coerchon_habasque_dlts_tp2

30 Octobre 2024

[1]:
```
#!pip install numpy
#!pip install sklearn
```

[2]:
```
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import pandas as pd

from scipy.signal import find_peaks

from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import  accuracy_score,confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier,␣
↪AdaBoostClassifier
from sklearn.svm import SVC
from sklearn.neighbors import KNeighborsClassifier
from sklearn.model_selection import GridSearchCV
from xgboost import XGBClassifier
```

[3]:
```
from IPython.core.display import HTML
HTML("""
<style>
.consignes{
  font-weight: bold;
  color: #3256a8;
  background-color: #edebdf
}
</style>
""")
```

[3]: <IPython.core.display.HTML object>

TP2: Détection de menaces radar

Listez les noms des étudiants (2 au maximum) ayant participé à ce notebook dans la cellule suivante

(prénom, nom, affectation).

Au moment du rendu, le notebook doit être nommé nom1_nom2_dlts_tp2.ipynb

2 séances de TP sur ce sujet : le 16 octobre (1h30) et le 23 octobre (3h). Deadline : 30 otobre 2024, 13h59, par mail à deepetsignal.mva@gmail.com

Pour installer les paquets nécessaires à la réalisation de ce TP vous pouvez utiliser dans le notebook

```
!pip install \< nom_du_paquet \>
```

merci de regrouper toutes les installations dans la première cellule du notebook. Essayez de faire en sorte que votre notebook puisse se lire comme un compte rendu, évitez de laisser du code mort et prennez le temps de commenter vos observations et résultats.

0.1 Mission

Vous commandez un avion de reconnaissance en mission top secrète. Malgré votre discretion et toutes les précautions prises vous dérangez et pouvez être sujet de menaces qu'il vous faut détecter pour garantir la sécurité de votre équipage et la succès de votre mission (comme elle est top secrète vous ne pouvez pas en savoir plus).

A votre disposition un réseau d'antennes et un système d'analyse spectrale perfectionnés vous permettent d'extraire et de caractériser des signaux en provenance des radars à altitude 0 qui parsèment votre parcours.

Un signal radar est composé d'impulsions. Le système d'analyse vous permettent de caractériser chaque impulsion reçue par un PDW (Pulse Description Word) qui contient:

- la date de début de détection de l'impulsion (en ms)
- la largeur ou durée de l'impulsion (en ms)
- la puissance de l'impulsion (en dB / référence)
- l'angle theta et l'angle phi décrivant la direction dans laquelle l'impulsion est détectée (en radians)
- la fréquence de l'impulsion (en Ghz)

Votre capteur n'est pas parfait et vous subissez notamment un phénomène de mitage: une certaine proportion des impulsions émises ne sont pas détectées. Cette proportion est d'autant plus grande que la puissance des impulsions est petite.

Votre vaisseau navigue à 10 km d'altitude, avec une vitesse constante de 1000 km/h vers le nord.

De précédentes missions ont permis de réaliser une base de données de signaux de 10 secondes. Chaque signal se présente sous la forme d'un fichier .npz qui contient l'ensemble des PDW reçus.

Un signal est donc un fichier dont le nom est de la forme 'pdw_.npz'.

Cette base de données est annotée: le destin de chaque mission a permis de déclarer chaque signal comme une 'menace' ou une 'nonmenace'.

Les signaux ont été divisés en deux ensembles indépendants:

- train
- test

Les annotations pour chaque ensemble sont disponibles dans le fichier labels_.json qui donne l'association nom de fichier -> menace ou nonmenace.

Votre mission (si vous l'acceptez) est de choisir et d'entrainer un algorithme d'apprentissage machine à détecter les menaces sur l'ensemble 'train' et à évaluer ses performances sur l'ensemble 'test'.

Fort de votre expérience passée, vous décidez de suivre la méthodologie suivante:

- Visualisation et analyse des données pour trouver les paramètres les plus pertinents à utiliser
- Sélection d'un algorithme naïf "baseline" de référence: vous définissez un premier algorithme simple et caractérisez ses performances qui serviront de point de comparaison pour qualifier l'apport d'algorithmes plus sophistiqués
- Vous utilisez la bibliothèque sklearn et notamment ses implémentations d'algorithmes de classification binaire pour définir et entrainer sur l'ensemble 'train' un ou plusieurs algorithmes en capacité de prédire le label menance ou nonmenace de chaque signal (https://scikit-learn.org/stable/auto_examples/classification/plot_classifier_comparison.html)
- Vous qualifiez les performances de vos algorithmes en termes d'accuracy (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html#sklearn.metrics.accuracy_score) ; vous calculez aussi les matrices de confusion (https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html#sklearn.metrics.confusion_mat

Pour ce TP nous ne cherchons pas à mettre en place des algorithmes d'apprentissage profond. Ces méthodes pourront être investiguées dans un deuxième temps.

