DÉTECTION PAR "BOOSTING"

PI - INF442

Mai 2018

Romain FOUILLAND et Antoine GROSNIT

1 INFORMATIONS PRATIQUES

Pour faire fonctionner notre programme, il suffit suivre les instructions disponibles dans le fichier readme.md ou de se rendre sur la page GitHub du projet : https ://github.com/romainfd/INF442-Detection-by-boosting-/.

2 CARACTÉRISTIQUES EMPLOYÉES

2.1 Calcul de l'image intégrale

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

2.2 Création des classifieurs faibles

Q2. Nous n'avons considéré ici que des caractéristiques « carré » (ayant le 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, 4466 caractéristiques (et de manière générale il y en a $O(n^3)$ où n est la longueur de l'image) :

$$\begin{aligned}\sum_{i=2}^{22}(28-i)(23-i) &= \sum_{i=1}^{21}(5+i)i \\ &= 5\sum_{i=1}^{21} i + \sum_{i=1}^{21} i^2 \\ &= 5 \times \frac{21 \times 22}{2} + \frac{21 \times 22 \times 43}{6} \\ &= 4466\end{aligned}$$

Cette valeur a pu être confirmée par la fonction *nbCaractsTot* du programme.

Le calcul des caractéristiques se fait 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).

3 CRÉATION D'UN CLASSIFIEUR

3.1 Création des classifieurs faibles

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.

— w_1 et w_2, vecteurs contenant les w_{i1} et w_{i2}, 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*).

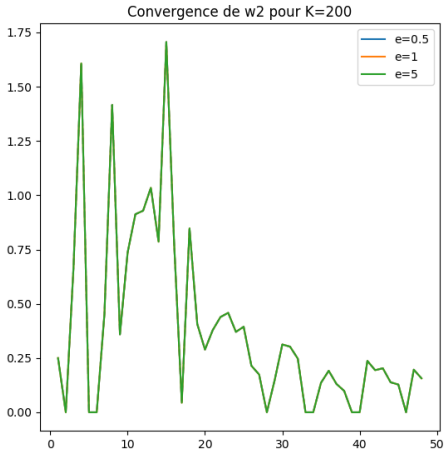
— Au bout de chaque série de p passages dans la boucle, les p processus ont appliqué une et une seule fois l'algorithme décrit dans la question précédente en tant que processus racine. Chaque processus dispose donc d'un vecteur complet de caractéristiques correspondant à une image distincte qui lui permet de calculer (suivant la logique décrite dans l'énoncé) la variation incrémentale de w_1 et w_2. On met alors à jour les valeurs de w_1 et w_2 pour tous les processus à l'aide d'un $MPI\_Allreduce$. Il faut noter que, contrairement à l'implémentation séquentielle pour laquelle les valeurs de w_1 et w_2 auraient été mises à jour à chaque boucle, ici nous ne procédons à cette mise à jour que tous les p passages en agrégeant en une fois les p modifications calculées en parallèles.

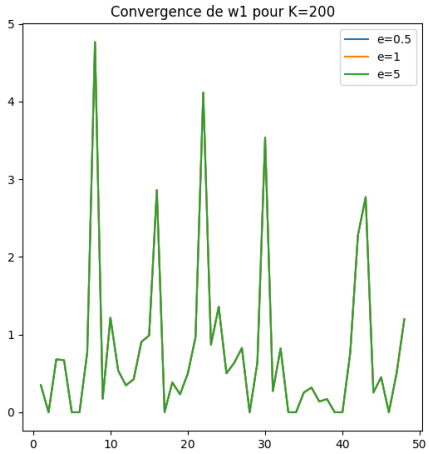
— Les valeurs stockées dans w_1 et w_2, obtenues à l'issue des K tirages au sort, déterminent nos classifieurs h_i.

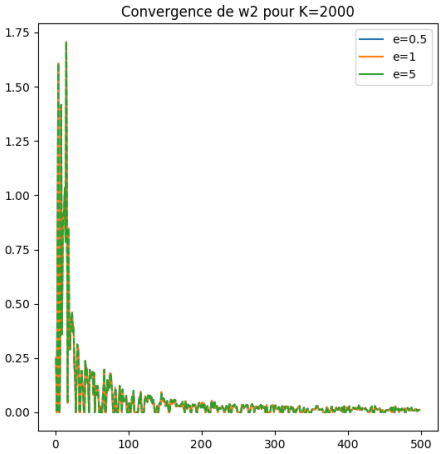
Convergence des paramètres en fonction de K et ϵ :
Nous avons considéré l'évolution des quantités $\frac{\|(w_1^{k+1}-w_1^k)\|}{\|w_1^k)\|}$ et $\frac{\|(w_2^{k+1}-w_2^k)\|}{\|w_2^k)\|}$ où $\|.\|$ désigne la norme euclidienne et w_i^k désigne le vecteur w_i à la boucle k. On mesure ainsi l'évolution relative des paramètres au cours de leur K *mod* p mises à jour.

