DATA MINING 2

Réseaux de Neurones, Mélanges de classifieurs, SVM avancé

I. Réseau Artificiel de Neurones

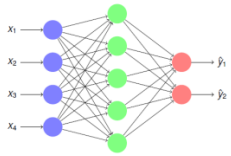
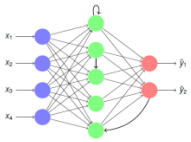
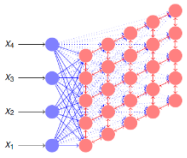
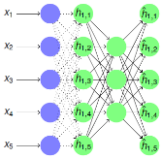
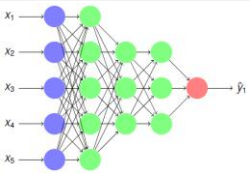
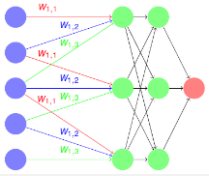
1. Neurone

$$\boxed{\hat{y} = f(w^\top x + b)}$$

Exemples de choix pour la fonction f :

- $\text{sigm}(x) \in [0,1]$
- $\tanh(x) \in [-1;1]$
- $\text{softmax}(x) \in [-1;1]$ suivant une loi de probabilité

2. Type de réseaux

Feedforward	*Récurrents*	*Carte auto-organisatrice*	*Auto-associateur*	*Réseaux profonds*	*Réseau convolutionnel*
Couches successives	Boucles de rétroaction	Neurones reliés	Objectif : réduction de dimension	Réseaux à multiples couches.	Poids liés, forme un filtre de convolution
					
MultiLayer Perceptron *Radial Basis Function Network*	*Time Delay Neural Networks* *NARMAX, Jordan, Elman, …*	*Self-Organized Map* *Carte de Cohonen*			

Les réseaux profonds nécessitent qu'on initialise les poids pour stabiliser les calculs. Pour cela, on pré-apprend les couches basses (vertes) avec des auto-associateurs (cf ex d'auto-associateur).

3. Perceptron Multi-Couches

a. Définition d'une couche

Pour une couche, on a n entrées notées i, p sorties notées j.

$$I = \begin{bmatrix} I_0 = 1 \\ I_1 \\ \vdots \\ I_i \\ \vdots \\ I_n \end{bmatrix}$$

Entrée

$$W = \begin{bmatrix} w_1 & \dots & w_j & \dots & w_p \end{bmatrix}$$

$$w_j = [w_{0j} \quad \dots \quad w_{nj}]^\top$$

$$\widetilde{W} = W(1:n, 1:p)$$

Poids

$$S = \begin{bmatrix} S_1 \\ \vdots \\ S_j = w_j^\top I \\ \vdots \\ S_p \end{bmatrix} = W^\top I$$

Sommage

$$O = \begin{bmatrix} O_1 \\ \vdots \\ O_j = f(S_j) \\ \vdots \\ O_p \end{bmatrix} = f(S)$$

Sortie

b. Principe pour une couche

Sortie

$$\boxed{ErrO_j = \frac{\partial L}{\partial O_j}}$$

$$ErrO = \nabla_O L$$

Paramètres

$$\boxed{\frac{\partial L}{\partial W_{ij}} = \underbrace{\frac{\partial L}{\partial O_j}}_{ErrO_j} \underbrace{\frac{\partial O_j}{\partial S_j}}_{f'(S_j)} \underbrace{\frac{\partial S_j}{\partial W_{ij}}}_{I_i}}$$

$$\Delta W = I \times \left(ErrO \cdot f'(S)\right)^\top$$

Entrée

$$\boxed{ErrI_i = \frac{\partial L}{\partial I_i} = \sum_j \underbrace{\frac{\partial L}{O_j}}_{ErrO_j} \underbrace{\frac{\partial O_j}{\partial S_j}}_{f'(S_j)} \underbrace{\frac{\partial S_j}{\partial I_i}}_{W_{ij}}}$$

$$ErrI = \widetilde{W} \times \left(ErrO \cdot f'(S)\right)$$

c. Calcul pour m couches et rétropropagation

La technique consiste à choisir une fonction L pour la sortie finale (ex : $L(O,Y) = \|O - Y\|^2$ où Y est l'objectif de sortie) et de calculer $\boxed{ErrO_j^{(m)} = \frac{\partial L}{\partial o_j^{(m)}}}$.

