Combinaison de Classifieurs

Méthodes pour la Construction d'Ensembles de Classifieurs

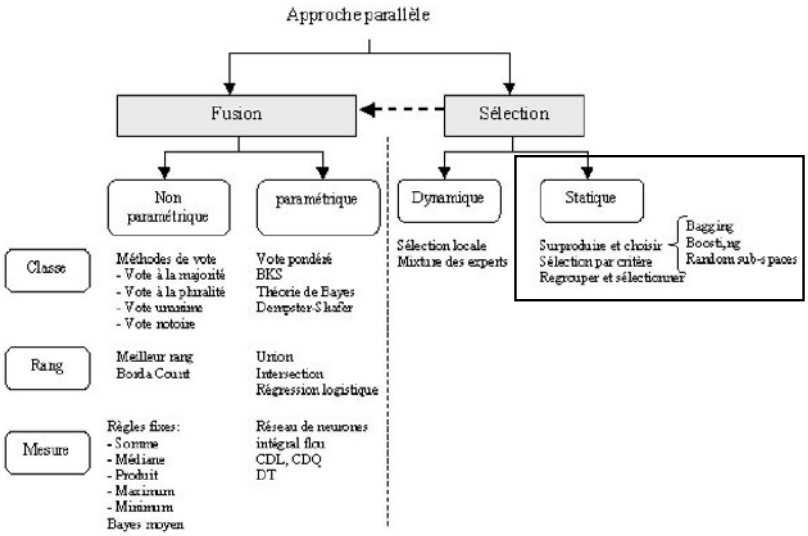
Laurent HEUTTE

Laurent.Heutte@univ-rouen.fr
http://www.univ-rouen.fr/psi/heutte

Plan du Cours

1. Qu'est-ce qu'un classifieur?
2. Pourquoi et comment les combiner?
3. Combinaison parallèle de classifieurs?
4. Méthodes pour la construction d'ensembles de classifieurs

Taxonomie des Méthodes de Combinaison Parallèle



Méthodes de Construction d'Ensembles

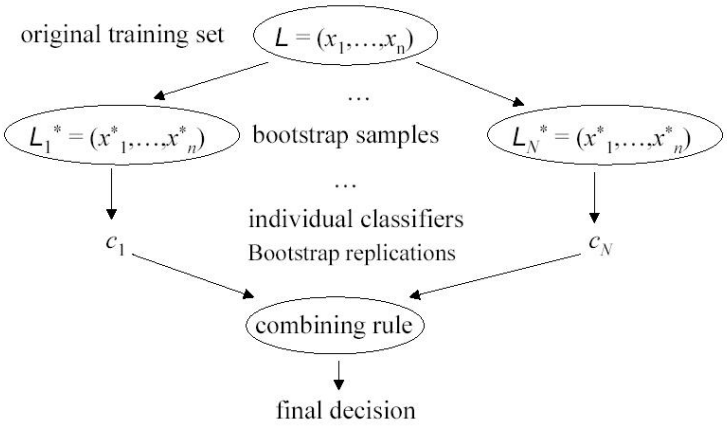
- L'efficacité d'un ensemble de classifieurs repose sur la combinaison de classifieurs complémentaires/divers
- De nombreuses approches ont été proposées pour construire des ensembles composés de classifieurs complémentaires:
  - Utiliser les connaissances a priori du problème et/ou de l'expert
    - Approche heuristique: faire varier le type de classifieur utilisé, l'architecture, les paramètres pour créer des classifieurs complémentaires
  - Injecter de l'aléatoire
    - Approche heuristique: l'algorithme de rétro-propagation du gradient dans un MLP est souvent exécuté plusieurs fois en utilisant différentes (aléatoires) conditions initiales (les poids synaptiques)
  - Manipuler les données d'apprentissage
    - Entrainer N classifieurs sur N bases différentes (cross-validated committees), bagging, boosting
  - Manipuler les caractéristiques d'entrée
    - Bruiter les caractéristiques d'entrée, Random subspace method
  - Manipuler les caractéristiques de sortie
    - Partitionner l'ensemble des classes de différentes façons (ECOC)

Manipuler les Données d'Apprentissage: Bagging (1)

