MEDICHAL PROJECT

AFRICA
CODALAB n°51

NDAO Ndieme — ISKOUNEN Feriel
PATAULT Paul — SERRE Gaëtan
FERREIRA Jules — ESTEVAN Benjamin

https://youtu.be/WtSt-l2n4hE
https://codalab.lri.fr/competitions/625
https://github.com/LE10EENFAIT/Africa

1. Description du problème

Le challenge MEDICHAL[1] consiste à détecter si une cellule est infectée par le parasite de la malaria grâce à une photo de la cellule en question. Nous avons choisi ce challenge pour l'aspect médical du projet, en effet nous pensons que c'est une des plus belles utilisations du machine learning.
Pour cela nous disposons d'un jeu de données contenant 16 534 images en couleur. Chaque image contient 19 *features* tel que le nombre de pixels noirs ou bien la couleur moyenne. La métrique du challenge est l'aire sous la *courbe ROC* (Receiver Operating Characteristic).
Pour réaliser ce projet, nous avons séparé notre groupe en trois sous-groupes. Un binôme s'est occupé du preprocessing du jeu de données, c'est à dire la transformation des données brutes en données utilisables plus efficacement par le classifieur (ex : la sélection de *features*) ; un autre binôme s'est chargé du choix du modèle, du classifieur et l'entraînement de ce dernier avec les données de test ; enfin le dernier binôme s'est occupé de l'affichage des résultats de manière compréhensible et pertinente.

2. Description des sous-groupes

2.1. Preprocessing — FERREIRA Jules & ESTEVAN Benjamin

Nous avons filtré les données de l'ensemble d'entraînement grâce à la forêt d'arbres décisionnels *Random forest* pour sa simplicité d'utilisation et son efficacité sur les grands ensembles de données. Pour cela nous avons d'abord séparé les données en fonction de leur classe car il est normal que des données provenant de deux classes distinctes diffèrent les unes des autres. Nous avons ensuite appliqué une *Isolation forest* sur les deux ensembles, ce qui nous a permis de distinguer les bonnes données des données aberrantes (outliers). Ainsi nous avons pu nous débarrasser des outliers avant de remettre les bonnes données ensemble. Nous avons ensuite analysé la variance des différentes *features* grâce à la fonction *var* de *panda.DataFrame* pour savoir lesquelles avaient les variances les plus grandes et allaient donc être conservées après avoir fait une *PCA*[2]. Puis, nous les avons comparées aux *features* considérées comme les plus importantes selon la fonction *feature_importances_* de *model.classifier* car ce sont celles qui

[1] Cours de L2 « Mini-projet ». Isabelle Guyon. 2020

[2] Principal Component Analysis (cf. https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html)

seraient conservées suite à une application du modèle *ExtraTreesClassiflier*[3] (cf. annexes 2, 3 et 4).
Après avoir ajusté les paramètres des deux modèles de façon à ne conserver dans les deux cas que 4 *features*, nous avons comparé les scores de *cross-validation* obtenus dans les 2 cas.
Ainsi le score de *cross validation* en ne conservant que 4 *features* ne dépasse pas 0,73 avec la *PCA* alors qu'on arrive à un score de 0,95 avec le modèle *ExtraTreesClassifier* de sélection de *features*, les *features* conservées par les deux modèles n'étant bien évidemment pas les mêmes. Nous avons donc décidé de n'utiliser que le modèle *ExtraTreesClassiflier* puisqu'il ne nécessite que 4 *features* pour atteindre un score de 0,95 là où la *PCA* en nécessite 7 pour arriver à un score de 0,94 avec les données analysées.

2.2. Modèle — PATAULT Paul & SERRE Gaëtan

Nous avons testé cinq modèles utilisant la *cross-validation* sur vingt ensembles de test tirés aléatoirement à partir de l'ensemble des données fournies par le sous-groupe preprocessing. Nous avons remarqué que les données sont relativement simples à séparer étant donné que le score moyen pour chaque modèle est toujours supérieur à 0.55. Cependant, un modèle est sorti du lot : la forêt d'arbres décisionnels (i.e. *Random forest* (cf. annexe 1))[4] avec un score moyen de 0.95 (cf. tableau 2).
Nous avons donc choisi d'utiliser la *Random forest* comme modèle de prédilection pour la séparation de nos données. Ensuite, grâce à la méthode *RandomizedSearchCV*[5] de la librairie *Scikit-Learn*, nous avons pu déterminer les meilleurs hyper-paramètres pour notre modèle *Random forest* afin de maximiser notre score et éviter le sur-apprentissage.

