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*
Lien de la vidéo : *https://www.youtube.com/watch?v=tLvfhfzNeCM*

Binôme 1 preprocessing : Alicia BEC (Groupe 2) & Charlotte TRUPIN (Groupe 2)
Binôme 2 model : Maxime VINCENT (Groupe 3) & Baptiste MAQUET (Groupe 3)
Binôme 3 visualization : Sakina ATIA (Groupe 1) & Mathilde LASSEIGNE (Groupe 4)

Introduction

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.

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, que nous avons départagé. La répartition de nos données est telle que trouvée dans le **Tableau 1**. Nous avons travaillé ce projet, dans un premier temps, sur le document README.ipynb sur le dépot github donné plus haut. Par la suite, nous avons créé un fichier model.py (qui contient une classe preprocessing et une classe modèle),à partir du README.ipynb qui nous a ensuite permis de déposer un fichier sur Codalab.

I/ Description des algorithmes et pseudo-code

Approche choisie

Pour commencer, nous avons choisi de faire une sélection de *features* pour exclure celles qui sont les moins utiles, une détection des outliers avec les méthodes fit et predict et une réduction de dimension avec une comparaison des méthodes *PCA (Principal Component Analysis)* et *TSNE (T-distributed Stochastic Neighbor Embedding)* qui nous a permis de transformer les *features* en une dimension inférieure. Ce qui nous a permis 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* et de *RandomizedSearch* 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).

Le binôme visualization a choisi une approche un peu particulière, étant donné que nous jouons un rôle dans le travail de chaque groupe:

Nous avons choisi dans un premier temps d'analyser les méthodes de visualizations déjà présentes dans le notebook, comme par exemple les matrices de confusions ou encore l'histogramme représentant l'importance des features. Nous trouvions important et nécessaire d'étudier les visualizations déjà fournies et ne pas se précipiter dans la création de nouvelles visualizations pour comprendre quels étaient les résultats attendus et sous quelle forme.

Ensuite nous avons décidé d'implémenter de nouvelles visualizations en utilisant les bibliothèques pyplot et panda d'abord en modifiant celles proposées pour une visualization plus pertinente. Modifier des features déjà présentes nous permettait de mieux comprendre les données avec lesquelles nous

travaillons et de tester des visualizations tout en ayant connaissance du résultat attendu. Nous avons tenté d'améliorer le graphique représentant l'analyse des features en utilisant une représentation en violon. Toutefois, il s'est avéré que l'histogramme original était plus intéressant et plus facile à comprendre à l'œil nu.

Description du travail fait

Dans le model.py, nous avons créé une class preprocess pour que le fichier soit plus clair.
On initialise notamment la pca et l'IsolationForest avec des paramètres respectifs n_pca = 6 et n_esti = 5.
Ensuite on a créé deux fonctions fit et transform. Dans fit on utilise donc la méthode fit d'IsolationForest (**Code 1**) puis dans transform on enlève les outliers avec predict et on appelle la méthode pca.transform (**Code 2**). La sélection de features se fait grâce à la **Figure 1.**

Le binôme 2 à retiré certains attributs et certaines conditions qui pouvaient entraîner des erreurs. Notamment, en ce qui concerne le nombre de données et le nombre de features.
Nous avons ajouté une classe pour le preprocessing, qui permet de réduire le nombre de features du jeu de données. Les méthodes de cette classe sont utilisées dans model::fit() et model::predict(). Nous avons également ajouté les méthodes model::optimize() (**Code 4**) pour optimiser les hyperparamètres, model::score() pour calculer le score, model::confusion_matrix() pour afficher une matrice de confusion.
Pour tester le modèle, nous avons utilisé des tests unitaires sous forme de assert(...)
Un premier test permet de vérifier que le score du modèle optimisé avec de meilleurs hyperparamètres, est meilleur que le score du modèle avec les paramètres pas défaut. Actuellement ce test renvoie une erreur car le preprocessing a faussé le résultat. Ce sera corrigé la semaine prochaine. Le deuxième test est en fait une succession de tests dans une boucle for. Il permet de vérifier le bon comportement de la méthode model::optimize(), qui utilise une RandomizedSearchCV. Plus concrètement, on teste que les hyperparamètres choisis sont bien dans les listes d'hyperparamètres entrés dans la RandomizedSearchCV.