Si les résultats vous semblent décevants, souvenez vous des conseils de votre prédecesseur et mentor G. Abitbol: "toujours donner un coup de pouce à un algo de machine learning en lui présentant les bonnes données"

Vous pouvez ouvrir un fichier de la façon suivante:

[4]:
```
pdws = np.load('./radars/train/pdw-0.npz')
dates = pdws['date']
largeurs = pdws['largeur']
frequences = pdws['frequence']
puissances = pdws['puissance']
theta = pdws['theta']
phi = pdws['phi']
```

et les fichiers d'annotations:

[5]:
```
import json

with open('./radars/train_labels.json') as f:
    dict_labels = json.load(f)

for i in range(10):
    print(f"Le signal pdw-{i}.npz est de type: {dict_labels[f'pdw-{i}']}")
```

```
Le signal pdw-0.npz est de type: nonmenace
Le signal pdw-1.npz est de type: nonmenace
Le signal pdw-2.npz est de type: menace
Le signal pdw-3.npz est de type: nonmenace
```

```
Le signal pdw-4.npz est de type: nonmenace
Le signal pdw-5.npz est de type: menace
Le signal pdw-6.npz est de type: menace
Le signal pdw-7.npz est de type: menace
Le signal pdw-8.npz est de type: menace
Le signal pdw-9.npz est de type: nonmenace
```

On compte sur vous, bonne chance !

Nom des étudiants : Habasque Chloé et Coërchon Colin

1 Sommaire

1.1 1. Présentation des données

1.1.1 1.1 Présentation générale de l'ensemble des données

1.1.2 1.2 Présentation des données suivant la variable cible

1.2 2. Premier travail naïf

1.2.1 2.1 Création naïve des premières features

1.2.2 2.2 Sélection de l'algorithme baseline

1.3 3. Présentation du travail final

1.3.1 3.1 Sélection réfléchie des features pour notre jeu de données

1.3.2 3.2 Exploitation et perfectionnement de différents modèles

1.4 4. Conclusion

1. Présentation des données ## 1.1 Présentation générale de l'ensemble des données

Pour cette présentation rapide des données, nous nous concentrerons sur les 20 premiers signaux.

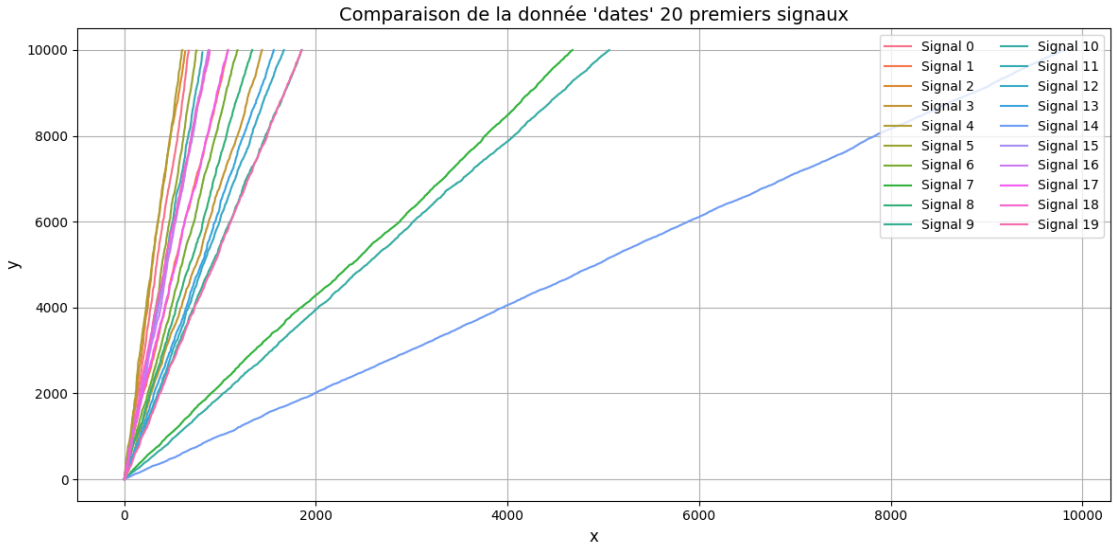
L'objectif est d'en faire une présentation simple et rapide, qui va mettre en lumière nos points de réflexion sur la future sélection de features pour notre jeu de données.

[6]:
```
dates_list = []
largeurs_list = []
frequences_list = []
puissances_list = []
theta_list = []
phi_list = []

for i in range(20):
    pdw_file = f'./radars/train/pdw-{i}.npz'
    pdws = np.load(pdw_file)

    dates_list.append(pdws['date'])
    largeurs_list.append(pdws['largeur'])
    frequences_list.append(pdws['frequence'])
```

```
    puissances_list.append(pdws['puissance'])
    theta_list.append(pdws['theta'])
    phi_list.append(pdws['phi'])
```