De plus, nous avons utilisé des *réseaux de neurones convolutifs* ou « CNN » (cf. annexe 9) pour classer les images brutes. La différence fondamentale entre un réseau de neurones classique et un *CNN* est que ce dernier va découper l'image entrée en plusieurs parties pour les analyser indépendamment contrairement au classique qui traite directement l'image dans son ensemble. C'est ce qui rend les CNN particulièrement efficace pour la reconnaissance d'image. Ainsi, avec la bibliothèque *Keras*[6] (« enfant » de *TensorFlow*), nous avons utilisé le *ResNet50v2* — un réseau de neurones convolutif spécialisé en reconnaissance d'images — que nous avons pré-entrainé sur la base de données *ImageNet*[7] (14e6 images). En le ré-entrainant sur 100 époques avec les *raw data* nous avons obtenus un score de 0.9824.

[3] Scikit-Learn ExtraTreeClassifier (cf. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html)

[4] Scikit-Learn RandomForest (cf. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)

[5] Scikit-Learn RandomizedSearchCV (cf. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html)

[6] Keras (cf. https://keras.io/layers/core/)

[7] Base de donnée ImageNet (cf. http://www.image-net.org/)

2.3. Visualisation — ISKOUNEN Feriel & NDAO Ndieme

Afin de bien explorer nos données, nous avons commencé par utiliser un algorithme de *clustering.* Nous avons choisi l'*algorithme k-mean*[8] car, contrairement aux autres *algorithmes de clustering,* il ne nécessite pas de paramètre difficile à trouver. Dans notre cas, le nombre de classes souhaité était connu. Nous avions donc deux possibilités : soit la cellule est infectée, soit elle ne l'est pas.
Nous avons voulu visualiser le résultat en fonction de *min_gray* et *max_gray* qui sont, d'après la matrice de corrélations, les *features* les plus corrélées avec le fait qu'une cellule soit infectée (cf. annexe 5). Le résultat obtenu ne permettait pas de distinguer les deux classes.
Nous avons ensuite fait une analyse en composantes principales pour réduire les dimensions des données et pouvoir mieux visualiser celles-ci en deux dimensions. Mais, il s'est avéré que la variance cumulée, exprimée par les deux composantes principales, ne représentait que 40% de la variance totale. Il nous est alors paru normal que la représentation en deux dimensions ne soit pas informative. Nous avons également affiché la surface de décision d'un *Decision Tree*, formé sur des paires de *features*[9] (cf. annexe 6). Comme ces deux résultats ne permettaient pas de discriminer les deux classes, nous avons utilisé l'algorithme *T-SNE*[10] de *Scikit-Learn.* Nous avons choisi cet algorithme car il est adapté à un grand nombre de données, et il permet de réduire les dimensions.
Nous avons essayé cet algorithme sur l'ensemble des données avec 19 *features* (cf. annexe 7) et sur nos données avec 4 *features* après notre preprocessing (cf. annexe 8). Nous remarquons que les données « parasitized » et « uninfected » forment deux groupes bien distincts uniquement sur les données avec 4 *features.*

3. Résultats

Grâce aux travaux de chaque binôme, nous avons réussi à réduire le nombre de *features*, passant de 19 à 4, et ce tout en gardant un score quasiment identique (d'environ 0.978, soit presque 1% de moins, mais un temps de calcul 4 fois plus rapide). Ce résultat a également été obtenu en utilisant comme modèle de classification de nos données la *Random forest* avec ses meilleurs paramètres, précédemment calculés. De plus, le second résultat obtenu est de 0.9824 avec un *réseau de neurones convolutif* (CNN) sur les *raw data.*

[8] Algorithme de *clustering*, pour plus d'information voir https://towardsdatascience.com/understanding-k-means-clustering-in-machine-learning-6a6e67336aa1