Le binôme 3 s'est chargé 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 sont plus compréhensibles, mais il a également été plus facile d'avancer sur le projet. Le Pie chart représentant l'importance des features donné en **Figure 1** a donc été modifié. Nous trouvions plus intéressant de l'avoir sous cette forme, car il est plus visuel : plus de couleurs, et on voit de suite ce qui est mis en avant.
Nous avons aussi créé deux graphiques simples montrant la répartition de cellules infectées parmi nos données. De plus, nous avons également appliqué les graphiques déjà existants à une plus grande base de données, tel que la matrice de confusion (**Code 3**) que nous avons appliquée à tous les modèles testés par le binôme Model, ce qui les a aidés à sélectionner le plus efficace (**Figure 2**).
C'est pourquoi nous n'avons finalement pas ajouté de nouvelles visualisations car les matrices étaient parfaitement adaptées.

II/ Les résultats

Les résultats de sur-apprentissage (**Tableau 2)** nous ont montré que seul les k-voisins étaient en sur-apprentissage, ce qui se reflétait sur le score. Notre choix de prendre le RandomForest devenait évident, il n'était pas en sur-apprentissage et on avait le meilleur score.
Score qui se confirme lorsqu'on observe la matrice de confusion associée à ce classifieur (**Figure 2**). En effet, le RandomForestClassifier a globalement les nombres de VN et de FP les plus équilibrés et les plus

bas. 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.
A partir de ces résultats nous avons donc créé le fichier model.py pour rassembler le code des groupes model et preprocessing en modifiant certaines choses de notre notebook pour s'adapter au nouveau format. Nous l'avons mis dans le dossier sample_code_submission avec un test.ipynb pour tester certaines fonctions avant de le soumettre sur codalab. Le plus gros challenge ici était de réunir notre code et de faire marcher les class model et preprocess ensemble. Nous avons eu beaucoup de difficultés à faire fonctionner notre code, de nombreux bogues çà et là qui ralentissaient notre progression et empêchaient une soumission sur codalab. Ainsi, une fois ces problèmes résolus, et le code soumis, nous avons obtenu un score de **0.9350383004.**

Conclusion

Ce projet fût très intéressant aussi bien par le sujet abordé que par la façon dont il est traité. Il nous aura permis de mettre en pratique les enseignements reçus au semestre dernier concernant l'apprentissage automatique sur un thème d'actualité. Nous avons pu mieux comprendre quel rôle joue cette étape dans la détection de maladies, et comment elle peut faciliter le travail des médecins.
Cela nous aura permis d'avoir un avant-goût du travail qui nous attend en tant qu'informaticiens: travailler à plusieurs sur un projet commun, avec des données à traiter, des enjeux (ici le nombre de vrai négatifs et de faux positifs), une répartition des tâches précise,et où chacun doit faire son travail dans les temps pour ne pas pénaliser les autres.

Figures :

