Pense-bête VIP : Modèles basés sur le réflex

Afshine Amidi et Shervine Amidi

8 septembre 2019

Prédicteurs linéaires

Dans cette section, nous allons explorer les modèles basés sur le réflex qui peuvent s'améliorer avec l'expérience s'appuyant sur des données ayant une correspondance entrée-sortie.

❒ **Vecteur caractéristique** – Le vecteur caractéristique (en anglais *feature vector*) d'une entrée x est noté $\phi(x)$ et se décompose en :

$$\boxed{\phi(x) = \begin{bmatrix} \phi_1(x) \\ \vdots \\ \phi_d(x) \end{bmatrix} \in \mathbb{R}^d}$$

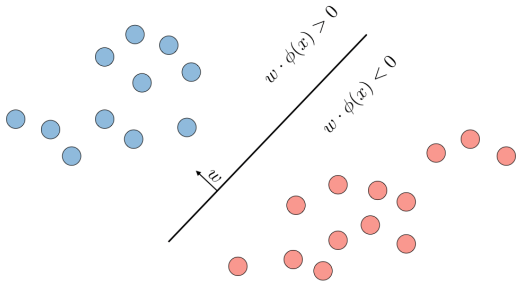
❒ **Score** – Le score $s(x,w)$ d'un exemple $(\phi(x),y) \in \mathbb{R}^d \times \mathbb{R}$ associé à un modèle linéaire de paramètres $w \in \mathbb{R}^d$ est donné par le produit scalaire :

$$\boxed{s(x,w) = w \cdot \phi(x)}$$

Classification

❒ **Classifieur linéaire** – Étant donnés un vecteur de paramètres $w \in \mathbb{R}^d$ et un vecteur caractéristique $\phi(x) \in \mathbb{R}^d$, le classifieur linéaire binaire f_w est donné par :

$$\boxed{f_w(x) = \text{sign}(s(x,w)) = \begin{cases} +1 & \text{if } w \cdot \phi(x) > 0 \\ -1 & \text{if } w \cdot \phi(x) < 0 \\ ? & \text{if } w \cdot \phi(x) = 0 \end{cases}}$$



❒ **Marge** – La marge (en anglais *margin*) $m(x,y,w) \in \mathbb{R}$ d'un exemple $(\phi(x),y) \in \mathbb{R}^d \times \{-1,+1\}$ associée à un modèle linéaire de paramètre $w \in \mathbb{R}^d$ quantifie la confiance associée à une prédiction : plus cette valeur est grande, mieux c'est. Cette quantité est donnée par :

$$\boxed{m(x,y,w) = s(x,w) \times y}$$

Régression

❒ **Régression linéaire** – Étant donnés un vecteur de paramètres $w \in \mathbb{R}^d$ et un vecteur caractéristique $\phi(x) \in \mathbb{R}^d$, le résultat d'une régression linéaire de paramètre w, notée f_w, est donné par :

$$\boxed{f_w(x) = s(x,w)}$$

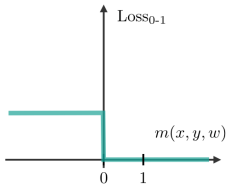
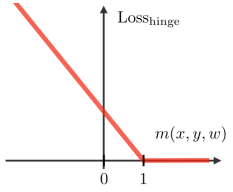
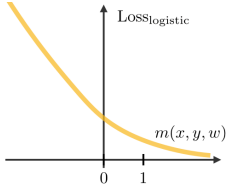
❒ **Résidu** – Le résidu $\text{res}(x,y,w) \in \mathbb{R}$ est défini comme étant la différence entre la prédiction $f_w(x)$ et la vraie valeur y :

$$\boxed{\text{res}(x,y,w) = f_w(x) - y}$$

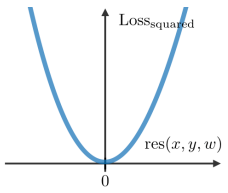
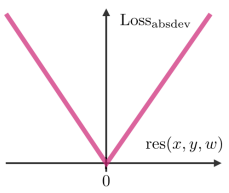
Minimisation de la fonction objectif

❒ **Fonction objectif** – Une fonction objectif (en anglais *loss function*) $\text{Loss}(x,y,w)$ traduit notre niveau d'insatisfaction avec les paramètres w du modèle dans la tâche de prédiction de la sortie y à partir de l'entrée x. C'est une quantité que l'on souhaite minimiser pendant la phase d'entraînement.

