Accélérer le Boosting avec partage de caractéristiques

Nicolas Honnorat, Hervé Le Borgne
CEA,LIST,Laboratoire d'Ingénierie de la Connaissance Multimédia Multilingue
18 route du Panorama, BP6, Fontenay-aux-roses F-92265, France.
nicolas.honnorat@cea.fr, herve.le-borgne@cea.fr

Résumé

Le Joint Boosting ou Boosting avec partage de caractéristiques est un algorithme permettant de construire des classifieurs multiclasses qui semble particulièrement adapté au cadre de l'indexation d'images. Cet algorithme a le défaut d'être assez lent, malgré les améliorations décisives apportées par son auteur [1]. Nous présentons ici trois manières d'accélérer les calculs sans perdre en performances.

Mots clefs

CBIR, Boosting , multiclasse , partage de caractéristiques, Fisher Linear Discriminant

1 Etat de l'art

Avec le développement des moteurs de recherche d'images l'annotation automatique d'images est devenue un domaine important. Les indexeurs extraient des images une série de caractéristiques qui leurs permettent de les classer [2]. De nombreuses méthodes de classifications existent [3, 4, 5]. Elles doivent en pratique pouvoir traiter rapidement des volumes importants d'images pouvant appartenir à plusieurs dizaines de classes souvent non exclusives et sont entraînées sur des corpus contenant quelques milliers d'images.

Parmi celles-ci, les techniques de Boosting construisent des classifieurs « forts » en combinant linéairement des classifieurs « faibles » , c'est-à-dire à peine meilleurs que le hasard [3]. La technique la plus connue est AdaBoost. Parmi les nombreuses variantes existantes, « Gentle AdaBoost » s'est révélée être la plus efficace sur les données réelles [6].

Gentle AdaBoost, comme de nombreuses autres méthodes, ne construit que des classifieurs binaires. Pour classifier des données appartenant à C classes on peut procéder de différentes manières : la méthode « un-contre-tous » consiste à entraîner un classifieur par classe, permettant de distinguer cette classe des autres et du fond ; la technique « un-contre-un » consiste à entraîner des classifieurs permettant de distinguer une classe d'une autre et à combiner leurs sorties par un vote à majorité. Toutes ces méthodes s'expriment dans le cadre plus général des codes correcteurs d'erreurs (ECOC, [7]) qui indiquent quelle classe assigner à une donnée connaissant les sorties d'un ensemble de classifieurs binaires.

Torralba et al. [1] proposent une variante de Boosting multiclasse dont les classifieurs faibles sont communs à plusieurs classifieurs forts un-contre-tous. Il s'agit de choisir à chaque étape le classifieur faible et la configuration de partage de ce classifieur permettant de réduire au maximum l'erreur de classification sur toutes les classes. Le classifieur multiclasse résultant est capable de traiter des classes non exclusives. A performances égales, le gain d'espace en mémoire par rapport à une batterie de classifieurs un-contre-tous est important, de même que la diminution du temps de calcul lors de la classification (phase de test). Néanmoins, malgré l'introduction d'une heuristique et d'une amélioration de calcul décisive dans [1] les temps d'apprentissage sont très longs pour des bases d'images courantes (plus de 48 heures pour 5000 images, 30 classes, 4000 caractéristiques extraites, sur un Pentium 4 cadencé à 3.19 GHz d'1 Go de RAM).

Dans cet article, après avoir présenté dans la section 2 les algorithmes de Boosting avec partage de caractéristiques développées dans [1] nous présentons dans la section 3 trois optimisations qui permettent de gagner en temps de calcul : nous commençons par décrire une nouvelle heuristique nettement plus rapide que celles utilisées jusqu'à présent, puis nous présentons deux techniques de sélection de caractéristiques qui ont donné de bons résultats avec cette heuristique et nous présentons enfin une variante utilisant d'autres classifieurs faibles susceptible de s'exécuter encore plus rapidement. La partie 4 présente tous nos résultats expérimentaux, notamment la vérification des gains en temps de calcul et du maintient des performances.