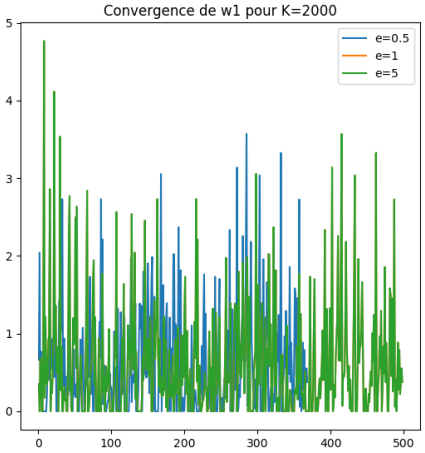
3.2 Choix de K

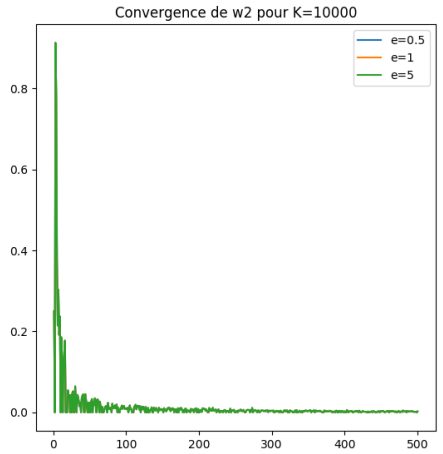
En traçant l'évolution de $\frac{\|(w_2^{k+1}-w_2^k)\|}{\|w_2^k)\|}$ pour $K = 200$, 2000 et 10000, on constate que cette quantité converge pour différentes valeurs de ϵ et que le comportement de $\frac{\|(w_1^{k+1}-w_1^k)\|}{\|w_1^k)\|}$ est non convergent pour ces trois valeurs de K. Il semble donc judicieux de prendre K de l'ordre de 2000 pour limiter le nombre de boucles tout en ayant une convergence suffisante de w_2.

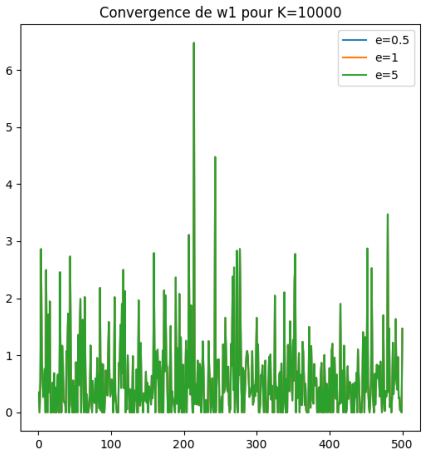










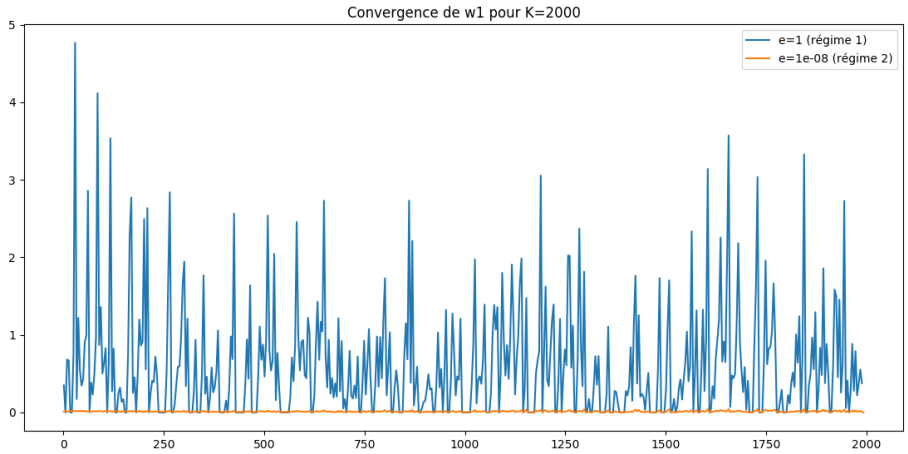


3.3 Choix de ϵ

En raisonnant sur les ordres de grandeurs des paramètres (et notamment de $X_{ki} \sim 10000$) nous avons distingué deux régimes distincts :

— Pour $\epsilon \gg \frac{1}{10000}$: w_{i2} est de l'ordre de ϵ et w_{i1} est de l'ordre de $\epsilon \times X_i \sim 10000 \times \epsilon$. Ainsi, w_{2i} ne joue globalement pas de rôle dans la classification qui est déterminée uniquement par le signe de $w_{i1} \gg w_{i2}$. De ce fait, la valeur particulière de ϵ dans ce régime a très peu d'impact sur le résultat. Etant donnée la variation des valeurs des X_{ki} et de la catégorie de l'image tirée au sort à chaque boucle, la quantité $\frac{\|(w_1^{k+1} - w_1^k)\|}{\|w_1^k)\|}$ reste de l'ordre de l'unité quel que soit K, et on n'observe pas de convergence.