❒ **Cas de la classification** – Trouver la classe d'un exemple x appartenant à $y \in \{-1,+1\}$ peut être faite par le biais d'un modèle linéaire de paramètre w à l'aide du prédicteur $f_w(x) \triangleq \text{sign}(s(x,w))$. La qualité de cette prédiction peut alors être évaluée au travers de la marge $m(x,y,w)$ intervenant dans les fonctions objectif suivantes :

Fonction objectif	Zéro-un	Hinge	Logistique
$\text{Loss}(x,y,w)$	$1_{\{m(x,y,w) \leqslant 0\}}$	$\max(1 - m(x,y,w), 0)$	$\log(1 + e^{-m(x,y,w)})$
Illustration			

❒ **Cas de la régression** – Prédire la valeur $y \in \mathbb{R}$ associée à l'exemple x peut être faite par le biais d'un modèle linéaire de paramètre w à l'aide du prédicteur $f_w(x) \triangleq s(x,w)$. La qualité de cette prédiction peut alors être évaluée au travers du résidu $\text{res}(x,y,w)$ intervenant dans les fonctions objectif suivantes :

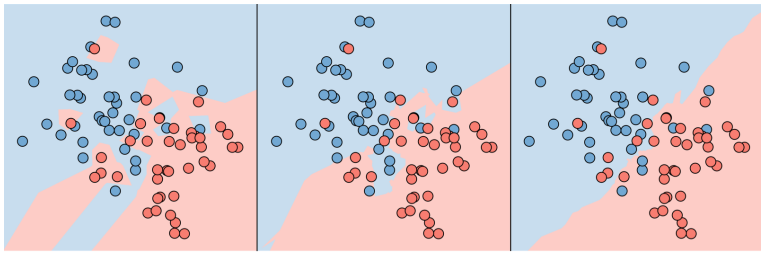
Nom	Erreur quadratique	Erreur absolue
Loss(x,y,w)	$(\text{res}(x,y,w))^2$	$\lvert\text{res}(x,y,w)\rvert$
Illustration		

❒ **Processus de minimisation de la fonction objectif** – Lors de l'entraînement d'un modèle, on souhaite minimiser la valeur de la fonction objectif évaluée sur l'ensemble d'entraînement :

$$\boxed{\text{TrainLoss}(w) = \frac{1}{|\mathcal{D}_{\text{train}}|} \sum_{(x,y)\in\mathcal{D}_{\text{train}}} \text{Loss}(x,y,w)}$$

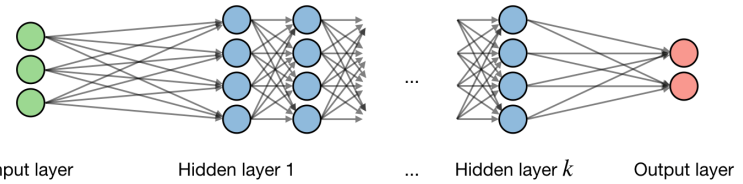
Prédicteurs non linéaires

❒ **k plus proches voisins** – L'algorithme des k plus proches voisins (en anglais *k-nearest neighbors* ou *k-NN*) est une approche non paramétrique où la réponse associée à un exemple est déterminée par la nature de ses k plus proches voisins de l'ensemble d'entraînement. Cette démarche peut être utilisée pour la classification et la régression.



Remarque : plus le paramètre k est grand, plus le biais est élevé. À l'inverse, la variance devient plus élevée lorsque l'on réduit la valeur k.