2 Joint Boosting

JointBoosting pour des stumps

Notons $H(v, c)$ le classifieur fort appris. Il fournit une confiance en l'appartenance de la donnée v à la classe c. Notons S un sous-ensemble de classe, représentant une configuration de partage, M le nombre de classifieurs faibles, C celui de classes et N celui

Algorithme 1 Joint Boosting

I. initialisations : $w_i^c = 1$, $H(v\ ,\ c) = 0, i = 1..N, c = 1..C$
II. pour $m = 1, 2, \ldots M$
  (1) recherche du meilleur classifieur faible h_m
  (2) ajout : $H(v\ ,\ c) := H(v\ ,\ c) + h_m(v\ ,\ c)$
  (3) modification des poids : $w_i^c := w_i^c e^{-z_i^c h_m(v\ ,\ c)}$

de données v_i, z_i^c les étiquettes des données, avec la convention $z_i^c = 1$ si v_i est de classe c et $z_i^c = -1$ sinon. Introduisons enfin un poids w_i^c pour chaque données i et chaque classe c. Le Joint Boosting se déroule alors selon l'algorithme 1.

Des stumps sont des classifieurs faibles de la forme suivante :

$$h_m^S(v\ ,\ c) = \begin{cases} a & \text{si } v_i^f > \theta \text{ et } c \in S \\ b & \text{si } v_i^f \leq \theta \text{ et } c \in S \\ k^c & \text{si } c \notin S \end{cases}$$

L'étape (1) du Joint Boosting est alors réalisée selon l'algorithme 2. Les paramètres a,b, et k^c de la stump sont donnés par les formules analytiques [1]. Le précalcul consiste en des sommes de valeurs, communes à toutes les configurations de partage.

Heuristiques

Une recherche exhaustive sur toutes les configurations de partage possibles (étape *ii*) étant en $O\left(2^C\right)$, il est nécessaire d'avoir recours à une heuristique limitant l'espace d'exploration. Torralba et al. [1] ont proposé une heuristique explorant C^2 configurations seulement, que nous avons conservée par la suite.

La complexité de cette heuristique étant en $O\left(C^3\right)$ et celle du précalcul en $O(NC)$, en notant F le nombre de caractéristiques (*features*) et N_θ le nombre de seuils précalculés pour chaque caractéristique, la complexité de l'algorithme présenté dans [1], que nous appellerons C^0 est finalement :

$$C^0 = O\left(MFN_\theta\left[NC + C^3\right]\right) \tag{1}$$

Algorithme 2 Recherche de la meilleure stump

(1) pour toute caractéristique f
  (a) pour tout seuil θ précalculé
    i) précalcul des erreurs et des paramètres des stumps
    ii) calcul pour chaque configuration de partage S :
      _ des paramètres de la meilleure stump $h_m^S(v\ ,\ c)$
      _ de l'erreur $J(S) = \sum_{c=1}^{C}\sum_{i=1}^{N} w_i^c\left(z_i^c - h_m^S(v\ ,\ c)\right)^2$

Algorithme 3 Nouvelle heuristique

(1a) précalcul du critère utilisé pour les caractéristiques
(1b) choix d'une configuration au hasard
(1c) pour $k = 1 \ldots K$
  i) recherche de la meilleure caractéristique
  ii) recherche du meilleur seuil
  iii) précalcul des erreurs et des paramètres des stumps
  iv) pour chaque configuration de partage S :
    calcul de la meilleure stump et de son erreur

3 Trois optimisations

3.1 Une meilleure heuristique

Nous nous sommes inspirés des techniques de relaxation par bloc pour proposer l'heuristique décrite par l'algorithme 3. Au lieu d'une recherche exhaustive de la meilleure stump, K étapes de réduction de l'erreur de classification selon chacun des trois paramètres variables successivement sont exécutées.

Remplacer l'étape *ii*) (la recherche d'un seuil optimal pour une caractéristique et une configuration données) par un tirage au hasard parmi les valeurs prises par la caractéristique sur les données d'entrée n'a pas réduit les performances.

