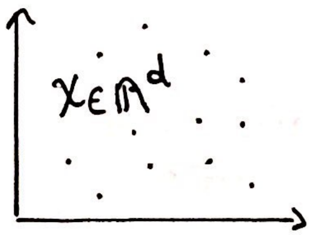
Metric learning (1)

Le choix de la similarité / distance est cruciale pour les performances.

=> Comment apprendre une similarité appropriée pour une tâche donnée ?

Le metric learning a pour objectif de construire une métrique de distance spécifiques à une tâche donnée à partir de données supervisées. La distance apprise peut ensuite être utilisée pour ≠ tâches (ex: KNN, clustering ...)

ex : vérification faciale.



Recette de metric learning :

1) choisir une distance paramétrique en amont

$D_M(x, x')$ paramétrisé par une matrice M

2) Collecter des jugements sur des paires d'images (couple (S, D), triplet (S, D, R))

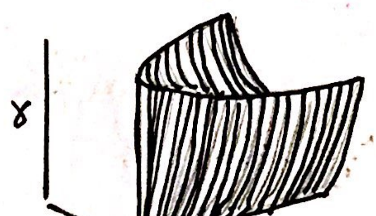
3) Estimer les paramètres (métrique qui s'accorde le + avec les jugements).

problème d'optimisation

$$M^* = \underset{M}{\arg\min} \Big[\underbrace{\ell(M, S, D, R)}_{\text{loss}} + \lambda \underbrace{reg(M)}_{\text{regul}} \Big]$$

(Frobenius, $L_{2,1}$ norm, Trace)

En général, on utilise la distance de Mahalanobis.



$$D_M(x, x') = \sqrt{(x - x')^T M (x - x')}$$

pseudo-distance

M est PSD $\in \mathbb{R}^{d \times d}$

c'est une distance euclidienne après transformation linéaire

L'ensemble des matrices PSD est un cône convexe donc cela va faciliter les problèmes d'optimisation.

Si $rg(M) = k \leq d$ alors $L \in \mathbb{R}^{k \times d}$ => réduction de dimension.

On a $x^T M x \geq 0 \ \forall x \in \mathbb{R}^d$ et $M = L^T L$ (Cholesky)

* Pairwise constraints.

=> tâche : clustering

Problème d'optimisation

$$\max_{M \in S_+^d} \sum_{(x_i,x_j)\in D} D_M(x_i,x_j)$$

concave car composée de racine + linéaire

$$\text{sous contraintes } \sum_{(x_i,x_j)\in S} D_M^2(x_i,x_j) \leq 1$$

change rien au pb.

convexe car linéaire en M

=> toujours faisable ($M=0$)

=> résolu avec une méthode de gradient projeté

complexité $O(d^2)$ si projeté sur les contraintes de distance.

$O(d^3)$ si " sur S_+^d

* Triplet constraints

=> tâche : recherche d'information.

$$\min_{M \in S_+^d, \xi \geq 0} \|M\|_F^2 + \lambda \sum_{i,j,k} \xi_{i,j,k}$$

norme de frobenius (regul)

Formulation convexe

$$\text{sous contraintes } D_M^2(x_i,x_k) - D_M^2(x_i,x_j) \geq 1 - \xi_{ijk}$$

1 contrainte par triplet.

=> on veut que les x_i soient plus près de x_j que x_k si possible avec une certaine marge. Si on n'arrive pas à avoir cette marge, on va payer un coût dans la fonction objectif.

* Large Margin Nearest Neighbor

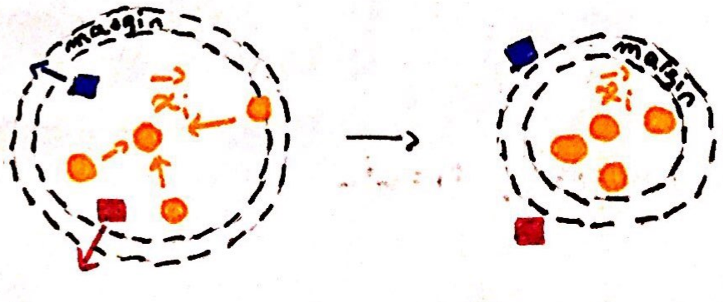
=> tâche : kNN classification

trade off parameter $\mu \in [0,1]$

Formulation convexe

$$\min_{M \in S_+^d, \xi \geq 0} (1-\mu) \sum_{(x_i,x_j)\in S} D_M^2(x_i,x_j) + \mu \sum_{ijk} \xi_{ijk}$$

$$\text{sous contraintes } D_M^2(x_i,x_k) - D_M^2(x_i,x_j) \geq 1 - \xi_{ijk}$$



on veut rapprocher les points d'une même classe et "éjecter" les autres avec une marge de sécurité.

