si elle a des valeurs dans $\mathbb{R}$.

- Construction: à l'aide des exemples, trouver des poids qui permettent de bien classer ces exemples. Processus itératif d'optimisation.
- Evaluation: calcul de s à partir d'une nouvelle donnée. $s = \phi(\sum_{i=1}^{P+1} w_i e_i)$.

Interprétation géométrique du neurone linéaire (où ϕ est l'identité): l'équation $s = \sum_{i=1}^{P+1} w_i e_i$ définit un hyperplan dans l'espace $\mathbb{R}^{P+1}$. On classe les données en fonction du côté de l'hyperplan où elles se trouvent.

S'il est possible de trouver un hyperplan qui classe les exemples correctement, on dit qu'elles sont linéairement séparables. Souvent, ce n'est pas le cas. On a alors peu de chance d'obtenir un bon classeur.

1.2 Apprendre les poids

Construction du perceptron = trouver de bons poids.

Initialisation: poids aléatoires, par exemple Gaussiens centrés en zéro. Question: quelle variance? Peut avoir une grande influence sur le processus d'apprentissage.

Pour trouver les poids on utilise une méthode itérative, avec plusieurs *épisodes*.

Première méthode: descente de gradient On choisit une fonction de perte $\ell(s, y) \geq 0$ et la perte totale de l'algorithme sur les exemples $\mathcal{X}$ est

$$L(w, \mathcal{X}) = \sum_{i=1}^{N} \ell(s_i, y_i)$$

où s_i est la sortie du neurone quand l'entrée est fixée à x_i.

La fonction de perte $w \mapsto L(w, \mathcal{X})$ est une fonction réelle de $P+1$ variables. On peut l'optimiser, c'est à dire chercher un minimum. Les paramètres w qui réalisent le minimum sont ceux qui permettent de mieux classer $\mathcal{X}$.

La méthode d'optimisation que l'on va utiliser est la descente de gradient.

- Entrée: w aléatoires; $\alpha \geq 0$; $\mathcal{X}$, ensemble de N données x_i avec P attributs $x_{i,j}$ et une classe y_i. Pour chaque donnée, $x_{i,0}$ est pris égal à 1.
- Entrée: un seuil $\varepsilon > 0$.
- Initialiser $E = +\infty$
- Tant que $E > \varepsilon$,
  - calculer $\nabla_w L(w, \mathcal{X})$ où $L(w, \mathcal{X}) = \sum_{i=1}^{N} \ell(s_i, y_i)$

  - calculer $E = \|\nabla_w L(w, \mathcal{X})\|$
  - mettre à jour $w \leftarrow w - \alpha \nabla_w L(w, \mathcal{X})$

Le gradient est $\nabla_x \ell(s_i, y_i) = (\frac{\partial \ell}{\partial w_j})_{j \in \{0,\dots,P\}}$

Exemple de perte: $\ell(s_i, y_i) = \frac{1}{2}(s_i - y_i)^2 = \frac{1}{2}(y_i - \phi(\sum_{j=0}^{P} w_j x_{i,j}))$.

Gradient pour l'exemple: $\frac{\partial \ell}{\partial w_j}(s_i, y_i) = (\phi(\sum_{j=0}^{P} w_j x_{i,j}) - y_i)\phi'(\sum_{j=0}^{P} w_j x_{i,j})(-x_{i,j})$.

En pratique: calculer le gradient est une tâche qui peut être réalisée par un algorithme.

Inconvénient de cette méthode d'optimisation: il faut calculer la perte de tous les exemples pour pouvoir faire un pas de gradient (une modification de w). C'est coûteux. Si on a un très grand nombre d'exemples, on peut imaginer qu'un sous-ensemble plus petit des exemples pourrait suffire à avoir une bonne estimation du gradient.

gradient stochastique On n'utilise pas tous les exemples pour calculer le gradient, mais juste un. On obtient beaucoup plus de mises à jour des poids à moindre coût, mais chaque mise à jour est une moins bonne estimation du gradient.

- Entrées: w, aléatoires; $\alpha \geq 0$; $\mathcal{X}$, ensemble de N données x_i avec P attributs $x_{i,j}$ et une classe y_i. Pour chaque donnée, $x_{i,0}$ est pris égal à 1.
- Entrée: un seuil $\varepsilon > 0$.
- Initialiser $E = +\infty$
- Tant que $E > \varepsilon$,
  - mélanger les données
  - $E \leftarrow 0$
  - Pour chaque donnée x_i:
    - calculer $\nabla_w \ell(s_i, y_i)$
    - mettre à jour $w \leftarrow w - \alpha \nabla_w \ell(s_i, y_i)$
    - $E \leftarrow E + \|\nabla \ell(s_i, y_i)\|/N$

Propriété : cette règle d'apprentissage converge asymptotiquement. C'est à dire que si on utilise assez d'épisodes les poids se stabilisent à une valeur.

Remarque : ajouter ou enlever un exemple dans $\mathcal{X}$ peut changer beaucoup la valeur limite de w.

1.3 Perceptron multi-couche: le premier réseau de neurones

Plusieurs neurones sont organisés par couches. Exemple pour la classification binaire:

- Premier niveau: $P + 1$ entrées correspondant aux attributs. N_1 neurones avec autant de sorties.
- Deuxième niveau: $N_1 + 1$ entrées (il y a toujours un biais), N_2 sorties.

- Troisième niveau: un neurone avec $N_2 + 1$ entrées et une sortie.

On n'est pas limité à trois niveaux mais on peut en empiler autant qu'on veut.

Optimisation: gradient stochastique comme pour le perceptron mais il faut calculer un gradient plus compliqué. Ce calcul peut être fait automatiquement, par un algorithme appelé "backpropagation".

On verra plus de détails sur les réseaux de neurones dans les cours suivants.