28 mars 2022

Rapport UE Introduction à l'Apprentissage Automatique – Projet Mer

DAUVIER Nino (L3 Mathématiques et Informatique, Saint Charles)

SANCHEZ BERMUDEZ Yaiza (L3 Mathématiques et Informatique, Saint Charles)

Monty python

Meilleure score estimé par validation croisée
0,83
Meilleur score obtenu sur images tests de mi-parcours
0,78

Table des matières

1. Introduction

Dans ce projet nous avons dû trouver une solution à un problème de classification d'image et coder cette solution en python. Le but ici est que l'ordinateur puisse distinguer, avec un minimum d'erreur, les images contenant de la mer des images d'ailleurs. Pour cela, nous allons entrainer un classifieur à partir d'un échantillon d'images données, on les appelle images d'apprentissages. Elles sont déjà classées en fonction de leur catégorie (image de mer ou image d'ailleurs) et reparties dans deux dossiers distincts. À la suite de l'apprentissage d'un classifieur sur ces images, nous pourrons deviner la classe d'une image étrangère.

Les classifieurs ont pour rôle de séparer en classes les échantillons ayant des propriétés similaires, en fonction des observations qu'ils ont déjà effectuer sur leur échantillon d'apprentissage. Il existe de nombreux classifieurs utilisant des approches différentes, par exemple des approches discriminantes (par séparation linaire ou non linéaire), des approches génératives et inclassables ou encore des méthodes ensemblistes. Pour apprendre un classifieur il faut tout d'abord représenter les images de sorte qu'elles puissent être lu par le classifieur en question. Les classifieurs que nous avons utilisez proviennent tous du package Sklearn et prennent en paramètre un numpy array (une matrice) comme représentation des images. Nous devons donc créer une matrice qui sur chaque ligne contient la description d'une image.

Nous avons, dans un premier, cherché les différentes manières de représenter les images. Notre première intuition a été de s'inspirer des caractéristiques d'une image contenant la mer par exemple, les couleurs notamment le bleu, la localisation du bleu dans l'image, les formes autours du bleu ou encore les vagues. La première représentation que nous avons faite consiste à ajouter consécutivement les valeurs de chaque pixel de l'image dans une matrice, nous nous sommes vite rendu compte que la durée de création et d'apprentissage de cette matrice été très important pour un résultat correct mais pas à la hauteur de nos attentes (0,73). Il y avait en fait trop de caractéristiques par image comparé au nombre d'image d'apprentissage. Nous avons donc compris que ce n'était pas la bonne manière de représenté nos données.

Afin de réaliser ce projet correctement nous avons dû nous organiser pour ne pas perdre de temps mais cela n'a pas été compliqué puisque naturellement, Nino c'est orienté vers les classifieurs et Yaiza vers la représentation des images. De ce fait, Yaiza a cherché la meilleure représentation possible en faisant attention à ce que les caractéristiques des images ne soient pas trop nombreuses par rapport au nombre d'image. Nino de son côté a cherché les meilleurs classifieurs pour la représentation de Yaiza ainsi que les meilleurs hyper paramètres pour ces derniers. Cependant nous ne nous sommes pas limités à notre orientation de départ, Nino a également généré des descriptions d'images et Yaiza a elle aussi cherchée les hyper paramètres et testée les différents classifieurs. N'étant que deux dans notre groupe, nous ne nous sommes pas vraiment attribués de rôle, nous avons participé tous deux à l'intégralité des taches accompli.

2. Prétraitement des données

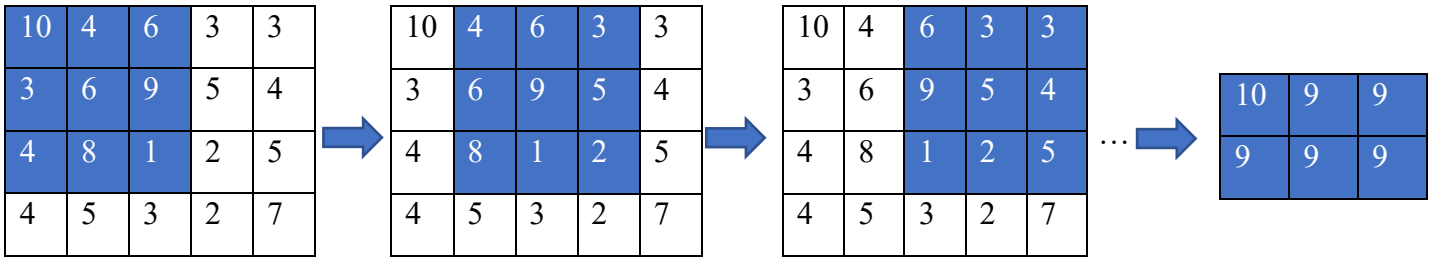
2.1. Représentation des données

L'échantillon d'images qui nous a été donné pour ce projet contient 414 images séparé en deux classes. Ces images sont de tailles et de formats différents, nous avons donc tout d'abord transformé les images PNG en JPG afin qu'elles aient toutes la même structure. En effet, les PNG ont une information en plus par pixel, le gamma, et les valeurs des couleurs rouge vert et bleu (RGB) sont comprise entre 0 et 1 alors que celles des JPG sont entre 0 et 255.

Pour pouvoir travailler avec les pixels, il fallait que les images soient toutes de même taille, puisque sinon les lignes du numpy array qui les représente serait de taille différente, or ceci est impossible. Nous avons donc créé une deuxième version du jeu de donné en changeant la taille de chaque image à 200 par 200 pixels.

