PI INF442 : Détection par « boosting »

Q1. Nous avons procédé, comme le suggère l’énoncé, par récurrence, en utilisant un principe de programmation dynamique, afin d’obtenir l’image intégrale à partir d’une matrice correspondant à une image.

Q2. Nous n’avons considéré ici que des caractéristiques « carrés » (de même nombre de pixels en longueur et en largeur). On obtient alors, pour une zone de 92 \* 112, un côté minimal de 8 pixels et un incrément de position et de taille de 4 pixels \*\*\*\* caractéristiques :

* O(n^3)
* Implémentation de façon parallèle avec MPI :
  + Chaque processus calcule entièrement l’image intégrale pour pouvoir calculer des caractéristiques en temps constant.
  + Chaque processus calcule, en fonction de son rang, certaines caractéristiques de l’image, de sorte que toutes les caractéristiques de l’image soient calculées par un unique processus.
  + Le processus racine reçoit (MPI\_Recv) de la part de tous les processus dans l’ordre de leur rang, un tableau des caractéristiques qu’ils ont calculées, et les agrège dans cet ordre dans un vecteur d’entier.
  + Ledit vecteur est retourné (on notera que quel que soit le processus racine, les caractéristiques retournées sont bien toujours dans le même ordre.

Q3. Dans le but d’entraîner nos classifieurs faibles de manière parallèle, nous avons procédé de la façon suivante avec MPI (avec p processus) :

* Nous récoltons tout d’abord les titres des images présentes dans la base d’apprentissage, et nous les stockons dans un vecteur de string pour pouvoir tirer consécutivement K images au sort.
* w1 et w2, vecteurs contenant les w1i et w2i, sont initialisés.
* Pour k variant de 1 à K, nous tirons une image au sort (la même pour chaque processus grâce à l’initialisation commune de la seed).
* Au bout de chaque série de p passages dans la boucle, les p processus ont appliqué une et une seule fois l’algorithme décrit dans la question précédente en tant que processus racine. Chaque processus dispose donc d’un vecteur complet de caractéristiques correspondant à une image distincte qui lui permet de calculer (suivant la logique décrite dans l’énoncé) la variation incrémentale de w1 et w2. On met alors à jour les valeurs de w1 et w2 pour tous les processus à l’aide d’un MPI\_Allreduce. Rq appretissage par bloque
* Les valeurs de w1 et w2 obtenues à l’issue des K tirages au sort correspondent à nos classifieurs faibles entraînés.

Etude de la convergence des paramètres en fonction de K et epsilon : nous avons choisi de définir la convergence des paramètres w1 et w2 en étudiant l’évolution de la quantité |w1k+1 – w1k|/|w1k| où |w1k| = sqrt(sum\_i^{nbC} {w1ki\*\*2})

Pour w2, n’importe quelle valeur de K supérieure à 1000 et d’epsilon comprise entre 0.5 et 5 semble convenir.

Pour w2, aucun choix de K et w2 ne semblent pleinement satisfaisant.

Pour epsilon suffisamment grand -> epsilon joue plus (w2 n’est plus dans le game)

Epsilon << 1 -> w1 = 1 et w2 = 0 (w1 n’évolue plus)

Q4. On met en œuvre ma méthode de boosting des classifieurs faibles qui permet de constituer un classifieurs plus performant que les classifieurs faibles, puisqu’il réalise la combinaison linéaire des plus pertinents d’entre eux pour la classification. On utilise à nouveau MPI pour obtenir le classifieur F.