[9] Scikit-Learn plot decision surface of a decision tree (cf. https://scikit-learn.org/stable/auto_examples/tree/plot_iris_dtc.html)

[10] Scikit-Learn TSNE (cf. https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html)

4. Tests & Pseudo-Code

Nous avons réalisé plusieurs tests unitaires pour chaque groupe. Toutes nos fonctions ont été appelées et nous avons réalisé plusieurs exemples en appelant *Scikit-Learn* en dehors de nos fonctions et, en vérifiant que les deux retournaient le même résultat. Lors de nos tests, nous avons choisi d'utiliser nos données prépocessées.
Pour ce qui est du pseudo-code, nous ne sommes pas en mesure d'en fournir un, étant donné que la grande majorité de notre code est constituée d'algorithmes implémentés dans *Scikit-Learn*, et nos connaissances actuelles ne nous permettent pas d'en comprendre toutes les subtilités.

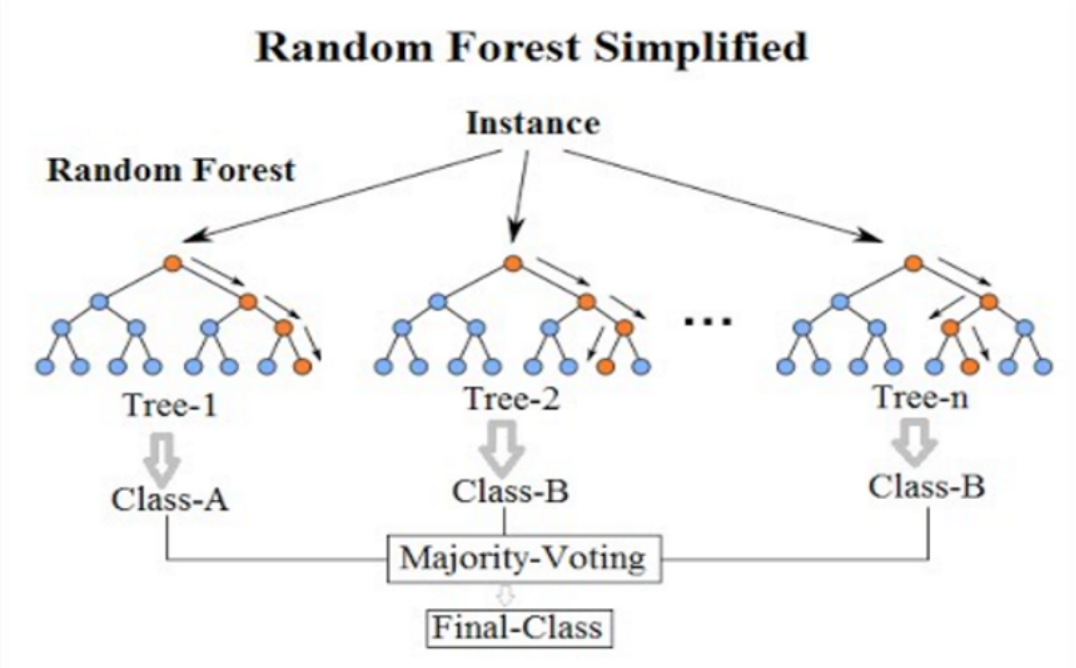
5. Conclusion

Tout au long de ce projet, nous avons appris les bases du machine learning et la résolution de problème de data science. Nous avons exploré différentes bibliothèques de *Python* comme *Numpy*, *Pandas*, *Matplotlib*, *Scikit-Learn ou encore Keras*. Nous avons alors, grâce à celles-ci, été en mesure de déterminer le meilleur algorithme pour ce problème ; c'est de loin une *Random Forest* pour le challenge préprocessé et un *CNN* pour le challenge RAW. En plus de ces découvertes, nous avons appris à nous organiser pour travailler efficacement en équipe. La distribution du groupe en 3 binômes à bien fonctionné et nous a permis d'être efficace. Ainsi grâce à ce projet, chacun de nous a développé de nombreuses compétences comme la rigueur, la répartition des tâches, le sérieux ou encore la confiance mutuelle entre chacun des membres de l'équipe.