- Bagging: méthode proposée par L. Breiman (1996) pour construire des ensembles de classifieurs, chaque classifieur étant entraîné sur une réplique différente de la base d'apprentissage
- Les différentes répliques de la base d'apprentissage sont obtenues par bootstrap:
  - Échantillon bootstrap: $L^*=(x_1^*, \ldots, x_n^*)$ est un échantillon aléatoire de taille n obtenu par tirage aléatoire avec remise dans l'échantillon original L=(x1, …, xn)
  - Chaque échantillon dans L peut apparaître dans L* zéro, une, deux, trois, … fois
- Idée de base: les différentes répliques L_i^* de la base d'apprentissage L sont très peu différentes de L mais suffisamment diverses pour obtenir des classifieurs différents qu'on va pouvoir combiner…

Manipuler les Données d'Apprentissage: Bagging (2)

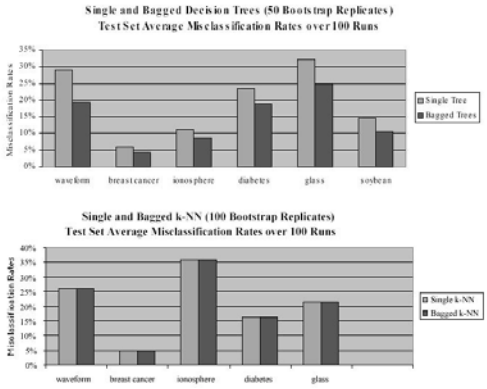
- Bagging: Bootstrap AGGregatING



Manipuler les Données d'Apprentissage: Bagging (3)

- Intérêt: bagging améliore la précision d'un classifieur instable
  - Classifieur instable: de faibles changements dans la base d'apprentissage entraînent de grands changements dans la construction du classifieur (frontières de décision)
  - Instables (MLP, arbres de décision, …) / stables (kppv, …)

Examples of bagging (Breiman, 1996)



Manipuler les Données d'Apprentissage: Bagging (4)

- ATTENTION: Bagging est une méthode pour construire des ensembles de classifieurs divers, pas une règle de combinaison
  - En théorie, n'importe quelle règle de combinaison peut être appliquée…
  - En pratique, vote majoritaire ou moyenne simple mais le problème est ouvert…
- Combien de répliques utiliser?
  - Les résultats expérimentaux montrent que 50 répliques sont en général suffisantes mais pas de résultats théoriques…
  - Attention aux temps de calcul en apprentissage…

Manipuler les Données d'Apprentissage: Boosting (1)

- L'algorithme AdaBoost (Freund and Schapire, 1995), le plus utilisé pour implémenter le boosting, cherche à produire des classifieurs "forts" (très précis) en combinant des instances "faibles" d'un classifieur donné
- AdaBoost construit de façon itérative un ensemble de N classifieurs complémentaires
  - Des classifieurs faibles sont ajoutés si nécessaire, et entraînés sur les échantillons que les classifieurs précédents n'ont pas correctement classés
  - Les classifieurs résultants sont combinés par un vote pondéré
- ATTENTION: AdaBoost est une méthode d'apprentissage d'ensemble, pas une méthode générale comme le Bagging pour construire des ensembles de classifieurs
  - Construction itérative de l'ensemble
  - Les exemples d'apprentissage mal classés à l'itération i ont une probabilité plus grande d'être sélectionnés à l'itération i+1
  - Le vote pondéré permet de tenir compte de la précision de chaque classifieur

Manipuler les Données d'Apprentissage: Boosting (2)

- Algorithme AdaBoost

Given n training samples $L = (x_1,\ldots,x_n)$

Initialize $D_1(i) = 1/n$, i=1,…,n; $L_1 = L$

For t=1,…,N:

- Train a classifier c_t on L_t
- Compute the error rate ε_t of c_t on L
- Set $\alpha_t = \frac{1}{2}\ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$
- Update: $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \begin{cases} e^{-\alpha_t} & \text{if } x_i \text{ is correctly classified} \\ e^{\alpha_t} & \text{if } x_i \text{ is misclassified} \end{cases}$

  where Z_t is a normalization factor so that D_t is a distribution

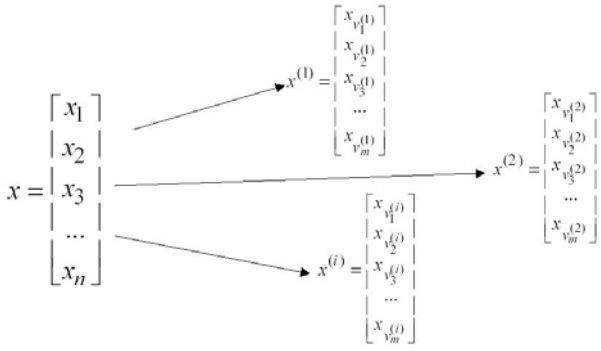
Combine the N classifiers by weighted majority voting, using the weights α_t