Les images de mer ont plusieurs caractéristiques comme la couleur (le bleu), les formes ou encore la texture de l'eau. De ce fait, nous avons construit notre description finale de manière à représenter un maximum de ces caractéristiques (cf. Figure 1). Notre représentation se compose tout d'abord d'un histogramme contenant le nombre de pixels par niveau de bleu de chaque image (via la fonction histogramme du package Math). Nous avons calculé au préalable le maximum de pixels parmi les niveaux de bleu des 414 images, afin d'améliorer l'histogramme de base (cf. Figure 2). En effet, nous nous sommes servis de ce maximum pour diviser le nombre de pixels de chaque niveau de bleu de chaque image. Cette division nous a permis de ramener les valeurs de l'histogramme (le nombre de pixels) entre 0 et 1 tout en donnant un poids plus important aux niveaux de bleu contenant un grand nombre de pixel.

Nous avons par la suite utilisé les images redimensionnées pour faire du « pooling » (mise en commun en français) de 15 par 15 pixels en avançant de 1pixel (cf. Figure 3). Il existe plusieurs sorte de « pooling », par exemple le « max pooling » ou encore « l'average pooling ». Nous avons utilisé le « max pooling » qui consiste à prendre un filtre, ici de 15x15, et de parcourir les pixel de l'image avec ce filtre en récupérant à chaque fois la valeurs maximale des pixels du filtre. Ici nous avons décidé de décale le filtre de 1 pixel à chaque fois.



Représentation du « pooling » sur une image 5x4 pixel en utilisant un filtre de 3x3.

Notre description contient ensuite les valeurs des pixels de la matrice obtenue par pooling sur chaque image. Le pooling permet de repérer certaine forme sur l'image notamment des lignes verticales, ceci apporte le complément des formes à couleur que nous avons déjà avec l'histogramme des bleu.

Enfin nous avons utilisé la matrice de cooccurrence de chaque image (cf. Figure 1). Une matrice de cooccurrence contient pour tout $P_{i,j}$ le nombre de couple de pixel de niveaux de gris i et j qui se situe à une distance n et un angle alpha (avec n et α des paramètres de la fonction qui génère la matrice). Une fois la matrice calculée, on récupère la dissimilarité de la matrice et son homogénéité.

La dissimilarité vaut $\sum_i \sum_j P_{i,j}|i-j|$ elle correspond à la somme des cases de la matrice coefficienté par la différence à la diagonale, c'est-à-dire la discontinuité et la disparité des niveaux de gris de l'image.

L'homogénéité vaut $\sum_i \sum_j \frac{P_{i,j}}{1+(i-j)^2}$ elle correspond quant à elle à l'inverse, plus une image sera uniforme plus l'homogénéité sera forte.

Nous avons essayé de nombreuses descriptions d'image mais aussi de nombreuses combinaisons de descriptions d'images avant de trouver notre description finale. Nous avons dans un premier temps travailler sur les pixels mais comme expliqué précédemment cette approche été trop couteuse. Par la suite, nous avons travaillé sur l'histogramme des niveaux de bleu mais cette représentation ne suffisait pas elle seule, nous l'avons donc combiné avec les gradients de l'image pour essayer de combiner la texture avec la couleur bleue. Cette description, qui pourtant devrait bien fonctionner en théorie, n'était pas à la hauteur de nos attentes. Nous nous sommes donc tournés vers la matrice de cooccurrence pour représenter la texture de l'image. La matrice en elle-même ne nous a pas beaucoup aidé ce sont plutôt les statistiques calculer à partir de celle-ci qui nous ont apporté la description de la texture dont nous avions besoin.

2.2. Augmentation des données

Nous avons remarqué que certaines images étaient mal classées dans l'échantillon d'apprentissage et nous avons longtemps hésité à les ranger. D'un côté cela est assez réaliste et fait donc partie de l'exercice, en effet il est normal qu'il y ait des erreurs dans les échantillons d'apprentissages. De plus, pour des échantillons plus conséquents il serait impossible de vérifier toutes les images une à une. Mais en même temps lorsque notre algorithme nous renvoie les erreurs qu'il a commise et que l'on constate qu'il s'agit d'image mal rangé, cela pourrait quand même avoir du sens de mettre ces images dans le bon dossier.

2.3. Éventuels commentaires

Afin d'améliorer notre description, nous avons programmer dans le classifieur le code (cf. Figure 5) qui permet d'afficher les images ou le classifieur s'est tromper. Grace à ce code nous avons remarqué qu'avec notre description avait du mal à différencier le désert de la mer. Ceci est surement dû aux traces sur le sable qui avec la description de la forme peut ressembler à des vagues, ainsi qu'au couleur clair du désert. Un traitement plus poussé des couleurs pourrait régler ce problème, car nous nous sommes limités au niveau de bleu.

3. Algorithme(s) d'apprentissage considérés

3.1. Algorithme retenu

Le classifieur final de notre projet est un « Soft Voting Classifier » qui choisit à partir d'un « Random Forest », d'un « Multi Layer Perceptron Classifier » et « Gaussian Process » (cf. Figure 6). Ces trois classifieurs donnaient des résultats très proche pour la description finale entre 0,74 pour la forêt, 0.78 pour le gaussian et 0,79 pour le MLP. C'est pourquoi nous avons décidé de les combiner dans un Soft Voting puisque celui-ci permet de profiter de chacun de leurs points forts. Son seul hyper-paramètre est le poids que l'on veut associer à chaque classifieurs. Nous avons fait plusieurs tests sur cet hyper-paramètre et nous avons trouvé qu'en mettant un poids plus fort sur le Random Forest le classifieur était plus efficace. Nous avons donc mis un poids de 0.2 pour le Gaussian Process, de 0.5 sur le Random Forest et un poids de 0.8 sur le MLP (Multi Layer Perceptron).