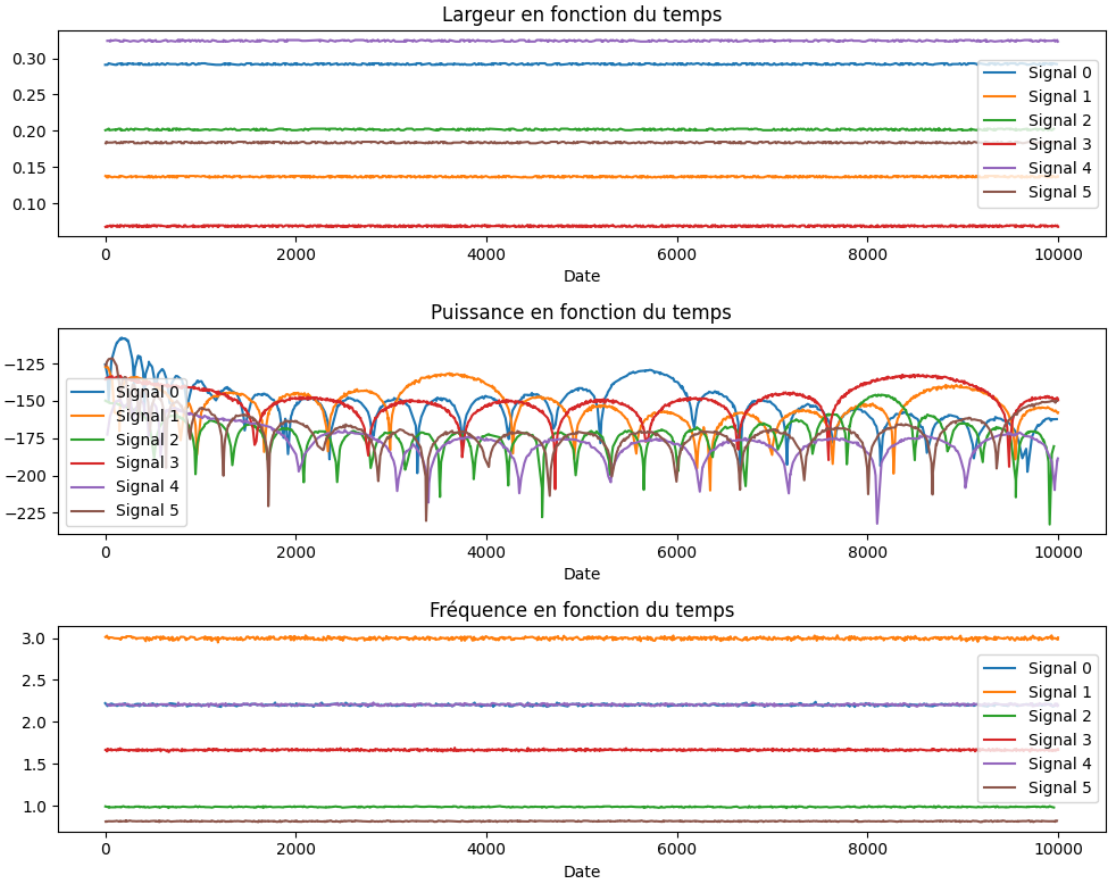
La première question que nous nous sommes posées concerne la donnée `date` de chacun des signaux.

[7]:
```
palette = sns.color_palette("husl", 20)

# Plot des 20 premiers signaux de 'puissance' en fonction de 'date'
plt.figure(figsize=(12, 6))

for i in range(20):
    plt.plot(dates_list[i], label=f'Signal {i}', color=palette[i])

plt.title("Comparaison de la donnée 'dates' 20 premiers signaux", fontsize=14)
plt.xlabel("x", fontsize=12)
plt.ylabel("y", fontsize=12)
plt.legend(loc='upper right', fontsize=10, ncol=2)
plt.grid(True)

plt.tight_layout()
plt.show()
```



Effectivement, les vecteurs de `date` vont tous de 0 à 10000, ce qui est **parfait** pour pouvoir comparer tous nos signaux. Nous remarquons seulement dans ce graphique que certains signaux (comme le signal 14 par exemple) ont une meilleure précision puisqu'ils possèdent davantage de points pour l'ensemble des autres données, ce qui permet de mieux observer les variations dans ces dernières comme la fréquence ou la puissance.

[8]:
```
fig, axs = plt.subplots(3, 1, figsize=(10, 8))

for i in range(6):
    axs[0].plot(dates_list[i], largeurs_list[i], label=f'Signal {i}')
    axs[1].plot(dates_list[i], puissances_list[i], label=f'Signal {i}')
    axs[2].plot(dates_list[i], frequences_list[i], label=f'Signal {i}')

axs[0].set_title('Largeur en fonction du temps')
axs[1].set_title('Puissance en fonction du temps')
axs[2].set_title('Fréquence en fonction du temps')

for ax in axs:
    ax.legend()
    ax.set_xlabel('Date')

plt.tight_layout()
plt.show()
```