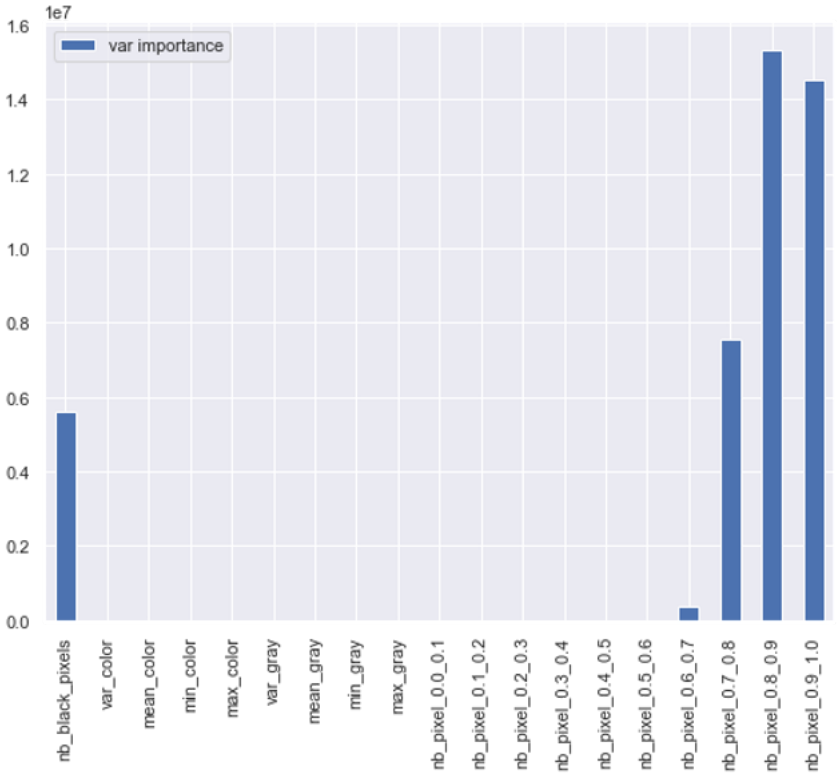
6. Bibliographie & Références

— Cours de L2 « Mini-projet ». Isabelle Guyon. 2020
— https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
— https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesClassifier.html
— https://scikit-learn.org/stable/modules/generated/
sklearn.ensemble.RandomForestClassifier.html
— https://scikit-learn.org/stable/modules/generated/
sklearn.model_selection.RandomizedSearchCV.html
— https://towardsdatascience.com/understanding-k-means-clustering-in-machine-
learning-6a6e67336aa1
— https://scikit-learn.org/stable/auto_examples/tree/plot_iris_dtc.html
— https://keras.io/layers/core/
— https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
— github.com/scikit-learn
— www.image-net.org/

