Rapport de projet

Groupe Malaria

Nom challenge : ***MediChal***
Nom de groupe : ***Malaria***
Membres : Charlotte TRUPIN <charlotte.trupin@u-psud.fr>, Mathilde LASSEIGNE <mathilde.lasseigne@u-psud.fr>, Alicia BEC <alicia.bec@u-psud.fr>, Sakina ATIA <sakina.atia@u-psud.fr>, Maxime VINCENT <maxime.vincent1@u-psud.fr>, Baptiste MAQUET <baptiste.maquet@u-psud.fr>.
URL challenge : *https://codalab.lri.fr/competitions/625*
Lien dépôt GitHub : *https://github.com/charlottetrupin/malaria*

Binôme 1 preprocessing : Alicia BEC & Charlotte TRUPIN
Binôme 2 model : Maxime VINCENT & Baptiste MAQUET
Binôme 3 visualization : Sakina ATIA & Mathilde LASSEIGNE

Contexte et motivation

La malaria autrement connue sous le nom de paludisme est une maladie infectieuse due à un parasite, propagé par certaines espèces de moustiques. Elle atteint majoritairement les enfants de moins de 5 ans et les femmes enceintes. Au total, près de 219 millions de personnes contaminées et 435 000 décès ont été enregistrés en 2017.
(Source : https://fr.wikipedia.org/wiki/Paludisme)
Le but de ce projet est donc de concevoir un algorithme simple permettant d'identifier et différencier les cellules saines des cellules infectées afin de faciliter la prise en charge de malades, tout particulièrement dans des zones à risque et ayant des problèmes sanitaires. Le recours à l'apprentissage automatique permettrait de dépister les personnes infectées à partir d'images de cellules, l'extraction de telles cellules demandant peu d'équipement.

Données et description du problème

Notre projet revient à traiter un problème de classification binaire. Nos données contiennent deux classes : les cellules infectées et les cellules saines, qu'il va nous falloir départager.
La répartition de nos données est telle que trouvée dans le **Tableau 1** *(faire le tableau de statistiques)*.
Nous travaillons ce projet sur le document README.ipynb sur le dépot github donné plus haut.

Approche choisie

Pour commencer, nous avons choisi de faire une sélection de *features* pour exclure celle qui sont moins utiles et une réduction de dimension avec une comparaison des méthodes *PCA (Principal Component Analysis)* et *TSNE (T-distributed Stochastic Neighbor Embedding)* qui permet de transformer les *features* en une dimension inférieure. Ce qui permet un gain de temps de calcul non négligeable lors de la classification.
Par la suite, nous avons sélectionné plusieurs classifieurs binaires que nous avons entraînés sur l'ensemble d'apprentissage à l'aide de *GridSearch* afin de déterminer les meilleurs hyperparamètres de chacun d'entre eux. Partant de ces algorithmes bien entraînés, nous avons cherché à savoir lesquels étaient en sur-apprentissage, en calculant leur score sur l'ensemble de test et à l'aide de l'outil de *cross-validation*. Notons que la méthode de calcul du score est basée sur l'AUC (aire sous la courbe ROC).

Description du groupe

Le binôme 1 s'occupe de sélectionner les features les plus importantes en fonction de la **Figure 1** donné, de la sélection des outliers (**Code 1**) et d'implémenter les méthodes *PCA* et *TSNE* et de tester laquelle fonctionne mieux.

Le binôme 2 se charge d'analyser les modèles et de sélectionner le modèle le plus efficace pour résoudre notre problème.

Le binôme 3 se charge de représenter les données et les résultats de manière à faire ressortir ce qui est étudié. En choisissant une bonne visualisation, non seulement les résultats seront plus compréhensibles, mais il sera également plus facile d'avancer sur le projet.

