PROJET : PROTECTION DES DONNEES

SENSENBRENNER Amaury / DIDON Valère / WEISS Nicolas

Scénario 1 : Detect malware at runtime

Amaury : Capture des données / Nettoyage des données / Classification binaire et multi-classes / fonction "test_model" pour du multi-classes / Apprentissage multi-classes / Standardisation / Rapport + présentation

Valère : Mise en place d'une fonction "test_model" pour appliquer les mêmes opérations sur les modèles / Début de la comparaison avec quelques indicateurs / mise en commun de "train_test_split" pour que le jeu de test soit le même entre les modèles / uniformisation des matrices de corrélations / Ajout indicateur de temps logarithmique / Fonction Tensorflow

Nicolas : Ajout de métriques pour la classification / Méthode LOFO / Cohérence des données / Métrique AUC-ROC

PRÉSENTATION DU JEU DE DONNÉES

CIC-MalMem-2022

- Un malware obfusqué est un malware qui se cache pour éviter la détection et l'extermination.

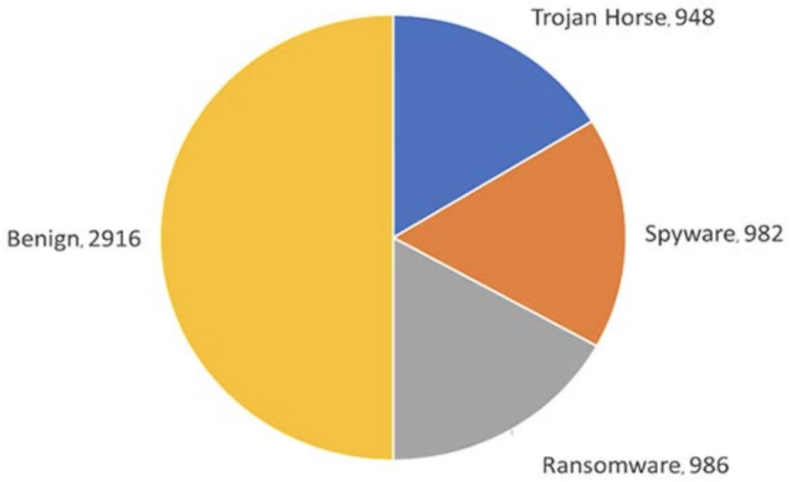
- L'ensemble de données sur les logiciels malveillants obfusqués est conçu pour tester les méthodes de détection des logiciels malveillants obfusqués en mémoire.

- L'ensemble de données a été créé pour représenter une situation aussi proche que possible de la réalité en utilisant des logiciels malveillants répandus dans le monde réel tel que Spyware, Ransomware ou encore Trojan Horse malware.

-Cet ensemble de données utilise le mode débogage pour le processus de vidage de la mémoire afin d'éviter que le processus de vidage n'apparaisse dans les vidages de la mémoire. Cela permet de représenter un exemple plus précis de ce qu'un utilisateur moyen aurait en cours d'exécution au moment d'une attaque de malware.

PRÉSENTATION DU JEU DE DONNÉES

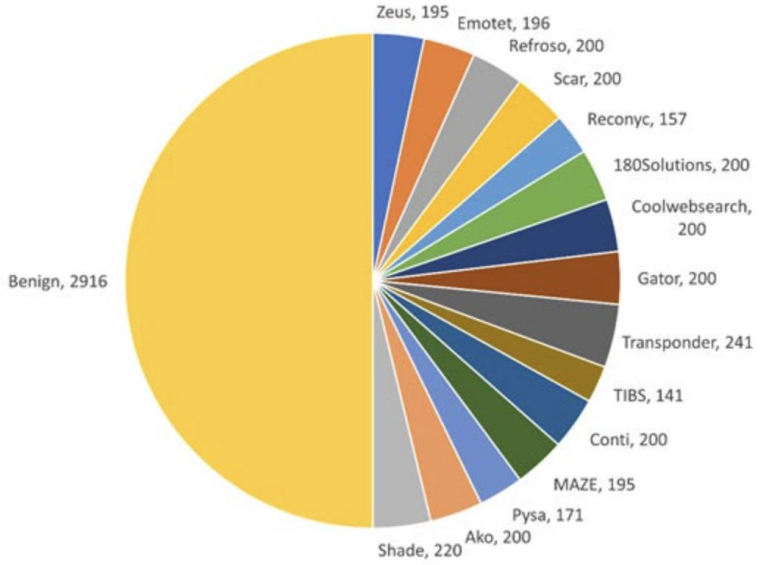
CIC-MalMem-2022



Données divisées en 4 classes

- Ensemble de données équilibré (50/50)

- Jeu de données contient 58 596 entrées



Données divisées en 16 classes

ANALYSE DU JEU DE DONNÉES DE RÉFÉRENCE

```
data.head()
```

	Category	pslist.nproc	pslist.nppid	pslist.avg_threads	pslist.nprocs64bit	pslist.avg_handlers	dlllist.ndlls	dlllist.avg_dlls_per_proc	handles.nhandles	handles.avg_handles_per_proc	...
0	Benign	45	17	10.555556	0	202.844444	1694	38.500000	9129	212.302326	...
1	Benign	47	19	11.531915	0	242.234043	2074	44.127660	11385	242.234043	...
2	Benign	40	14	14.725000	0	288.225000	1932	48.300000	11529	288.225000	...
3	Benign	32	13	13.500000	0	264.281250	1445	45.156250	8457	264.281250	...
4	Benign	42	16	11.452381	0	281.333333	2067	49.214286	11816	281.333333	...

5 rows × 57 columns

Chargement des données + Observation du jeu de données + Visualisation des ensembles de données + Nettoyage des données

NETTOYAGE DES DONNÉES

VALEURS MANQUANTES & VALEURS UNIQUES

```
dfs = [data]
for df in dfs:
  print()
  for c in df.columns:
    wtf = ""
    nulls = df[c].isnull().sum()
    ratio = nulls/len(df)

    if ratio > 0.1:
      print(wtf + c + " : "  + str(nulls) + f" nulls, ratio : {Fore.RED}" + str(round(ratio*100,2)) + f"%{Style.RESET_ALL} de nulls")
    else:
      print(wtf + c + " : "  + str(nulls) + " nulls, ratio : " + str(round(ratio*100,2)) + "% de nulls")
```

Code pour obtenir le pourcentage de valeurs manquantes par colonnes

Code pour obtenir les colonnes ayant une variable unique

```
features_name = list(data.columns)
print(features_name)

for element in features_name:
    if(len(data[element].unique()) < 2):
      print(element)
      print()
      print(data[element].unique())
      print()
```

58596 rows × 54 columns

NETTOYAGE DES DONNÉES

MÉTHODE LOFO (Leave One Feature Out)

```
from sklearn.model_selection import KFold
from lofo import LOFOImportance, Dataset

X_dataset = Dataset(df=pd.concat([X,Y], axis=1), target="Class", features=[col for col in X.columns])

cv = KFold(n_splits=4, shuffle=True, random_state=42)

lofo = LOFOImportance(X_dataset, scoring="roc_auc", cv=cv)

importance_df = lofo.get_importance()
```

```
features_to_delete = []
for feature in X.columns:
    if(feature not in importance_df['feature'][importance_df['importance_mean'] > 0].unique()):
        features_to_delete.append(feature)
```