❒ **Réseaux de neurones** – Les réseaux de neurones (en anglais *neural networks*) constituent un type de modèle basés sur des couches (en anglais *layers*). Parmi les types de réseaux populaires, on peut compter les réseaux de neurones convolutionnels et récurrents (abbréviés respectivement en CNN et RNN *en anglais*). Une partie du vocabulaire associé aux réseaux de neurones est détaillée dans la figure ci-dessous :



En notant i la i-ème couche du réseau et j son j-ième neurone, on a :

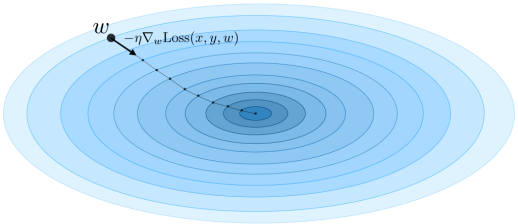
$$\boxed{z_j^{[i]} = {w_j^{[i]}}^T x + b_j^{[i]}}$$

où l'on note w, b, x, z le coefficient, le biais ainsi que la variable de sortie respectivement.

Algorithme du gradient stochastique

❒ **Descente de gradient** – En notant $\eta \in \mathbb{R}$ le taux d'apprentissage (en anglais learning rate ou step size), la règle de mise à jour des coefficients pour cet algorithme utilise la fonction objectif Loss(x,y,w) de la manière suivante :

$$\boxed{w \longleftarrow w - \eta\nabla_w \text{Loss}(x,y,w)}$$



❒ **Mises à jour stochastiques** – L'algorithme du gradient stochastique (en anglais *stochastic gradient descent* ou *SGD*) met à jour les paramètres du modèle en parcourant les exemples $(\phi(x),y) \in \mathcal{D}_{\text{train}}$ de l'ensemble d'entraînement un à un. Cette méthode engendre des mises à jour rapides à calculer mais qui manquent parfois de robustesse.

❒ **Mises à jour par lot** – L'algorithme du gradient par lot (en anglais *batch gradient descent* ou *BGD*) met à jour les paramètre du modèle en utilisant des lots entiers d'exemples (e.g. la totalité de l'ensemble d'entraînement) à la fois. Cette méthode calcule des directions de mise à jour des coefficients plus stable au prix d'un plus grand nombre de calculs.

Peaufinage de modèle

❒ **Classe d'hypothèses** – Une classe d'hypothèses $\mathcal{F}$ est l'ensemble des prédicteurs candidats ayant un $\phi(x)$ fixé et dont le paramètre w peut varier :

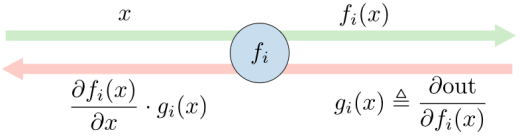
$$\boxed{\mathcal{F} = \left\{f_w : w \in \mathbb{R}^d\right\}}$$

❒ **Fonction logistique** – La fonction logistique σ, aussi appelée en anglais *sigmoid function*, est définie par :

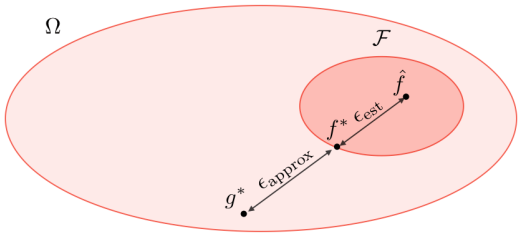
$$\forall z \in]-\infty, +\infty[, \quad \sigma(z) = \frac{1}{1+e^{-z}}$$

Remarque : la dérivée de cette fonction s'écrit $\sigma'(z) = \sigma(z)(1-\sigma(z))$.

