TME 6 - SVM

Exercice 1 – Module scikit-learn

Le module scikit-learn (`sklearn` : http://scikit-learn.org/stable/documentation.html) est le principal module d'apprentissage statistique de python. Le sous-module `sklearn.linear_model` propose en particulier une implémentation du perceptron, `sklearn.neighbors` une implémentation des k-nn, `sklearn.tree` des arbres de décision. Explorez rapidement ses implémentations, comparez les résultats de vos tmes précédents avec les implémentations de `sklearn`.

Exercice 2 – SVM et *Grid Search*

Le module `sklearn.svm` propose une implémentation des SVMs. Sur les jeux de données des précédents TMEs (2d artificiels et reconnaissance de chiffre), explorez plusieurs noyaux (linéaire, gaussien, polynomial) et plusieurs paramètrages des noyaux. Vous étudierez en particulier les frontières de décisions et les vecteurs supports. Comment évolue le nombre de ces derniers en fonction du noyau et de son paramètrage ? Est-ce normal ?
Afin de trouver les meilleurs paramètres, on opère par validation croisée sur une grille des paramètres (grid search). Pour les différents noyaux et différents nombre d'exemples d'apprentissage, opérez un grid search afin de trouver les paramètres optimaux. Tracez les courbes d'erreurs en apprentissage et en test. Les résultats sont-ils cohérents ?

Exercice 3 – Apprentissage multi-classe

Soit Y un ensemble de k classes $y_1, y_2, \ldots, y_k$. Deux méthodes courantes permettent de traiter des cas multi-classes à partir de classifieurs binaires :

- One-versus-one : on apprend $k(k-1)/2$ classifieurs permettant de départager tout couple de classes y_i, y_j ; la classification se fait en comptant le nombre de vote pour chaque classes.
- One-versus-all : k classifieurs sont appris correspondant à la séparation de chaque classe y_i de l'ensemble des classes $Y/\{y_i\}$. La classification se fait en considérant le classifieur qui obtient le meilleur score.

Testez ses deux méthodes sur la reconnaissance de chiffres.

Exercice 4 – String Kernel

Le string kernel est un noyau défini sur les mots Σ^* d'un alphabet Σ. Pour un mot s, on dénote $|s|$ la longueur du mot :$s = s_1, s_2, \ldots, s_{|s|}$, $s[i:j]$ le sous-mot $s_i, \ldots, s_j$ de s. On dit que u est une sous-séquence de s s'il existe une suite d'indices $\mathbf{i} = (i_1, \ldots, l_{|u|})$ avec $1 \leq i_1 < i_2 < \ldots < i_{|u|} \leq |s|$ tels que $u_j = s_{i_j}$ pour $j = 1, \ldots, |u|$, ce que l'on notera $u = s[\mathbf{i}]$. La longueur $l(\mathbf{i})$ de la sous-séquence dans s est $i_{|u|} - i_1 + 1$.
La projection utilisée est l'ensemble des coordonnées $\{\phi_u(s) = \sum_{\mathbf{i}:u=s[\mathbf{i}]} \lambda^{l(i)}\}$ $u \in \Sigma^*$ avec $\lambda \leq 1$.
Ainsi on peut définir le noyau $K_n(s,t) = \sum_{u \in \Sigma^n} < \phi_u(s), \phi_u(t) >= \sum_{u \in \Sigma^n} \sum_{\mathbf{i}:u=s[\mathbf{i}]} \sum_{\mathbf{j}:u=t[\mathbf{j}]} \lambda^{l(\mathbf{i})+l(\mathbf{j})}$.
Programmez un string kernel, visualisez la matrice de similarité sur des exemples de textes de différents auteurs, puis testez l'apprentissage.