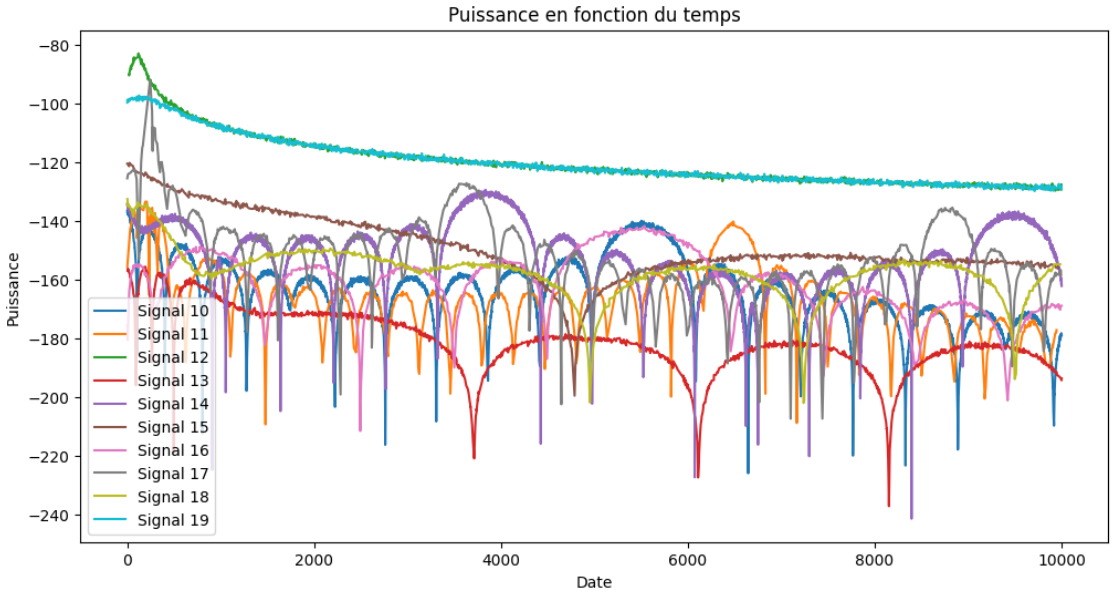


Les graphiques montrent que la `largeur` et la `fréquence` des signaux sont relativement **stables**

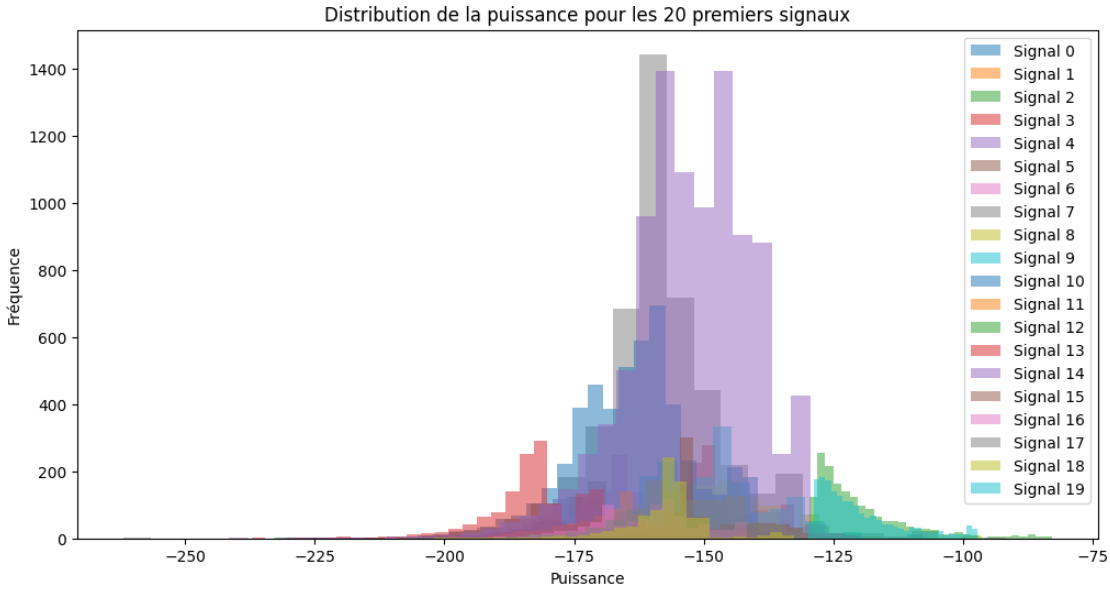
et **distinctes** pour chaque signal, indiquant une faible variation dans le temps et facilitant leur différenciation. En revanche, la `puissance` varie fortement avec des pics et des chutes (sûrement dus aux impulsions de ces différents signaux), suggérant des fluctuations spécifiques à chaque signal. Cette stabilité en largeur et fréquence, contrastée avec les variations de puissance, pourrait aider à identifier des caractéristiques propres à chaque signal.

Détaillons un peu les différentes puissances que nous pouvons avoir en visualisant d'autres signaux.