Pour les autres couches ($l < m$), on applique la rétropropagation du gradient, soit $\boxed{ErrO_i^{(l)} = ErrI_i^{(l+1)}}$

II. Mélange de classifieurs

1. Généralités sur les mélanges

Principe Consiste à combiner les décisions de plusieurs classifieurs.

Limitations d'un classifieur unique

Limitation	*Problème du classifieur seul*	*Avantage du mélange*
Statistique (variance)	Il existe plusieurs classifieurs avec les mêmes performances en test.	Moyenner plusieurs classifieurs aussi performants.
Représentation (biais)	On peut créer un nombre limité de classifieurs.	Obtenir un classifieur qu'on ne pourrait pas construire directement.
Computationnelle	On tombe souvent dans des minimas locaux.	S'approcher du minimum global.

Types de mélanges Hétérogènes (classifieurs de types ≠) / Homogènes (classifieurs de même type)

Approches Injecter de l'aléatoire / Manipuler les données (apprentissage, entrée, sortie) / …

2. Bootstrap

Méthode de ré-échantillonnage permettant d'estimer une grandeur $s(X)$ d'un jeu initial X de n valeurs en créant B jeux de données $\tilde{X}_k$ de m valeurs par tirage aléatoire avec remise sur le jeu.

$$X = \{x_1, \dots, x_i, \dots x_n\} \quad \rightarrow \quad \begin{matrix} \tilde{X}_1 = \{x_4, x_6, x_4, x_1, x_3, \dots\} \\ \vdots \\ \tilde{X}_B = \{x_2, x_1, x_n, x_2, x_2, \dots\} \end{matrix}$$

$$s(X) = \sum_{k=1}^{B} s(\tilde{X}_k) \Big/ B \qquad \sigma_s = \sum_{k=1}^{B} \left(s(\tilde{X}_k) - s(X)\right)^2 \Big/ (B-1)$$

3. Bagging

a. Principe

Le bagging applique l'idée du bootstrap à la classification. A partir du jeu initial X, on crée B (souvent 50) jeux par tirage bootstrap (souvent $m = n$) sur lesquels on apprends des classifieurs que l'on combine finalement (souvent vote majoritaire).

b. Out-Of-Bag

L'OOB d'un jeu $\tilde{X}_k$ est l'ensemble de valeurs x_i de X qui ne sont pas dans $\tilde{X}_k$ (en moyenne 37%).

Cet ensemble peut servir comme jeu de test lors de l'apprentissage sur le jeu $\tilde{X}_k$.

4. Boosting

a. Principe

Apprentissage itératif/adaptatif d'un ensemble de classifieurs par pondération de chaque donnée en fonction de son erreur par les classifieurs précédents.

b. Algorithme d'apprentissage AdaBoost

$D_1(x_i) = \frac{1}{m} \quad \forall\, i$ — Initialisation des poids

pour $t = 1, \dots, T$ faire

$h_t = \mathcal{L}(X, D_t)$ — Apprentissage de h_t

$\widehat{e_t} = \sum_{\substack{i \\ h(x_i) \neq y_i}} D_t(x_i)$ — Calcul de l'erreur pondérée de h_t

$\alpha_t = \frac{1}{2} \ln \frac{1 - \widehat{e_t}}{\widehat{e_t}}$ — Calcul du coefficient de pondération de h_t

$D_{t+1}(x_i) = \frac{D_t(x_i) \exp -\alpha_t y_i h_t(x_i)}{\sum_i D_{t+1}(x_i)} \quad \forall\, i$ — Mise à jour des poids des données

fin pour

c. Classification

On fait de la classification par vote pondéré, c'est-à-dire :

$$\hat{y} = signe\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right)$$

5. Foret aléatoires

a. Arbres de décision

1. Créer un premier groupe avec l'ensemble de données
2. Apprendre une règle pour séparer le groupe en 2 sous-groupes les plus purs possibles
3. Répéter l'opération 2 sur chaque sous-groupe, sauf si on a atteint le critère d'arrêt