Pour l'étape *i*), nous avons choisi d'utiliser le critère de Fisher. Pour une caractéristique f, notons S la configuration de partage courante et $\bar{S}$ son complémentaire dans $1 \ldots C$, notons m_X la moyenne des v_i^f appartenant à l'une des classes de X et V_X la variance correspondante. Ce critère s'écrit :

$$F = \frac{(m_S - m_{\bar{S}})^2}{V_S + V_{\bar{S}}}$$

Le précalcul de ce critère consiste à calculer pour chaque caractéristique la somme des valeurs (et de leur carré) prises sur chaque classe, d'où un $O(NFC)$. On obtient finalement une complexité C^1 moindre que C^0 de l'ordre de quelques N_θ :

$$C^1 = O\left(M\left[NFC + K\left(FC + NC + C^3\right)\right]\right) \tag{2}$$

3.2 Sélection des caractéristiques

Dans les situations courantes, les termes prépondérants pour le temps de calcul sont proportionnels au nombre de caractéristiques. Le réduire permet donc de construire les classifieurs plus rapidement.

Les méthodes de sélections de caractéristiques se déclinent selon trois type : « Filter » [8], « Wrapper » [9] et « Online » [10]. Les premières reposent sur des tests statistiques réalisés sur les données d'apprentissage pour réduire leur dimension ; elles

sont donc appliquées a priori et indépendamment de la classification. Les secondes, au contraire, réalisent des classifications sur des sous-ensembles de caractéristiques pour sélectionner celles qui doivent être conservées. Les dernières, enfin, permettent de choisir les caractéristiques en cours de calcul.

Pour réduire la dimension de nos données, nous avons testé quatre critères de sélection des caractéristiques de type « filter » puis vérifié les performances d'une méthode de type « wrapper » qui permet de gagner en temps de calcul.

Un bon critère de sélection.

Nous avons comparé quatre critères de sélection : deux critères ne prenant pas en compte les étiquettes des données et deux critères mesurant les disparités entre les ditributions des données des différentes classes.

Pour les premiers, nous avons conservé uniquement les caractéristiques de plus forte entropie ou de plus forte variance considérant qu'elles étaient les plus susceptibles de nous permettre de distinguer les données des différentes classes.

En cherchant à sélectionner les caractéristiques qui permettent à une classe de se distinguer des autres, nous avons testé deux autres critères. En notant D la divergence de Jensen-Shannon, F^f et p^f la fonction de répartition et la distribution de probabilités pour la caractéristique f sur toutes les classes et F_c^f et p_c^f celles sur la classe c seulement, les deux autres critères sont :

$$JS^f = \max_{c=1..C} \left[D\left(p^f, p_c^f\right) \right] \quad (3)$$

$$KS^f = \max_{c=1..C} \|F_c^f - F^f\|_\infty \quad (4)$$

Un « wrapper » simple et rapide.

Nous avons constaté que les performances limites sont généralement atteintes en un nombre d'étapes inférieur au triple du nombre de caractéristiques extraites (lui-même vingt fois supérieur au nombre de classes) et que les classifieurs obtenus ne prennent en compte qu'un peu moins de la moitié des variables. Ce résultat nous a amené à proposer un apprentissage en deux étapes : sur les éléments d'une partition de l'ensemble des caractéristiques dans un premier temps, puis sur celles qui ont été retenues à la première étape seulement par la suite. Cette manière de procéder serait jusqu'à quatre fois plus rapide.

3.3 Des discriminants de Fisher comme classifieurs faibles

Souhaitant conserver des classifieurs faibles qui soient rapides à utiliser tout en étant plus efficaces que les stumps nous avons testé des séparatrices hyperplanes de la forme :

$$h_m^S(v\ ,\ c) = \begin{cases} a & \text{si } \sum_j \lambda_j v_i^{f_j} > \theta \text{ et } c \in S \\ b & \text{si } \sum_j \lambda_j v_i^{f_j} \leq \theta \text{ et } c \in S \\ k^c & \text{si } c \notin S \end{cases}$$

Afin de toujours utiliser des caractéristiques de score de Fisher maximal et considérant ces nouveaux classifieurs comme des stumps sur les combinaisons linéaires de caractéristiques $\sum_j \lambda_j v^{f_j}$, nous avons choisi de prendre des discriminants de Fisher pondérés (WFLD) pour ces combinaisons, comme dans [11].

