Percepton linéaire, perceptron multi-couches

Liva Ralaivola
Laboratoire d'Informatique Fondamentale de Marseille
UMR 6166 CNRS – Université de Provence
liva.ralaivola[at]lif.univ-mrs.fr

5 février 2008

Plan

Outline

Contexte de la classification supervisée

Classification supervisée (*pattern recognition*)

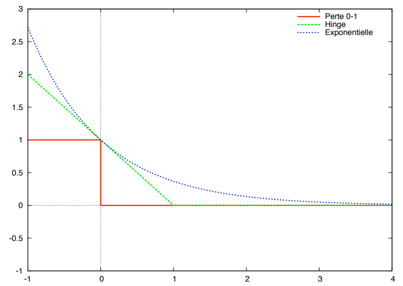
- classification binaire (bi-classe)
- ou multi-classe (catégorisation multi-classe)
- $\mathcal{S} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\}$ ensemble d'apprentissage
- $\mathcal{S}$ réalisation d'un échantillon de v.a. identiquement et indépendamment distribuées selon $\mathcal{D}$ distribution *fixe* mais *inconnue*
- Objectif : classifieur f^* ayant une erreur de généralisation la plus petite possible
$$f^* = \arg\max_{f \in \mathcal{F}} L(f) = \int_{\mathcal{X}\times\mathcal{Y}} l(\mathbf{x}, y, f(\mathbf{x}))dD(\mathbf{x}, y)$$
où $\mathcal{F} \subset \mathcal{Y}^{\mathcal{X}}$

Contexte de la classification supervisée

Fonctions de perte ($f \in \mathbb{R}^{\mathcal{X}}$)

$$f^* = \arg\max_{f \in \mathcal{F}} L(f) = \int_{\mathcal{X} \times \mathcal{Y}} l(\mathbf{x}, y, f(\mathbf{x})) dD(\mathbf{x}, y)$$

- 0-1 loss, step loss : $l(\mathbf{x}, y, f(\mathbf{x})) = \mathbf{1}_{f(\mathbf{x}) \neq y}$
- Exponentielle : $l(\mathbf{x}, y, f(\mathbf{x})) = \exp(-yf(\mathbf{x}))$
- Hinge-Loss : $l(\mathbf{x}, y, f(\mathbf{x})) = |1 - yf(\mathbf{x})|_+$



Contexte de la classification supervisée

Préoccupations

- Algorithmes
  - temps d'exécution finis
  - nombre d'exemples requis raisonnable
  - ... cf. cadre *Probably Approximately Correct* (Valiant, 1984)
- Contrôle de la généralisation
  - bornes sur l'erreur $R(f) = P_{(\mathbf{x},y) \sim D}(f(\mathbf{x}) \neq y)$
  - cf. Théorie statistique de l'apprentissage (Vapnik, 79), ...

Outline

Aprentissage supervisé

Perceptrons linéaires
- Historique
- Digression : espaces vectoriels
- Perceptron linéaire

Perceptron multi-couches

Conclusion

Réseaux de neurones artificiels

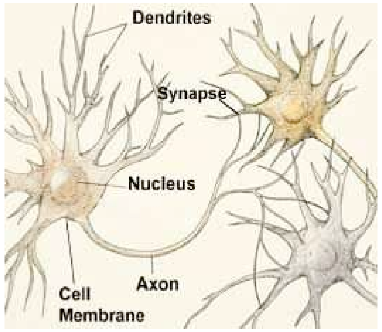
Contexte

- Un des premiers modèles statistiques d'apprentissage
- Perceptron linéaire
- Perceptron multi-couches

Historique

Motivations biologiques

- systèmes apprenants composés de réseaux connectés de plusieurs unités
- capacités de mémoire/adaptabilité de ces systèmes



Historique

Réseaux de neurones biologiques

- nombre de neurones dans le cerveau : 10^{11} neurones chacun étant connecté à 10^4 autres neurones
- temps de transmission de l'information entre deux neurones du cerveau : 10^{-3}
- mais temps d'activation du réseau très rapide : 10^{-1} secondes pour la reconnaissance d'un proche
- connexions en boucles

