Large Scale Kernel methods

=> cf rappel des méthodes à noyau (SVM)

Noyau

une fonction symmétrique $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ est un noyau si il existe une fonction mapping $\phi : \mathcal{X} \to \mathcal{H}$ de l'espace de départ $\mathcal{X}$ à un espace de Hilbert $\mathcal{H}$ tel que K peut s'écrire comme un produit scalaire dans $\mathcal{H}$:

$$K(x, x') = \langle \phi(x), \phi(x') \rangle$$

<=> K est un noyau si il est PSD.

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_i c_j K(x_i, x_j) \geq 0$$

Exemples de noyaux :

- Linéaire $K(x, x') = x^T x'$
- Polynomial $K(x, x') = (x^T x' + c)^d$
- Gaussian RBF $K(x, x') = e^{-\gamma \|x - x'\|_2^2}$
- Laplace RBF $K(x, x') = e^{-\gamma \|x - x'\|_1}$

=> permet d'obtenir des variantes non linéaires pour la plupart des algorithmes de Machine Learning.

Problèmes de scalabilité

* **Training time complexity**

=> besoin du calcul de la matrice de Gram

$$G = \begin{bmatrix} K(x_1, x_1) & \dots & K(x_1, x_n) \\ \vdots & & \\ K(x_n, x_1) & \dots & K(x_n, x_n) \end{bmatrix}$$

=> construire G : $O(n^2)$ long
=> inverser G : $O(n^3)$

Donc dès que n grand, trop long.

* **Test time complexity.**

=> méthodes à noyau non paramétrique
=> besoin de garder en mémoire au moins certains points d'apprentissage pour la prédiction.

exemple SVM : intérêt de la forme duale, on peut passer au noyau.

$$f(x) = \text{sign}\left(\sum_{i=1}^{n} \alpha_i y_i K(x, x_i) + b\right)$$

=> le nombre de vecteurs support ↗ linéairement avec n

Donc dès que n est grand, les prédictions sont lentes.

Deux méthodes :

* Random Kernel Features : approximation du mapping ϕ
* Nystrom approximation : " matrice de Gram G

* Random Kernel Features.

Exemple SVM : quand ϕ est de dimension infinie, on ne peut résoudre le problème que dans sa forme duale

=> Trouver une approximation $\hat{\phi}$ de dimension finie de ϕ.

$$\langle \hat{\phi}(x), \hat{\phi}(x') \rangle \simeq K(x,x') \qquad \hat{\phi} \in \mathbb{R}^c$$

Choix de c

- $c \ll n^2$: apprentissage plus rapide
- $c \ll n$: prédiction plus rapide.
- $c \to +\infty$: on retrouve le noyau

Choisir c en fonction des contraintes de calcul qu'on peut avoir.

Noyau shift. invariants

$$K(\Delta) = K(x-a, x'-a) = K(x,x') = K(x-x')$$

=> invariance par translation (tous les noyaux qui utilisent les dist).

Exemples :

- Noyau Gaussien RBF
- Noyau de Laplace.

Théorème de Bochner

Un noyau invariant par translation $K(x,x') = K(\Delta)$ est positif défini si et seulement si $K(\Delta)$ est la transformation de Fourier d'une mesure de probabilité non négative.

=> on va utiliser le résultat de ce théorème qui dit que.

$$K(x-x') = E_{\omega \sim p,\, b \sim U(0,2\pi)} \Big[\underbrace{\sqrt{2}\cos(\omega^T x + b)\sqrt{2}\cos(\omega^T x' + b)}_{\text{forme de produit scalaire}} \Big]$$

Si je peux échantillonner ω et b, je peux avoir $\hat{\phi}$ qui ressemble à un produit scalaire et je vais converger vers la vraie valeur avec un gd nb d'échantillonnage.

=> Il suffit de trouver p pour approximer K par échantillonnage. (transformation de Fourier de K).

Pour un noyau gaussien

$$p^{rbf}(\omega) = \mathcal{N}(0, 2\gamma I_p)$$

Pour un noyau de Laplace,

$$p^{lap}(\omega) = \prod_p \frac{1}{\pi(1+\omega_p^2/2\gamma)} \text{ (Cauchy)}$$

Large Scale Kernel Methods (2)

Algorithme

1) Initialiser le nombre de kernel features c
2) Générer $\omega_1, \dots \omega_c \sim P(\omega)$ et $b_1, \dots, b_c \sim U(0, 2\pi)$
3) Mapper les points d'entraînement dans leur noyau :

$$\hat{\phi}_j(x) = \sqrt{\frac{2}{c}} \cos(\omega_j^T x + b_j)$$

$\frac{1}{\sqrt{c}}$ on avait une espérance donc besoin de normaliser

4) Entraîner un modèle linéaire sur les données transformées.