On peut prendre par exemple comme critères d'arrêts une pureté minimale à atteindre et/ou une hauteur maximale de l'arbre.

b. Principe de la forêt aléatoire

Apprendre un ensemble d'arbres de décision à partir de données différentes $\tilde{X}_k$ mais suivant la même distribution statistique et combiner leurs décisions.

c. Construction des jeux de données

Pour augmenter les performances des forêts, on ajoute de l'aléatoire dans la création du jeu d'apprentissage de chaque arbre $\tilde{X}_k$ à partir des données X.

- Création des jeux $\tilde{X}_k$ par bootstrap
- Construction des arbres par rapport à une seule variable (Random Feature Selection TODO a revoir…)
- Sélection d'un sous-ensemble de $\tilde{p}$ variables parmi les p variables de X

d. Décision de la forêt

- Vote majoritaire simple des décisions des arbres
- Pondération des arbres par leur performance sur l'OOB

III. SVM et noyaux

1. Rappel : le SVM linéaire

a. Problème primal

$$X_{app} \in \mathbb{R}^{n \times p} \quad y_{app} \in \{1; -1\}^n \quad \Rightarrow \quad \boxed{\begin{cases} \min_{w,\xi_i} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i \\ y_i(w^\top x_i + b) \geq 1 - \xi_i \ \ \forall i \\ \xi_i \geq 0 \end{cases}} \quad \Rightarrow \quad \boxed{\begin{matrix} f(x) = w^\top x + b \\ \hat{y} = sign(f(x)) \end{matrix}}$$

Données initiales — *Problème SVM linéaire primal* — *Fonction de décision*

b. Problème dual

♦ *Lagrangien du problème primal*

$$\mathcal{L}(w, b, \xi, \alpha, \beta) = \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \alpha_i(y_i(w^\top x + b) - 1 + \xi_i) - \sum_{i=1}^{n} \beta_i \xi_i$$

♦ *Focus sur la complémentarité : cas possibles de classement des points*

$\alpha_i(y_i(w^\top x + b) - 1 + \xi_i) = 0$

Bien classés I_0 (inutiles)	*Vecteurs supports I_α* (définissent la frontière)	*Mal classés I_C* (influent)
$y_i(w^\top x_i + b) > 1$	$y_i(w^\top x_i + b) = 1$	$y_i(w^\top x_i + b) = 1 - \xi_i < 1$
$\alpha_i = 0$	$0 < \alpha_i < C$	$\alpha_i = C$
$\xi_i = 0$	$\xi_i = 0$	$\xi_i > 0$

♦ *Problème dual*

$$\boxed{\begin{cases} \min_{\alpha} \frac{1}{2}\alpha^\top G\alpha - \alpha^\top e \\ 0 \leq \alpha_i \leq C \\ \alpha^\top y = 0 \end{cases}} \quad \text{avec} \quad \boxed{G_{ij} = x_i^\top x_j y_i y_j} \quad \Rightarrow \quad \boxed{\begin{matrix} w = \sum_{i=1}^{n} \alpha_i y_i x_i \\ f(x) = \sum_{i=1}^{n} \alpha_i y_i x_i^\top x + b \\ \hat{y} = sign(f(x)) \end{matrix}}$$

Problème SVM linéaire dual — *Fonction de décision*

2. Passage au problème non-linéaire : noyau

On se propose de rendre linéaire un problème qui ne l'est pas en changeant d'espace via une *feature map* ϕ passant des x_i aux t_i, on utilisera donc les t_i dans le problème.

L'espace d'arrivée est très grand. Dans le primal, le nombre de variable explose. Dans le dual, on a toujours autant de variables mais il faut calculer les produits scalaires entre les t_i.

On définit le kernel permettant de calculer $t_i^\top t_j$ à partir de x_i, x_j.