Réseaux de neurones artificiels

- nombre de neurones : de l'ordre de quelques centaines au maximum avec quelques dizaines de connexions
- temps de transmission de l'information entre deux neurones : 10^{-10} secondes
- difficulté d'apprentissage avec des connexions en boucle

Historique

Popularité

1943–1995 modèle dominant

- Neurone artificiel (McCulloch & Pitts, 1943)
- Perceptron linéaire (Rosenblatt, 1957)
- Limite du XOR (Papert, Minsky, 1969)
- Perceptron multi-couches (1980)
- Reconnaissance de chiffres manuscrits (ATT)

1995–2005 perte de vitesse

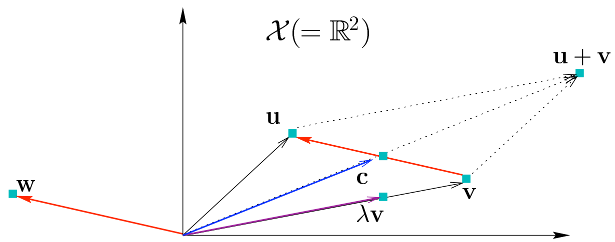
- Méthodes à noyaux, SVM
- Boosting
- Statistical Learning Theory

2006– regain d'intérêt

- Architecture profonde

is through the composition of many non-linearities, i.e. with a **deep architecture**. For example, the parity function with d inputs requires $O(2^d)$ examples and parameters to be represented by a Gaussian SVM (Bengio et al., 2006), $O(d^2)$ parameters for a one-hidden-layer neural network, $O(d)$ parameters and units for a multi-layer network with $O(\log_2 d)$ layers, and $O(1)$ parameters with a recurrent neural network. More generally, boolean functions (such as the function that computes

Vectors and inner product (1/3)

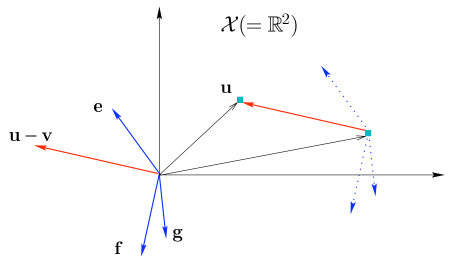


- $\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{c}$ are vectors
- $\mathbf{w} = \mathbf{u} - \mathbf{v}$ (red arrows)
- $\mathbf{c} = \frac{1}{2}(\mathbf{u} + \mathbf{v})$
- Here : $0 < \lambda < 1$

Vectors and inner product (2/3)

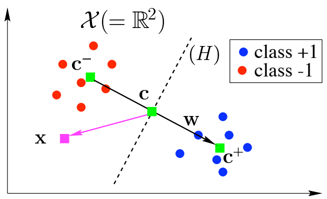
- Inner product $\langle \cdot, \cdot \rangle : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$:
  - symmetric : $\langle \mathbf{u}, \mathbf{v} \rangle = \langle \mathbf{v}, \mathbf{u} \rangle$
  - bilinear : $\langle \lambda \mathbf{u}_1 + \gamma \mathbf{u}_2, \mathbf{v} \rangle = \lambda \langle \mathbf{u}_1, \mathbf{v} \rangle + \gamma \langle \mathbf{u}_2, \mathbf{v} \rangle$
  - positive : $\langle \mathbf{u}, \mathbf{u} \rangle \geq 0$
  - definite : $\langle \mathbf{u}, \mathbf{u} \rangle = 0 \Rightarrow \mathbf{u} = 0$
- An inner product
  - provides $\mathcal{X}$ with a structure
  - can be viewed as a 'similarity'
  - defines a norm $\| \cdot \|$ on $\mathcal{X}$: $\|\mathbf{u}\| = \sqrt{\langle \mathbf{u}, \mathbf{u} \rangle}$
- Example in $\mathbb{R}^2$
  - $\mathbf{u} = \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}, \mathbf{v} = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} : \langle \mathbf{u}, \mathbf{v} \rangle = u_1 v_1 + u_2 v_2$

Vectors and inner product (3/3)



