Théorie et Codage de l'Information

Dejan PARIS

13 Novembre 2020

Table des matières

1 Introduction

Ce rapport rend compte d'un travail de codage, ayant pour but la compression optimisée d'images. Il comprend cinq parties :

- L'implémentation de fonctions permettant d'évaluer l'efficacité des codages mis en place dans la suite.
- Une première approche de la compression d'images en noir et blanc, à l'aide d'un RLE.
- Des essais de compression d'image à perte, et l'examen de la relation entre taux de compression et distorsion de l'information.
- L'implémentation et le test du calcul de l'information mutuelle entre deux jeux de données, nécessaire pour la suite.
- La sélection de caractéristiques à l'aide de la méthode mRMR, et la mise au point d'un classeur de données basé sur cette sélection.

L'intégralité du code écrit à ces effets se trouve dans les fichiers *chef.py* (fonctions et classes) et *test.py* (visualisation des résultats).

2 Calculs d'entropie

L'entropie est une mesure essentielle dans ce projet : elle permet d'évaluer la dispersion de l'information dans un jeu de données, d'implémenter des algorithmes de compression et d'en calculer l'efficacité.

La fonction *entropy* remplit ce rôle, et illustre aussi un modèle récurrent dans ce compte-rendu. Comme la plupart des fonctions qui vont suivre, elle prend en paramètre un vecteur de données X et un ensemble (optionnel) dans lequel X prend ses valeurs. Passer cet ensemble en paramètre évite simplement à l'algorithme de le calculer lui-même.

On peut en vérifier la cohérence en traçant l'entropie en fonction de la proportion d'un élément dans X suffisamment long (par exemple la proportion de 0 dans un vecteur binaire).
C'est le rôle de *draw_h* qui calcule l'entropie d'un vecteur numérique, en faisant varier la proportion de 0. Comme on peut le prévoir, l'entropie est maximale pour l'équiprobabilité, i.e lorsque
P[X=0] = 1/n où n est le nombre de valeurs dans *setX*.

3 Compression à mémoire

3.1 Source à mémoire

Un système intéressant pour le codage de source binaire est de considérer une source à mémoire. Ici, on analyse l'image d'un texte (en noir et blanc) comme une chaîne de Markov d'ordre 1 : on considère les probabilités de rencontrer un pixel noir/blanc en fonction du pixel précédent.

La fonction *estimate_M* permet d'estimer ces probabilités. Pour un texte, en codant un pixel blanc comme 1 et noir comme 0, la probabilité *p11* (blanc après blanc) est extrêmement élevée, ce qui permet de définir un codage approprié.

Le calcul de l'entropie d'une telle source nous permet de constater le gain en efficacité potentiel. Pour une mémoire d'ordre 1, l'entropie H est :

$$H = -p_0 * (p_{00} * log_2(p_{00}) + p_{01} * log_2(p_{01})) - p_1 * (p_{10} * log_2(p_{10}) + p_{11} * log_2(p_{11}))$$

En calculant l'entropie d'un scan de 8 415 000 de pixels, on obtient 0,14 bits/pixel en considérant la source avec mémoire, contre 0,28 sans mémoire.

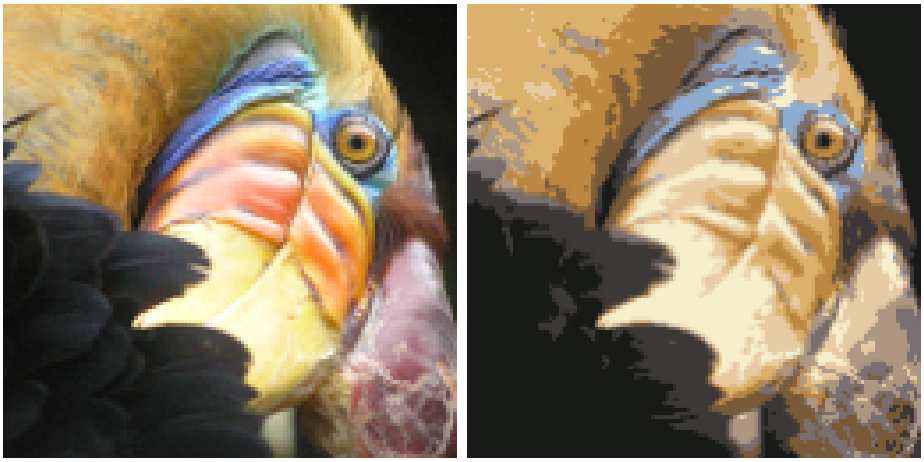
3.2 Run-length-encoder

Etant donné la valeur de *p11*, le codage par RLE (run-length-encoder) apparaît approprié pour ce genre d'image. Il s'agit de coder les "runs", suites ininterrompues de blanc ou de noir. La fonction *run_length_encoder* implémente ce codage.