En pratique donc, on ne calcule pas les t_i, on intègre la fonction kernel au problème quand on a besoin des produits scalaires des t_i.

Le problème et la fonction de décision deviennent donc :

Exemple de ϕ polynomiale :

$$\phi : \mathbb{R}^2 \rightarrow \mathbb{R}^p \qquad x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \rightarrow t = \begin{bmatrix} x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \\ x_1 x_2 \\ \vdots \end{bmatrix}$$

Exemple de kernels :

$$k_{polynomial}(x_i, x_j) = (x_i^\top x_j + 1)^m$$

$$k_{gaussien}(x_i, x_j) = e^{-\frac{\|x_i - x_j\|^2}{2\sigma^2}}$$

$$\boxed{\begin{cases} \min_{f,b,\xi_i} \frac{1}{2}\|f\|^2 + C\sum_{i=1}^{n} \xi_i \\ y_i(f(x_i) + b) \geq 1 - \xi_i \\ \xi_i \geq 0 \end{cases}} \quad \Rightarrow \quad \boxed{\begin{cases} \min_{\alpha} \frac{1}{2}\alpha^\top G\alpha - \alpha^\top e \\ 0 \leq \alpha_i \leq C \\ \alpha^\top y = 0 \end{cases}} \quad \Rightarrow \quad \boxed{\begin{matrix} w = \sum_{i=1}^{n} \alpha_i y_i t_i \\ f(x) = \sum_{i=1}^{n} \alpha_i y_i k(x_i, x) + b \\ \hat{y} = sign(f(x)) \end{matrix}}$$

$\boxed{f(\cdot) = \sum_{i=1}^{n} \alpha_i y_i \, k(\cdot, x_i)}$ — $\boxed{G_{ij} = k(x_i, x_j) y_i y_j}$

Problème primal — *Problème dual* — *Fonction de décision*

IV. Choix des paramètres par validation croisée

1. Estimation de l'erreur par validation croisée

$$X \quad \rightarrow \quad \{X_k\} \quad \rightarrow \quad \forall\, k, \begin{cases} X_k \text{ app } \overline{X_k} \text{ test} \\ \overline{X_k} \text{ app } X_k \text{ test} \end{cases} \quad \rightarrow \quad e_k \quad \rightarrow \quad e = \frac{1}{K}\sum_k e_k \qquad \sigma^2 = \frac{1}{K-1}\sum_k (e_k - e)^2$$

Erreur : $\boxed{e \pm t_{\alpha/2,K-1}\sqrt{\frac{\sigma^2}{K}}}$

Data *Splitées* *Appretissages* *Erreurs* *Erreur finale*

2. Optimisation des hyperparamètres

Pour optimiser P paramètres, on génère une grille de dimension p et on calcule l'erreur pour chaque point de la grille. On peut répéter l'opération sur plusieurs grilles de plus en plus précises.

V. Support Vector Data Description

1. Minimum enclosing ball problem

$$\boxed{\begin{cases} \min\limits_{c,R,\xi} R^2 + C\sum\limits_{i=1}^{n} \xi_i \\ \|x_i - c\|^2 \le R^2 + \xi_i \\ \xi_i \ge 0 \end{cases}}$$

Problème primal

$$\boxed{\begin{cases} \min\limits_{\alpha} \alpha^\top G\alpha - \alpha^\top \operatorname{diag} G \\ \alpha^\top e = 1 \qquad (\mu) \\ 0 \le \alpha_i \le C \end{cases}} \quad \text{avec } G_{ij} = x_i^\top x_j$$

$$R^2 = \mu + \alpha^\top G\alpha = \mu + \|c\|^2$$

Problème dual

2. Ajout d'un kernel

$$\boxed{\begin{cases} \min\limits_{c,R,\xi} R^2 + C\sum\limits_{i=1}^{n} \xi_i \\ \|k(\cdot, x_i) - c(\cdot)\|^2 \le R^2 + \xi_i \\ \xi_i \ge 0 \end{cases}}$$