Premiers résultats

Notre notebook README.ipynb donne les premiers résultats sur lesquels nous avons travaillé. Dans notre problème, on doit prêter une attention toute particulière à ce que le nombre de personnes réellement infectées mais classées comme saines (VN = Vrai Négatif != FP = Faux Positif), soit le plus faible possible. En ce sens, l'algorithme qui présente le meilleur score est le *RandomForestClassifier*, score qui se confirme lorsqu'on observe la matrice de confusion associée à ce classifieur (**Figure 3**). En effet, le *RandomForestClassifier* a globalement les nombres de VN et de FP les plus équilibrés et les plus bas. De plus, ce classifieur n'est pas en *over-fitting*. Cela dit le Perceptron a le nombre de faux négatif le plus faible mais un nombre de faux positif assez élevé ce qui l'enlève de notre sélection du meilleur algorithme.

Figures :

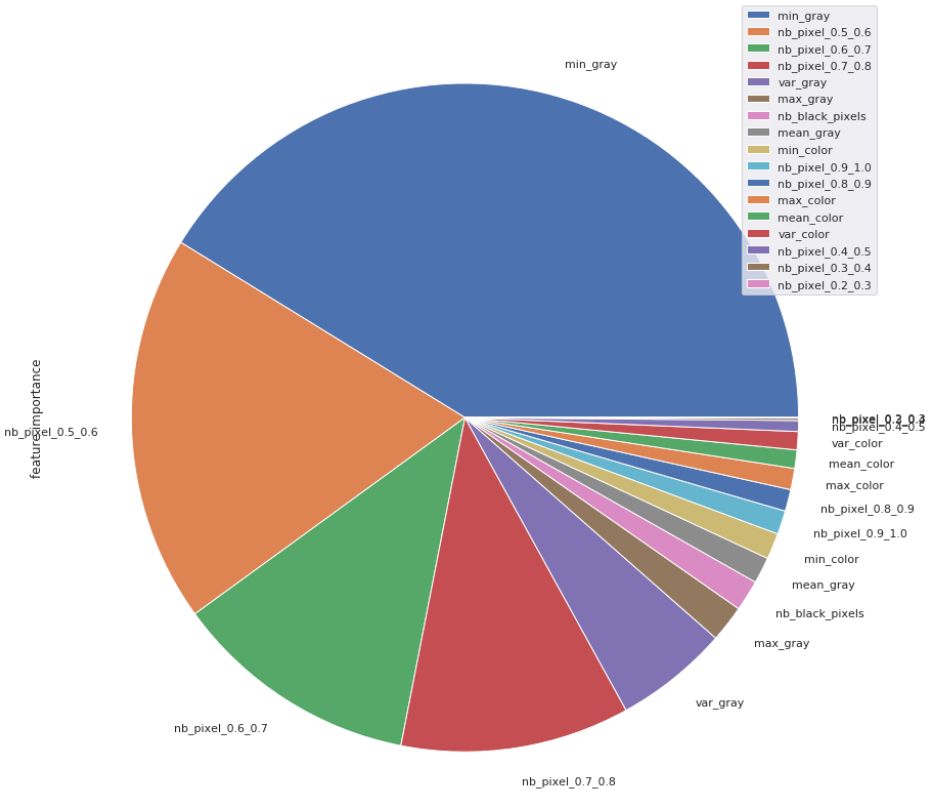


Figure 1: Importance des features proposées par le challenge

Figure 2 : Scores et sur-apprentissage des algorithmes

```
# Affichage des scores de chaque algorithme avec
# les meilleurs hyperparamètres
print(algorithm_scores)
# Meilleur modèle parmi ceux testés
best_model = max([(algorithm_scores[algo], algo) for algo in algorithm_scores.keys()])[1]
print("Best model =", best_model)
```

```
{'Perceptron': 0.8428046592896256, 'MLPClassifier': 0.9340020256250104, 'KNeighborsClassifier': 1.0, 'DecisionTree
Classifier': 0.9481340463512504, 'RandomForestClassifier': 0.9845173615546335}
Best model = KNeighborsClassifier
```

```
check_over_fitting()
```

```
{'Perceptron': False,
 'MLPClassifier': False,
 'KNeighborsClassifier': True,
 'DecisionTreeClassifier': False,
 'RandomForestClassifier': False}
```