Nous avons choisi ce classifieur car c'est celui qui nous renvoie le taux d'erreur le plus faible tout en ayant un écart type sur un cross test à 15 partie plutôt correct (0.04 - 0.06). Nous expliquerons plus en détails ce qu'est un cross test à 15 parties dans la suite de ce rapport.

Pour les deux autre classifieurs nous nous sommes servis du HalvingGridSearch de Scikitlearn qui est une variante de GridSearch afin de trouver leurs meilleurs hyper-paramètres, par exemple cf. Figure 7. L'idée étant de proposer une liste de valeurs pour chaque hyper paramètre, le programme réalise un cross test en 5 parties sur toutes les combinaisons de valeurs possible. Au fur et à mesure il élimine le moins bon résultat recommence le cross test et ainsi de suite jusqu'à ce qu'il n'y ait que le meilleur ensemble d'hyper paramètre.

3.2. Explication de(s) algorithme(s) retenus

Le « Soft Voting » consiste à calculer les résultats de plusieurs classifieurs, puis à multiplier les probabilités, retourné par les différents classifieurs, d'appartenir à chacune des classes afin d'obtenir une nouvelle probabilité et déduire la classe de l'image.

Le Random Forest consiste à entrainer plusieurs Decision Tree sur les données et à croiser leurs estimations afin d'obtenir l'estimation finale. Ces hyper-paramètres sont

- « criterion » qui peut valoir « entropy » ou « giny » et définit la qualité de séparation. On utilise "entropy" ;
- «n_estimators» le nombre de Decision Tree que l'on construit que l'on a fixé à 30 ;
- «max_depth» qui donne la limite de profondeur des arbres que nous avons mis à 10 ;
- «max_features» le nombre maximal d'attribut à prendre en compte pour le critère de séparation que nous avons fixé à 2.

Les deux derniers sont les paramètres des Decision Tree qui leurs seront transmis lors de leurs constructions.

Les Decisions Tree sont des arbres construit par récurrence sur leurs profondeurs. On prend l'ensemble des images, on cherche le critère de séparation qui sépare le mieux les classes puis on associe à un des fils du nœuds l'ensemble des images qui satisfont ce critère et à l'autre l'ensemble des images qui ne le satisfont pas. Et on recommence sur les fils avec les images qui lui ont été associé jusqu'à atteindre des séparations entre les classes assez nettes.

Il suffit alors pour ranger une image de parcourir l'arbre en choisissant les fils associés au résultat qu'obtient l'image à chaque nœud, jusqu'à arriver à une feuille et qui va nous rendre la classe associée.

Le MLP est un réseau de neurone, nous nous sommes cantonnés à des réseaux à une seule couche lors de la recherche des hyper-paramètres. Il s'agit donc d'une couche de perceptron qui reçoivent chacun tous les attributs de notre description, leurs appliques des coefficients et calcule la distance entre le vecteur de $\mathbb{R}^n$ ainsi formé et un hyperplan de $\mathbb{R}^n$(qui est une caractéristique du perceptron). Chaque perceptron renvoie ensuite sa distance au perceptron final qui répète le processus mais en renvoyant la probabilité d'appartenir à la classe ou non. Toute la complexité vient des algorithmes d'optimisation des poids qui sont pris en argument du classifieur. Il y en a 3 qui sont implémenté dans Sklearn, *'lbfgs', 'sgd', 'adam'* aux débuts nous avions de meilleurs résultats avec 'lbfgs', mais dans la version finale 'adam' était plus efficace. Les autres hyper-paramètres sont :

- « hidden_layer_sizes » qui décrit la structure des couches de perceptron que nous avons laissé à sa valeur initiale d'une couche de 100. Nous avons essayé de passer à deux ou trois couches de différentes tailles mais le calcul devenait beaucoup plus long pour des différences non-significative, et même pour la taille de la couche 100 donnait les meilleurs résultats avec le HalvingGridSearch ;
- « Alpha » le paramètre de régularisation de la finesse du fitting pour éviter l'overfitting et que nous avons laissé à sa valeur par default, 0.0001 ;
- « max_iter » qui correspond au maximum d'itération de la fonction d'optimisation des poids (elle s'arrêteras toujours même si elle ne converge pas). Dans notre cas nous avions de meilleur résultat si elle ne convergeait pas en particulier pour 300 itérations (voir graphique en section 4.3) ;
- "random_states" décrit l'état initial des poids, random state 1 rends de meilleurs résultats.

Le GaussianProcess associe à l'espace de description un champ scalaire qui donne pour tout point la probabilité que l'élément associé à cette description soit dans l'une des classes. Initialement ces probabilités sont toute à 0.5 et pour chaque élément de l'échantillon d'entrainement on ajoute au champ une gaussienne centré sur la description de l'élément. Il suffit alors pour tester un nouvel élément de prendre la probabilité associée à sa description par le champ. Le seul paramètre auquel on a touché est le kernel car c'est le seul dont on a bien compris le sens, il représente la fonction que l'on ajoute au champ de probabilité. Il vaut initialement ConstantKernel(1.0, constant_value_bounds="fixed" * RBF(1.0, length_scale_bounds= "fixed"). On a modifié le premier 1.0 ce qui a pour conséquence d'augmenter les poids des éléments de l'échantillon d'apprentissage. Ces changements n'ont pas eu de réel impact sur le score. Peut-être modifier le Alpha aurais pu avoir du sens mais comme on n'a pas compris son sens on n'y a pas touché.