Problème SVDD primal

$$\boxed{\begin{cases} \min\limits_{\alpha} \alpha^\top G\alpha - \alpha^\top \operatorname{diag} G \\ \alpha^\top e = 1 \qquad (\mu) \\ 0 \le \alpha_i \le C \end{cases}} \quad \text{avec } G_{ij} = k(x_i, x_j)$$

Problème SVDD dual

$$c(\cdot) = \sum_{i=1}^{n} \alpha_i k(\cdot, x_i)$$

Centre

$$f(x) = \|k(x,\cdot) - c(\cdot)\|^2 - R^2 = -2\sum_{i=1}^{n} \alpha_i k(x, x_i) + k(x,x) - \mu$$

Fonction de décision

VI. SVM multi-classe

	Approche	*Problem size*	*Nb of sub-problems*	*Total size*	*Discrimination*	*Rejection*
	All together	$n \times c$	1	$n \times c$	+	--
	1 vs. all	n	c	$n \times c$	++	-
Décomposition	*1 vs. 1*	$2n/c$	$c(c-1)/2$	$n \times (c-1)$	++	-
	c SVDD	n/c	c	n	-	++
	Couplage CH	n	1	n	+	+

All together : $\begin{cases} \min\limits_{w,\xi_i} \frac{1}{2}\sum\limits_{\ell=1}^{c} \|w_\ell\|^2 \\ x_i^\top (w_{y_i} - w_\ell) + b_{y_i} - b_\ell \ge 1 \quad \forall\, i, \ell \ne y_i \end{cases}$

VII. Multi kernel SVM

1. Principe

$$K(\cdot, x_i) = \sum_{m=1}^{M} d_m k_m(\cdot, x_i) \quad \text{avec} \quad \sum_{m=1}^{M} d_m = 1 \quad \text{et} \quad 0 \leq d_m$$

Mélange de kernels : combinaison linéaire de kernels

$$f(\cdot) = \sum_m \underbrace{d_m \sum_i \alpha_i y_i k_m(\cdot, x_i)}_{f_m(\cdot)}$$

Fonction de décision

2. Problème d'optimisation

$$\begin{cases} \min\limits_{f_m, b, \xi, d} \frac{1}{2} \sum\limits_m \frac{1}{d_m} \|f_m\|^2 + C \sum\limits_{i=1}^{n} \xi_i \\ y_i \left(\sum\limits_m f_m(x_i) + b \right) \geq 1 - \xi_i \\ \xi_i \geq 0 \quad ; \quad \sum\limits_m d_m = 1 \quad ; \quad d_m \geq 0 \end{cases}$$

Problème MKL primal

$$\begin{cases} \min\limits_d J(d) = \begin{cases} \min\limits_{f_m, b, \xi} \frac{1}{2} \sum\limits_m \frac{1}{d_m} \|f_m\|^2 + C \sum\limits_{i=1}^{n} \xi_i \\ y_i \left(\sum\limits_m f_m(x_i) + b \right) \geq 1 - \xi_i \\ \xi_i \geq 0 \end{cases} \\ \sum\limits_m d_m = 1 \quad ; \quad d_m \geq 0 \end{cases}$$

Problème MKL primal bi-niveaux

3. Résolution

On résout le problème par descente de gradient. A chaque itération on résout $J(d)$ et on calcule son gradient $\nabla_d J$, on en déduit la direction de descente D et le pas optimal dans cette direction γ.

$$J(d) = \frac{1}{2} \alpha^\top G \alpha - e^\top \alpha$$

$$G = \sum_m d_m G_m \qquad G_{m,ij} = k_m(x_i, x_j)$$

Cout à l'optimum

$$\Rightarrow \quad \nabla_{d_m} J = \frac{1}{2} \alpha^\top G_m \alpha$$

Gradient

$$\sum_m d_m = 1 \Rightarrow \sum_m D_m = 0$$

$$\Rightarrow \begin{cases} D_m = \nabla_{d_m} J - \nabla_{d_1} J \\ D_1 = -\sum\limits_{m=2}^{M} D_m \end{cases}$$

Direction de descente