Fondements de l'apprentissage machine

Automne 2014

Roland Memisevic

Lecon 10

Plan

- k-nearest neighbors (kNN) = k plus proches voisins (kPP)
- des noyaux
- support vector machines (SVM)

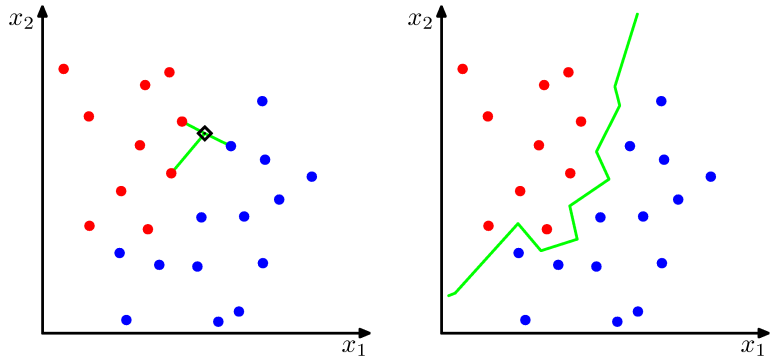
K-Nearest Neighbors

- Étant donnée des observations $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$ et des cibles $\{t_1, \ldots, t_N\}$, l'algorithme K-nearest neighbors (KNN/KPP) classe un exemple de test, $\mathbf{x}$, comme suit :

K-nearest neighbors (KNN)

- En utilisant une mesure de distance $d(\mathbf{x}_1, \mathbf{x}_2)$ entre des obervations, trouvez les k plus proches voisins de $\mathbf{x}$. Un choix commun pour la mesure de distance est la distance euclidienne : $d(\mathbf{x}, \mathbf{x}_i) = \|\mathbf{x} - \mathbf{x}_i\|$
- Definissez comme cible pour $\mathbf{x}$ la majorité des labels parmi les k plus proches voisins.

Exemple



(a) (b)

- (a) Classification d'un exemple de test, en utilisant $K = 3$.
- (b) Frontièr de classification d'un classifieur 1-NN.

régression KNN

- Bien que kNN est plus courant dans les problèmes de la classification, on peut l'utiliser également pour la régression : Assignez à l'exemple de test, $\mathbf{x}$, la moyenne des cibles (reélle) de K voisin les plus proches. Le résultat est un modèle constant par segments (piece-wise linear).

Kernel density estimation (estimation par noyau)

- On peut "lisser" les prédictions des méthodes de KNN et les fusionner avec la modélisation probabiliste :

Kernel density estimation (KDE)

$$p(\mathbf{x}) = \frac{1}{N}\sum_{n=1}^{N} k(\mathbf{x}, \mathbf{x}_n)$$

La fonction $k(\mathbf{x}, \mathbf{x}_n)$ doit satisfaire

$$\int k(\mathbf{x}, \mathbf{x}_n)\, \mathrm{d}\mathbf{x} = 1, \quad k(\mathbf{x}, \mathbf{x}_n) \geq 0$$

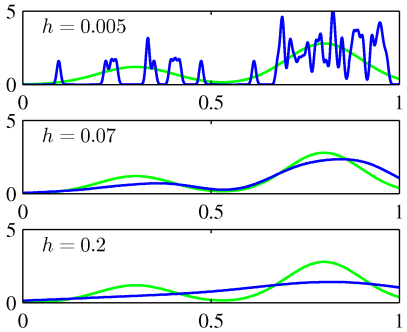
Dans ce cas, $k(\cdot, \cdot)$ est appelée "kernel function", ou "Parzen window" (fenêtre de Parzen).

Kernel density estimation (estimation par noyau)

- Un choix très commun pour $k(\cdot, \cdot)$ est le *noyau gaussien* :

$$k(\mathbf{x}, \mathbf{x}_n) = \frac{1}{(2\pi h^2)^{D/2}} \exp\Big(-\frac{\|\mathbf{x} - \mathbf{x}_n\|^2}{2h^2}\Big)$$

- Exemple de l'estimation d'un densité avec des noyaux gaussiens :



Utilisation d'estimation par noyau

- Après avoir estimé les distributions, il est facile de résoudre diverses tâches d'apprentissage machine en utilisant les lois de la probabilité.
- Par exemple, pour résoudre un problème de classification nous pouvons définir une densité pour chaque classe et utiliser la régle de Bayes.
- Pour résoudre une tâche de régression, nous pouvons définir la densité jointe sur les entrées $\mathbf{x}$ et les sorties t.