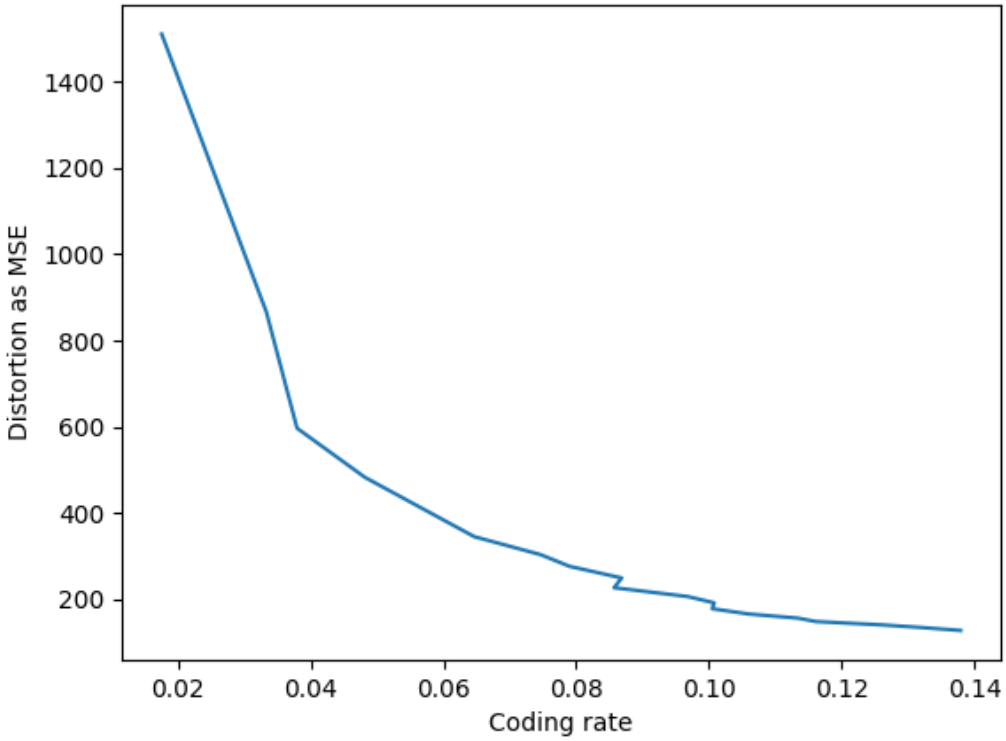
Avec l'image mentionnée, on obtient une longueur de code (calculée par *code_length*) de 855 374 bits, soit un gain de presque 90% par rapport au codage pixel par pixel.

4 Compression à perte

On souhaite maintenant examiner les résultats de la compression à perte (i.e une simplification de l'information pour un codage plus efficace).
Pour cela, on utilise la classe *KMeans* de *sklearn*. La fonction *k_means* englobe la compression : on lui passe l'image à compresser (aplatie en une ligne de pixels) et le nombre de clusters (couleurs "moyennes" de l'image) souhaité en sortie.
Ainsi, on réduit le nombre de couleurs pour une compression plus simple : ci-dessous, on peut constater la perte de qualité entre l'image de test (128*128 pixels) et celle créée par *KMeans* avec 10 clusters :



sklearn nous permet aussi de calculer le niveau de distorsion de l'image originelle, à l'aide de *mean_squared_error*. En traçant cette distorsion en fonction du taux de compression (obtenu avec notre algorithme RLE), on remarque une évolution exponentielle : plus on gagne en efficacité, plus la distorsion augmente rapidement.



Avec le codage minimal (2 clusters, donc l'équivalent d'une image noir et blanc), on obtient un taux de compression similaire à la partie précédente, environ 2%.