3.3. Alternatives

Au cours du projet nous avons essayé de nombreux classifieurs sur les différentes descriptions à savoir :

- Naif Bayes, un algorithme qui part de l'idée que chaque attribut est indépendant des autres, il suffit alors de calculer la probabilité d'être dans chaque classe pour chaque valeur des attributs et de combiner tous les résultats ;
- KClosestNeighbors, on considère la description comme un espace à n dimension et on calcule pour une image les k voisins les plus proches dans cet espace et ainsi que sa classe en fonction de celle de ces voisins. Ce classifieur est plus efficace sur les descriptions avec peu d'attribut, la nôtre étant plutôt lourde, soit 860 attributs pour 414 images, il n'est pas étonnant que ce ne soit pas le meilleur ;
- Quadratic Discriminant Analysis, qui est l'évolution de l'analyse linéaire, on essaie dans ce cas de séparer les données non par un hyperplan mais par une surface décrite par des équations quadratique ;
- SVC, l'idée est de trouver un hyperplan séparant les deux classes en optimisant la marge entre l'hyperplan et les descriptions. On peut ajouter des « kernel », ce sont des fonctions qui envoient l'espace de description sur d'autres espace dans lequel la séparation entre les classes sera linéaire. Pour nous ni les kernel linéaire ni les kernel polynomiaux n'ont été efficaces ;

Les résultats obtenus par validation croisée avec ces classifieurs étant plus faible nous ne les avons pas retenues dans le model final. Le Gaussian Process avait de bon résultat et avait, dans un premier temps, été intégré au Soft Voting. Cependant nous nous sommes rendu compte que ça contribution n'était pas essentielle et nous l'avons donc retiré dans la dernière version de l'algorithme.

4. Évaluation des performances des classifieurs

4.1. Protocole d'estimation des performances

Pour estimer la performance de notre classifieur en fonction notre description nous avons réalisé quasi-intégralement que des validations croisées de 10 parties. Cela consiste à séparer la description des images d'apprentissages et leur classe en 10 parties disjointes puis pour chaque partie à entrainer le classifieur sur les 9 restants et à le tester sur la partie correspondante. Nous sommes par la suite passés sur des validations croisées à 15 parties, le principe est le même, le calcul est à peine plus long et le résultat est plus précis. Dans les deux cas nous avons regardé la moyenne des résultats obtenues et leurs écart-types, afin que notre estimation soit la plus précise possible. Nous avons donc considéré qu'un classifieur été performant en fonction d'une description si la moyenne été élevé et que l'écart type été le plus petit possible. En effet, cela veut dire que le résultat de chaque partie et proche de celui des autres, il y a donc une certaine stabilité.

En implémentant le Soft Voting Classifier, nous nous sommes rendu compte que regarder la réussite ne suffisait pas et n'était donc probablement pas le plus intéressant. Au moment de classer une image, un classifieur va calculer la probabilité que l'image appartienne à une certaine classe. En suite en fonction de cette probabilité il va donner la classe estimée à l'image en question. En effet, il est intéressant d'ajouter un classifieur Soft Voting, si celui ne se trompe pas lorsqu'il calcule une probabilité forte sur une image. Cependant s'il se trompe quand il donne des probabilité moyenne cela a peu d'importance dans le calcul final. Nous aurions donc pu changer le paramètre « scoring function » de notre HalvingGridSearch, de sorte qu'elle calcule la somme des réussites fois la probabilité qui lui a été associé moins les échecs fois leurs probabilités associé. En optimisant ainsi les hyper paramètres des sous classifieur du Soft Voting, il aurait été plus efficace. Cependant nous n'avons pas eu le temps de mettre cette idée en pratique.

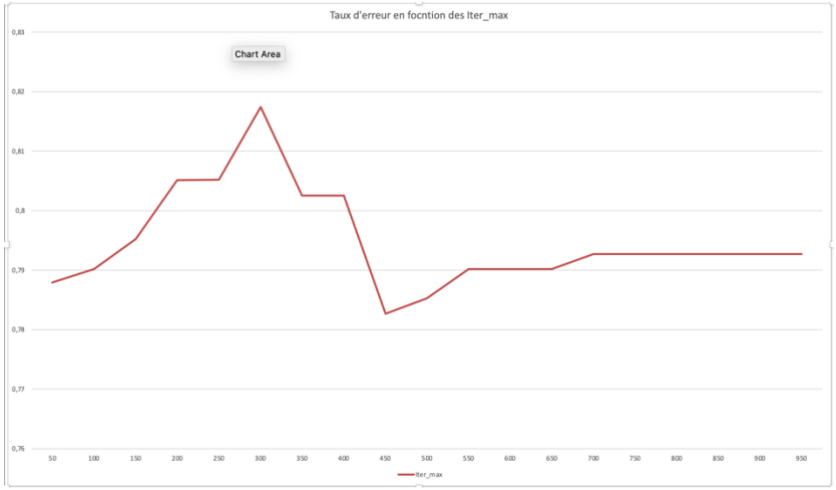
4.2. Performances obtenues

La meilleure performance que nous ayons eu est 0,83, elle a été obtenue en faisant la moyenne des résultats d'une validation croisées à 15 parties. Ce résultat a été calculé à l'aide du classifieur Soft Voting que nous avons présenté précédemment avec la description finale de nos images d'apprentissage. Cette estimation varie cependant entre 0,80 et 0,83 avec un écart type qui lui varie entre 0,03 et 0,06.