Les temps de calcul de ces discriminants étant assez longs (puisque de l'ordre du cube du nombre de dimensions combinées) nous nous sommes restreints à combiner les caractéristiques de meilleurs scores de Fisher seulement.

4 Résultats expérimentaux

4.1 Bases de données et caractéristiques

Bases d'images

Nous avons utilisé trois bases d'images :

- COIL-100 contient les images de cent objets photographiés sur fond noir sous 72 vues différentes, tous les 5 ° . L'apprentissage sur cette base a lieu sur 1, 2, 4, 8 ou 18 vues équiréparties, dont la vue à 0° , et le test est réalisé sur les images restantes. La base entière a donc 100 classes exclusives.
- VCDT contient 1827 photographies de vacances pour l'apprentissage et 1000 pour le test, pouvant appartenir à 17 classes non exclusives [12]
- PascalVOC 2008 [13] contient 2111 photographies pour l'apprentissage et 2221 pour la validation (que nous avons utilisées pour le test), pouvant contenir des objets appartenant à 20 classes

Caractéristiques extraites

Selon les bases nous avons extrait différents jeux de caractéristiques décrits dans [14]. Il s'agissait toujours d'histogrammes calculés après redimensionnement des images. Nous avons extrait jusqu'à 2030 caractéristiques, le jeu de meilleures performances en contenant 640.

Métriques

Chaque base est associée à un protocole de test dépendant de la campagne d'évaluation. Sur COIL-100 (classes exclusives) nous avons calculé des taux de classification (nous avons assigné à chaque image testée la classe de sortie maximale et mesuré la proportion de bonnes étiquettes) ; sur VCDT, nous avons calculé l'Equal Error Rate pour chaque classe et sa moyenne (l'EER est le taux d'erreur au point de la courbe ROC pour lequel les taux de faux positifs et de faux négatifs sont égaux ; plus il est faible, meilleure est la classification) ; sur PascalVOC, enfin, nous avons

calculé la précision moyenne pour chaque classe (notée MAP par la suite).

4.2 Validation des heuristiques

Temps de calcul et performances

Fixer K à 5 nous a donné les meilleurs résultats. Nous avons comparé notre nouvelle heuristique avec un perfectionnement de l'heuristique de [1] : au lieu de prendre des seuils répartis de manière homogène (ce qui donne des résultats catastrophiques dès que des valeurs anormales sont présentes) nous avons pris des quantiles des données d'apprentissage comme seuils et supprimé les valeurs en doublon pour ne pas effectuer de calculs inutiles. Nos données étant entières et distribuées de manière peu homogène cette variante s'est exécutée deux fois plus rapidement (3075 seuils ont été calculés au lieu de 6400 pour 10 objets, par exemple, pour fourir une « précision » correspondant à des déciles).

Comme le montre le tableau 1, les gains en temps de calcul sont bien du même ordre de grandeur que ceux prévus et même supérieurs. C'est en partie dû au fait que l'heuristique de départ demande un précalcul des seuils (simplifié lors du calcul de la complexité) alors qu'avec la nouvelle on entre directement dans la boucle principale. De manière générale plus le nombre d'images d'apprentissage par classe est faible et le nombre de classe est élevé et plus l'écart entre notre nouvelle heuristique et l'ancienne est marqué.

La figure 1 montre en complément que les performances à la limite (pour un grand nombre de classifieurs faibles) sont les mêmes et qu'elles sont atteintes en seulement deux à trois fois plus d'étapes de calcul. Notre heuristique permet donc bien d'atteindre les mêmes performances nettement plus rapidement.

expérience	gain observé	gain prévu
10 objets 3075 seuils	15.37	22.23
20 objets 2984 seuils	50.12	42.35
30 objets 3025 seuils	84.4	62.59

Tableau 1 – Gain en temps de calcul sur COIL-100, pour deux images d'apprentissage par objet et 640 caractéristiques, sur 500 étapes de calcul.