- $\langle \mathbf{u} - \mathbf{v}, \mathbf{e} \rangle > 0$: $\mathbf{u} - \mathbf{v}$ and $\mathbf{e}$ point to the 'same direction'
- $\langle \mathbf{u} - \mathbf{v}, \mathbf{f} \rangle = 0$: $\mathbf{u} - \mathbf{v}$ and $\mathbf{f}$ are orthogonal
- $\langle \mathbf{u} - \mathbf{v}, \mathbf{g} \rangle < 0$: $\mathbf{u} - \mathbf{v}$ and $\mathbf{g}$ point to 'opposite directions'

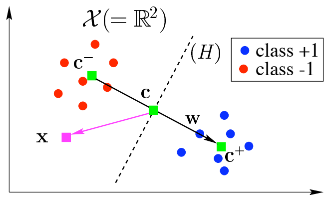
A simple linear classifier



- $\mathbf{c}^+ = \frac{1}{m^+} \sum_{\{i:y_i=+1\}} \mathbf{x}_i$
- $\mathbf{c}^- = \frac{1}{m^-} \sum_{\{i:y_i=-1\}} \mathbf{x}_i$
- $\mathbf{c} = \frac{1}{2}(c^+ + c^-)$
- $\mathbf{w} = c^+ - c^-$

- Idea (see [Schölkopf and Smola, 2002] for details) : classify points $\mathbf{x}$ according to which of the two class means $\mathbf{c}^+$ or $\mathbf{c}^-$ is closer :
  - for $\mathbf{x} \in \mathcal{X}$, it is sufficient to take the sign of the inner product between $\mathbf{w}$ and $\mathbf{x} - \mathbf{c}$
  - if $h(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} - \mathbf{c} \rangle$, we have the classifier $f(\mathbf{x}) = \text{sign}\,(h(\mathbf{x}))$
  - the (dotted) hyperplane (H), of normal vector $\mathbf{w}$, is the decision surface

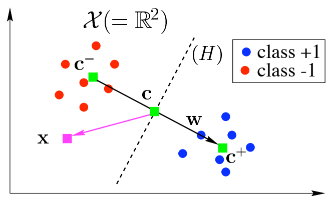
A simple linear classifier



- $\mathbf{c}^+ = \frac{1}{m^+} \sum_{\{i:y_i=+1\}} \mathbf{x}_i$
- $\mathbf{c}^- = \frac{1}{m^-} \sum_{\{i:y_i=-1\}} \mathbf{x}_i$
- $\mathbf{c} = \frac{1}{2}(c^+ + c^-)$
- $\mathbf{w} = c^+ - c^-$
- On evaluating $h(\mathbf{x})$

$$\begin{aligned} h(\mathbf{x}) &= \langle \mathbf{w}, \mathbf{x} - \mathbf{c} \rangle = \langle \mathbf{w}, \mathbf{x} \rangle - \langle \mathbf{w}, \mathbf{c} \rangle \\ &= \langle \mathbf{c}^+, \mathbf{x} \rangle - \langle \mathbf{c}^-, \mathbf{x} \rangle - \langle \mathbf{c}^+, \mathbf{c} \rangle + \langle \mathbf{c}^-, \mathbf{c} \rangle \\ &= \sum_{i=1,\ldots,m} \alpha_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b, \quad \text{with } b \text{ a real constant} \end{aligned}$$

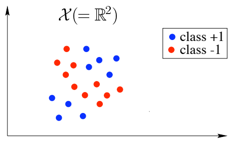
A simple linear classifier



- $\mathbf{c}^+ = \frac{1}{m^+} \sum_{\{i:y_i=+1\}} \mathbf{x}_i$
- $\mathbf{c}^- = \frac{1}{m^-} \sum_{\{i:y_i=-1\}} \mathbf{x}_i$
- $\mathbf{c} = \frac{1}{2}(c^+ + c^-)$
- $\mathbf{w} = c^+ - c^-$

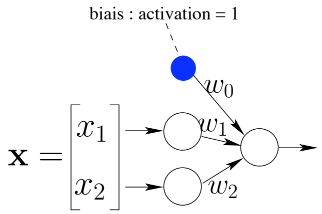
- To summarize : $\boxed{h(\mathbf{x}) = \sum_{i=1,\ldots,m} \alpha_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b}$ [Schölkopf et al., 2000]

- Question : what if the dataset is not linearly separable, i.e. (H) fails to separate red and blue disks ?