Notre meilleur score sur les données test de mi-parcours (0.78) a été obtenue avec le même algorithme qui nous a permis d'obtenir 0.83 au cross test. L'écart entre le résultat du cross test et celui sur les données test est dû au fait que les données test ne sont représentatives des données d'apprentissages. Nous avons également essayé d'ajouter les images du test dans nos images d'apprentissage, pour voir l'effet que cela pouvait avoir sur le résultat du cross test, et notre score a diminué. On en a donc conclu que les images du test de mi-parcours sont plus difficiles à reconnaitre car elles ne sont bien pas représentatives de notre échantillon d'apprentissage.

4.3. Éventuelles courbes de performance

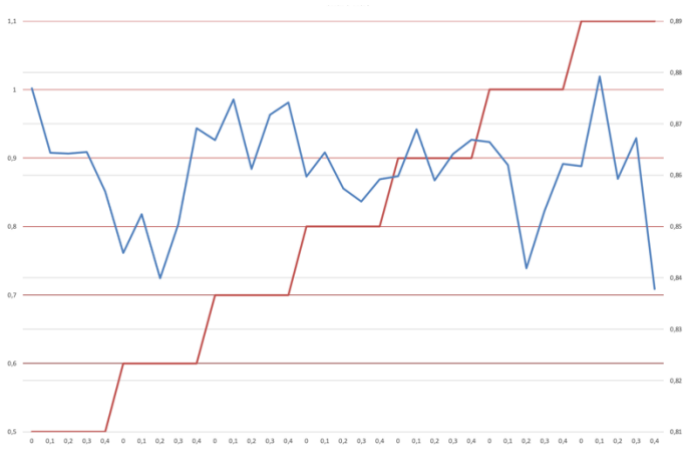
Comme nous l'avons expliqué précédemment nous avons fait des HalvingGridSearch pour trouver nos hyper-paramètres, de ce fait, nous avons seulement deux courbes de performance. La première porte sur le maximum d'itération du classifieur MLP. En effet, nous avions remarqué que ce classifieur était plus efficace lorsqu'il ne convergeait pas, nous avons donc voulu le vérifier. Voici la courbe de performance obtenue lorsque nous avons fait varier le maximum d'itération de 50 à 950 en allant de 50 en 50.



Graphe représentant le taux de réussite du MLP en fonction de la valeur de son maximum d'itération

Cette courbe a été obtenue avec toutes les caractéristiques de la description finale sauf le pooling. Cette représentation étant tout de même très proche de notre description finale nous avons conservé ce résultat. Le classifieur MLP est effectivement plus efficace lorsqu'il ne converge pas comme on peut le constater avec le pique au niveau des 300 itérations.

Nous avons eu une erreur de description qui nous a permis d'atteindre 0.88 lors d'une validation croisé. Nous avons sans faire exprès mélangé la description des images de mer entre elles et d'ailleurs entre elles. Avec description erroné nous avons fait plusieurs teste sur notre classifieur finale, en fonction du poids appliqué aux Random Forest et au MLP. Ayant déjà effectué ce teste auparavant avec des valeurs entre 0 et 1 pour chaque classifieurs, nous nous sommes rendu compte que le Soft Voting était plus efficace lorsque le poids des Random Forest été entre 0.5 et 1 et celui du MPL entre 0 et 0.5. Malheureusement, les valeurs du premier test n'ont pas pu être conserver puisque l'exécution été trop longue (plus d'une heure). Pour ce deuxième test nous avons donc obtenue la courbe ci-dessous, où le courbe bleu correspond au taux de réussite du Soft Voting en fonction des poids, son axe ordonné est celui de droite. La courbe rouge elle correspond au poids des Random Forest, valeur sur l'axe des ordonnés à gauche et l'axe abscisse correspond au poids du MLP.