Régression Nadaraya-Watson

- Si les noyaux joints prennent la forme d'un produit : $k(\mathbf{x}, \mathbf{x}_n)k(t, t_n)$ la distribution conditionelle est

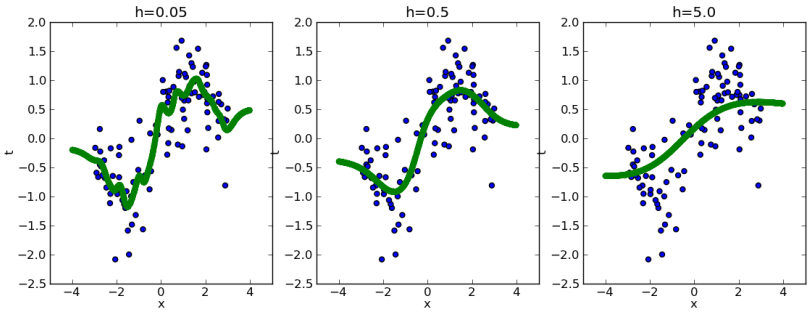
$$p(t|\mathbf{x}) = \frac{p(\mathbf{x}, t)}{\int p(\mathbf{x}, t)\, \mathrm{d}t} = \frac{\sum_n k(\mathbf{x}, \mathbf{x}_n)k(t, t_n)}{\sum_m k(\mathbf{x}, \mathbf{x}_m)}$$

- Pour les nouyaux gaussiens, l'espérance conditionnelle devient :

$$y(\mathbf{x}) := \mathbb{E}[t|\mathbf{x}] = \sum_n \frac{k(\mathbf{x}, \mathbf{x}_n)}{\sum_m k(\mathbf{x}, \mathbf{x}_m)} t_n$$

- Ceci est connu comme le modèle de régression *Nadaraya-Watson*, ou *kernel smoothing*.
- Comme les constantes de normalisation s'annulent, les fonctions du noyau n'ont besoin d'être normalisée.

Exemple de régression Nadaraya-Watson



KDE commentaires

- Les méthodes KNN / estimation par noyau sont également appelés *méthodes non-parametriques* parce qu'elles n'utilisent pas d'hypothèses explicites sur la forme fonctionnelle de la fonction apprise.
- À l'exception des hyperparamètres il n'y a pas vraiment de phase d'apprentissage : il suffit d'enregistrer l'ensemble de entraînement.
- Désavantage : La complexité des prédictions est une fonction linéaire de la taille de l'ensemble d'entraînement. Il existe plusieurs approches pour accélérer ces algorithmes (par exemple, les arbres KD (=KD-trees)).

Méthodes de noyaux non-probabiliste

- Il existe une autre classe de modèle basé sur l'utilisation des noyaux $k(\cdot, \cdot)$.
- Pour ces méthodes, les noyaux ne sont pas utilisés pour définir des probabilités.
- Ils sont basés sur le fait qu'une fonction $k(\mathbf{x}, \mathbf{x}')$ qui est symétrique et definie positive correspond toujours à un produit scalaire d'une transformation des $\mathbf{x}, \mathbf{x}'$:

$$k(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^{\mathrm{T}} \phi(\mathbf{x}')$$

(Mercer's Theorem). La function $\phi(\cdot)$ dépend du noyau.

Kernel Trick

Kernel Trick

- Re-définissez l'algorithme d'apprentissage pour qu'il utilise les données seulement sous forme d'un produit intérieur.
- Remplacez chaque produit intérieur dans l'algorithme par l'application d'un noyau :

$$\mathbf{x}^{\mathrm{T}}\mathbf{x}' \quad \rightarrow \quad k(\mathbf{x}, \mathbf{x}')$$

- Maintenant l'algorithme utilise implicitement une transformation non-linéare $\phi(\mathbf{x})$ sans le calculer réellement.

Régression linéaire en produits intérieurs

- Coût :

$$E(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n - t_n\right)^2 + \frac{\lambda}{2}\mathbf{w}^{\mathrm{T}}\mathbf{w}$$

- Solution (en mettant la dérivée à zéro) :

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{n=1}^{N}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n - t_n\right)\mathbf{x}_n =: \sum_{n=1}^{N} a_n\mathbf{x}_n = \mathbf{X}^{\mathrm{T}}\mathbf{a}$$

où

$$a_n = -\frac{1}{\lambda}\left(\mathbf{w}^{\mathrm{T}}\mathbf{x}_n - t_n\right)$$

Régression linéaire en produits intérieurs

- Remplacez $\mathbf{w} = \mathbf{X}^{\mathrm{T}}\mathbf{a}$ dans le coût :

$$E(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{a} + \mathbf{a}^{\mathrm{T}}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{t} + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{X}\mathbf{X}^{\mathrm{T}}\mathbf{a}$$

avec $\mathbf{t} = (t_1, \ldots, t_N)^{\mathrm{T}}$

- Donc nous pouvons utiliser les noyaux :

$$E(\mathbf{a}) = \frac{1}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{K}\mathbf{a} + \mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{t} + \frac{1}{2}\mathbf{t}^{\mathrm{T}}\mathbf{t} + \frac{\lambda}{2}\mathbf{a}^{\mathrm{T}}\mathbf{K}\mathbf{a}$$