Performances en généralisation et nombre de classes

Sur COIL-100 entière nous avons d'abord comparé les performances de notre heuristique à celle d'une batterie de classifieurs un-contre-tous (construite avec recherche exhaustive sur toutes les caractéristiques pour 20 seuils possibles) contenant le même nombre de classifieurs faibles au total (ce qui signifie qu'elle classe aussi rapidement de nouvelles données). Notre

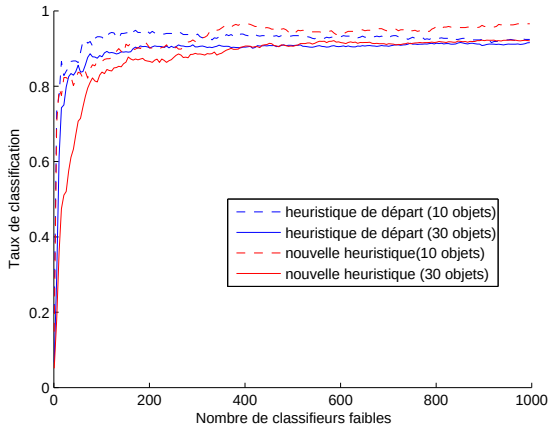

Figure 1 – *performance des heuristique sur COIL-100 (2 images d'apprentissage par objet)*

heuristique (cf. figure 2 pour 600 classifieurs) apparaît nettement meilleure.

Le Joint Boosting construisant les classifieurs plus lentement, nous avons construit des batteries plus grosses et des classifieurs multiclasses plus petits jusqu'à obtenir des temps d'apprentissage du même ordre.

Finalement, un classifieur multiclasse de 300 classifieurs faibles a présenté de bien meilleures performances qu'une batterie de 60 x 100 classifieurs faibles entraînée 3 fois plus lentement. Cette batterie a été construite aussi lentement qu'un classifieur multiclasse de 950 classifieurs faibles (dont les performances sont bien supérieures).

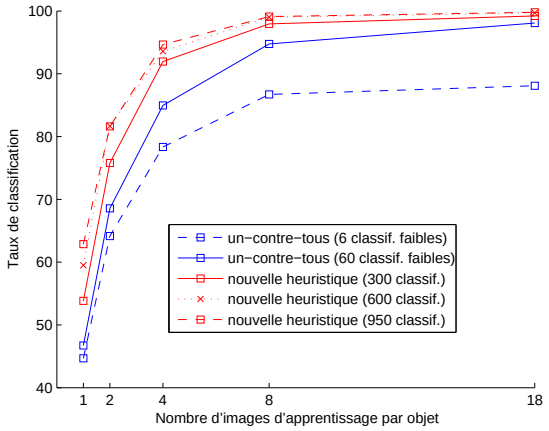

Figure 2 – *JointBoosting et un-contre-tous sur COIL-100 entière*

Par ailleurs, nous avons noté que les performances s'améliorent nettement lorsque le nombre de classes apprises en même temps augmente : nous avons choisi 12 fois 5 classes parmi les 20 de PascalVOC,

puis 10 fois 10 et enfin 10 fois 15 et construit avec notre nouvelle heuristique (à l'aide de toute la base) des classifieurs de 800 classifieurs faibles concernant uniquement ces classes. Le tableau 2 donne les comparaisons des précisions moyennes obtenues avec celles atteintes par un apprentissage sur la base entière. Ceci montre que notre heuristique possède de meilleures performances que la technique un-contre-tous et que ses performances augmentent avec le nombre de classes apprises simultanément.

nombre de classes	5	10	15
ratio des MAP (%)	85.92 ± 7.49	97.7 ± 2.22	99.28 ± 3.87

Tableau 2 – *performances sur des sous-ensemble de classes*

Comparaison aux autres techniques sur VCDT

En utilisant toutes les caractéristiques dont nous disposions, nous avons obtenus en 1500 étapes de calcul un EER moyen d'environ 22.5 % sur la base entière et de seulement 21.9 % après 6000 étapes de calcul. Ceci nous aurait placé en quatrième position parmi les 53 soumissions à la campagne ImageClef de 2008 [12]. Ces calculs ayant été menés uniquement à l'aide de caractéristiques globales sur les images, nous aurions probablement encore amélioré les performances en utilisant des « bag-of-features » en complément.