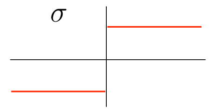


Perceptron linéaire

Séparateur linéaire



$\sigma(\sum_{i=1}^{d} w_i x_i + w_0)$



Classification de $\mathbf{x}$ effectuée en fonction de l'hyperplan

$$\mathbf{w} \cdot \tilde{\mathbf{x}} = 0 \text{ où } \mathbf{w} = \begin{bmatrix} w_0 \\ w_1 \\ w_2 \end{bmatrix} \text{ et } \tilde{\mathbf{x}} = \begin{bmatrix} 1 \\ \mathbf{x} \end{bmatrix}$$

Question ?

Quelle est l'utilité du '1' supplémentaire ajouté à chaque vecteur ?

Perceptron linéaire

Algorithme I

$\mathcal{X} = \mathbb{R}^n, \mathcal{Y} = \{-1, +1\}$

- Initialisation $\mathbf{w} = \mathbf{0}$
- Répéter jusqu'à convergence ou bien atteinte d'un nombre max d'itérations
  - pour tous les exemples $(\mathbf{x}_p, y_p)$ faire
    - si $\sigma(\mathbf{w} \cdot \tilde{\mathbf{x}}_p) = y_p$
      ne rien faire
    - sinon
      $\mathbf{w} \leftarrow \mathbf{w} + y_p \tilde{\mathbf{x}}_p$

Perceptron linéaire

Algorithme d'apprentissage II (descente de gradient)

- Initialiser $\mathbf{w}$ aléatoirement
- Répéter jusqu'à convergence ou atteinte d'un nombre max d'itérations
  - Initialiser Δw_i à 0
  - Pour tous les exemples $(\mathbf{x}_p, y_p)$
    - calculer la sortie $s = \mathbf{w} \cdot \tilde{\mathbf{x}}_p$
    - pour chaque composante $\mathbf{w}_i$

      $$\Delta w_i \leftarrow \Delta w_i + \eta(y_p - s)x_{pi}$$
  - pour chaque composante $\mathbf{w}_i$ faire

    $$w_i \leftarrow w_i + \Delta w_i$$

Un peu de théorie

Theorem

Convergence Soit $\mathcal{S} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\}$ un ensemble de points étiquetés. Si $\|\mathbf{x}_i\| \leq R$ *et* s'il *existe* $\mathbf{w}^*$ *et* $\gamma > 0$ *tels que* $\mathbf{w}^* \cdot x_i \geq \gamma$ alors *l'algorithme du perceptron l converge en moins de R^2/γ^2 itérations. (preuve en TD)*

Theorem

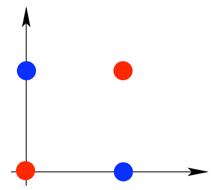
Généralisation (Vapnik) Soit D un distribution de probabilités sur $\mathbb{R}^d \times \mathcal{Y}$, soit $\delta > 0$, soit l un entier tq $l > n$, soit $\mathcal{S}$ un ensemble de l exemples tirés indépendamment selon D et soit f un classifieur linéaire.
Alors, avec une probabilité supérieure à $1 - \delta$, on a

$$R(f) \leq R_{emp}(f) + \sqrt{\frac{n+1}{l}\left(1 + \log\frac{2l}{n+1}\right) + \frac{1}{l}\log\frac{4}{\delta}}$$

A retenir sur le perceptron linéaire

Propriétés du perceptron linéaire

- convergence de l'algorithme du perceptron garantie si séparabilité des exemples possible
- séparation impossible du XOR



Extensions

- perceptron multi-couches
- kernel adatron [Friess et al., 1998]
- voted perceptron [Freund and Schapire, 1999]