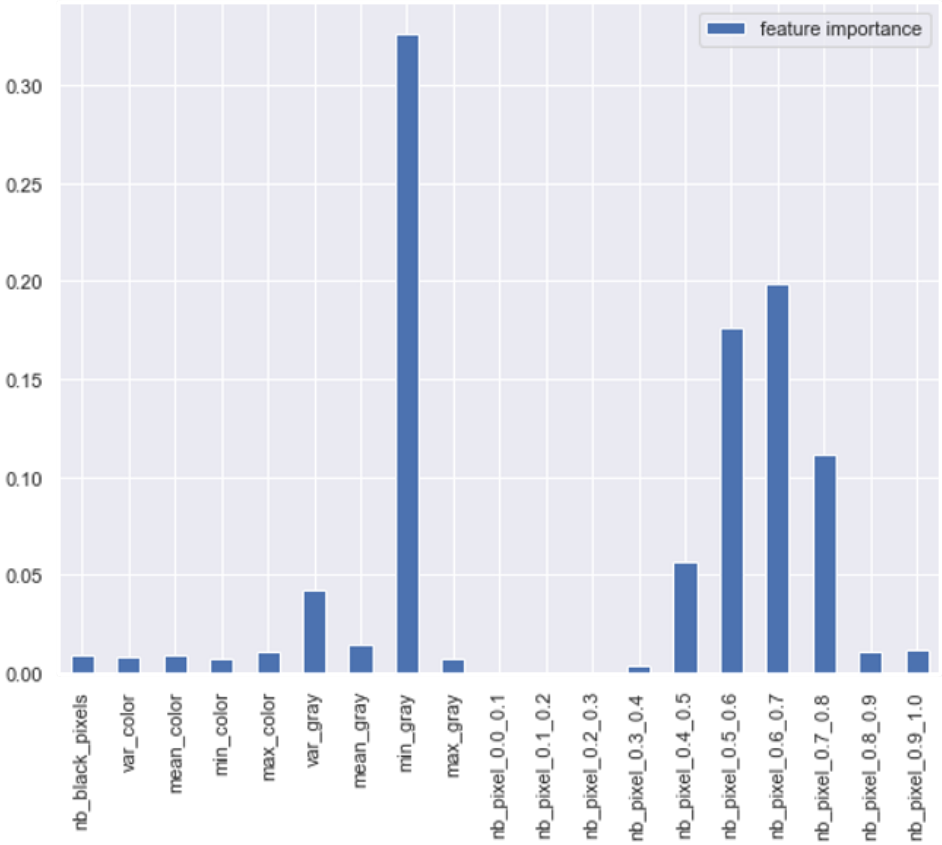
7. Annexes



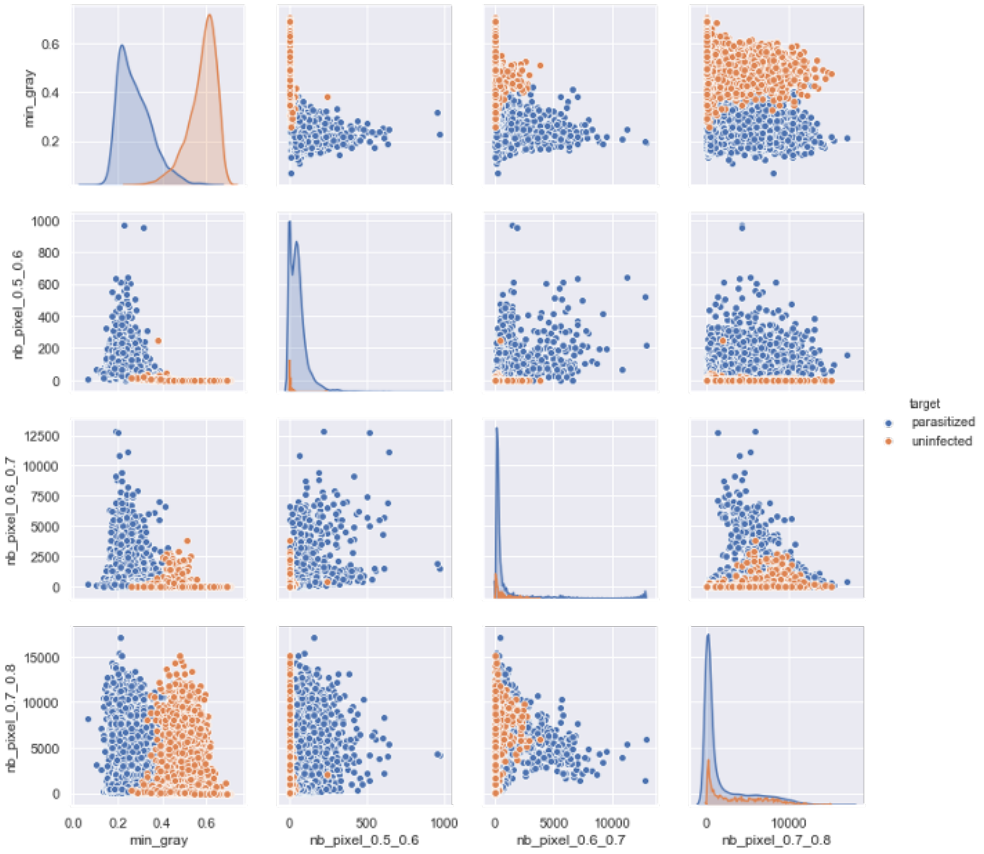
Annexe 1 : Schéma simplifié du fonctionnement d'une *Random forest*