4.3 Sélection des caractéristiques

Comparaison des critères

Sur VCDT nous avons extrait 2030 caractéristiques et utilisé les quatre critères pour en écarter 1500. Nous avons lancé des apprentissages jusqu'à convergence et comparé les performances obtenues à celles d'apprentissages sur l'ensemble des caractéristiques.

Comme le montre le tableau 3, sélectionner les caractéristiques de forte entropie n'est pas une bonne idée ; les critères prenant en compte les classes des données sont bien meilleurs.

Le critère KS est le meilleur : l'EER n'a diminué que d'un point pour la forte réduction du nombre de variable imposée. Il donne donc une mesure fiable de l'importance des caractéristiques pour le Joint Boosting.

Apprentissage en deux étapes

Nous avons choisi quatre jeux de respectivement 640, 400, 432 et 433 caractéristiques que nous noterons *a*, *b*, *c* et *d*. Nous avons entraîné un classifieur durant 500 étapes sur chaque jeu de caractéristique, puis réalisé 8 apprentissages de 500 étapes : sur les jeux *ab*, *ac*, *cd* et *bcd* puis sur les réunions correspondantes de caractéristiques de ces jeux sélectionnées par les apprentissages sur chaque jeu. Le tableau 4 montre que nous n'avons pas observé de dégradation importante des performances du fait de la sélection de caractéristiques.

Critère	entropie	variance	JS	KS	hasard
EER (%)	31.1	25.4	25.2	22.8	23.5 ± 0.5

Tableau 3 – *sélection d'un quart des caractéristiques. Sur 2030 caractéristiques on obtient un EER de 22.38% en 1500 étapes de calcul (21.86 % en 6000 étapes). 11 sélections au hasard ont été réalisées*

Jeux	ab	ac	cd	bcd
sans sélection	22.95	23.51	27.43	25.49
après sélection	23.83	23.86	27.45	25.06

Tableau 4 – *EER (en %) obtenus avec ou sans sélection des caractéristiques par JointBoosting. la différence (*0.2% *en moyenne) est inférieure aux variations d'EER observées en cours d'apprentissage (*0.4% *environ)*

4.4 Discriminants de Fisher

Pour tester les WFLD nous avons réalisé des apprentissages sur VCDT pour les 2030 caractéristiques dont nous disposions. Nous avons testé différentes combinaisons : des 3, des 6, des 20 ou des 40 caractéristiques de meilleurs scores de Fisher. Les résultats sont présentés dans le tableau 5.

Nb étapes	Sans	F-3	F-6	F-20	F-40
200	24.86	24.82	24.24	24.16	23.42
700	23.35	22.97	22.48	23.09	-
1200	22.55	22.63	22.68	-	-

Tableau 5 – *EER obtenus avec les discriminants de Fisher (en %) en fonction du nombre d'étapes. La baseline est un apprentissage sans discriminant de Fisher, l'heuristique « F-x » correspond à un WFLD des x caractéristiques de score de Fisher maximal.*

Les performances à la limite sont atteintes d'autant plus rapidement que le nombre de caractéristiques combinées est grand comme on pouvait s'y attendre.

Le gain en nombre d'étapes de calcul a cependant été compensé durant nos expériences par une perte en temps de calcul à chaque étape : dans le meilleur des cas (F-6) les performances limites ont été atteintes en trois fois moins d'étapes, mais les étapes ont duré presque trois fois plus longtemps...

Nous avons par la suite vérifié l'augmentation du score de Fisher obtenue grâce aux WFLD et l'avons traduite en terme d'erreur de classification. Les caractéristiques combinées se sont révélées à peine meilleures que celles de départ. Nous pensons que c'est la raison principale de ce résultat mitigé et que sur d'autres données, où la combinaison de 2 ou 3 caractéristiques suffirait à améliorer nettement les

scores, un gain en temps de calcul d'un facteur 2 reste envisageable par cette méthode.