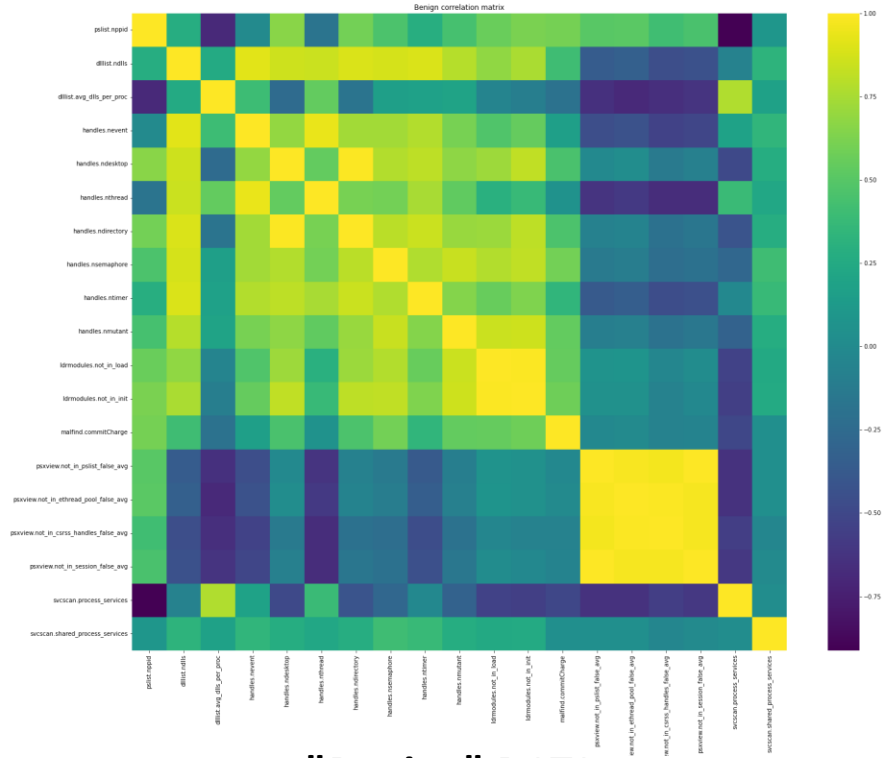
```
for feature in features_to_delete:
    del X[feature]
    del X_4class[feature]
    del data[feature]
```

	feature	importance_mean	importance_std	val_imp_0	val_imp_1	val_imp_2	val_imp_3
3	handles.nevent	1.989817e-06	3.446464e-06	0.000000e+00	7.959269e-06	-2.220446e-16	0.000000e+00
8	svcscan.process_services	5.871592e-07	1.016990e-06	0.000000e+00	2.348637e-06	0.000000e+00	0.000000e+00
20	handles.nsemaphore	5.125993e-07	8.878480e-07	0.000000e+00	2.050397e-06	0.000000e+00	0.000000e+00
5	pslist.nppid	4.473594e-07	7.748492e-07	0.000000e+00	1.789437e-06	0.000000e+00	0.000000e+00
11	psxview.not_in_ethread_pool_false_avg	4.473594e-07	7.748492e-07	0.000000e+00	1.789437e-06	0.000000e+00	0.000000e+00
37	ldrmodules.not_in_init	2.656196e-07	4.600667e-07	0.000000e+00	1.062479e-06	0.000000e+00	0.000000e+00
48	dlllist.ndlls	2.609596e-07	4.519953e-07	0.000000e+00	1.043839e-06	0.000000e+00	0.000000e+00
6	handles.ntimer	2.283397e-07	3.954959e-07	-2.220446e-16	9.133587e-07	0.000000e+00	0.000000e+00
40	psxview.not_in_session_false_avg	1.491198e-07	2.582831e-07	0.000000e+00	5.964792e-07	0.000000e+00	0.000000e+00
44	handles.ndesktop	7.455990e-08	1.291415e-07	0.000000e+00	2.982396e-07	0.000000e+00	0.000000e+00
13	handles.ndirectory	6.989990e-08	1.210702e-07	0.000000e+00	2.795996e-07	0.000000e+00	0.000000e+00
30	svcscan.shared_process_services	4.659993e-08	8.071345e-08	1.110223e-16	1.863997e-07	0.000000e+00	0.000000e+00
47	psxview.not_in_csrss_handles_false_avg	4.193994e-08	7.264211e-08	0.000000e+00	1.677598e-07	0.000000e+00	1.110223e-16
38	handles.nmutant	4.193994e-08	7.264211e-08	0.000000e+00	1.677598e-07	0.000000e+00	0.000000e+00
2	ldrmodules.not_in_load	1.863997e-08	3.228538e-08	0.000000e+00	7.455990e-08	0.000000e+00	0.000000e+00

58596 rows × 19 columns

MATRICE DE CORRÉLATION

SÉPARATION DU DATASET POUR ANALYSE

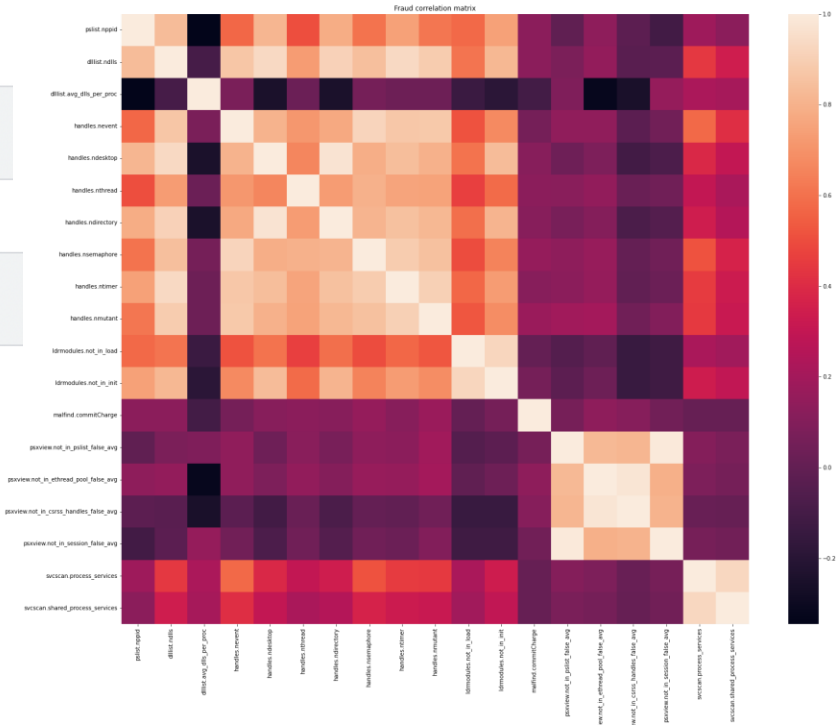


"Benign" DATA

```
isBenign_data = data.loc[data['Category'] == 'Benign']
```

```
isFraud_data = data.loc[data['Category'] != 'Benign']
```

On peut observer de fortes corrélations entre les différentes colonnes du Dataset pour les deux jeux de données.