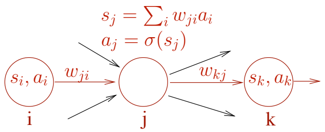
Outline

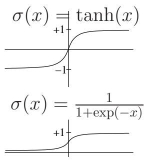
Contexte

- Classification supervisée (*pattern recognition*)
  - $\mathcal{S} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell)\}$ ensemble d'apprentissage
  - $\mathcal{X} = \mathbb{R}^d$, $\mathcal{Y} = \{0, 1\}^m$
    - $\mathbf{x}_i$ de classe $c \Leftrightarrow y_i = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$ '1' en c^{eme} position
- Utilisation
  - temps d'apprentissage long autorisé
  - et/ou apprentissage *en ligne* requis
  - nombre de données assez grand
  - interprétabilité du modèle non nécessaire
  - possible bruit sur les données

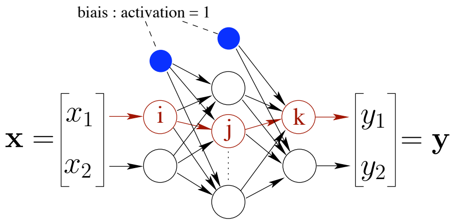
Perceptron multi-couches (1/9)

- Neurone formel





- Perceptron multi-couches



Perceptron multi-couches (2/9)

- Fonction implémentée : $f : \mathbb{R}^d \to \mathbb{R}^m$
- Erreur quadratique, en posant $\mathbf{o}_p = f(\mathbf{x}_p)$

$$E(\mathbf{w}) = \sum_{p=1}^{\ell} E_p(\mathbf{w}) \quad \text{avec} \quad E_p(\mathbf{w}) = \frac{1}{2}\|\mathbf{o}_p - \mathbf{y}_p\|^2 = \frac{1}{2}\sum_{q=1}^{m}(o_{pq} - y_{pq})^2$$

autres fonctions possibles (e.g. cross-entropy)

- Descente de gradient

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta \nabla_{\mathbf{w}} E(\mathbf{w}), \ \eta > 0$$

- Descente de gradient stochastique, à la présentation de l'exemple $(\mathbf{x}_p, \mathbf{y}_p)$

$$\mathbf{w}^{t+1} = \mathbf{w}^t - \eta_t \nabla_{\mathbf{w}} E_p(\mathbf{w}), \ \eta > 0$$

Perceptron multi-couches (3/9)

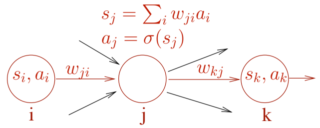
- η, η_t : pas d'apprentissage, pas d'apprentissage adaptatif
- Propriétés de η_t pour descente de gradient stochastique

$$\sum_t \eta_t \to \infty, \qquad \sum_t \eta_t < \infty$$

- Exercices
  - Montrer que pour η assez petit la descente de gradient permet de diminuer à chaque étape l'erreur E
  - De quelle forme peut être η_t ?

Perceptron multi-couches (4/9)

- Rétropropagation du gradient (j neurone caché, k neurone de sortie), activation logistique



  - dérivation en chaîne : $\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial s_j}\frac{\partial s_j}{\partial w_{ji}} = \frac{\partial E_p}{\partial s_j} a_i = \delta_j a_i$
  - Montrer que pour k et tous les neurones de la couche de sortie

$$\delta_k = \frac{\partial E_p}{\partial s_k} = -(y_{pk} - a_k)a_k(1 - a_k)$$

  - Montrer que pour j et tous les neurones de la couche cachée

$$\delta_j = a_j(1 - a_j) \sum_{k \in Succ(j)} \delta_k w_{kj}$$

Perceptron multi-couches (5/9)

Algo. apprentissage : 1 couche cachée, $\sigma(x) = (1 + e^{-x})^{-1}$

- Répéter jusqu'à convergence
  - pour chaque exemple $(\mathbf{x}_p, \mathbf{y}_p)$ faire
    - propager l'information
    - pour chaque neurone de sortie k calculer
      $\delta_k \leftarrow a_k(1 - a_k)(y_{pk} - a_k)$
    - pour chaque neurone caché j calculer
      $\delta_j \leftarrow a_j(1 - a_j) \sum_{k \in Succ(j)} \delta_k w_{kj}$
    - calculer le pas Δw_{ij}^t par
      $\Delta w_{ij}^t = -\eta \delta_j a_i$
    - mettre à jour les w_{ji} avec
      $w_{ji} \leftarrow w_{ji} + \Delta w_{ij}^t$
    - passer à l'itération suivante
      $t \leftarrow t + 1$