Manipuler les Données d'Apprentissage: Boosting (3)

- L'erreur sur la base d'apprentissage décroit à vitesse exponentielle si chaque classifieur est un peu meilleur que l'aléatoire (Freund and Schapire, 1997):
  - C'est-à-dire si Erreur(t) < Erreur* pour Erreur* < 0.5 (cas d'un classifieur faible)
- AdaBoost transforme de façon efficace un classifieur faible en classifieur fort (erreur faible)
- Les résultats expérimentaux montrent que AdaBoost ne sur-apprend pas (N=1000) même si le principe de l'algorithme est de se focaliser sur les exemples "durs"
- La capacité de généralisation est directement reliée au concept de classifieur maximisant la marge
  - Les exemples "durs" sur lesquels se focalise le boosting sont ceux qui ont la marge la plus faible (Freund and Schapire, 1997)

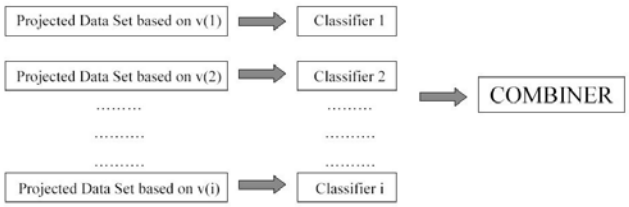
Manipuler les Caractéristiques d'Entrée: RSM (1)

- La méthode des sous-espaces aléatoires (Random Subspace Method) consiste à sélectionner aléatoirement un certain nombre de sous-espaces de l'espace de caractéristiques original et d'entraîner un classifieur sur chaque sous-espace (Ho, 1998)
- On projette tous les points de la base d'apprentissage dans des sous-espaces tirés aléatoirement



Manipuler les Caractéristiques d'Entrée: RSM (2)

- On combine ensuite les informations fournies par les classifieurs, chacun étant entraîné sur un sous-espace différent
  - On peut combiner avec une simple moyenne des sorties



- RSM est particulièrement bien adaptée aux espaces de caractéristiques de grande dimension contenant des caractéristiques redondantes donc ne souffre pas du problème de la malédiction de la dimensionalité
- Paramètres critiques pour les performances:
  - Le nombre de sous-espaces aléatoires
  - La dimensionalité des sous-espaces aléatoires

Remarques sur le Concept de Classifieur Faible

- Les méthodes telles que Bagging, Boosting, RSM utilisent des classifieurs faibles (performances un peu meilleures que 50%)
- Pourquoi chercher à utiliser des classifieurs faibles si l'on peut concevoir des classifieurs "forts"?
  - Parce que concevoir un classifieur fort en combinant plusieurs classifieurs faibles est souvent plus simple (simplicité pour le concepteur)
  - Entraîner un classifieur faible est souvent plus rapide qu'entraîner un classifieur fort
  - Les classifieurs faibles, avec une faible variance, sont souvent moins sensibles aux effets dus à l'apprentissage sur des bases de petite taille

Manipuler les Caractéristiques de Sortie (1)

- Idée de base:
  - Construire des classifieurs complémentaires en partitionnant l'ensemble des K classes de différentes façons
  - Entraîner un classifieur pour résoudre un sous-problème du problème à K classes (par exemple one-vs-all, all-vs-all,…)
  - Utiliser une "bonne" méthode de combinaison pour "récupérer" une décision dans l'espace des K classes
- Méthode efficace pour un grand nombre de classes: ECOC (Error-Correcting Output Coding) proposé par Dietterich et Bakiri, 1991
  - Le problème d'apprentissage est vu comme un problème de communication à travers un canal (caractéristiques d'entrée, exemples d'apprentissage, algo d'apprentissage):

  - Le canal est vu comme un canal bruité par lequel l'information "classe" est corrompue:

  - Utiliser des "codes" sur les classes pour se protéger du bruit

Manipuler les Caractéristiques de Sortie (2)

- Principe d'ECOC:
  - X: espace de caractéristiques (n-dimensional)
  - K classes: c_1, …, c_k
  - m fonctions disciminantes f_0, …, f_{m-1} telles que f_i : X dans {0,1}
  - A chaque classe c_j, on associe un codeword $b^{(j)}=b_0…b_{m-1}$ avec b_i dans {0,1}
  - On construit alors une matrice de décodage dont les lignes sont les classes c_j et les colonnes sont les bits b_i du codeword associé à la classe

Exhaustive ECOC for $c = 4$ classes ($L = 7$ classifiers)

	D_1	D_2	D_3	D_4	D_5	D_6	D_7
ω_1	0	0	0	1	0	1	1
ω_2	0	0	1	0	0	0	0
ω_3	0	1	0	0	1	0	1
ω_4	1	0	0	0	1	1	0

Manipuler les Caractéristiques de Sortie (3)

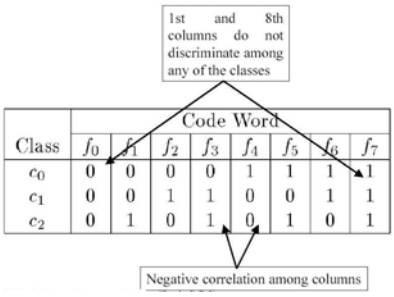
- Classification par ECOC
  - Chaque fonction booléenne f_i est entraînée pour chaque position du code correcteur d'erreur
  - Pour classer un nouvel exemple x, chaque fonction de l'ensemble f(x)={f_0(x), … f_{m-1}(x)) est évaluée pour produire une chaine de m bits
  - On calcule alors la distance au plus proche des K codes à l'aide d'une distance de Hamming

$$\text{class} = \text{argmin}_k \, d(b^{(k)}, f(x))$$

- Le processus de classification doit permettre de corriger les classes de sortie "corrompues" produites par le "canal" et restaurer la classe de sortie "correcte"
- Problème: comment générer un bon ECOC?
  - 2 propriétés à satisfaire: séparation des colonnes et séparation des lignes

Manipuler les Caractéristiques de Sortie (4)

- Séparation des lignes:
  - La distance entre chaque codeword et les autres codewords définit la distance de la classe considérée aux autres classes
- Séparation des colonnes:
  - Chaque fonction f_i doit être non corrélée avec chaque fonction f_j: la distance entre chaque colonne doit être suffisamment grande (au sens de la distance de Hamming).
  - Eviter les corrélations positives ou négatives entre colonnes.



Manipuler les Caractéristiques de Sortie (5)

- Méthodes pour construire ECOC:
  - En général, s'il y a K classes, il y a 2^{k-1}-2 colonnes utilisables après avoir éliminé les compléments et les colonnes toutes à 0 ou toutes à 1
- Comment construire un bon ECOC?
  - Codes exhaustifs (toutes les colonnes utilisables) 3 <= K <= 7
  - Sélection de colonnes à partir des codes exhaustifs
  - Optimisation combinatoire
  - BCH codes (méthodes algébriques dérivées de la théorie des corps de Galois)
- Les expériences montrent que:
  - Les performances ne dépendent pas de la taille des données d'apprentissage
  - Les performances ne dépendent pas d'une affectation particulière des codewords aux classes
  - ECOC fournit une mesure de confiance de classification qui est aussi bonne que les autres méthodes

Références Bibliographiques

- L. Breiman. *Bagging preditors*. Machine Learning, vol. 24, no. 2, pp. 123-140, 1996.
- Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. Proc. of 13th International Conference on Machine Learning, pp. 148-156, 1996.
- T.K. Ho. *The random subspace method for constructing decision forests*. IEEE Trans. on PAMI, vol. 20, no. 8, pp. 832-844, 1998.
- T.G. Dietterich and G. Bakiri. *Solving multi-class learning problems via error-correcting output codes*. Journal of Artificial Intelligence Research, vol. 2, pp. 263-286, 1995.
- L.I. Kuncheva. *Combining Pattern Classifiers. Methods and Algorithms*. John Wiley & Sons, New Jersey, 2004.
- …