"Fraud" DATA

ALGORITHMES DE CLASSIFICATION

Les algorithmes de classification utilisés dans ce projet :

- CART (Classification And Regression Trees)
- XGBoost
- RF (Random Forest)
- MLP (MultiLayer Perceptron)
- KNN (K-nearest neighbors)
- SVM (Support Vector Machines)

```
test_model("clf",tree.DecisionTreeClassifier,random_state=42)
test_model("xbg",XGBClassifier,objective="binary:logistic",random_state=42)
test_model("rf",RandomForestClassifier,n_estimators=100)
test_model("mlp",MLPClassifier,max_iter=300,random_state=42)
test_model("knn",KNeighborsClassifier, n_neighbors=3)
test_model("svm",svm.SVC,kernel='linear', C=1, probability=True,random_state=42)
```

DIFFÉRENTES MÉTRIQUES

[17]:

```
import time
models = {}
time_fit = {}
time_pred = {}
def test_model(model_name, model_f, **model_args):
    models[model_name] = model_f(**model_args)
    deb = time.time()
    models[model_name].fit(X_train, y_train)
    time_fit[model_name]=time.time()-deb
    print("temps de fit : ", time_fit[model_name])
    deb = time.time()
    prediction = models[model_name].predict(X_test)
    time_pred[model_name]=time.time()-deb
    print("temps de prédiction : ", time_pred[model_name])
    proba = models[model_name].predict_proba(X_test)
    print("prediction : ", prediction)
    print("proba : ", proba)
    prediction = pd.DataFrame(prediction, columns = ['Class'])
    print("confusion matrix : ")
    conf_matrix = confusion_matrix(y_test,prediction)
    print("pp_matrix_from_data : ")
    pp_matrix_from_data(y_test,prediction)
    print("classification_report : ")
    print(classification_report(y_test, prediction))
    print("balanced_accuracy_score :", balanced_accuracy_score(y_test, prediction))
    print("matthews_corrcoef :", matthews_corrcoef(y_test, prediction))
    FP = conf_matrix.sum(axis=0) - np.diag(conf_matrix)
    FN = conf_matrix.sum(axis=1) - np.diag(conf_matrix)
    TP = np.diag(conf_matrix)
    TN = conf_matrix.sum() - (FP + FN + TP)
    print("TNRs :", TN/(TN+FP))
    print("TPRs :", TP/(TP+FN))
```

Informations pour chaque algorithme :

- Temps d'entraînement du modèle
- Temps de prédiction du modèle
- Prédiction
- Probabilité de chaque prédiction
- Matrice de confusion
- Classification du modèle (precision / recall / f1-score)
- "Balanced accuracy report"
- "Matthews Correlation Coefficient"
- TNR (True negative rate)
- TPR (True positive rate)

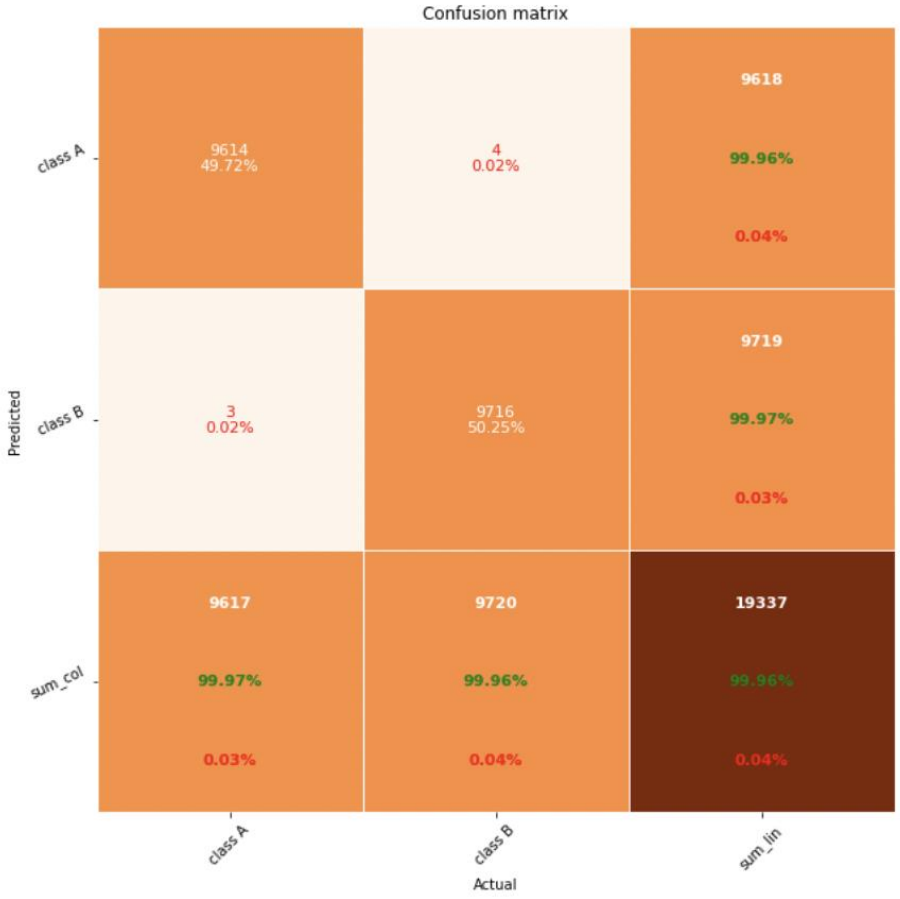
EXEMPLE DE RÉSULTATS DES MÉTRIQUES

Résultats obtenus pour l'algorithme CART (classification binaire)

```
temps de fit :  0.5171411037445068
temps de prédiction :  0.008604288101196289
prediction :  ['Benign' 'Benign' 'Benign' ... 'Malware' 'Malware' 'Malware']
proba :  [[1. 0.]
 [1. 0.]
 [1. 0.]
 ...
 [0. 1.]
 [0. 1.]
 [0. 1.]]
```

```
classification_report :
              precision    recall  f1-score   support

      Benign       1.00      1.00      1.00      9617
     Malware       1.00      1.00      1.00      9720

    accuracy                           1.00     19337
   macro avg       1.00      1.00      1.00     19337
weighted avg       1.00      1.00      1.00     19337

balanced_accuracy_score : 0.9996382648867254
matthews_corrcoef : 0.9992759845812595
TNRs : [0.99958848 0.99968805]
TPRs : [0.99968805 0.99958848]
```

pp_matrix_from_data :