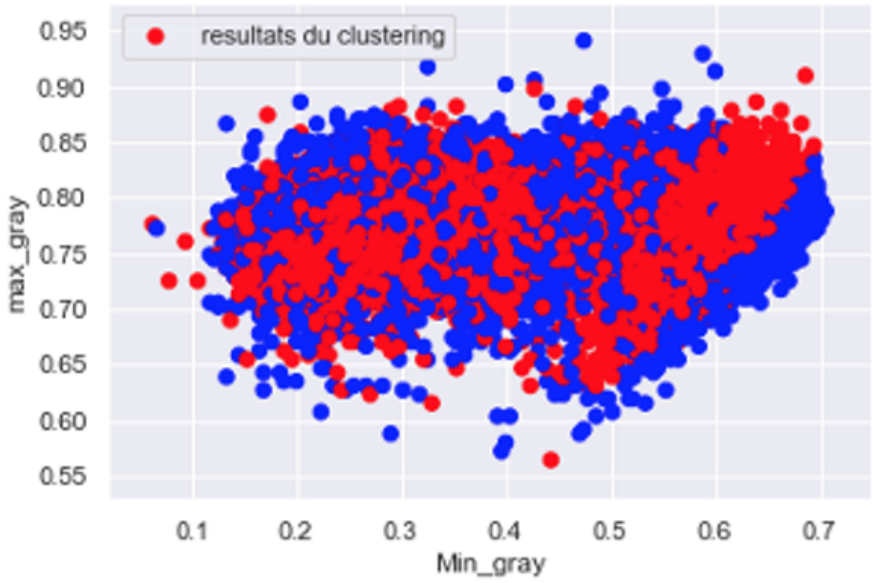
Annexe 2 : Histogramme de la variance pour chaque feature



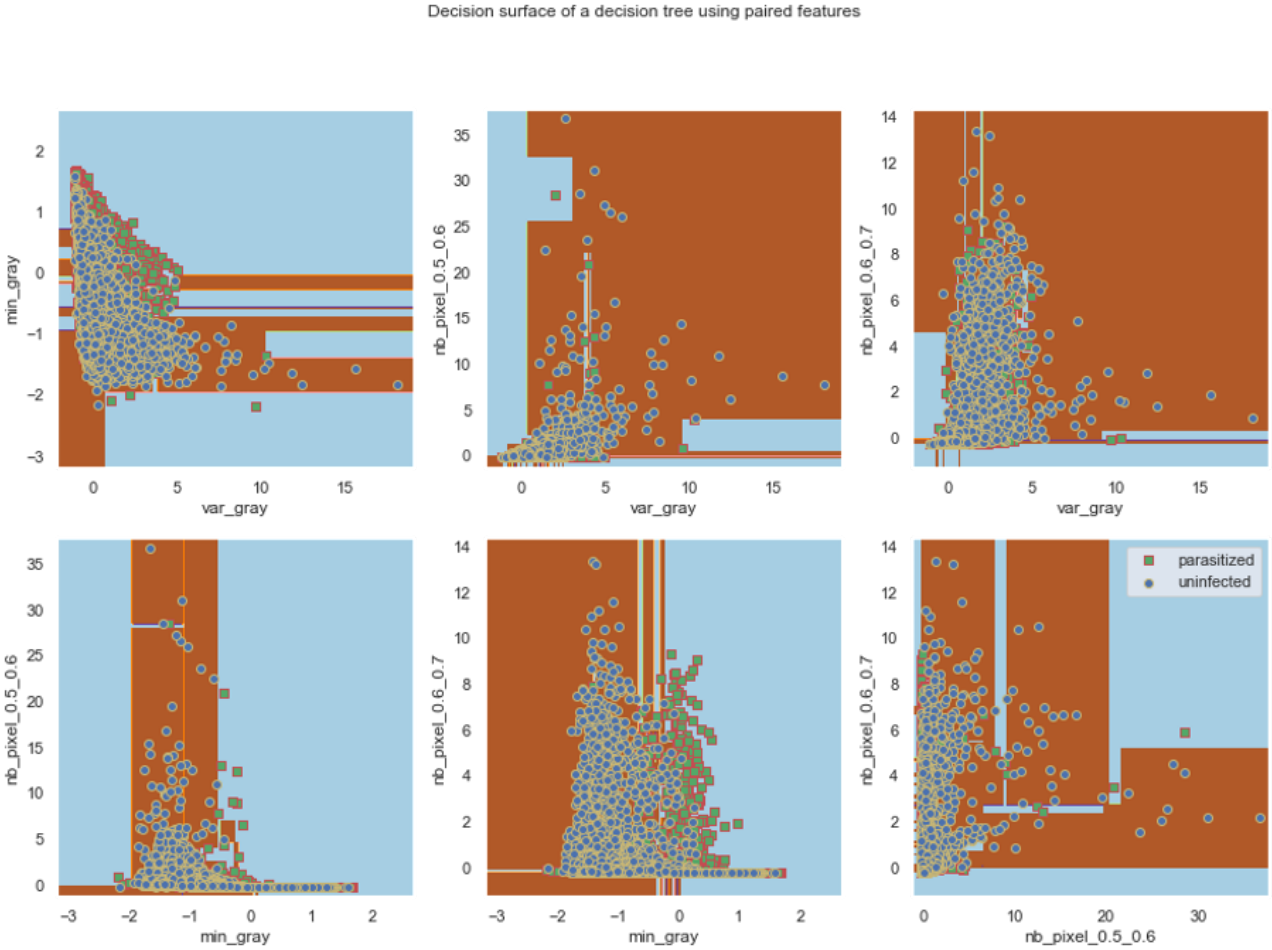
Annexe 3 : Histogramme de l'importance de chaque *feature*