5 Quantité d'information mutuelle

Pour la dernière approche que constitue la sélection de caractéristiques, nous avons besoin de l'information mutuelle entre deux vecteurs de même taille. C'est le rôle de la fonction *mutual_info_quantity*, qui calcule l'information mutuelle à l'aide de la matrice de probabilités conjointes de X et Y. Cette matrice, calculée par *estimate_joint_P*, permet le calcul des probabilités individuelles des deux vecteurs. Ensuite, le formule :

$$I(X,Y) = \sum P(X=x, Y=y) * log_2(\frac{P(X=x, Y=y)}{P(X=x) * P(Y=y)})$$

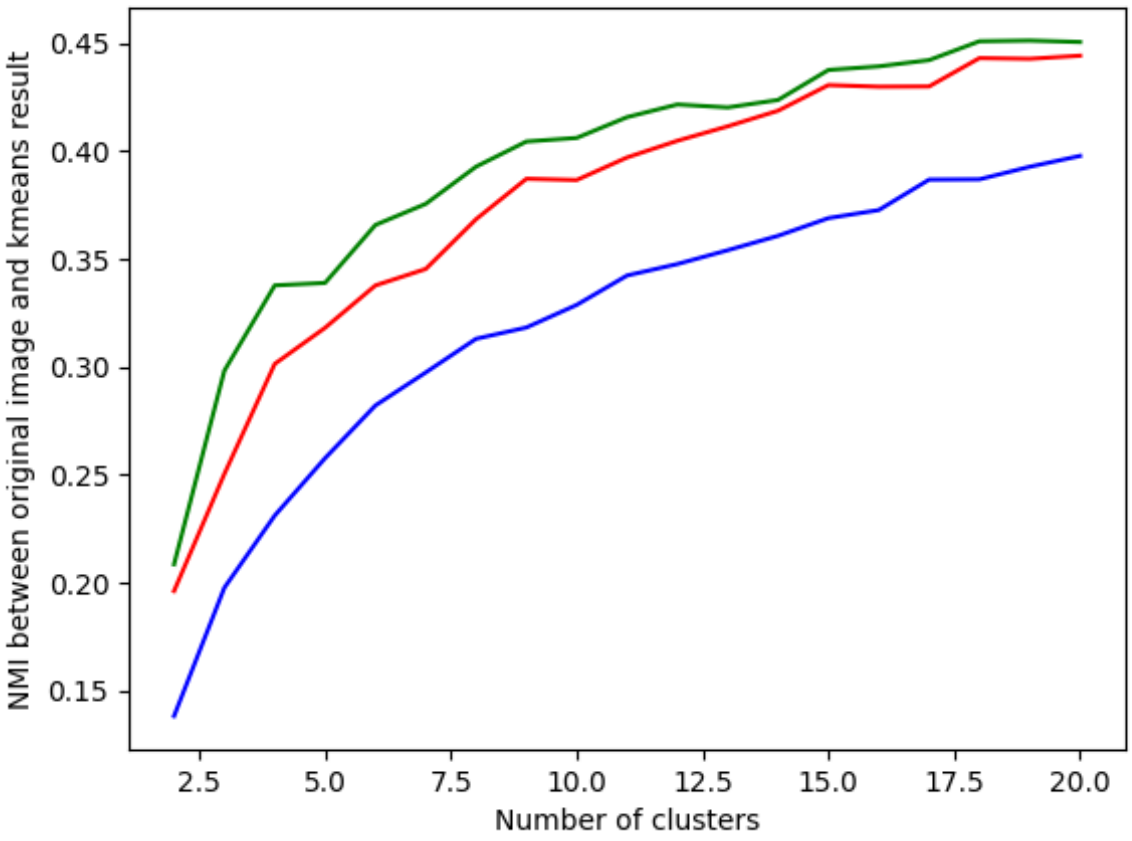
donne l'information mutuelle recherchée.
Enfin, on normalise cette quantité pour obtenir :

$$NMI(X,Y) = \frac{2 * I(X,Y)}{H(X) + H(Y)}$$

5.1 Canaux RVB

On peut mettre ce calcul à profit pour déterminer quelles couleurs sont les plus "liées" dans une image (i.e connaître la répartition d'une couleur nous donne des informations sur la deuxième). En comparant les NMI entre les couleurs RVB de l'image utilisée pour la compression à perte, on remarque que la plus faible est celle entre le rouge et le bleu, la plus élevée entre le rouge et le vert.
En effet, l'image contient du rouge et du bleu séparés, mais très peu de couleurs proches du magenta ou violet, mélange des deux. A l'inverse, l'image est majoritairement jaune (qui se traduit par une très grande valeur de rouge et une grande valeur de vert), ce qui permet de relier les composantes R et V.