— Pour $\epsilon \ll \frac{1}{10000}$: w_{i2} reste de l'ordre de ϵ et w_{i1} est de l'ordre de l'unité (son initialisation). Là encore, w_{2i} ne joue globalement pas de rôle dans la classification qui est déterminée uniquement par le signe de $w_{i1} \gg w_{i2}$ (car $1 \gg \epsilon$). De ce fait, la valeur particulière de ϵ dans ce régime a très peu d'impact sur le résultat (le classifieur est déterminé par la valeur de w_{i1} qui varie très faiblement autour de sa valeur initiale, indépendante de ϵ - ce qui tend d'ailleurs à rendre l'entraînement superflu). On observe donc une relative stabilité de w_1 dans ce régime ($\frac{\|(w_1^{k+1} - w_1^k)\|}{\|w_1^k)\|} \ll 1$).

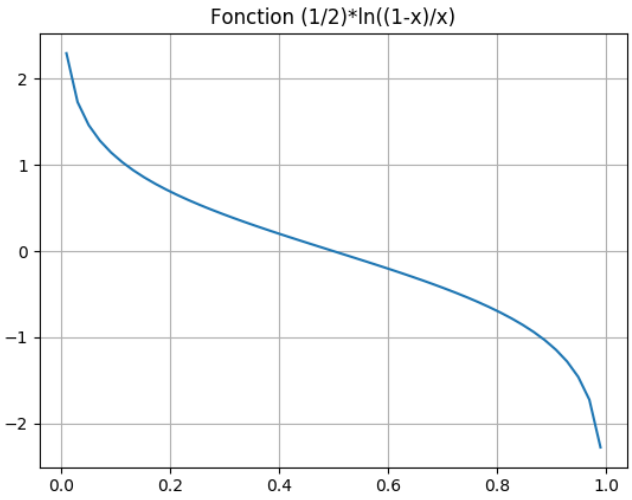


Nous essayons donc dans la suite de nous placer à la frontière entre ces deux régimes.

3.4 Boosting des classifieurs faibles

Q4. Dans cette partie, nous cherchons une combinaison linéaire des classifieurs faibles entraînés dans la première partie.
Pour cela nous itérons à chaque étape sur toutes les images et tous nos classifieurs afin de déterminer le classifieur avec la plus faible erreur pondérée. Nous avons, à ce titre, utilisé MPI_MINLOC pour déterminer l'indice du meilleur classifieur. A chaque étape, nous avons ensuite mis à jour les poids des images pour rendre moins efficace le classifieur faible sélectionné au cours de cette étape (on a baissé le poids des images qu'il classait déjà correctement) afin de sélectionner d'autres classifieurs par la suite.
L'idée est de finalement obtenir une combinaison de classifieurs qui sont efficaces sur l'ensemble des images.

Pour mettre en oeuvre cette méthode, nous avons utilisé la fonction $\frac{1}{2}\log(\frac{1-x}{x})$ qui nous permet d'obtenir une nouvelle pondération en fonction de l'erreur pondérée du meilleur classifieur.



Nous voyons que cette fonction change de signe autour d'une erreur de 0.5 et cela a conduit à la limitation du nombre de classifieurs faibles sélectionnés par notre algorithme. En effet, un classifieur avec une erreur un peu plus faible que cette valeur était sélectionné puis, à cause de la modification des poids, il devenait moins efficace et passait à une erreur supérieure à 0.5 et donc, lorsqu'il était repris, on améliorait cette fois ses performances, le faisant à nouveau passer sous la barre de 0.5 d'erreur pondérée. Ce sueil de l'erreur de 0.5 nous a empêché d'obtenir une combinaison riche de classifieurs puisqu'en général nous avons obtenu moins d'une dizaine de classifieurs, indépendamment de la valeur de N.

4
TEST DU CLASSIFIEUR FINAL

Q5. Notre classifieur final était très peu sensible à θ. Les paramètres $K = 2000$, $\epsilon = 0.1$, $n = 200$, $N = 20$, $\theta = 0$ et une normalisation de 0.001 nous ont néanmoins permis d'obtenir un $F - score$ de 0.444 sur toutes les images du dossier test.

Nous avons cependant été limités dans l'apprentissage par le nombre d'images disponibles, et notamment le fait qu'elles soient identiques dans les 3 dossiers (app, test et dev). Nous aurions peut-être dû passer à des caractéristiques rectangulaires pour obtenir davantage d'informations. Enfin, nous pourrions envisager de diviser la valeur de chaque caractéristique par sa taille afin de normaliser toutes les valeurs et de pouvoir ainsi les comparer entre elles et qu'elles soient toutes de l'ordre de l'unité (pour que w_1 et w_2 puissent avoir des influences du même ordre de grandeur (cf. 3.3 Choix de).