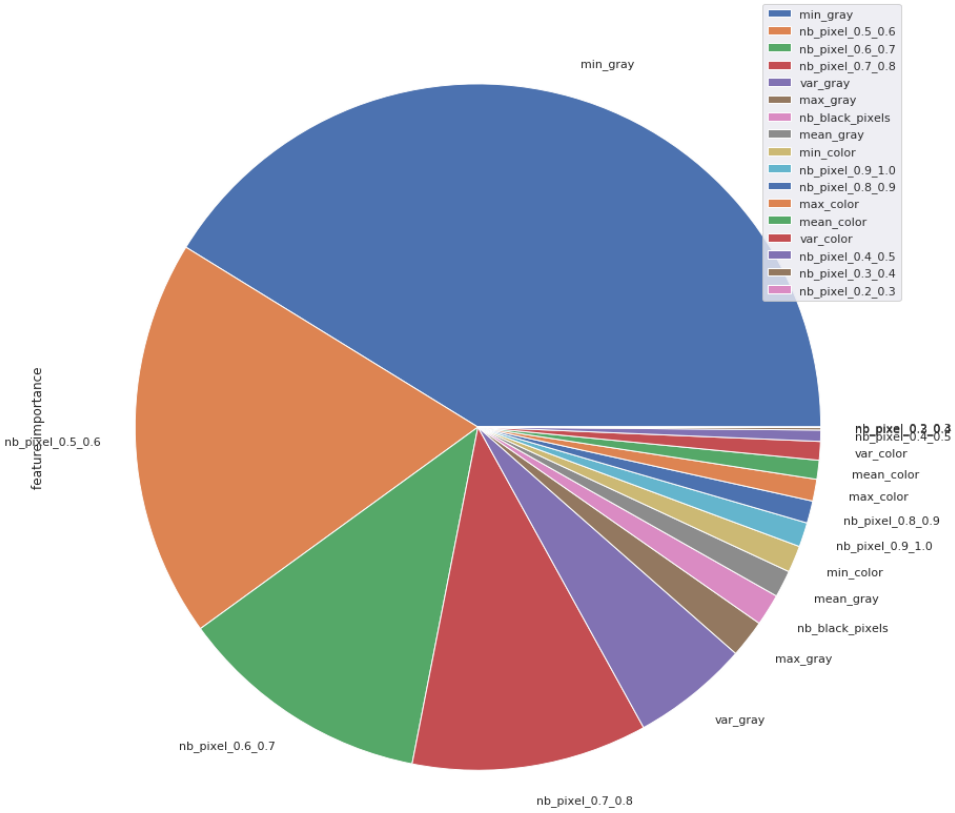
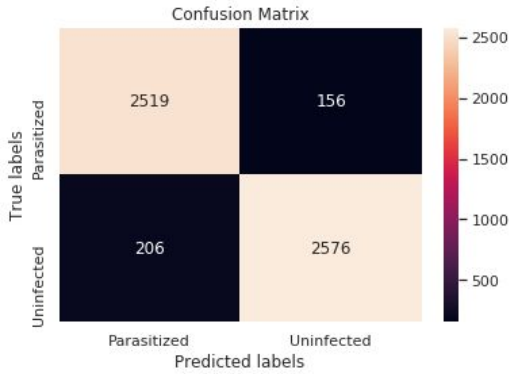


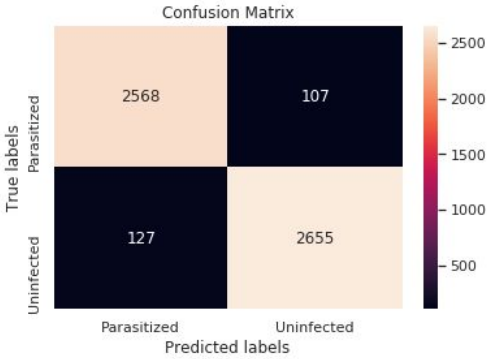
Figure 1: Importance des features proposées par le challenge

Figure 2 : Matrices de confusion

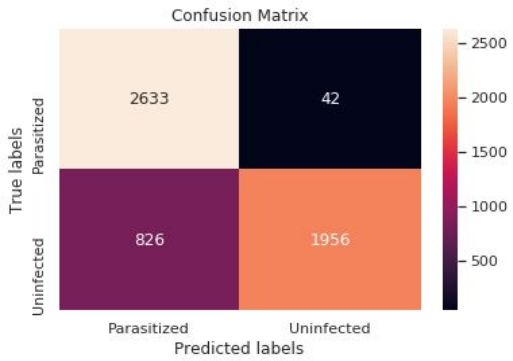
Perceptron Multicouche:



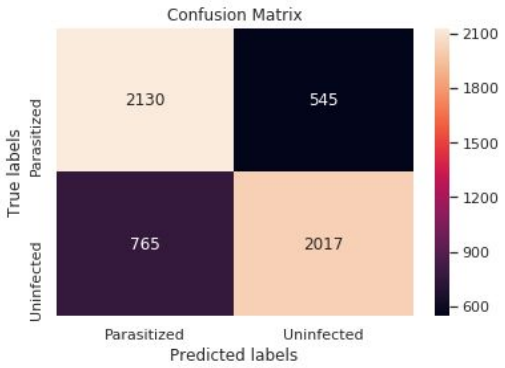
*Forêt d'arbres de décisio*n:



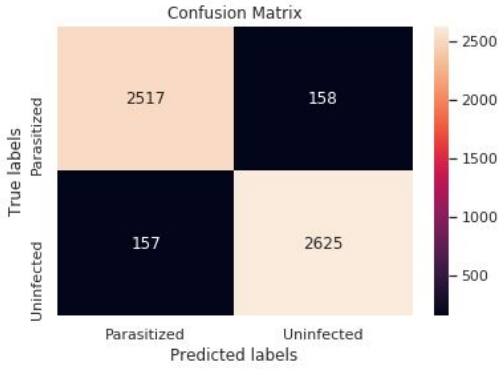
K-voisins:



Perceptron:



Arbre de décision:



Code 1 : fonction fit

```
def fit(self, X):
    X = X[:,[0,5,7,14,15,16]]
    self.pca.fit(X)
    self.estimator.fit(X)
    self.is_trained = True
```

Code 2 : fonction transform

```
def transform(self, X, y, remove_outliers = True):
    """ Preprocessing du jeu de données X """
    X = X[:,[0,5,7,14,15,16]]
    if remove_outliers :
        liste = []
        for i in range(X.shape[0]):
            if self.estimator.predict(X)[i] != -1 :
                liste.append(i)
        X = X[liste, :]
        y = y[liste]
    X = self.pca.transform(X) # reduce dimension
    return X,y
```

Code 3: Pour créer et afficher les matrices de confusion

```
def confusion_matrix(self, X, y):
        ax = plt.subplot()
        # annot=True to annotate cells
        sns.heatmap(confusion_matrix(y, self.classifier.predict(X)), annot=True, fmt='g', ax=ax)

        # labels, title and ticks
        ax.set_xlabel('Predicted labels');
        ax.set_ylabel('True labels');
        ax.set_title('Confusion Matrix');
        ax.xaxis.set_ticklabels(['Parasitized', 'Uninfected']);
        ax.yaxis.set_ticklabels(['Parasitized', 'Uninfected']);
        plt.show()
```

Code 4: Optimisation des hyperparamètres

```
def optimize(self, X, y, n_iter=100):
        parameters={'bootstrap':[True,False],
                'criterion':["gini", "entropy"],
                'n_estimators':[i for i in range(10,300,10)],
                'max_depth':[i for i in range(1,10)]+[None],
                'min_samples_split':[i for i in range(2,5)],
                'min_samples_leaf':[i for i in range(1,5)],
                'random_state':[i for i in range(1,100)]}
        grid = RandomizedSearchCV(self.classifier, parameters,
scoring=make_scorer(self.scoring_function), n_jobs=-1, n_iter=n_iter)
        print(grid.param_distributions)
        grid.fit(X, y)
        print(grid.best_estimator_)
        self.classifier = grid.best_estimator_
```

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 Bien classé/ mal classé**	
Training	11077	19 (label compris)	Non	10 905	172
Test	5457	18	Non	5203	254

Tableau 2 :Sélection du meilleur modèle

Algorithmes	Sur-apprentissage	Score sur l'ensemble d'apprentissage	Score sur l'ensemble de test
Perceptron	False	0.82	0.80
MLPClassifier	False	0.93	0.93
KNeighborsClassifier	True	1	0.77
DecisionTreeClassifier	False	0.95	0.94
RandomForestClassifier	False	0.98	0.95

Bibliographie :

- https://fr.wikipedia.org/wiki/Paludisme
- https://scikit-learn.org/stable/modules/outlier_detection.html
- https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html
- https://scikit-learn.org/stable/modules/feature_selection.html
- https://scikit-learn.org/stable/modules/generated/sklearn.manifold.TSNE.html
- https://pandas.pydata.org/pandas-docs/stable/index.html
- https://matplotlib.org/api/pyplot_api.html
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html
- https://scikit-learn.org/stable/auto_examples/index.html#dataset-examples
- https://matplotlib.org/3.1.1/gallery/
- https://seaborn.pydata.org/examples/index.html