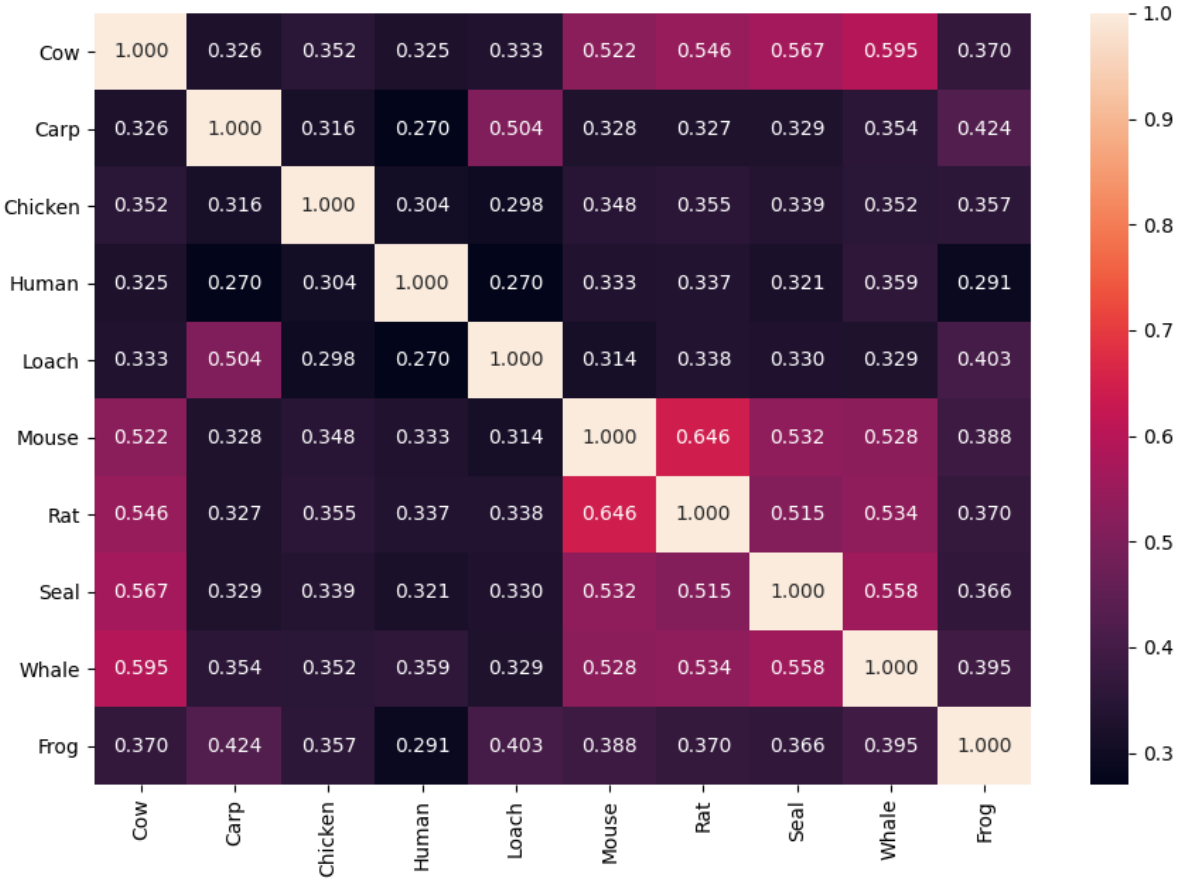
On peut aussi examiner l'information donnée par le résultat de *KMeans* par rapport à l'image originale. Ci-dessous, l'information mutuelle entre les valeurs RVB des deux images, chaque courbe représentant sa couleur :



Forcément, plus on réduit le nombre de couleurs, moins on a de précision sur les couleurs d'origine.

5.2 Séquences ADN

Une autre utilisation est la comparaison de séquences ADN. Avec des échantillons ADN de dix animaux (dont l'homme), on remarque que la NMI entre ADN humain et autre ne dépasse pas 0,36 (elle atteint plus de 0,6 entre la souris et le rat). Avec nos données de test, l'ADN le plus proche de l'homme est celui de la baleine.