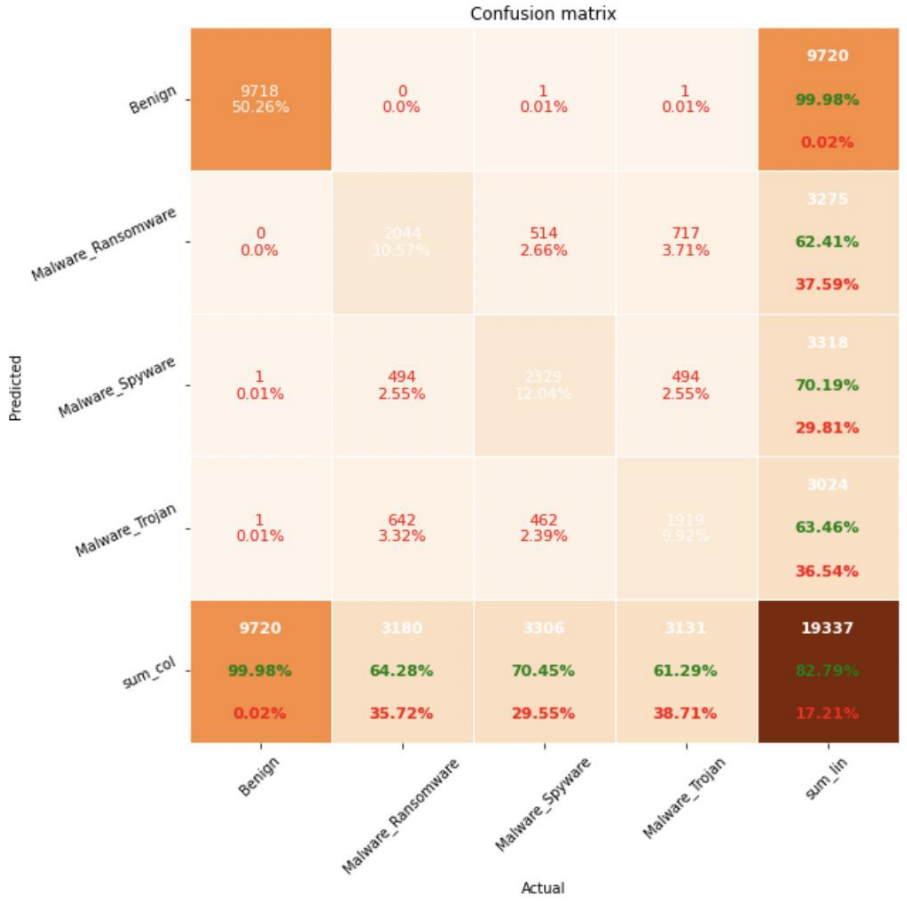
EXEMPLE DE RÉSULTATS DES MÉTRIQUES

Résultats obtenus pour l'algorithme CART (classification multi-classe)

```
--->  clf_4class
temps de fit :  0.46209001541137695
temps de prédiction :  0.008464574813842773
antiscore :  0.1720535760459223
prediction :  ['Malware_Trojan' 'Malware_Ransomware' 'Malware_Spyware' ... 'Benign'
 'Benign' 'Benign']
proba :  [[0. 0. 0. 1.]
 [0. 1. 0. 0.]
 [0. 0. 1. 0.]
 ...
 [1. 0. 0. 0.]
 [1. 0. 0. 0.]
 [1. 0. 0. 0.]]
```

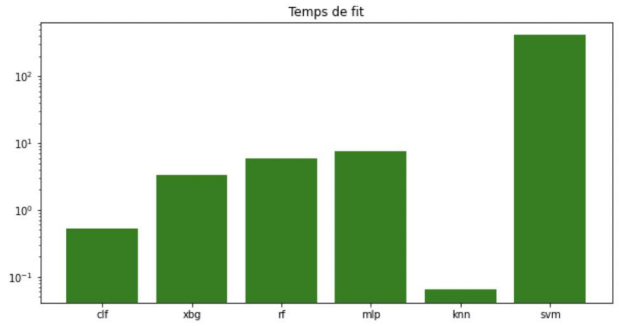
```
classification_report :
                   precision    recall  f1-score   support

           Benign       1.00      1.00      1.00      9720
Malware_Ransomware      0.62      0.64      0.63      3180
   Malware_Spyware      0.70      0.70      0.70      3306
    Malware_Trojan      0.63      0.61      0.62      3131

         accuracy                           0.83     19337
        macro avg       0.74      0.74      0.74     19337
     weighted avg       0.83      0.83      0.83     19337

balanced_accuracy_score : 0.7399853672752444
matthews_corrcoef : 0.741233517825787
TNRs : [0.99979203 0.92381011 0.93830703 0.93181538]
TPRs : [0.99979424 0.6427673  0.70447671 0.61290323]
```

```
confusion matrix :
pp_matrix_from_data :
```

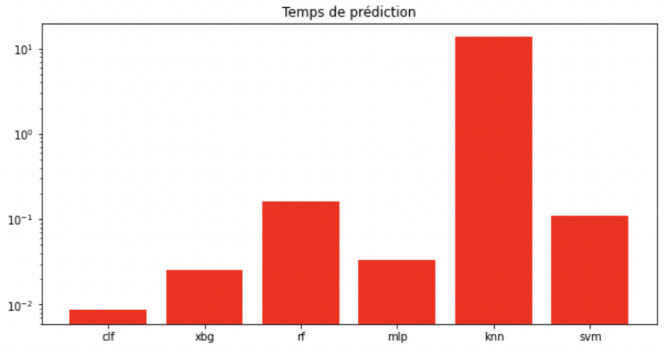


COMPARAISON DES MODÈLES

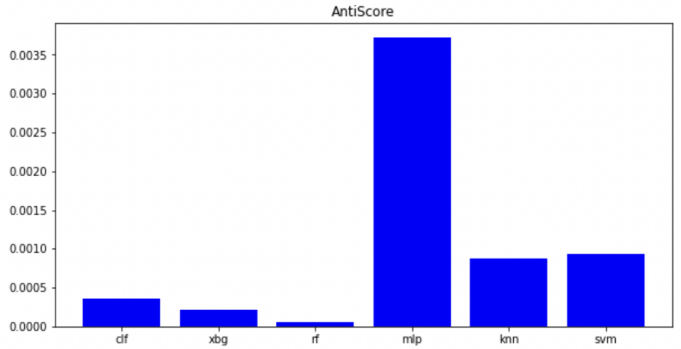
Mesure du temps de prédiction et du temps de fit de chaque modèle effectué à l'aide du module "time" disponible sur Python. Le module time en Python est une des façons les plus simple de mesurer le temps dans un programme.



Temps de fit



Temps de prédiction



Antiscore

TEMPS DE FIT

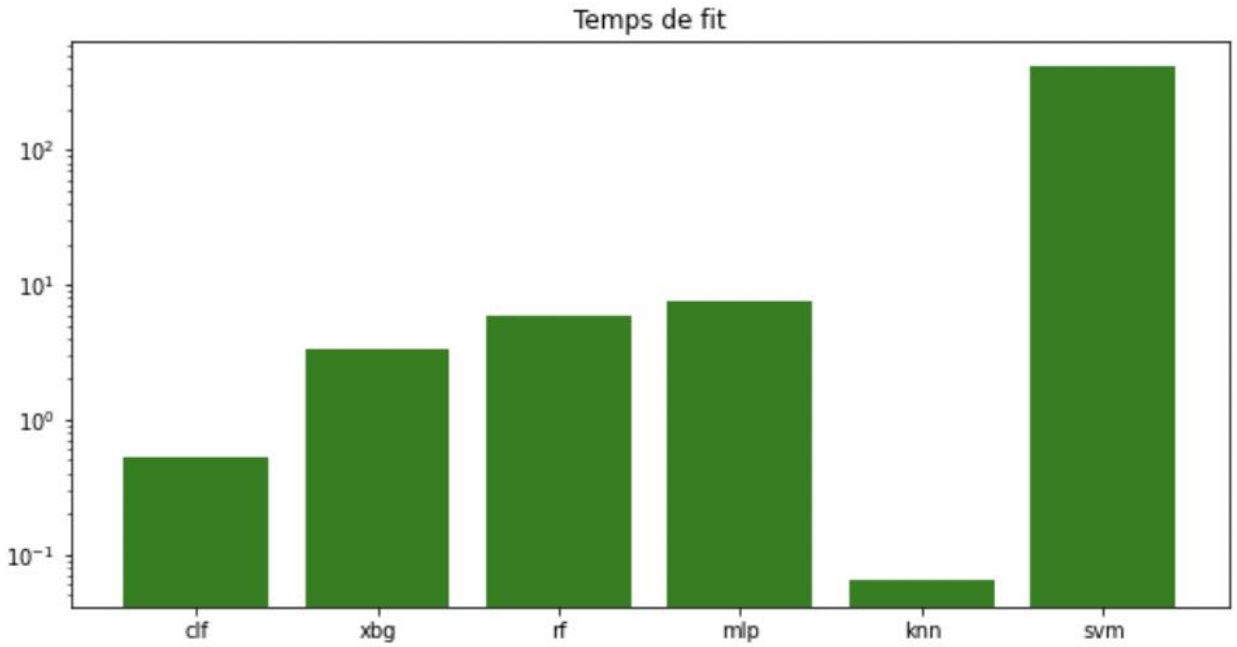
- Temps de fit effectué sur le même jeu de données pour l'ensemble des modèles.
- Unité de mesure : temps logarithme.

MEILLEUR MODÈLE

- KNN (K-nearest neighbors)
- CART (Classification And Regression Trees)

MOINS BON MODÈLE

- SVM (Support Vector Machines)



TEMPS DE PRÉDICTION

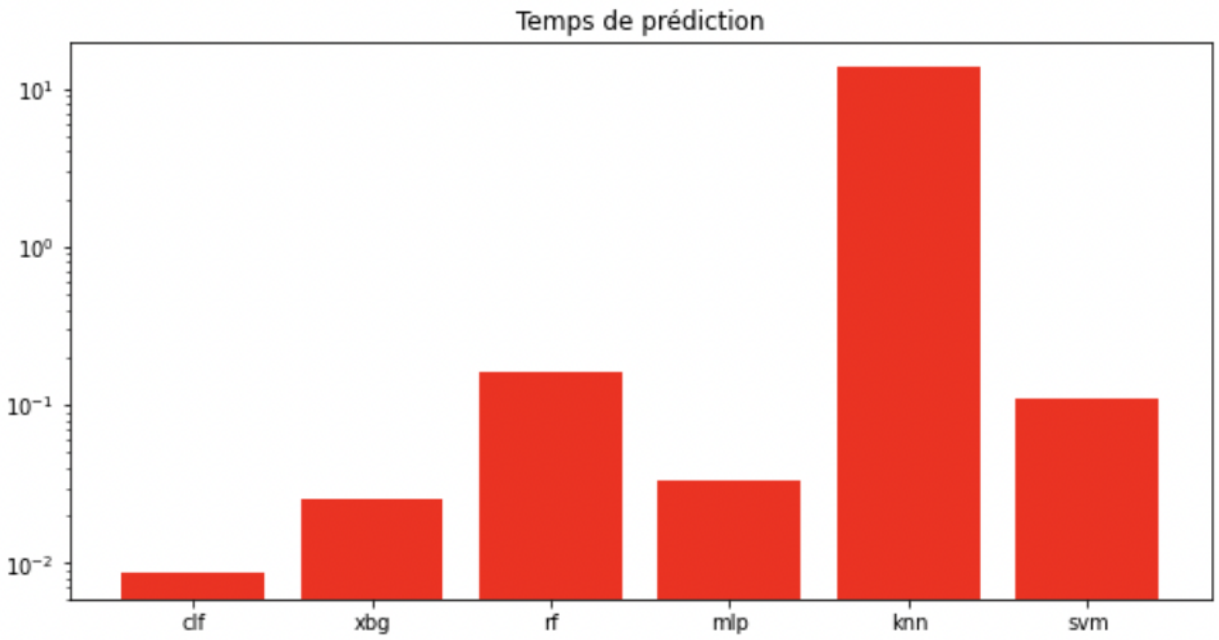
- Temps de prédiction effectué sur le même jeu de données pour l'ensemble des modèles.
- Unité de mesure : temps logarithme.

MEILLEUR MODÈLE

- CART (Classification And Regression Trees)
- XGBoost

MOINS BON MODÈLE

- KNN (K-nearest neighbors)
- RF (Random Forest)



ANTISCORE

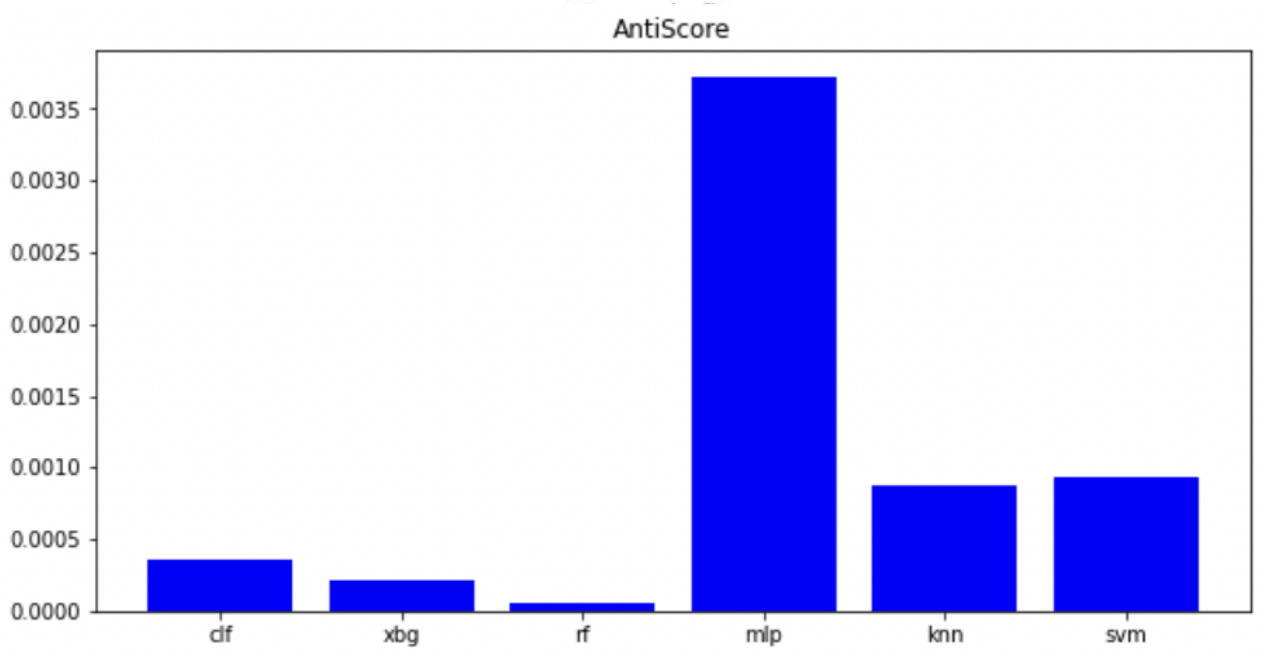
- Antiscore effectué sur le même jeu de données pour l'ensemble des modèles.
- Formule : 1 - model.score

MEILLEUR MODÈLE

- RF (Random Forest)
- XGBoost

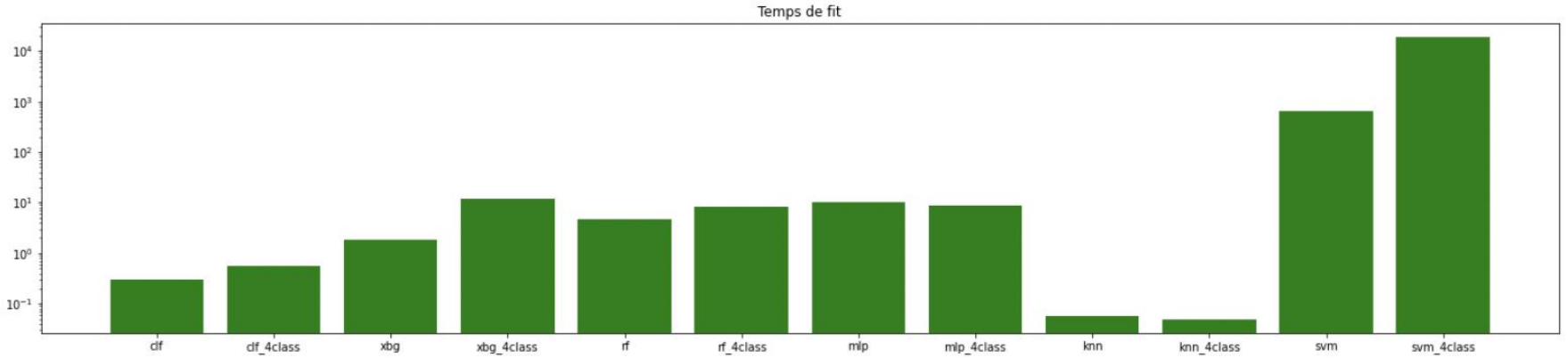
MOINS BON MODÈLE

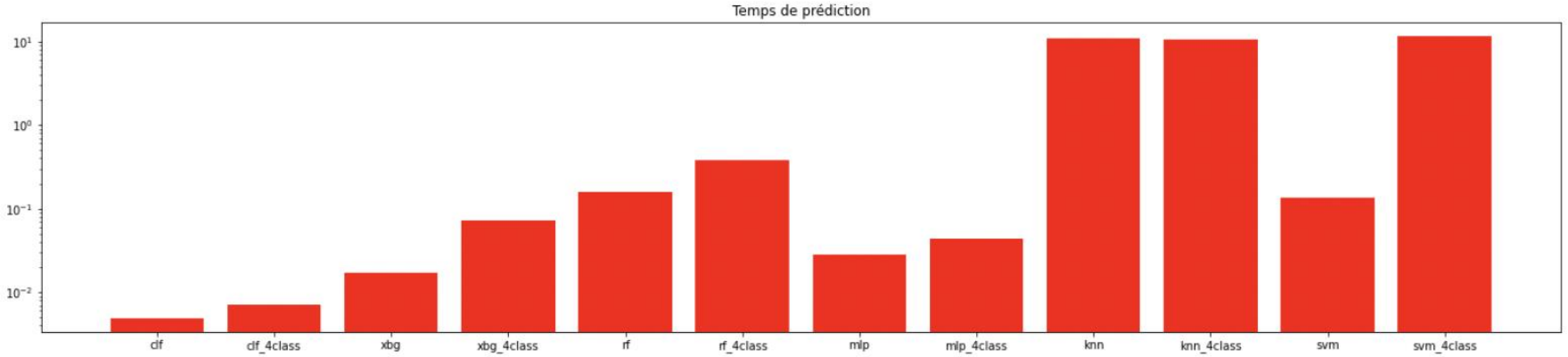
- MLP (MultiLayer Perceptron)

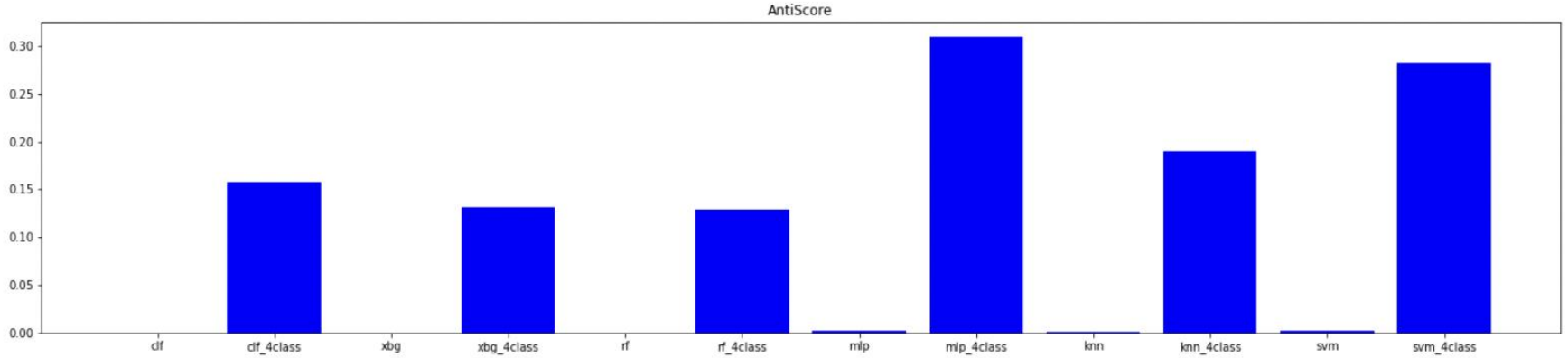


CLASSIFICATION BINAIRE & MULTI-CLASSES

TEMPS FIT
TEMPS PRÉDICTION
ANTISCORE







TRAVAUX RÉALISÉS PAR LA SUITE

(AVEC UN EXEMPLE : CART)

- **Ajout et compraison de variante** (choix des options utilisées) pour chaque modèle cité ci-dessus.

- **Enregistrement des modèles pré-entrainés** dans un fichier.

- 558 **modèles (multi-classes et binaire) à comparer** donc beaucoup de travail.

- **Analyse et extraction** des variantes de modèle les plus intérressantes.

RECHERCHE DU MEILLEUR PARAMÈTRE

(AVEC UN EXEMPLE : CART AVEC CLASSIFICATION BINAIRE)

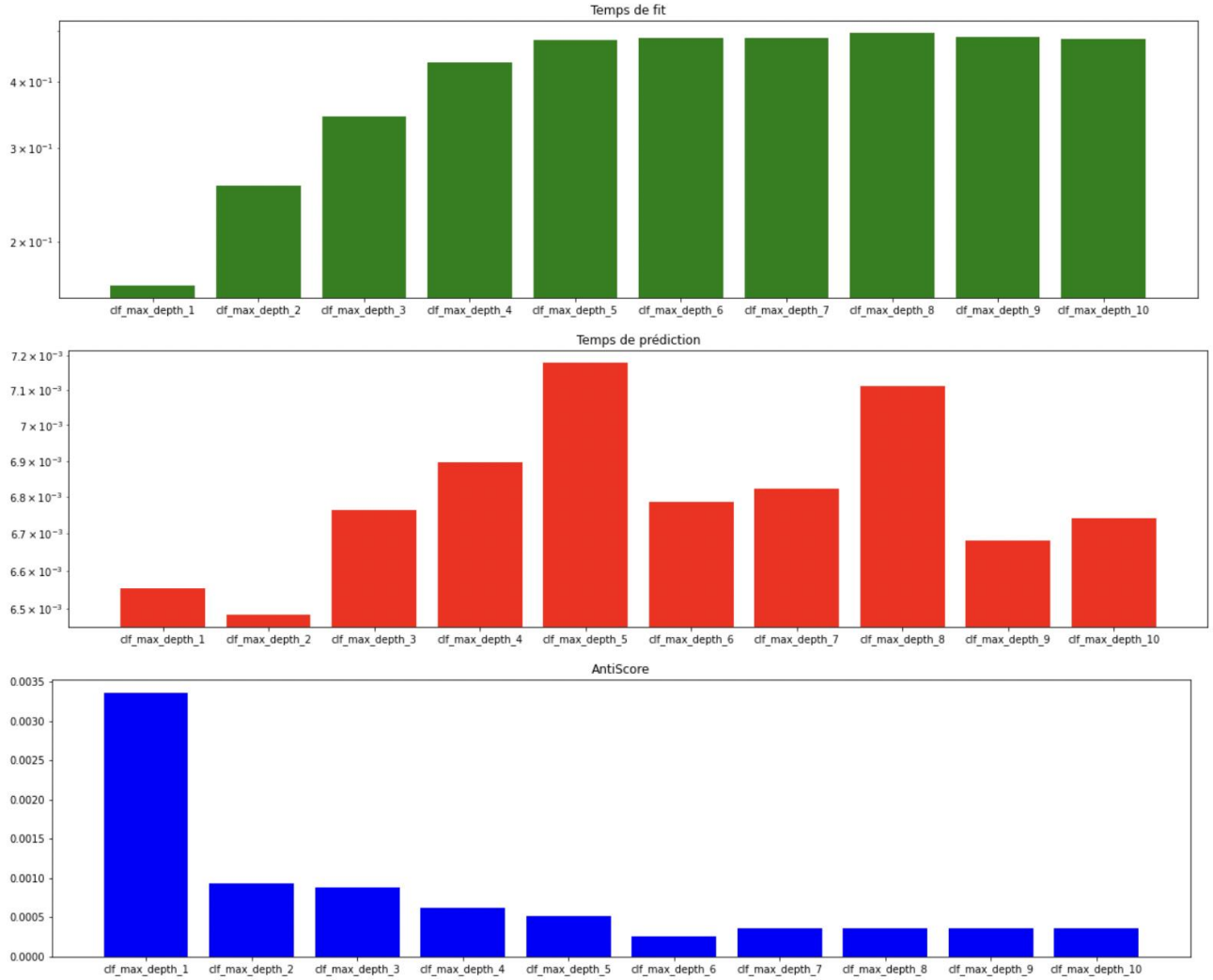
ANALYSE MULTI-CLASSE DISPONIBLE DANS LE NOTEBOOK

```
for max_depth in [1,2,3,4,5,6,7,8,9,10]:
    test_model("clf_max_depth_"+str(max_depth),tree.DecisionTreeClassifier,random_state=42,max_depth=max_depth)
compare(time_fit, time_pred, models)
clean_tabs()
for class_weight in [None, 'balanced']:
    test_model("clf_class_weight_"+str(class_weight),tree.DecisionTreeClassifier,random_state=42,class_weight=class_weight)
compare(time_fit, time_pred, models)
clean_tabs()
for criterion in ['gini', 'entropy']:
    test_model("clf_criterion_"+str(criterion),tree.DecisionTreeClassifier,random_state=42,criterion=criterion)
compare(time_fit, time_pred, models)
clean_tabs()
for splitter in ['best', 'random']:
    test_model("clf_splitter_"+str(splitter),tree.DecisionTreeClassifier,random_state=42,splitter=splitter)
compare(time_fit, time_pred, models)
clean_tabs()
for max_features in [None, 'auto', 'sqrt', 'log2']:
    test_model("clf_max_features_"+str(max_features),tree.DecisionTreeClassifier,random_state=42,max_features=max_features)
compare(time_fit, time_pred, models)
clean_tabs()
for min_samples_split in [2,3,4,5,6,7,8,9,10]:
    test_model("clf_min_samples_split_"+str(min_samples_split),tree.DecisionTreeClassifier,random_state=42,min_samples_split=min_samples_split)
compare(time_fit, time_pred, models)
clean_tabs()
for min_samples_leaf in [1,2,3,4,5,6,7,8,9,10]:
    test_model("clf_min_samples_leaf_"+str(min_samples_leaf),tree.DecisionTreeClassifier,random_state=42,min_samples_leaf=min_samples_leaf)
compare(time_fit, time_pred, models)
clean_tabs()
for min_weight_fraction_leaf in [0.0,0.1,0.2,0.3,0.4,0.5]:
    test_model("clf_min_weight_fraction_leaf_"+str(min_weight_fraction_leaf),tree.DecisionTreeClassifier,random_state=42,min_weight_fraction_leaf=min_weight_fraction_leaf)
compare(time_fit, time_pred, models)
clean_tabs()
for max_leaf_nodes in [None,2,3,4,5,6,7,8,9,10]:
    test_model("clf_max_leaf_nodes_"+str(max_leaf_nodes),tree.DecisionTreeClassifier,random_state=42,max_leaf_nodes=max_leaf_nodes)
compare(time_fit, time_pred, models)
clean_tabs()
for min_impurity_decrease in [0.0,0.1,0.2,0.3,0.4,0.5]:
    test_model("clf_min_impurity_decrease_"+str(min_impurity_decrease),tree.DecisionTreeClassifier,random_state=42,min_impurity_decrease=min_impurity_decrease)
compare(time_fit, time_pred, models)
clean_tabs()
for ccp_alpha in [0.0,0.1,0.2,0.3,0.4,0.5]:
    test_model("clf_ccp_alpha_"+str(ccp_alpha),tree.DecisionTreeClassifier,random_state=42,ccp_alpha=ccp_alpha)
compare(time_fit, time_pred, models)
clean_tabs()
```

ANALYSE DES RÉSULTATS

(AVEC UN EXEMPLE : CART AVEC CLASSIFICATION BINAIRE)

TOUS LES MODÈLES SONT STOCKÉS AFIN D'ÊTRE ANALYSÉS PAR LA SUITE.



RECHERCHE DU MEILLEUR MODÈLE

TOUS ALGORITHMES DE CLASSIFICATION CONFONDUS

```
# Meilleur modèle par l'antiscore
best_model = min(antiscore_all, key=antiscore_all.get)
print("Meilleur modèle par le score : ", best_model)
print("Score : ", 1 - antiscore_all[best_model])
print("Temps de fit : ", time_fit_all[best_model])
print("Temps de prédiction : ", time_pred_all[best_model])
print("Paramètres : ", models_all[best_model].get_params())
# Meilleur modèle par time_fit
best_model = min(time_fit_all, key=time_fit_all.get)
print("Meilleur modèle par le temps de fit : ", best_model)
print("Score : ", 1 - antiscore_all[best_model])
print("Temps de fit : ", time_fit_all[best_model])
print("Temps de prédiction : ", time_pred_all[best_model])
print("Paramètres : ", models_all[best_model].get_params())
# Meilleur modèle par time_pred
best_model = min(time_pred_all, key=time_pred_all.get)
print("Meilleur modèle par le temps de prédiction : ", best_model)
print("Score : ", 1 - antiscore_all[best_model])
print("Temps de fit : ", time_fit_all[best_model])
print("Temps de prédiction : ", time_pred_all[best_model])
print("Paramètres : ", models_all[best_model].get_params())
```

RECHERCHE DU MEILLEUR MODÈLE

TOUS ALGORITHMES DE CLASSIFICATION CONFONDUS

```
Meilleur modèle par le score :  rf
Score :  1.0
Temps de fit :  5.018779039382935
Temps de prédiction :  0.15602922439575195
Paramètres :  {'bootstrap': True, 'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gin
i', 'max_depth': None, 'max_features': 'auto', 'max_leaf_nodes': None, 'max_samples': None,
'min_impurity_decrease': 0.0, 'min_samples_leaf': 1, 'min_samples_split': 2, 'min_weight_fra
ction_leaf': 0.0, 'n_estimators': 100, 'n_jobs': None, 'oob_score': False, 'random_state': N
one, 'verbose': 0, 'warm_start': False}
Meilleur modèle par le temps de fit :  knn_4class
Score :  0.807984692558308
Temps de fit :  0.050388336181640625
Temps de prédiction :  11.087274312973022
Paramètres :  {'algorithm': 'auto', 'leaf_size': 30, 'metric': 'minkowski', 'metric_params':
None, 'n_jobs': None, 'n_neighbors': 3, 'p': 2, 'weights': 'uniform'}
Meilleur modèle par le temps de prédiction :  clf_min_impurity_decrease_0.5
Score :  0.497336712002896
Temps de fit :  0.11419987678527832
Temps de prédiction :  0.004103183746337891
Paramètres :  {'ccp_alpha': 0.0, 'class_weight': None, 'criterion': 'gini', 'max_depth': Non
e, 'max_features': None, 'max_leaf_nodes': None, 'min_impurity_decrease': 0.5, 'min_samples_
leaf': 1, 'min_samples_split': 2, 'min_weight_fraction_leaf': 0.0, 'random_state': 42, 'spli
tter': 'best'}
```

RECHERCHE DU MEILLEUR MODÈLE

UTILISATION DE TENSORFLOW POUR TROUVER LES MEILLEURS PARAMÈTRES

```python
def find_best_parameters(models_all,type,indicators,indicators_val):
    df = pd.DataFrame(columns=indicators)
    for model in models_all.keys():
        if model.startswith(type):
            for param in models_all[model].get_params():
                df.join(pd.DataFrame(columns=[param]))
    for model in models_all.keys():
        if model.startswith(type):
            for ind in indicators:
                df.loc[model,ind] = indicators_val[ind][model]
            for param in models_all[model].get_params():
                df.loc[model,param] = models_all[model].get_params()[param]
    X = df[indicators]
    y = df.drop(columns=indicators)
    y = pd.get_dummies(y)
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
    X_train = np.asarray(X_train).astype(np.float32)
    X_test = np.asarray(X_test).astype(np.float32)
    y_train = np.asarray(y_train).astype(np.float32)
    y_test = np.asarray(y_test).astype(np.float32)
    model_tf = keras.Sequential([keras.layers.Dense(128, activation='relu', input_shape=[len(X_train[0])]),keras.layers.Dense(128, activation='relu'),keras.layers.Dense(128, activation='relu'),keras.layers.Dense(128, activation='relu'),keras.layers.Dense(128, activation='relu'),keras.layers.Dense(128, activation='relu'),keras.layers.Dense(128, activation='relu'),keras.layers.Dense(128, activation='relu'),keras.layers.Dense(128, activation='relu'),keras.layers.Dense(len(y_test[0]))])
    #model_tf = keras.Sequential([])
    model_tf.compile(optimizer='adam', loss='mean_squared_error', metrics=['accuracy'])
    history = model_tf.fit(X_train, y_train, epochs=1000, validation_split=0.2, verbose=0)
    best = model_tf.predict([[0]*len(indicators)])
    for i in range(len(y.keys())):
        print(f"{y.keys()[i]} = {best[0][i]}")
    return model_tf, history, y, best
```

```python
model_tf, history, y, best = find_best_parameters(models_all,"mlp",["time_fit","time_pred","antiscore"],{"time_fit":time_fit_all,"time_pred":time_pred_all,"antiscore":antiscore_all})
```

```
alpha = 4.963172912597656
beta_1 = -0.028665196150541306
beta_2 = 3.7918806076049805
epsilon = -0.9422153234481812
hidden_layer_sizes = 136.49241638183594
learning_rate_init = 0.5834686756134033
max_fun = 12673.32421875
max_iter = 282.2931213378906
momentum = 0.08763692528009415
n_iter_no_change = 4.4831390380859375
power_t = -10.529962539672852
random_state = 35.790523529052734
tol = 3.6669726371765137
validation_fraction = 0.7735840678215027
activation_identity = -1.5252374410629272
activation_logistic = -3.8054447174072266
activation_relu = 0.12042991071939468
activation_tanh = 0.955704391002655
batch_size_auto = 2.046154022216797
early_stopping_False = 0.31483888626098633
early_stopping_True = 3.5943212509155273
learning_rate_adaptive = -2.2086594104766846
learning_rate_constant = 1.6380572319030762
learning_rate_invscaling = 2.427823066711426
nesterovs_momentum_False = -0.9295732975006104
nesterovs_momentum_True = -2.6827199459075928
shuffle_False = -0.28794020414352417
shuffle_True = 2.3815858364105225
solver_adam = -0.41793352365493774
solver_lbfgs = -0.8703501224517822
solver_sgd = 0.2866825759410858
verbose_False = 4.319645404815674
warm_start_False = 1.9286432266235352
```

EXEMPLE AVEC LE MODÈLE DE CLASSIFICATION MLP

RECHERCHE DU MEILLEUR MODÈLE

UTILISATION DE TENSORFLOW POUR TROUVER LES MEILLEURS PARAMÈTRES

RÉSULTAT FINAL (CLASSIFICATION BINAIRE) :

```
test_model("mlp_best",MLPClassifier,alpha = 0, beta_1 = 0.99999999999, beta_2 = 0, epsilon =
0.00001, hidden_layer_sizes = 110, learning_rate_init = 0.23760008811950684, max_fun = 1314
8.3505859375, max_iter = 296, momentum = 0, n_iter_no_change = 8.396625518798828, power_t =
4.656700611114502, random_state = 42, tol = -1.833219051361084, validation_fraction = 0, act
ivation="tanh", batch_size="auto", early_stopping=False, learning_rate="constant", nesterovs
_momentum=True, shuffle=False, solver="sgd")
```

```
--->  mlp_best
temps de fit :  73.74049401283264
temps de prédiction :  0.0379335880279541
antiscore :  0.497336712002896
```

Conclusion : résultats des test non concluants avec Tensorflow.

CONCLUSION

- Trouver le modèle qui possède le meilleur antiscore, le modèle qui possède le meilleur temps de fit et le modèle possédant le meilleur temps de prédiction car nous avons effectué des tests sur l'ensembles des paramètres possibles.
- Mais comme vous avez pu le voir, cela prend du temps de construire et d'analyser un aussi grand nombre de données.
- Utilisation de Tensorflow afin de trouver plus rapidement les meilleurs paramètres pour un modèle de classification.
- Conclusion : résultats de test que nous avons effectuées ne sont pas concluants.

MERCI POUR VOTRE ATTENTION !

SENSENBRENNER Amaury / DIDON Valère / WEISS Nicolas