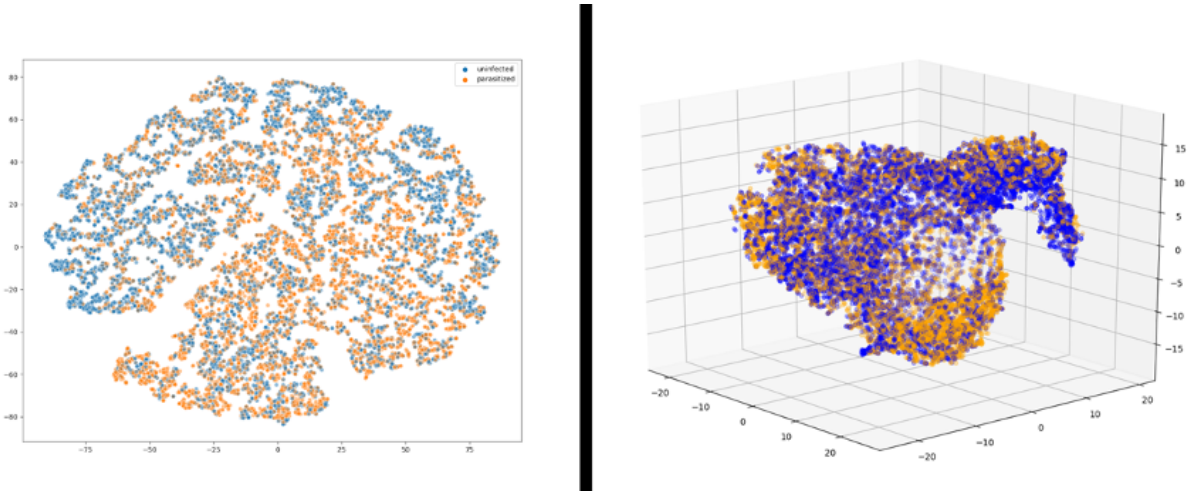
Annexe 4 : Schémas comparatifs des cellules saines et infectées par paires de *features*