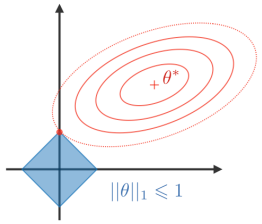
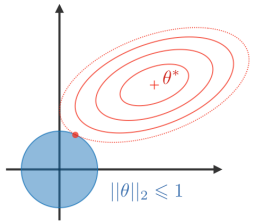
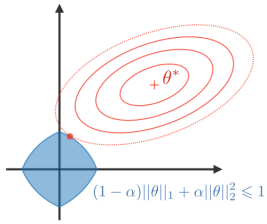
❒ **Rétropropagation du gradient (en anglais *backpropagation*)** – La propagation avant (en anglais *forward pass*) est effectuée via f_i, valeur correspondant à l'expression appliquée à l'étape i. La propagation de l'erreur vers l'arrière (en anglais *backward pass*) se fait via $g_i = \frac{\partial \text{out}}{\partial f_i}$ et décrit la manière dont f_i agit sur la sortie du réseau.



❒ **Erreur d'approximation et d'estimation** – L'erreur d'approximation ϵ_{approx} représente la distance entre la classe d'hypothèses $\mathcal{F}$ et le prédicteur optimal g^*. De son côté, l'erreur d'estimation ϵ_{est} quantifie la qualité du prédicteur $\hat{f}$ par rapport au meilleur prédicteur f^* de la classe d'hypothèses $\mathcal{F}$.



❒ **Régularisation** – Le but de la régularisation est d'empêcher le modèle de surapprendre (en anglais *overfit*) les données en s'occupant ainsi des problèmes de variance élevée. La table suivante résume les différents types de régularisation couramment utilisés :

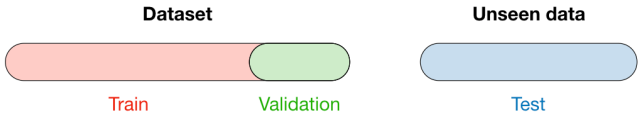
LASSO	**Ridge**	**Elastic Net**
- Réduit les coefficients à 0 - Bénéfique pour la sélection de variables	Rapetissit les coefficients	Compromis entre sélection de variables et coefficients de faible magnitude
		
$\ldots + \lambda\|\|\theta\|\|_1$ $\lambda \in \mathbb{R}$	$\ldots + \lambda\|\|\theta\|\|_2^2$ $\lambda \in \mathbb{R}$	$\ldots + \lambda\left[(1-\alpha)\|\|\theta\|\|_1 + \alpha\|\|\theta\|\|_2^2\right]$ $\lambda \in \mathbb{R}, \quad \alpha \in [0,1]$

❒ **Hyperparamètres** – Les hyperparamètres sont les paramètres de l'algorithme d'apprentissage et incluent parmi d'autres le type de caractéristiques utilisé ainsi que le paramètre de régularisation λ, le nombre d'itérations T le taux d'apprentissage η.

❒ **Vocabulaire** – Lors de la sélection d'un modèle, on divise les données en 3 différentes parties :

Données d'entraînement	**Données de validation**	**Données de test**
- Le modèle y est entrainé - Constitue normalement 80% du jeu de données	- Le modèle y est évalué - Constitue normalement 20% du jeu de données - Aussi appelé données de développement (en anglais *hold-out* ou *development set*)	- Le modèle y donne ses prédictions - Données jamais observées

Une fois que le modèle a été choisi, il est entrainé sur le jeu de données entier et testé sur l'ensemble de test (qui n'a jamais été vu). Ces derniers sont représentés dans la figure ci-dessous :



Apprentissage non supervisé

Les méthodes d'apprentissage non supervisé visent à découvrir la structure (parfois riche) des données.

k-moyennes (en anglais k-*means*)