- Pour optimiser par rapport à $\boldsymbol{a}$ mettez la derivée à zéro :

$$\mathbf{a} = \left(\mathbf{K} + \lambda\mathbf{I}_N\right)^{-1}\mathbf{t}$$

Régression linéaire en produits intérieurs

- Pour faire des prédictions :

$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} = \mathbf{a}^{\mathrm{T}}\mathbf{X}\mathbf{x}$$

- En empilant les produits intérieurs (ou des application du noyau) dans le vecteur $\mathbf{k}(\mathbf{x})$:

$$y(\mathbf{x}) = \mathbf{k}(\mathbf{x})^{\mathrm{T}}\left(\mathbf{K} + \lambda\mathbf{I}_N\right)^{-1}\mathbf{t}$$

- D'autres méthodes peuvent être transformé pour utiliser les produits intérieurs (et donc les noyaux) : régression logistique, PCA, kNN, k-means, ...
- Désavantage : La complexité des prédictions est toujours une fonction linéaire de la taille de l'ensemble d'entraînement, et l'apprentissage une fonction quadratique.

Support vector machines

- Les support vector machines (SVM) ont joué un rôle central dans la popularisation des méthodes du noyau. Ils sont donc souvent considéré comme les méthodes du noyau prototypiques.
- Mais dans leur forme élémentaire ils sont simplement une façon d'entraîner un classifieur linéare binaire

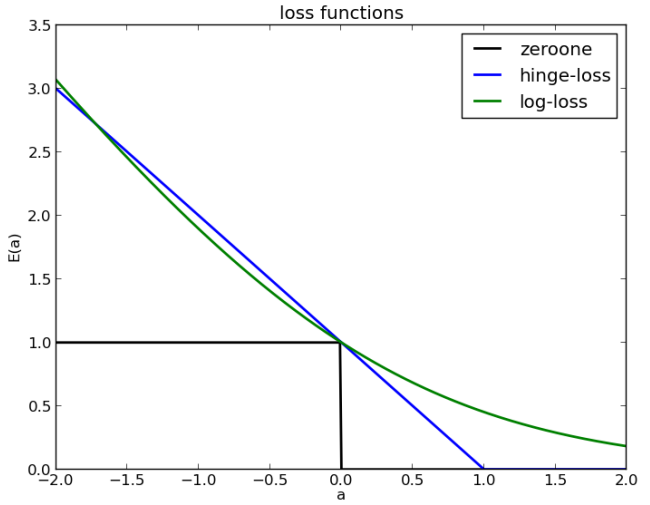
$$y(\mathbf{x}) = \mathbf{w}^{\mathrm{T}}\mathbf{x} + b$$

(avec la fonction de décision $\mathrm{sgn}\big(y(\mathbf{x})\big)$)
- Ils utilisent une fonction de coût qui est similaire au coût de régression logistique régularisé :

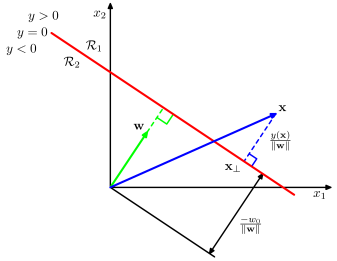
$$\sum_{n=1}^{N} \big[1 - y(\mathbf{x}_n)t_n\big]_+ + \lambda\|\mathbf{w}\|^2$$

où $[\,\cdot\,]_+$ mets les arguments negatifs à zéro
- $\big[1 - y(\mathbf{x}_n)t_n\big]_+$ est appelée hinge loss.

Hinge loss



La marge (margin)



- Comme les points sur la frontière satisfont $\mathbf{w}^{\mathrm{T}}\mathbf{x} + b = 0$, la distance entre la frontière et l'origine est $-\frac{b}{\|\mathbf{w}\|}$
- Donc la distance entre n'importe quel point $\mathbf{x}$ et la frontière est

$$\left|\mathbf{x}^{\mathrm{T}}\frac{\mathbf{w}}{\|\mathbf{w}\|} + \frac{b}{\|\mathbf{w}\|}\right| = \left|\frac{\mathbf{x}^{\mathrm{T}}\mathbf{w} + b}{\|\mathbf{w}\|}\right| = \frac{|y(\mathbf{x})|}{\|\mathbf{w}\|}$$

- Donc maximiser la confiance et minimiser $\|\mathbf{w}\|$ va maximiser *la marge* (et donc la robustesse de décisions).

SVM avec des noyaux

- On peut re-écrire le problème d'optimisation et les décisions en utilisant des produits interieurs seulement.
- Comme pour la régression linéare, on obtient un nouveau problème avec une variable pour chaque exemple d'entraînement.
- Dans la solution il se trouvent qu'un certain nombre de variables prennent la valeur 0. Ces points s'appellent *vecteur support* (*support vectors*). Il ne font pas partie de la solution.
- Comme toutes les méthodes non-paramétriques, les SVM avec des noyaux sont difficiles à appliquer à des problèmes qui ont un grand nombres d'exemples.

Frontière de décisions en utilisant un noyau gaussien