$$\hat{\phi}(x_1), \dots, \hat{\phi}(x_n) \in \mathbb{R}^c$$

Borne

Soit $c \geq 1$ et $\hat{\phi} : \mathbb{R}^p \to \mathbb{R}^c$ la feature map obtenue en générant les ω de $P(\omega)$:

$$\sup_{x, x' \in \mathcal{X}} \left| \langle \hat{\phi}(x), \hat{\phi}(x') \rangle - k(x, x') \right| \leq O\left(\sqrt{\frac{p}{c}} \right)$$

=> Avec grande probabilité, la plus grande différence que je peux avoir est de l'ordre de $\sqrt{\frac{p}{c}}$ (tend vers 0 quand $c \to +\infty$).

=> Il y a une dépendance aux features de départ
Plus j'ai de features au départ, plus j'ai besoin que c soit grand.

=> on peut aussi obtenir une borne de l'erreur de généralisation.

* Approximation Nyström

=> on va chercher à approximer la matrice de Gram de façon compacte. ($k < n$).

=> Comme elle est PSD, on peut la décomposer.

$$G = U \Lambda U^T$$

valeurs propres — vecteurs propres

=> On peut générer la meilleure approximation de rang k au sens de Frobenius

toujours $n \times n$ — $G_k = U_k \Lambda_k U_k^T$

en seuillant et en ne gardant que les k plus grandes valeurs propres

Mais, on a :

- toujours besoin de construire G : $O(n^2)$
- besoin de calculer la décomposition de rang k, G_k : $O(n^2)$ à $O(n^3)$

=> coûteux donc on va changer de stratégie.

=> Trouver une bonne approximation $\hat{G}_k$ de G_k en temps linéaire $O(n)$ (on aura alors une matrice exprimée avec un nombre de paramètres beaucoup plus petit, coût de stockage plus faible et entraînement plus rapide).

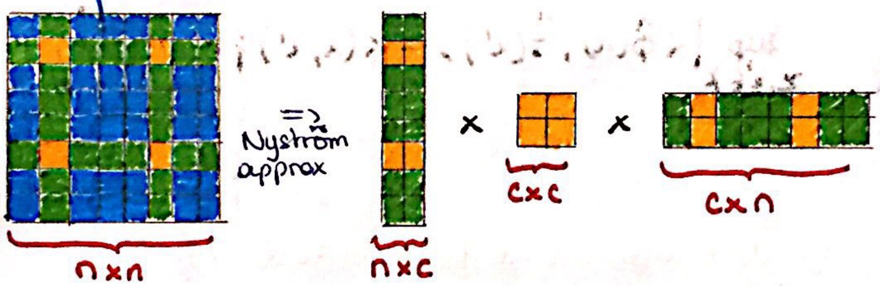
Approximation de Nyström :

$$\hat{G}_k = C W^+ C^T$$

$C \in \mathbb{R}^{n \times c}$

pseudo inverse de Moore Penrose de W
là pour besoin de normalisation
existe toujours..

pas besoin de calculer les entrées en bleu



on calcule juste une petite partie de la matrice

Algorithme

1) on tire un jeu de c indices uniformément
2) Calcul de $C \in \mathbb{R}^{n \times c}$ avec $C_{ij} = k(x_i, x_j)$
3) Former la matrice $W \in \mathbb{R}^{c \times c}$ avec $W_{ij} = k(x_i, x_j)$
4) Calcul de $W_k \in \mathbb{R}^{c \times c}$, la meilleure approximation de rang k de W
5) Approximation finale de rang k de G

on va mettre toutes les vp très petites à 0 pour éviter d'avoir des valeurs qui explosent

$$O(c^3 + nck)$$

Borne

terme incompressible

$$\|G - \hat{G}_k\|_F \leq \|G - G_k\|_F + O\left(\sqrt{\frac{n}{c}}\right)$$

- converge dépend de n

matrice de Gram — Approxim° Nyström — meilleure approxi. de rang k — erreur additive si c est grand, $\hat{G}_k$ est presque aussi bon que G_k

Génération de $\hat{\phi}$

$$\hat{G}_k = C W_k^+ C^T \qquad W_k^+ = V_k \Sigma_k V_k^T \qquad \hat{\phi}(x_i) = C_i V_k \Sigma_k^{1/2} \qquad \hat{\phi}(x) = k_{x,I} V_k \Sigma_k^{1/2}$$

(entraînement) (nouveaux points).