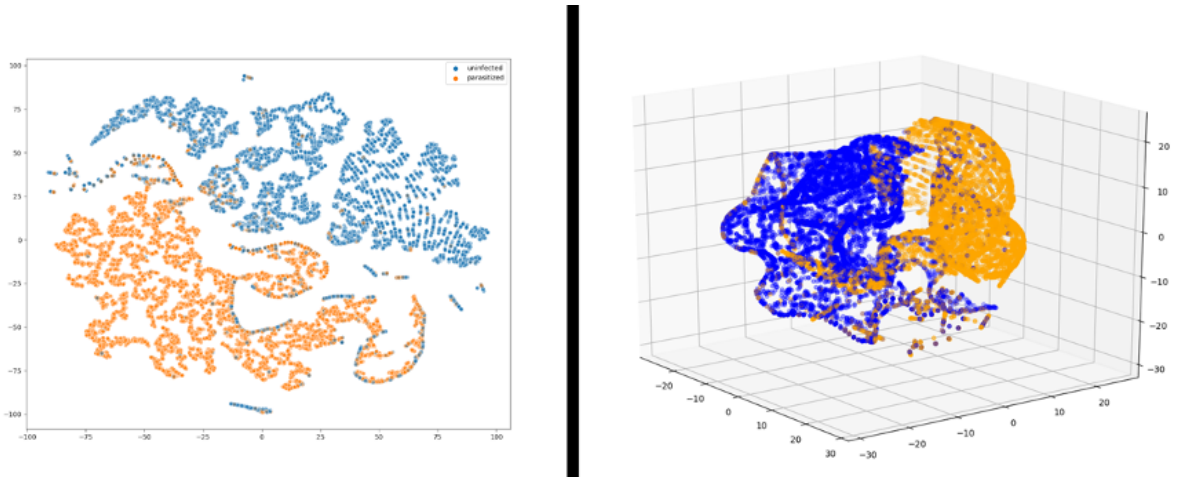
Annexe 5 : Résultat du *Clustering*
Rouge : parasitized
Bleu : uninfected



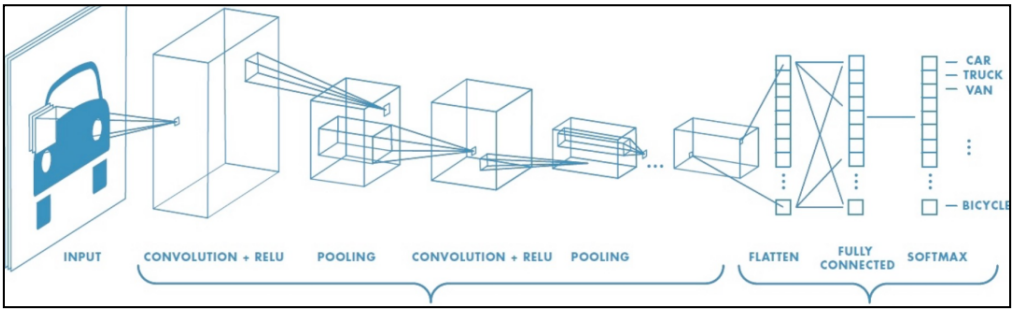
Annexe 6 : Surface de décision d'un arbre de décision formé sur des paires de *features*
Bleu : surface parasitized
Marron : surface uninfected



Annexe 7 : T-SNE 2D (gauche) T-SNE 3D (droite) sur données avec 19 *features*
Orange : parasitized
Bleu : uninfected



Annexe 8 : T-SNE 2D (gauche) T-SNE 3D (droite) sur données avec 4 *features*
Orange : parasitized
Bleu : uninfected



Annexe 9 : CNN

Dataset	Num. Examples	Num. Variables/ Features	Has categorical variables ?	Has missing data ?	Nums. Examples in each class
Training	16534	4	No	No	8276: Parasitized 8258: Uninfected
Valid(ation)	5512	4	No	No	Unknown
Test	5512	4	No	No	Unknown

Tableau 1 : Informations sur les données

Method	Random Forest	Neural Network (MLP)	K-Neighbors	Naive Bayes	Perceptron
Training	0.006	0.39	0.12	0.1	0.05
CV	0.06	0.38	0.11	0.09	0.07
Valid(ation)	0.0514	0.4339	0.0675	0.0845	0.1055

Tableau 2 : Renseignements des résultats préliminaires

Ces résultats ont été obtenus avec les données du challenge classique sans preprocessing de notre part. On renseigne ici le taux d'erreur sur [0, 1] (avec 0 comme valeur souhaitable) de nos modèles sur les différents ensembles de données à prédire. On remarque ici que *Random Forest* est le meilleur sur tous les jeux de données. De plus, les autres modèles ne sont pas nécessairement meilleurs sur l'ensemble d'entraînement que sur les autres.

Challenge	Score	Classement
Medichal - Classique	0.9870	1er
Medichal - Raw Data	0.9824	1er

Tableau 3 : Leaderboard