Perceptron multi-couches (6/9)

- Ajout d'un moment : mise à jour selon $w_{ji} \leftarrow w_{ji} + \Delta w_{ij}^t$ avec

$$\Delta^t w_{ij} \leftarrow -\eta \delta_j a_i + \alpha \Delta_{ij}^{t-1} w_{ij}, \quad 0 \leq \alpha < 1$$

  - accélère la convergence de l'algorithme
  - n'a pas d'effet sur la généralisation
- Couches cachées
  - rétropropagation du gradient marche pour n'importe quel nombre de couches cachées
  - couche cachée : changement de représentation (cf exercices)

Perceptron mutli-couches (7/9)

- Exemple d'autres fonctions d'erreurs
  - Erreur quadratique pénalisée (*weight decay*)

$$E(\mathbf{w}) = \sum_{p=1}^{\ell} E_p + \gamma \sum_{i,j} w_{ij}^2, \quad \gamma > 0$$

    - pénalise les réseaux avec des poids importants
    - exercice : ré-écrire l'équation de mise à jour de w_{ij} correspondante
  - Entropy croisée (cross-entropy), cas binaire $y_p \in \{0, 1\}$, 1 neurone de sortie

$$E(\mathbf{w}) = -\sum_{p=1}^{\ell} \left(y_p \log(o_p) + (1 - y_p) \log(1 - o_p)\right), \quad o_p = f(\mathbf{x}_p)$$

Perceptron multi-couches (8/9)

- Capacités de modélisation des perceptrons multi-couches [Cybenko, 1988, Cybenko, 1989]
  - toute fonction continue bornée peut être approchée avec une erreur arbitrairement petite par un PMC à une couche cachée
  - toute fonction peut être approchée avec une erreur arbitrairement petite par un PMC à deux couches cachées
- Importance de la régularisation [Bartlett, 1997]
  - contrôle de la taille des poids pour la généralisation

Perceptron multi-couches (9/9)

- Notions non présentées
  - partage de poids
  - sélection automatique de la structure
    - par ajout de neurones
    - par suppression de connexions
  - gradient conjugué
  - gradient exponentiel
  - ...
- Réseaux récurrents
- Réseaux RBF

Outline

Conclusion

Perceptron linéaire

- algorithmes d'apprentissage
- limitation du perceptron linéaire
- généralisation

Perceptron multi-couches

- neurone formel avec activation sigmoidale
- calcul de gradient par rétropropagation
  - qualité de l'apprentissage malgré les minima locaux
- gradient stochastique
- choix de l'architecture
- régularisation

Handwritten digits recognition

Bartlett, P. L. (1997).
For valid generalization the size of the weights is more important than the size of the network.
In *Adv. in Neural Information Processing Systems*, volume 9, page 134.

Cybenko, G. (1988).
Continuous valued neural networks with two hidden layers are sufficient.
Technical report, Department of Computer Science, Tufts University, Medford, MA.

Cybenko, G. (1989).
Approximation by superpositions of a sigmoidal function.
Mathematics of Control, Signals, and Systems, 2 :303–314.

Freund, Y. and Schapire, R. E. (1999).
Large Margin Classification Using the Perceptron Algorithm.
Machine Learning, 37(3) :277–296.

Friess, T., Cristianini, N., and Campbell, N. (1998).
The Kernel-Adatron Algorithm : a Fast and Simple Learning Procedure for Support Vector Machines.
In Shavlik, J., editor, *Machine Learning : Proc. of the 15th Int. Conf.* Morgan Kaufmann Publishers.

Schölkopf, B., Herbrich, R., and Smola, A. J. (2000).
A generalized representer theorem.
Technical Report NC-TR-00-081, NeuroCOLT.

Schölkopf, B. and Smola, A. J. (2002).
Learning with Kernels, Support Vector Machines, Regularization, Optimization and Beyond.
MIT University Press.