[9]:
```
plt.figure(figsize=(12, 6))
for i in range(10,20):
    plt.plot(dates_list[i], puissances_list[i], label=f'Signal {i}')

plt.title('Puissance en fonction du temps')
plt.xlabel('Date')
plt.ylabel('Puissance')
plt.legend()
plt.show()
```



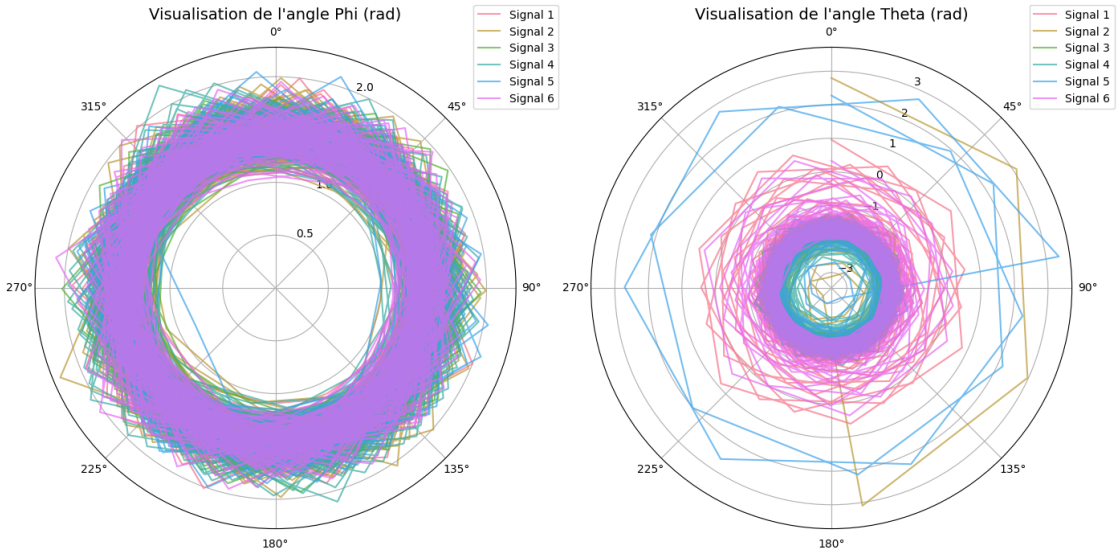
[10]:
```
plt.figure(figsize=(12, 6))
for i in range(20):
    plt.hist(puissances_list[i], bins=30, alpha=0.5, label=f'Signal {i}')

plt.title('Distribution de la puissance pour les 20 premiers signaux')
plt.xlabel('Puissance')
plt.ylabel('Fréquence')
plt.legend()
plt.show()
```



Certains signaux ont donc des puissances sans “pics”, tandis que d’autres en ont énormément. On remarque également que certains signaux ont clairement une moyenne de puissance bien plus forte que d’autres. Reste à voir si ces informations peuvent être intéressantes pour notre **mission**.

[11]:
```
palette = sns.color_palette("husl", 6)

# On initialise un graphique en coordonnées polaires
fig, ax = plt.subplots(1, 2, figsize=(14, 7),␣
 ↪subplot_kw=dict(projection='polar'))

for i in range(6):
    # 'phi' sur le premier graphique
    ax[0].plot(phi_list[i], label=f'Signal {i+1}', color=palette[i], alpha=0.7)
    # 'theta' sur le deuxième graphique
    ax[1].plot(theta_list[i], label=f'Signal {i+1}', color=palette[i], alpha=0.7)

ax[0].set_title("Visualisation de l'angle Phi (rad)", fontsize=14)
ax[0].set_theta_zero_location("N")
ax[0].set_theta_direction(-1)  # Sens horaire

ax[1].set_title("Visualisation de l'angle Theta (rad)", fontsize=14)
ax[1].set_theta_zero_location("N")
ax[1].set_theta_direction(-1)  # Sens horaire

ax[0].legend(loc='upper right', bbox_to_anchor=(1.1, 1.1))
ax[1].legend(loc='upper right', bbox_to_anchor=(1.1, 1.1))
```

```
plt.tight_layout()
plt.show()
```



Les données présentes dans `phi` et `theta` semblent, pour le coup, très erratiques et difficilement exploitables.

1.2 Présentation des données suivant la variable cible

Cette première approche des données nous a beaucoup aidé dans notre compréhension de ce jeu de données. Mais l'intérêt réside tout de même à travers la recherche de signaux menaçant ou non-menaçant.

Il est donc nécessaire de se créer une variable cible `Y_train` et `Y_test`, et de visualiser nos données en conséquences. Dans cette donnée binaire, "menace" = 1, et "nonmenace" = 0.