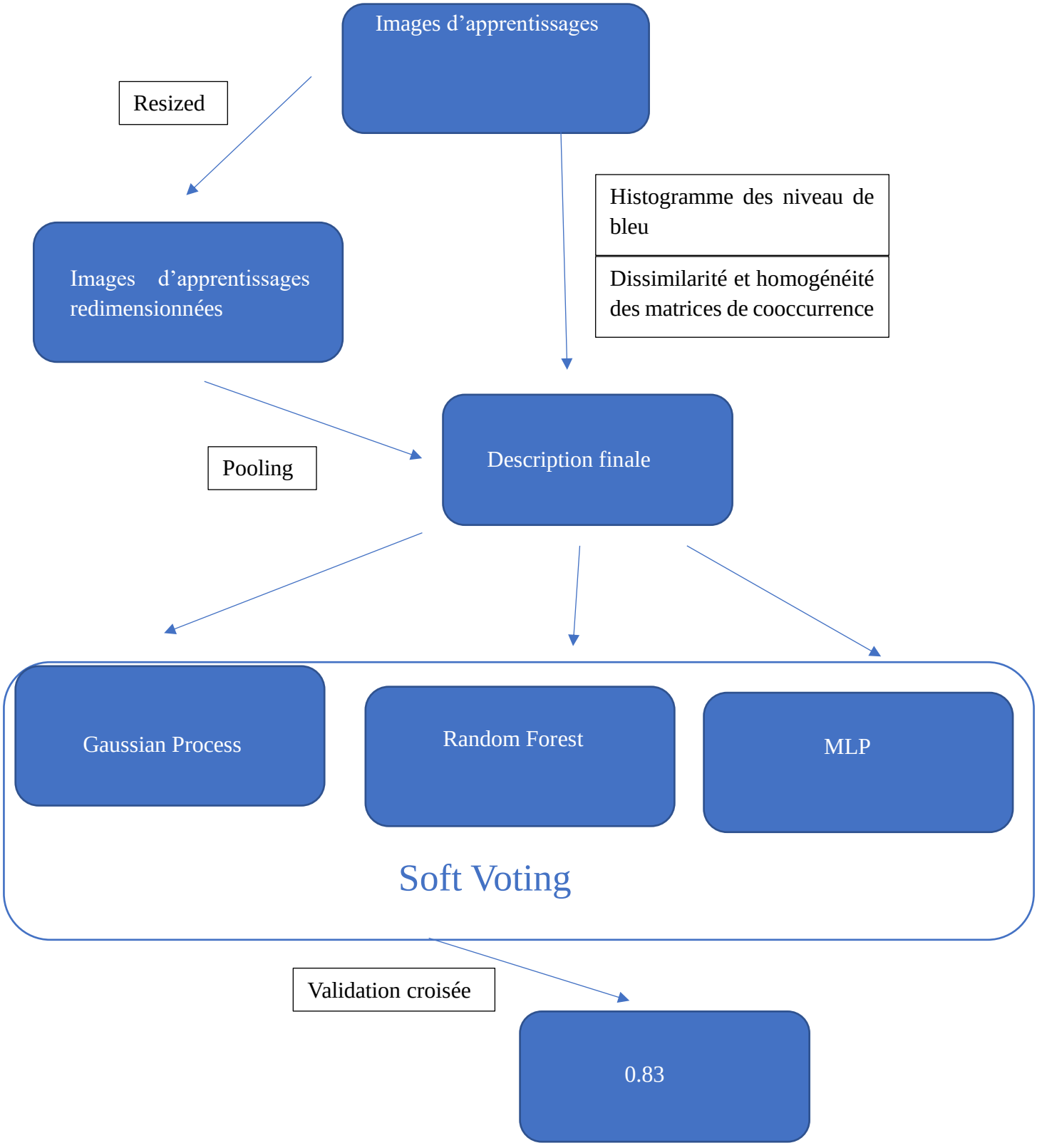
Graphe représentant le taux de réussite du soft voting en fonction du poids des classifieurs qui le compose avec la description finale.

Si notre description avait été erroné nous aurions pu appliquer un poids de 1.1 au Random Forest et 0.25 au MLP afin d'obtenir 0.88.

Sur les deux courbes chaque valeur du taux de réussite correspond à la moyenne d'une validation croisée à 15 parties.

5. Résultats obtenus

5.1. Pipeline complet



5.2. Alternatives

Nous avons sur notre pipeline, tout d'abord sur la manière de rejoindre les descriptions faites sur les resized et sur les images initiales. En effet la raison pour laquelle nous avons eu une erreur dans la description et parce que nous avions fait d'abord le pooling de toutes les images redimensionnées en amont et avions classer les données obtenues dans l'ordre de parcours des images. Ce qu'est que par la suite que nous avons remarqué que les images n'était jamais parcouru dans le même ordre et donc lorsque que l'on parcourrait les images normal l'ordre de parcours ne correspondait pas à celui dur les « resized » bien que les images soient triées dans le même ordre dans les deux dossiers.

5.3. Résultats et commentaires

Au tout début du projet le temps de nous familiariser avec sklearn, nous avons fait des descriptions simples. Ce que nous voulons dire par description simple c'est que nous avons pris, par exemple l'histogramme des niveaux de bleu et cet histogramme était la seule description de nos images. Ce type de représentation nous a permis d'avoir un taux de réussite autours de 0.76 en fonction du classifieur utilisé. Nous nous sommes par la suite dit qu'en associant plusieurs de ces descriptions nous pourrions avoir de meilleur résultat. Dans un premier temps, les associations que nous avons fait, par exemple les gradient et l'histogramme des bleu, ne donnait pas un meilleur résultat. Nous avons donc continué à tester des associations et nous avons trouvé qu'en associant la couleur avec la texture et les formes nous avions un taux de réussite de 0.83 par validation croisé. Le résultat de tous les tests que nous avons fait ont été estimé par validation croisée, afin d'être le plus précis possible. A mi-parcours nous avons testé une de nos descriptions sur l'échantillon test donnée par le professeur et comme expliquer précédemment nous n'avons obtenue 0.74. Par la suite avec notre description finale et notre Soft Voting Classifier nous avons obtenu 0.78 sur les images de mi-parcours. Nous pensons comme nous l'avons expliqué précédemment que cela est dû au fait que les images test n'appartiennent pas à la même distribution que notre jeu de donnée d'apprentissage.

Durant la plus grande partie du projet, nous avons avancé sur le traitement de données et sur les algorithmes d'apprentissage en parallèle, en testant tous nos algorithmes sur chaque description, puis lorsque nous avons découvert HalvingGridSearch, nous avons arrêté de chercher de nouveau algorithme. Nous nous sommes essentiellement focalisés sur le tunning des hyper-paramètre même si nous avons un peu augmenté les descriptions avec le pooling. Tout au long du projet, nous nous sommes focalisés sur les résultats obtenu par le classifieur sur les données d'apprentissage plutôt que sur celles du test de mi-parcours.

6. Conclusion

En conclusion, ce projet nous a permis de découvrir la bibliothèque sklearn ainsi que tous les concepts et les algorithmes de base lié à l'apprentissage automatique.