Scanné avec CamScanner

Metric learning (2)
Extension non-linéaire

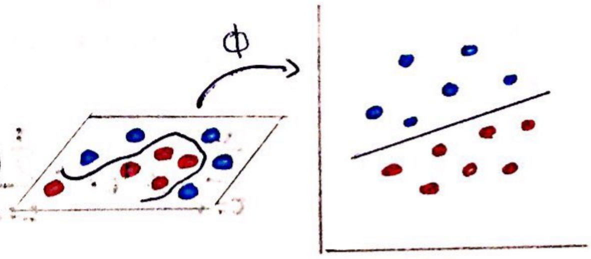
* Approche par noyau

on fixe une transformation riche $K(x, x') = \langle \phi(x), \phi(x') \rangle$

=> on apprend une métrique linéaire dans l'espace noyau.

distance de Mahalanobis

$$D^2_M(\phi_i, \phi_j) = (\phi_i - \phi_j)^T M (\phi_i - \phi_j)$$
$$= (\phi_i - \phi_j)^T L^T L (\phi_i - \phi_j)$$



en posant $L^T = \phi U^T$

$$D^2_M(\phi(x), \phi(x')) = (k - k')^T M (k - k')$$

les paramètres ne dépendent pas de la taille de l'espace ϕ

on peut calculer ces distances que par évaluation du noyau.

=> problème de passage à l'échelle pour cette méthode. $O(n^2)$

* Approche par apprentissage d'une transformation linéaire

on fixe un mapping ϕ

=> on apprend une distance euclidienne dans ce mapping.

$$D_\phi(x, x') = \| \phi(x) - \phi(x') \|_2$$

Marche bien mais ne conserve pas la convexité.

Exemple d'application : image avec réseau siamois (on peut plugger la fonction objectif au bout)
paroles.

* Apprentissage multiple de métrique locale (LMNN)

=> grouper les données en C clusters.

Pour chaque groupe, on apprend une métrique :

$$\min_{\xi \geq 0} (1-\mu) \sum_{(x_i, x_j) \in S} D^2_{M_{c(x_j)}}(x_i, x_j) + \mu \sum_{ijk} \xi_{ijk}$$

sous contraintes $D^2_{M_{c(x_k)}}(x_i, x_k) - D^2_{M_{c(x_j)}}(x_i, x_j) \geq 1 - \xi_{ijk}$

garde la convexité, mais overfitting.

Large-Scale Metric learning

Deux problèmes :
- large datasets => n grand ($O(n^2)$, $O(n^3)$)
- Grande dimension => d grand ($O(d^2)$, $O(d^3)$)

★ Cas de n large

1- Online learning

moyen de passer à l'échelle

=> des données arrivent séquentiellement (on a pas besoin de gérer tous les points à la fois).

On cherche à minimiser le regret :

fonction de décroissance qui converge vers 0

$$\sum_{t=1}^{T} \ell_t(M_t) - \sum_{t=1}^{T} \ell_t(\underbrace{M^*}_{M \text{ optimal}}) \leq f(T) \quad (T \text{ étapes})$$

(A quel point j'ai été moins bon que si j'avais travaillé avec toutes les données en batch).

=> On peut montrer qu'on a des garanties : pour certains algorithmes, le regret est en $\sqrt{T}$ ou $\log(T)$

Algorithme OASIS.

A chaque étapes t,

pour ne pas oublier ce que j'ai vu avant

$$M^t = \underset{M,\xi}{\operatorname{argmin}} \frac{1}{2} \|M - M^{t-1}\|_F^2 + C\xi$$

paramètre d'aggressivité

$$\text{sous contraintes} \quad 1 - S_M(x_i, x_j) + S_M(x_i, x_k) \leq \xi \quad \xi \geq 0$$

puis $S_M(x, x') = x^T M x'$ $C \geq 0$

à quel point on est prêt à changer la métrique si on s'est trompé sur un triplet

Solution donnée par $M^t = M^{t-1} + \beta V$

pas (si C faible, β faible)

$V = x_i (x_j - x_k)^T$

2- Méthodes de gradient stochastique

$$\min_{M} \frac{1}{|R|} \sum_{(x_i,x_j,x_k)\in R} \ell(M, x_i, x_j, x_k)$$

=> utilise un échantillon aléatoire pour estimer le gradient
=> mieux que le gradient classique quand n est grand.
=> peut être combiné avec de l'optimisation distribuée.

* Cas de d large

1- Apprendre une matrice diagonale

=> on apprend d paramètres

=> on apprend qu'une pondération des features (pas très riche).

2- Apprendre la métrique après réduction de dimension (PCA)

=> on perd de l'information

=> difficile à interpréter.

3- Matrices décompositions

Décomposition de faible rang $M = L^T L \quad L \in \mathbb{R}^{r \times d}$

=> on apprend $r \times d$ paramètres

=> en général non convexe, on doit tuner r

Décomposition rang-1

$$M = \sum_{i=1}^{K} w_k b_k b_k^T$$

=> on apprend K paramètres.

=> on doit choisir un bon set de base.

Régularisation

=> pour éviter l'overfitting.

* norme de Frobenius $\|M\|_F^2 = \sum_{i,j=1}^{d} M_{ij}^2$ convexe, facile à optimiser

* norme mixte $L_{2,1}$ $\|M\|_{2,1} = \sum_{i=1}^{d} \|M_i\|_2$ amène certaines colonnes à 0 (sélection features). convexe

* norme trace $\|M\|_* = \sum_{i=1}^{d} \sigma_i(M)$ favorise les matrices de rg faible (réduction dimension), convexe.

- Composantes qu'on retrouve dans plein d'algo
- Solutions existes pour passer à l'échelle
- Gap entre recherche et software

=> librairie **metric-learn** sur Python.
compatible avec scikit-learn.