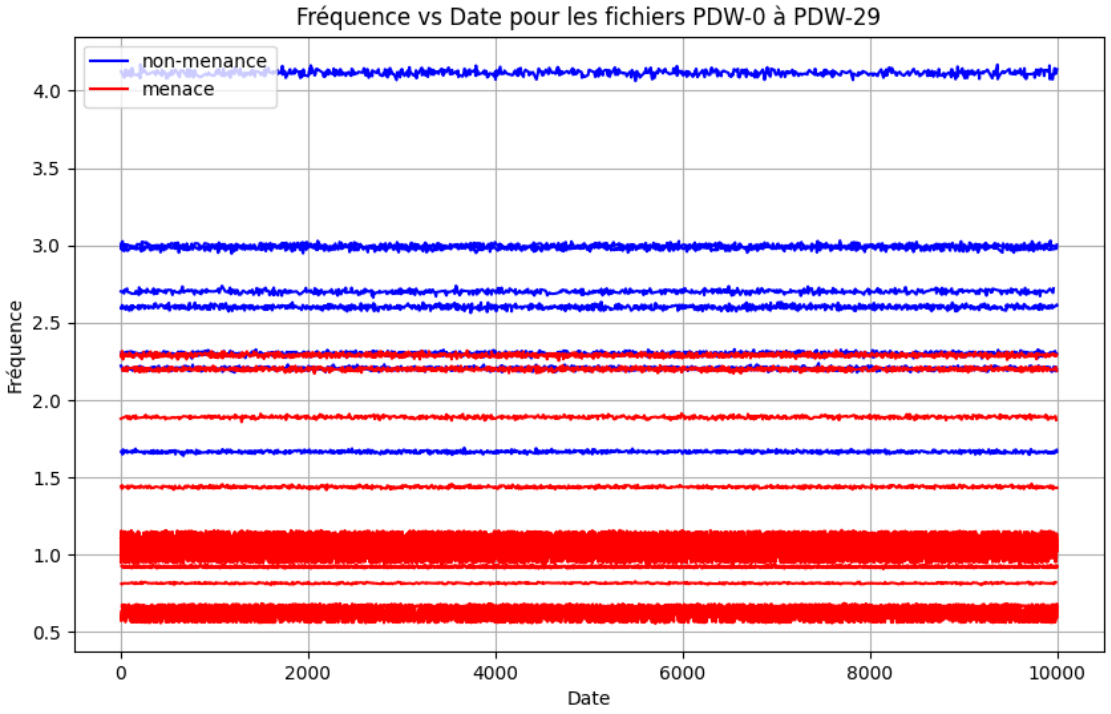
[12]:
```
# Charger les labels du train depuis le fichier JSON
with open('./radars/train_labels.json') as f:
    dict_labels_train = json.load(f)

Y_train = []

for i in range(len(dict_labels_train)):  # Parcourir tous les signaux de train
    label = dict_labels_train[f'pdw-{i}']
    if label == 'menace':
        Y_train.append(1)
    else:
        Y_train.append(0)

Y_train = np.array(Y_train)
```

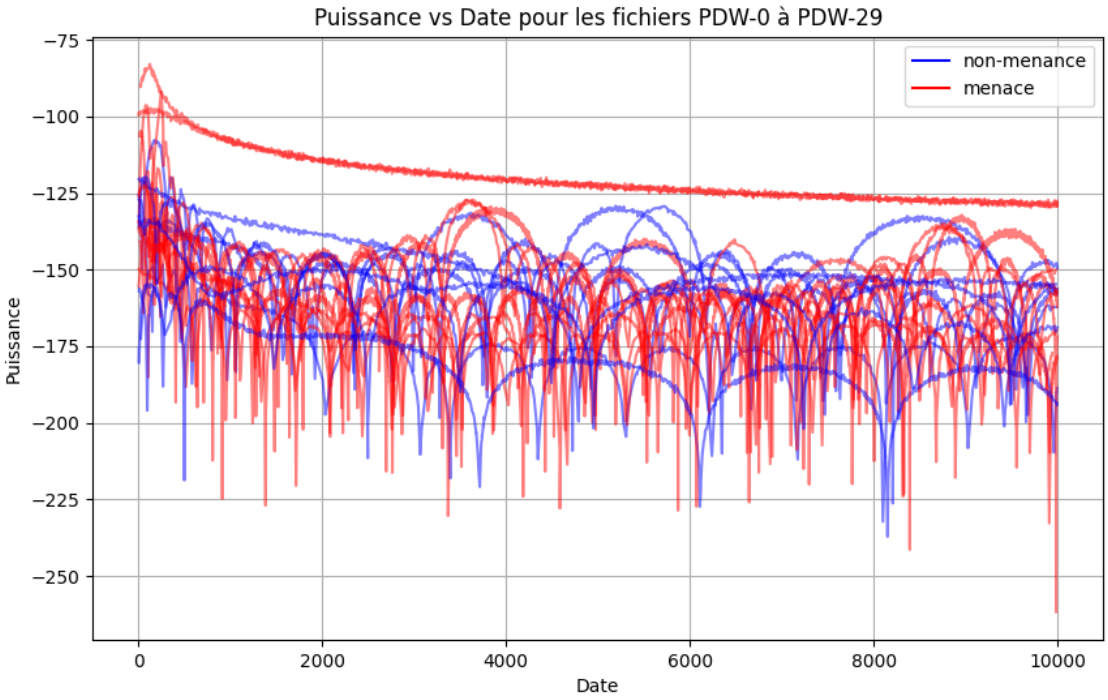
```
# Charger les labels du test depuis le fichier JSON
with open('./radars/test_labels.json') as f:
    dict_labels_test = json.load(f)

Y_test = []

for i in range(len(dict_labels_test)):  # Parcourir tous les signaux de test
    label = dict_labels_test[f'pdw-{i}']
    if label == 'menace':
        Y_test.append(1)
    else:
        Y_test.append(0)

Y_test = np.array(Y_test)
```

Affichons maintenant nos différentes données, en nous concentrant sur cette variable cible de “menace”.