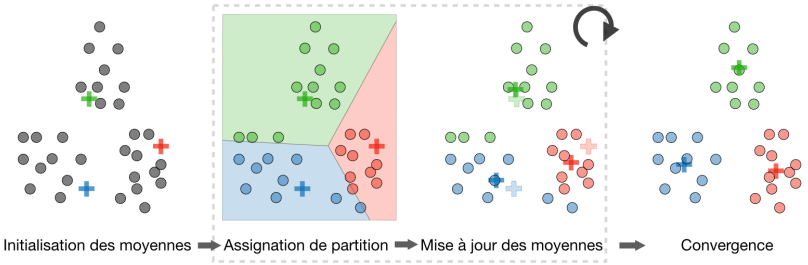
❒ **Partitionnement** – Étant donné un ensemble d'entraînement $\mathcal{D}_{\text{train}}$, le but d'un algorithme de partitionnement (en anglais *clustering*) est d'assigner chaque point $\phi(x_i)$ à une partition $z_i \in \{1,...,k\}$.

❒ **Fonction objectif** – La fonction objectif d'un des principaux algorithmes de partitionnement, k-moyennes, est donné par :

$$\boxed{\text{Loss}_{\text{k-means}}(x,\mu) = \sum_{i=1}^{n} ||\phi(x_i) - \mu_{z_i}||^2}$$

❒ **Algorithme** – Après avoir aléatoirement initialisé les centroïdes de partitions $\mu_1,\mu_2,...,\mu_k \in \mathbb{R}^n$, l'algorithme k-moyennes répète l'étape suivante jusqu'à convergence :

$$\boxed{z_i = \arg\min_j ||\phi(x_i) - \mu_j||^2} \quad \text{and} \quad \boxed{\mu_j = \frac{\sum_{i=1}^{m} 1_{\{z_i=j\}}\phi(x_i)}{\sum_{i=1}^{m} 1_{\{z_i=j\}}}}$$



Analyse des composantes principales

❒ **Valeur propre, vecteur propre** – Étant donnée une matrice $A \in \mathbb{R}^{n\times n}$, λ est dite être une valeur propre de A s'il existe un vecteur $z \in \mathbb{R}^n \backslash \{0\}$, appelé vecteur propre, tel que :

$$\boxed{Az = \lambda z}$$

❒ **Théorème spectral** – Soit $A \in \mathbb{R}^{n\times n}$. Si A est symétrique, alors A est diagonalisable par une matrice réelle orthogonale $U \in \mathbb{R}^{n\times n}$. En notant $\Lambda = \text{diag}(\lambda_1,...,\lambda_n)$, on a :

$$\boxed{\exists \Lambda \text{ diagonal}, \quad A = U\Lambda U^T}$$

Remarque : le vecteur propre associé à la plus grande valeur propre est appelé le vecteur propre principal de la matrice A.

❒ **Algorithme** – La procédure d'analyse des composantes principales (en anglais *Principal Component Analysis* ou *PCA*) est une technique de réduction de dimension qui projette les données sur k dimensions en maximisant la variance des données de la manière suivante :

— Étape 1 : Normaliser les données pour avoir une moyenne de 0 et un écart-type de 1.

$$\boxed{x_j^{(i)} \leftarrow \frac{x_j^{(i)} - \mu_j}{\sigma_j}} \quad \text{where} \quad \boxed{\mu_j = \frac{1}{m}\sum_{i=1}^{m} x_j^{(i)}} \quad \text{and} \quad \boxed{\sigma_j^2 = \frac{1}{m}\sum_{i=1}^{m}(x_j^{(i)} - \mu_j)^2}$$

— Étape 2 : Calculer $\Sigma = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}x^{(i)^T} \in \mathbb{R}^{n\times n}$, qui est symétrique avec des valeurs propres réelles.

— Étape 3 : Calculer $u_1,...,u_k \in \mathbb{R}^n$ les k valeurs propres principales orthogonales de Σ, i.e. les vecteurs propres orthogonaux des k valeurs propres les plus grandes.

— Étape 4 : Projeter les données sur $\text{span}_{\mathbb{R}}(u_1,...,u_k)$. Cette procédure maximise la variance sur tous les espaces à k dimensions.