Notre organisation nous a permis de mener à bien ce projet et d'obtenir des résultats satisfaisants. En effet, notre meilleur score lors de la validation croisée étant 0.83 nous avons bien réussit à nous améliorer par rapport à nos résultats initiaux qui ne dépassait pas les 0.76. Nous sommes plutôt contents du résultat obtenu compte tenu du temps et des ressources que nous avions pour réaliser ce projet. Nous avons lors de ce projet était livré à nous même, même si notre professeur restait disponible. Nous avons donc dû réaliser un vrai travail de documentation tout au long du projet mais également de sélection d'information.

Nous n'avons pas réellement rencontré de difficulté si ce n'est au niveau de la compréhension du fonctionnement de certain algorithme trouvé, par exemple la matrice de cooccurrence. La compréhension de ce que l'on fait est essentiel car il est compliqué d'améliorer une description dont on ne connait pas l'intérêt la couleur, la texture, … Les pistes que nous avons suivies afin d'améliorer notre score ont donc découlé de cette compréhension. Nous avons également eu plusieurs faux espoirs quant au taux de réussite, du a des erreurs de programmation. Par exemple nous avons atteint 0.93, cependant ce résultat était dû à un changement de la description des images de mer qui n'a pas été reporté sur les images d'ailleurs. Nous nous sommes cependant vite rendu compte de l'erreur et n'avons heureusement pas perdu beaucoup de temps. Ou encore lorsque nous avons obtenu 0.87 la veille de la soutenance et que ce résultat été du a un mélange des descriptions des images de mer entre elle et des images d'ailleurs entre elle.

Nous avons plutôt été sérieux en cours ce qui nous a permis de bien avance sur le projet sans avoir à fournir beaucoup de travail à l'extérieur. Il est fortement probable qu'avec plus de temps nous pourrions obtenir des résultats encore meilleurs que ce que nous avons déjà obtenu. En effet, nous avions encore de nouvelles idées pour améliorer le classifieur comme évoqué précédemment. Une fonction de scoring plus adapté, une description plus fine prenant en compte les différentes couleurs sont de bonne piste pour aller plus loin. Un réseau de neurone profond serait aussi beaucoup plus efficace que notre algorithme, même si ce n'était pas l'objectif ici.

Références

- https://kongakura.fr/article/Matrice-de-cooccurence-haralick
- https://www.mathworks.com/help/images/ref/graycomatrix.html
- https://fr.mathworks.com/help/images/ref/graycoprops.html
- https://scikit-learn.org/stable/modules/cross_validation.html
- https://dspace.univ-ouargla.dz/jspui/bitstream/123456789/11639/1/CHATTI-KOUL.pdf
- https://scikit-learn.org/stable/

Annexes

- *Figure 1: Code génération de la description finale*

```
''' Cette fonction combine l'histogramme des niveaux de bleu avec
les probalibilités de la matrice de cooccurence et le pooling l'image
donnée en parametre et ainsi frome sa description '''

def coo_pooling():
    names = []
    y = []
    imgs = []
    path = "Data/Mer"
    valid_images = [".jpg", ".jpeg"]
    maximum = max_bleu()
    i = 0
    for f in os.listdir(path):
        ext = os.path.splitext(f)[1]
        if ext.lower() in valid_images:
            names.append(path + "/" + f)
            img = cv2.imread(path + "/" + f, 0)
            data = Image.open(path + "/" + f)
            block = pooling(path + "/" + f)
            r, g, b = data.split()
            liste = []
            for nb in b.histogram():
                liste.append(nb / maximum)

            for pixel in block:
                liste.append(pixel / 255)

            glcmimg = skimage.feature.graycomatrix(img, [1, 2], [0, np.pi / 4, np.pi / 2, 3 *
np.pi / 4],
                                                   symmetric=True,
                                                   normed=True)

            homogeneite = skimage.feature.graycoprops(glcmimg, 'homogeneity')
            dissimilarite = skimage.feature.graycoprops(glcmimg, 'dissimilarity')
            # print("h:",homogeneite, "\n",dissimilarite)

            for ligne in homogeneite:
                for pixel in ligne:
                    liste.append(pixel)

            for ligne in dissimilarite:
                for pixel in ligne:
                    liste.append(pixel)
            i += 1
            imgs.append(liste)
            y.append(1)

    path = "Data/Ailleurs"
    valid_images = [".jpg", ".jpeg"]
    i = 0
    for f in os.listdir(path):
        ext = os.path.splitext(f)[1]
        if ext.lower() in valid_images:
            names.append(path + "/" + f)
            img = cv2.imread(path + "/" + f, 0)
            data = Image.open(path + "/" + f)
            r, g, b = data.split()
            block = pooling(path + "/" + f)
            liste = []
            for nb in b.histogram():
                liste.append(nb / maximum)
```

```
            for pixel in block:
                liste.append(pixel / 255)

            glcmimg = skimage.feature.graycomatrix(img, [1, 2], [0, np.pi / 4, np.pi / 2, 3 *
np.pi / 4],
                                                   symmetric=True,
                                                   normed=True)

            homogeneite = skimage.feature.graycoprops(glcmimg, 'homogeneity')
            dissimilarite = skimage.feature.graycoprops(glcmimg, 'dissimilarity')
            # print("h:",homogeneite,"\n", dissimilarite)

            for ligne in homogeneite:
                for pixel in ligne:
                    liste.append(pixel)

            for ligne in dissimilarite:
                for pixel in ligne:
                    liste.append(pixel)
            i += 1
            imgs.append(liste)
            y.append(0)

    print("taille imgs", len(imgs))
    print("taille descr 1 image", len(imgs[0]))
    return imgs, y, names
```