[13]:
```
color_mapping = {1: 'red', 0: 'blue'}  # Rouge pour "menace" (1) et bleu pour␣
↪"non-menance" (0)
num_files = 20

plt.figure(figsize=(10, 6))

plt.plot([], [], color='blue', label='non-menance')  # Bleu pour "non-menance"
plt.plot([], [], color='red', label='menace')        # Rouge pour "menace"

# Affichage de la fréquence en fonction du temps avec les couleurs basées sur Y
for i in range(num_files):
    label_value = Y_train[i] if i < len(Y_train) else Y_test[i - len(Y_train)]
    color = color_mapping[label_value]

    plt.plot(dates_list[i], frequences_list[i], color=color)

plt.title('Fréquence vs Date pour les fichiers PDW-0 à PDW-29')
plt.xlabel('Date')
plt.ylabel('Fréquence')
plt.legend()
plt.grid(True)
plt.show()

plt.figure(figsize=(10, 6))

plt.plot([], [], color='blue', label='non-menance')  # Bleu pour "non-menance"
plt.plot([], [], color='red', label='menace')        # Rouge pour "menace"

# Affichage de la puissance en fonction du temps avec les couleurs basées sur Y
```

```python
for i in range(num_files):
    label_value = Y_train[i] if i < len(Y_train) else Y_test[i - len(Y_train)]
    color = color_mapping[label_value]

    plt.plot(dates_list[i], puissances_list[i], color=color, alpha=0.5)

plt.title('Puissance vs Date pour les fichiers PDW-0 à PDW-29')
plt.xlabel('Date')
plt.ylabel('Puissance')
plt.legend()
plt.grid(True)
plt.show()
```



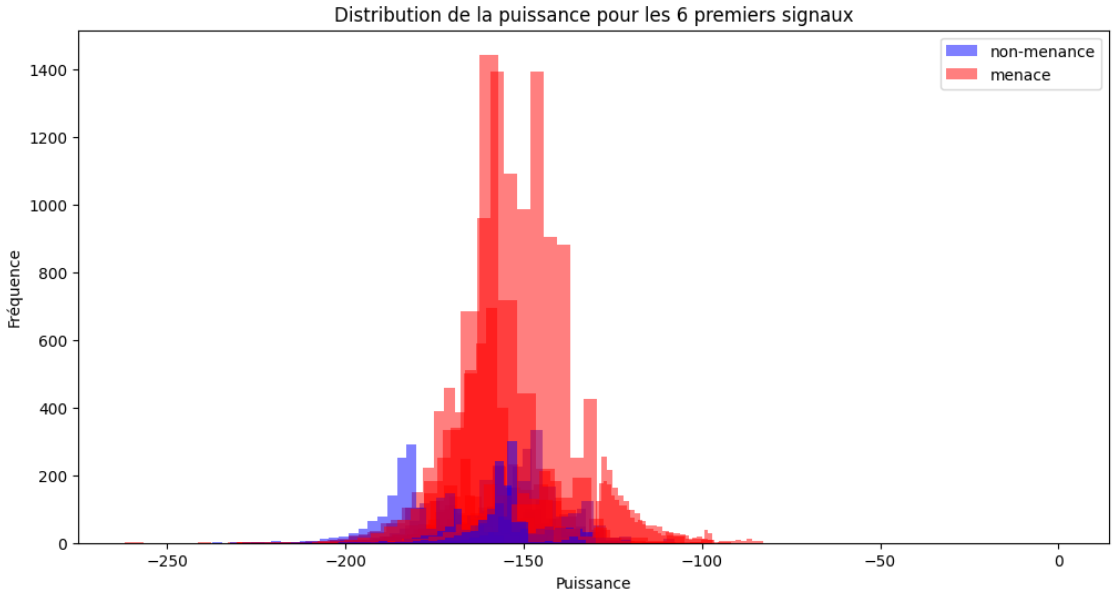


Le premier graphique montre l'évolution des fréquences en fonction des classes, les signaux de menace en rouge et ceux de non-menace en bleu, au fil du temps. On observe que les signaux de classe menace ont souvent une fréquence plus faible que ceux de la classe non-menace. La fréquence pourrait donc constituer un bon paramètre pour classifier les signaux.

Le deuxième graphique illustre l'évolution des puissances, toujours en fonction des catégories menace en rouge et non-menace en bleu, au fil du temps. On remarque que, parmi les 20 puissances affichés, l'allure des puissances des signaux menaces présente soit aucun pics, soit beaucoup de pics. L'utilisation d'un paramètre qui compte le nombre de pics des puissances de chaque signal pourrait donc s'avérer intéressant pour notre classification.