5 Remarques et conclusion

Nous avons présenté une heuristique nettement plus rapide que celles utilisées jusqu'à présent et vérifié que ses performances restent à la hauteur. Elle s'est révélée meilleure que la technique classique « un-contre-tous » et a donné des résultats intéressants sur VCDT [12]. Nous avons ensuite présenté deux manières simples de sélectionner les caractéristiques extraites des images et nous avons étudié un autre type de classifieurs faibles.

En nous permettant de gagner de manière significative en temps de calcul à performances égales, tous ces perfectionnements permettent de traiter des bases d'images plus importantes, jusqu'alors inaccessibles.

Les performances du Joint Boosting augmentant avec le nombre de classes l'heuristique que nous avons présentée, qui est d'autant plus intéressante que le nombre de classes est élevée, démontrera toute son utilité en pratique.

Les techniques de sélection de caractéristiques que nous avons présentées, enfin, permettraient de paralléliser une partie des calculs : les jeux de caractéristiques seraient réduits indépendamment les uns des autres avant de servir à construire un classifieur fort multiclasse par Joint Boosting.

6 Remerciements

Nous remercions la Direction Générale des Entreprises pour son financement via les pôles Systematic (projet POPS) et Cap Digital (projet Mediatic).

Nous tenons également à remercier les relecteurs de cet article pour leurs conseils précis.

Références

[1] A. Torralba, K. P. Murphy, et W.T. Freeman. Sharing visual features for multiclass and multiview object detection. *IEEE T. PAMI*, 29(5) :854–869, mai 2007.

[2] R. Datta, D. Joshi, J. Li, et J.Z. Wang. Image retrieval : Ideas, influences and trends of the new age. *ACM Transactions on Computing Surveys*, 40(2), avril 2008.

[3] R.E. Shapire. The boosting approach to machine learning : an overview. Dans *MSRI Workshop on Nonlinear Estimation and Classification*, décembre 2001.

[4] B. Schölkopf, A. Smola, R. Williamson, et P. L. Bartlett. New support vector algorithms. *Neural Computation*, pages 1207–1245, décembre 2000.

[5] Leo Breiman. random forests. *Machine Learning*, 45 :5–32, janvier 2001.

[6] R. Lienhart, A. Kuranov, et V. Pisarevsky. Empirical analysis of detection cascades of boosted classifiers for rapid object detection. Dans *DAGM 25th Pattern Recognition Symposium*, 2003.

[7] Dietterich et Bakiri. Solving multiclass learning problems via ecocs. *Journal of AI Research*, 2 :263–286, 1995.

[8] Isabelle Guyon. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3 :1157–1182, 2003.

[9] Avrim L. Blum et Pat Langley. Selection of relevant features and examples in machine learning. *Artificial Intelligence*, 97 :245–271, 1997.

[10] Simon Perkins et James Theiler. Online feature selection using grafting. Dans *In International Conference on Machine Learning*, pages 592–599. ACM Press, 2003.

[11] Ivan Laptev. Improvements of object detection using boosted histograms. Dans *In Proc. BMVC*, volume 3, pages 949–958, Edinburgh, UK, 2006. BMVC.

[12] Thomas Deselaers et Allan Hanbury. The visual concept detection task in ImageCLEF 2008. Dans Carol Peters, Danilo Giampiccol, Nicola Ferro, Vivien Petras, Julio Gonzalo, Anselmo Peñas, Thomas Deselaers, Thomas Mandl, Gareth Jones, et Mikko Kurimo, éditeurs, *Evaluating Systems for Multilingual and Multimodal Information Access – 9th Workshop of the Cross-Language Evaluation Forum*, Lecture Notes in Computer Science, Aarhus, Denmark, Septembre 2008 (printed in 2009).

[13] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, et A. Zisserman. The PASCAL Visual Object Classes Challenge 2008 Results. http ://www.pascal-network.org/challenges/VOC/voc2008/ workshop/index.html.

[14] M Joint, P-A Moellic, P. Hede, et P. Adam. Piria : A genaral tool for indexing, search and retrieval of multimedia content. Dans *SPIE Electronic Imaging*, San Jose, California, USA, janvier 2004.