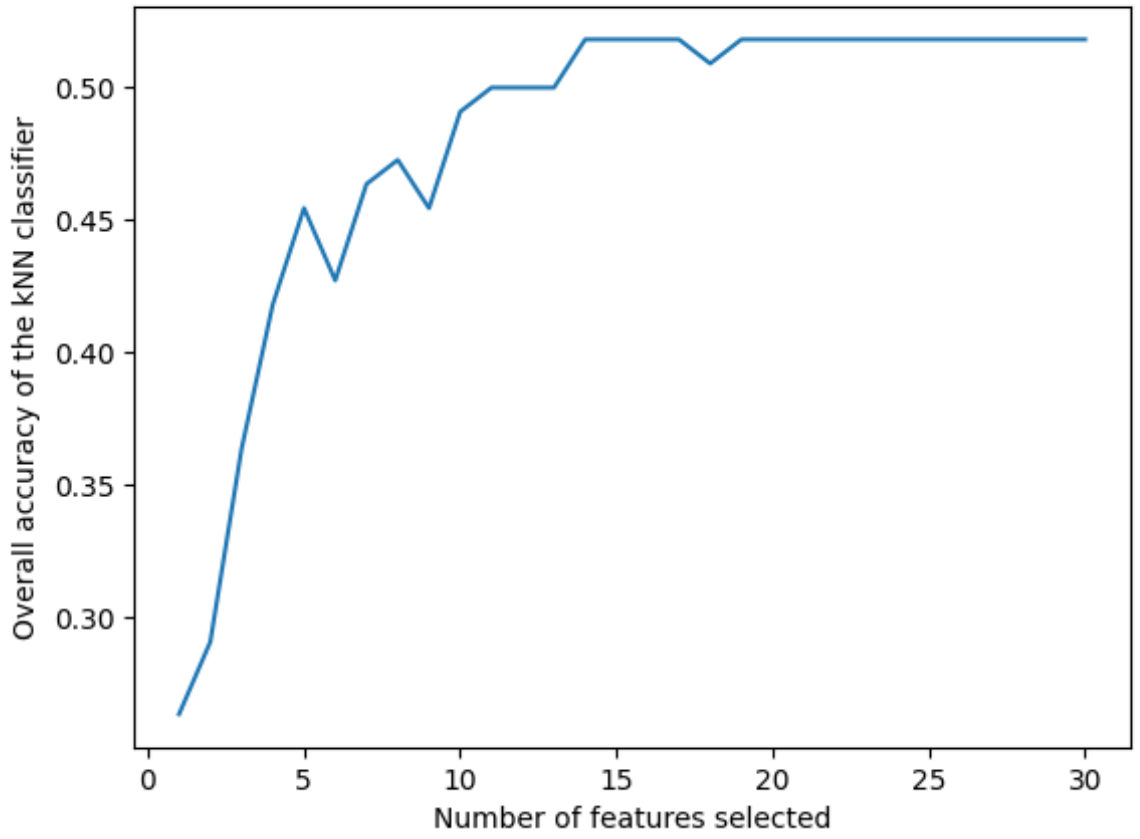


6 Sélection de caractéristiques

La sélection de caractéristiques avec une image se traduit en général par la sélection des canaux (rouge, bleu, vert, alpha, luminosité...) les plus représentatifs - ceux qui, isolés, donnent le plus d'information sur l'image complète. On peut étendre cela à des images hyperspectrales (avec un nombre de caractéristiques arbitraire). On travaille ici avec une image composée de 220 bandes spectrales.

Afin de sélectionner les meilleures bandes en terme de codage, on implémente la méthode mRMR (minimal redundancy maximal relevance), qui consiste à sélectionner les bandes offrant le plus d'information "nouvelle" sur l'image, par rapport aux bandes déjà sélectionnées. La fonction *mRMR_feature_selection* implémente cet algorithme, et renvoie les bandes sélectionnées ainsi que leurs positions dans la matrice d'entrée.

La classe *kNN_Classifier* permet d'en vérifier l'efficacité, en utilisant ces caractéristiques pour prédire la classe d'un ou plusieurs pixels, avec l'algorithme des plus proches voisins.
Les fonctions utilisées sont insensibles à la permutation des données, l'ordre des caractéristiques n'a donc pas d'importance pour *kNN_Classifier*. Ci-dessous, les résultats de la classification pour 1 à 20 caractéristiques sélectionnées, avec 50% des pixels utilisés comme données d'entraînement :



A partir de 15 caractéristiques, on atteint un palier de précision, indiquant soit qu'aucune caractéristique n'est vraiment très représentative des autres, soit que le nombre de classes dans le vecteur cible ne permet pas de précision plus grande que 52%.