Code 1: Sélection des valeurs non aberrantes:

```
liste = []
# le modèle qu'on a utilisé
from sklearn.ensemble import IsolationForest
estimator = IsolationForest(n_estimators = 10)
estimator.fit(X_train)
for i in range(data.shape[0]):
    if estimator.predict(X_train)[i] != -1 :
        # liste des valeurs que l'on veut garder
        liste.append(i)
# on ne garde dans X_train que les valeurs de la liste
X_train = X_train[liste,:]
```

Code 2:

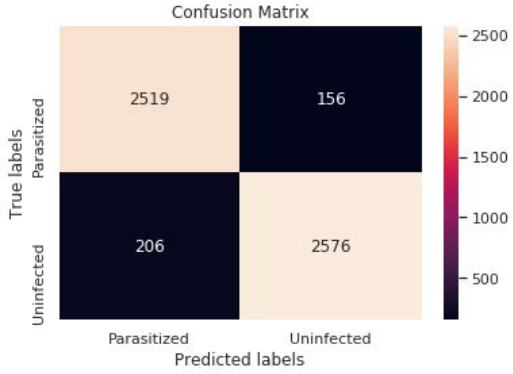
```
# Notre propre algorithme d'apprentissage
class AlgorithmLearning:

        # Constructeur très complexe
        def __init__(self):
                self.learning = False
                self.learnt = [ ]

        # Apprentissage légèrement en sur-apprentissage
        def fit(self, data, label):
                self.learnt = label

        # Prédiction parfaitement dépendante des données
        def predict(self, data):
                return self.learnt
```

Tableau 1 : Statistiques des données sur l'algorithme du RandomForest:

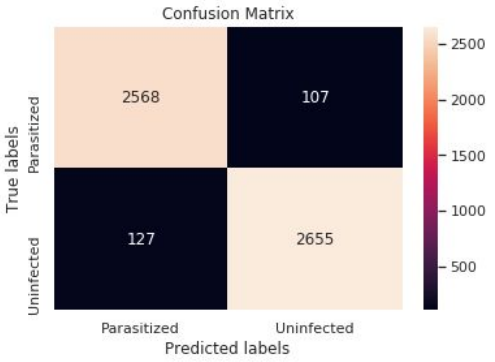
Dataset	Nombre d'exemples	Nombre de features	A des données manquantes ?	Nombre d'exemples dans chaque classe Saines / Infectées	
Training	11077	19 (label compris)	Non	10 905	172
Test	5457	18	Non	5203	254

Figure 3 : Matrices de confusion

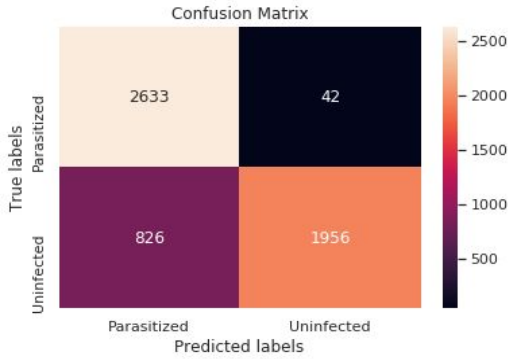
Perceptron Multicouche:



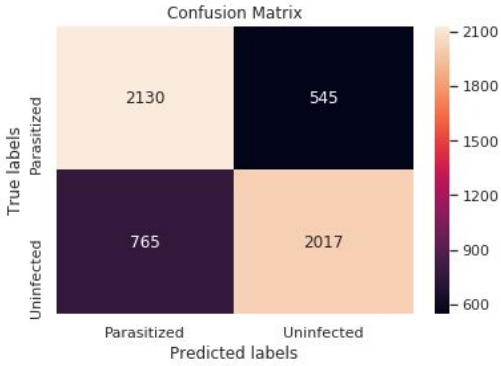
Forêt d'arbres de décision:



K-voisins:



Perceptron:



Arbre de décision: