Challenge - 2020 - MPRI

Spinelli Isaïa

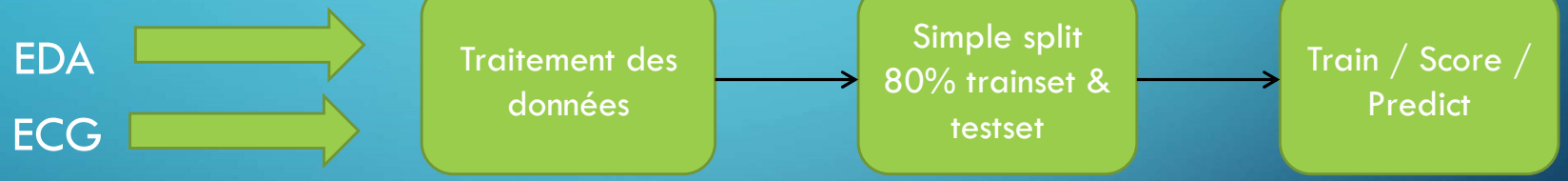
(RF)

Challenge - MPRI

Le 15 décembre 2020

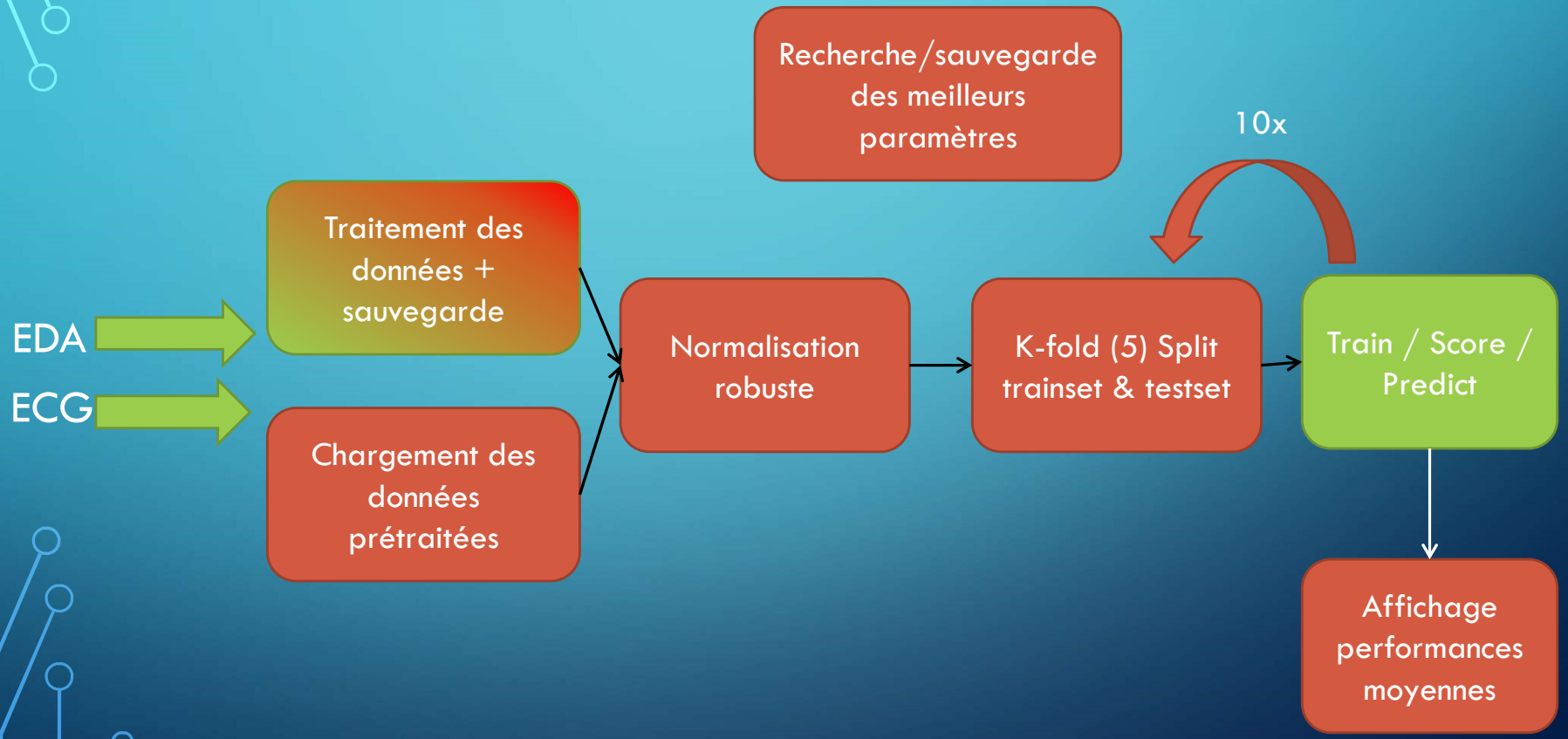
Tables des matières

Présentation générale



Output : 0 -> NST
1 -> ST

Présentation générale



Chargement des données prétraitées

```
file_pre_proces_features = "preProcessFeatues.pkl"

REPROCESSING_EDA = 0
REPROCESSING_ECG = 0
```

```
# get all old features (ECG + EDA)
if REPROCESSING_ECG == 0 and REPROCESSING_EDA == 0:
    dataset = pd.read_pickle(file_pre_proces_features)
else:
    load_raw_data()
    plot_ECG()
    # Get old features ECG
    if REPROCESSING_ECG == 0:
```

```
if REPROCESSING_EDA == 1:
    eda_features_df = fe.compute_EDA_features(raw_df, segmentation_level=SEGMENTATION_LEVEL)
if REPROCESSING_ECG == 1:
    ecg_features_df = fe.compute_ECG_features(raw_df, segmentation_level=SEGMENTATION_LEVEL)
subject_features = pd.concat([label_df, eda_features_df, ecg_features_df], axis=1, sort=False)
```

```
# Save pre-processed features dataset
dataset.to_pickle(file_pre_proces_features)
```

Traitement des données

```
# STD DIFF
std_d = df_subset["EDA"].std()
std_b = baseline_df["EDA"].std()
features_df['s{}_EDA_std_diff'.format(segment_count)] = [std_d - std_b]

# MEDIAN DIFF
median_d = df_subset["EDA"].median()
median_b = baseline_df["EDA"].median()
features_df['s{}_EDA_median_diff'.format(segment_count)] = [median_d - median_b]
```

Normalisation robuste

```
X = dataset.drop(['label'], axis=1)

scalerRobust = RobustScaler()
# apply the scaler on all set (for search for best hyper parameter)
X_scaled = scalerRobust.fit_transform(X)

X_scaled_df = pd.DataFrame(X_scaled)
```

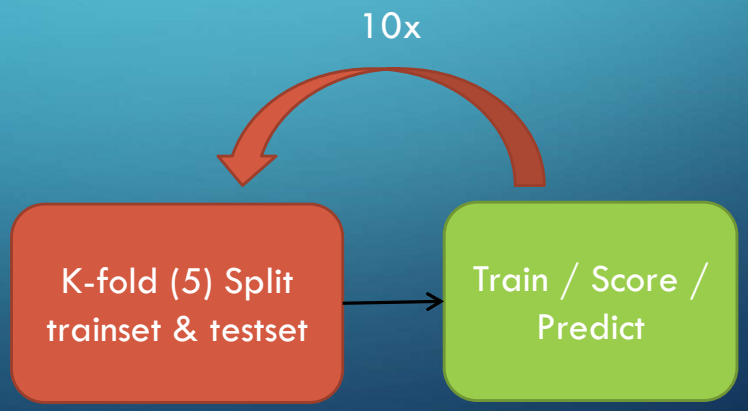
Recherche/sauvegarde des meilleurs paramètres

```python
def searchBest(self, X, y):

    # load best score for now
    bestScore = self.loadBestScore()

    classifier_2 = RandomForestClassifier()

    # create the grid of parameters to explore
    param_grid = {"n_estimators": [5, 10, 20, 50, 100],
                  "criterion": ["gini", "entropy"],
                  "max_depth": [1, 2, 5, 10, None],
                  "min_samples_split": [2, 3, 5, 10],
                  "min_samples_leaf": [1, 3, 5, 10],
                  "max_features":  ["auto", "sqrt", "log2"],
                  "bootstrap": [True, False]}

    # use k-fold cross-validation to select the best set of parameters
    grid_search = GridSearchCV(classifier_2, param_grid, cv=5)
    grid_search.fit(X, y)
    results = grid_search.cv_results_
```

```python
"""
    Initialize algorithm
"""
classifier = algo.RNG_Algorithm()
```

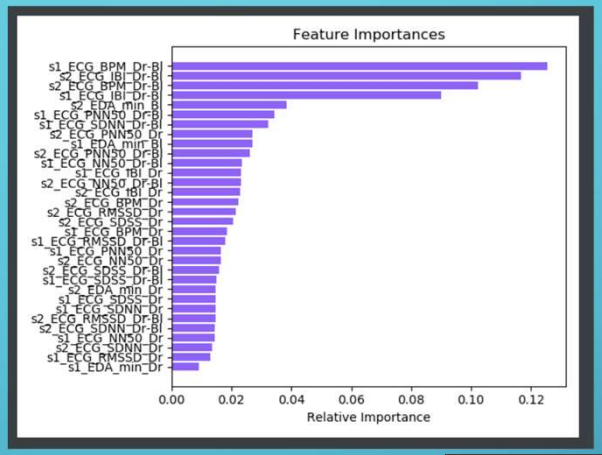
K-fold (5) Split trainset & testset

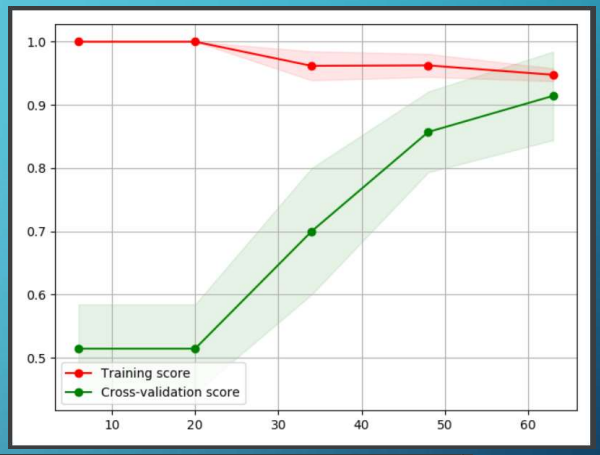
```
cv = StratifiedKFold(n_splits=5, random_state=seed, shuffle=True)
for (train, test), i in zip(cv.split(X_scaled_1, y), range(5)):
    X_train_kfold = X_scaled_1.iloc[train]
    X_test_kfold = X_scaled_1.iloc[test]

    train_labels = y.iloc[train]
    test_labels = y.iloc[test]
```



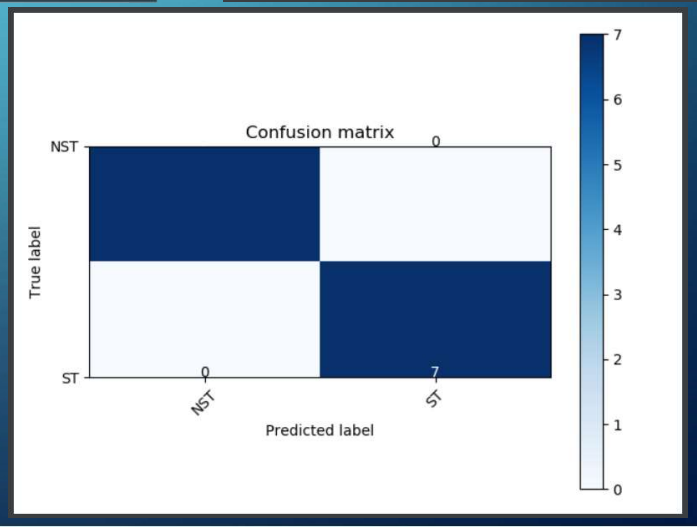
Affichage des performances moyennes

```
Real Accuracy Mean = 84.143 %
Real F1 Mean min = 0.831 %
Real F1 Mean max = 0.833 %
```

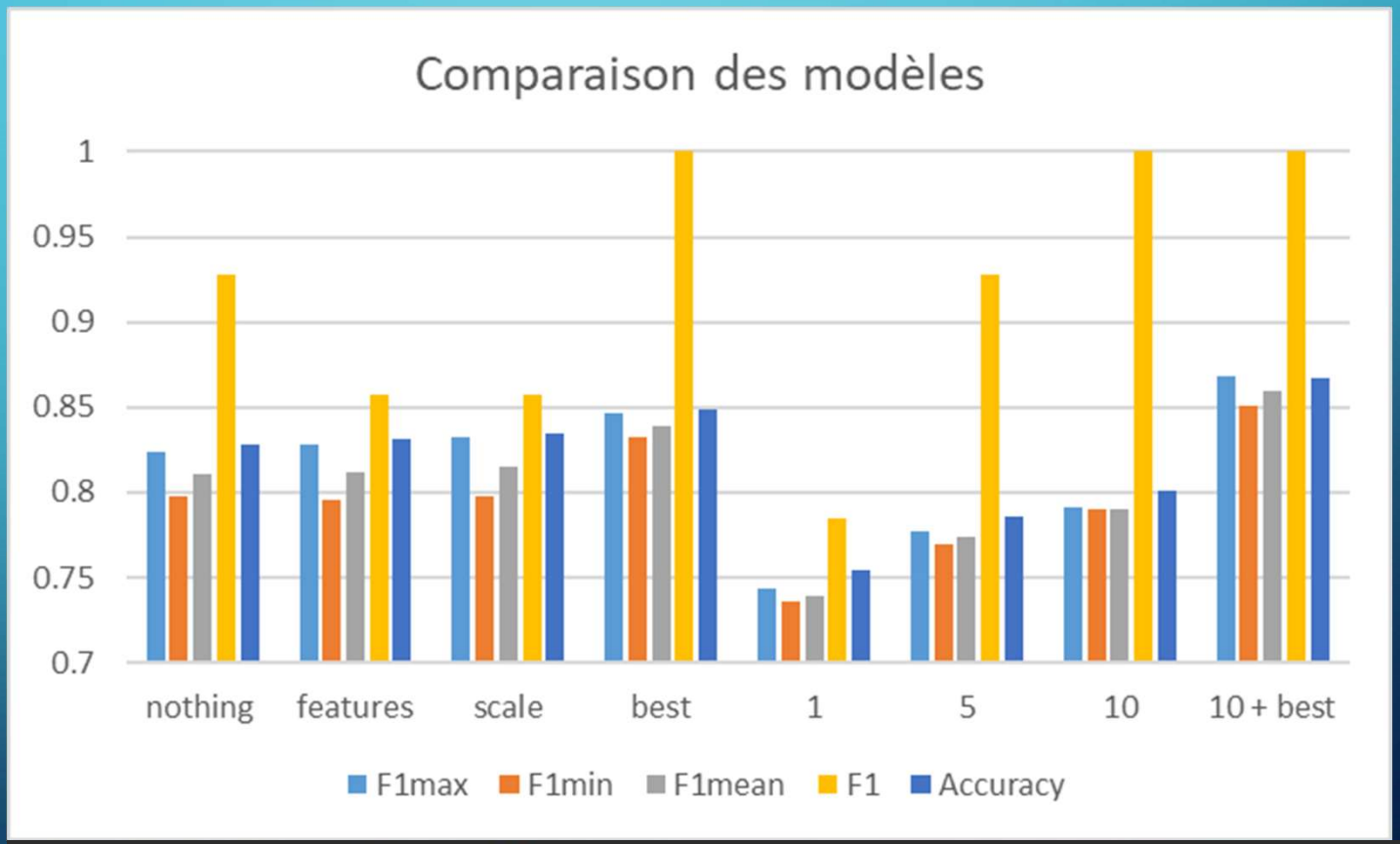





```
[[7 0]
 [0 7]]
Metrics per classes
accuracy_score: 1.0
precision_score: [1. 1.]
recall_score: [1. 1.]
f1_score: [1. 1.]
Metrics (average)
accuracy_score: 1.0
precision_score: 1.0
recall_score: 1.0
I am a RF algorithm, I predict this : [0 1 1 0 0 0 0 1 0 1 1 0]
```



Évolution des résultats




```
Load my best model : RandomForestClassifier(max_depth=2, min_samples_leaf=3, min_samples_split=10,
                                            n_estimators=50)
```

Fusion

A : 0
B : 1
C : 1
D : 0
E : 0
F : 0
G : 0
H : 1
I : 0
J : 1
K : 1
L : 1
AM : 87.2