- *Figure 2 : Code du calcul du maximum de bleu*

```
''' Cette fonction récupère le plus grand nombre de pixel contenant
le meme niveau de bleu toutes les images '''
def max_bleu():
    maxi_bleu = 0
    path = "Data/Mer"
    valid_images = [".jpg", ".jpeg"]
    for f in os.listdir(path):
        ext = os.path.splitext(f)[1]
        if ext.lower() in valid_images:
            data = Image.open(path + "/" + f)
            r, g, b = data.split()
            # la focntion max récupère le plus grand nombre de pixel
            # contenant le même niveau de bleu
            if maxi_bleu <= max(b.histogram()):
                maxi_bleu = max(b.histogram())

    path = "Data/Ailleurs"
    valid_images = [".jpg", ".jpeg"]
    for f in os.listdir(path):
        ext = os.path.splitext(f)[1]
        if ext.lower() in valid_images:
            data = Image.open(path + "/" + f)
            r, g, b = data.split()
            if maxi_bleu <= max(b.histogram()):
                maxi_bleu = max(b.histogram())

    return maxi_bleu
```

- *Figure 3 : Code du pooling*

```
''' Cette fonction fait le pooling d'une image donnée en paramètre
à partir de son image redimensionnée '''


def pooling(file):
    deb, fin = os.path.split(file)
    if deb == 'Data/Mer':
        fin = os.path.splitext(fin)[0]
        path = "Data/Resized/Mer/" + fin + "resized.jpg"
        img = cv2.imread(path)
        # fonction de skimage qui fait le pooling d'une image
        block = skimage.measure.block_reduce(img, (15, 15, 1), np.max)
        liste = []
        # block est de dimension 3,  il faut donc le mettre en dimension 1
        for ligne in block:

            for ele in ligne:
                for pixel in ele:
                    liste.append(pixel)
        return liste

    elif deb == 'Data/Ailleurs':
        fin = os.path.splitext(fin)[0]
        path = "Data/Resized/Ailleurs/" + fin + "resized.jpg"
        img = cv2.imread(path)
        # fonction de skimage qui fait le pooling d'une image
        block = skimage.measure.block_reduce(img, (15, 15, 1), np.max)
        liste = []
        # block est de dimension 3,  il faut donc le mettre en dimension 1
        for ligne in block:
            for ele in ligne:
                for pixel in ele:
                    liste.append(pixel)
        return list
```

- *Figure 4 : Code qui affiche les images où le classifieurs s'est trompé*

```
def voting_test(X, y, names):
    X, y, names = shuffle(X, y, names)
    X_train, X_test, y_train, y_test, name_train, name_test = train_test_split(X, y, names,
test_size=0.20)
    classifiers = [
        RandomForestClassifier(bootstrap=False, criterion='entropy', max_depth=10,
max_features=2, n_estimators=30),
        MLPClassifier(random_state=1, max_iter=300)
    ]
    ere = VotingClassifier(estimators=[(str(i), classifiers[i]) for i in
range(len(classifiers))], voting='soft')
    ere.fit(X_train, y_train)
    y_predit = ere.predict(X_test)
    print(accuracy_score(y_test, y_predit))
    print(y_test, len(y_test))
    print(y_predit, len(y_predit))
    count = 0
    for i in range(len(y_test)):
        if y_test[i] != y_predit[i]:
            count += 1
            plt.axis('off')
            plt.imshow(mat.image.imread(name_test[i]), cmap=plt.cm.gray)
            plt.title(name_test[i] + 'réel' + str(y_test[i]) + 'prédit' + str(y_predit[i]))
            plt.show()
    print(count)
```

- *Figure 5 : Code de la validation croisée du classifieur*

```
def voting_cross_val(X, y):
    # on melange notre échantillon pour ne pas toujours avoir les mêmes parties lors
du cross test
    X, y = shuffle(X, y)
    classifiers = [
        GaussianProcessClassifier(1.0 * RBF(1.0)),
        RandomForestClassifier(bootstrap=False, criterion='entropy', max_depth=10,
max_features=2, n_estimators=30),
        MLPClassifier(random_state=1, max_iter=300)
    ]
    ere = VotingClassifier(estimators=[(str(i), classifiers[i]) for i in
range(len(classifiers))],
                           voting='soft', weights=(0.2,0.3, 0.9))
    scores = cross_val_score(estimator=ere, X=X, y=y, cv=15)
    print(scores)
    ecart_type, moyenne = np.std(scores), np.mean(scores)
    print(ecart_type, moyenne)
    return moyenne
```

- *Figure 6 : Code HalvingGridSearch sur les Random Forest*

```
def halving_grid_random_forest(x_array, y_array):
    imgs_array = np.array(x_array)
    y_array = np.array(y_array)

    X_train, X_test, y_train, y_test = train_test_split(imgs_array, y_array, test_size=0.20)

    dico = {"max_depth": [3, 5, 10, 20, 30, 35], "n_estimators": [10, 15, 30, 35, 40, 45, 100],
            "max_features": [2, 6, 10, 15, None], "criterion": ["entropy"]# "gini"
        , "bootstrap": [True, False]}
    classifieur = HalvingGridSearchCV(RandomForestClassifier(), dico, verbose=2, min_resources="exhaust")
    classifieur.fit(X_train, y_train)
    print(classifieur.best_params_)
    print(classifieur.score(X_test, y_test))
```