[14]:
```
color_mapping = {1: 'red', 0: 'blue'}

plt.figure(figsize=(12, 6))

plt.hist([], color='blue', bins=1, alpha=0.5, label='non-menance')  # Bleu pour␣
↪"non-menance"
plt.hist([], color='red', bins=1, alpha=0.5, label='menace')        # Rouge pour␣
↪"menace"

for i in range(20):
    label_type = Y_train[i]
    color = color_mapping[label_type]
    plt.hist(puissances_list[i], color=color, bins=30, alpha=0.5)
```

```
plt.title('Distribution de la puissance pour les 6 premiers signaux')
plt.xlabel('Puissance')
plt.ylabel('Fréquence')
plt.legend()
plt.show()
```



Cet histogramme des puissances suggère, dans le cas des signaux de classe menace, que leur puissance est généralement plus élevée. La puissance constitue donc un paramètre intéressant pour notre classification.

2. Premier travail naïf

2.1 Création naïve des premières features

Après cette présentation succinte des données, il est temps de créer nos premières features pour pouvoir ensuite tester différents modèles de machine learning. Ces premières features doivent simplement servir à nous créer un modèle baseline de référence.

1.4.1 Création de X_train

[15]:
```
# Fonction pour extraire des statistiques d'un vecteur
def calculate_statistics(vector, prefix):
    stats = {
        f'{prefix}_mean': np.mean(vector),
        f'{prefix}_median': np.median(vector),
        f'{prefix}_std': np.std(vector),
        f'{prefix}_max': np.max(vector),
        f'{prefix}_min': np.min(vector)
    }
```

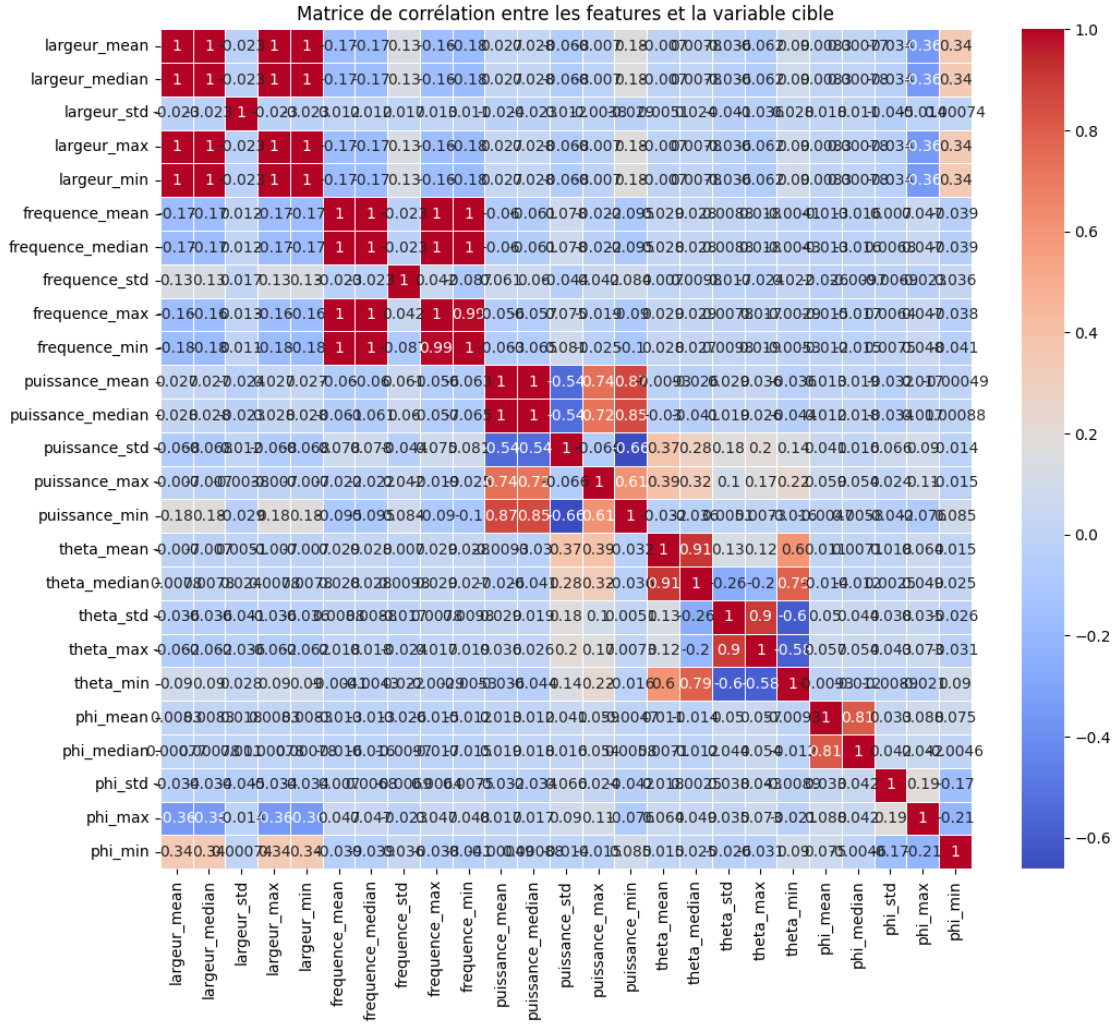
```
    return stats

all_features = []

num_signals = 2000  # C'est la taille du dossier .\train

# Boucle sur chaque fichier pdw
for i in range(num_signals):
    pdw_file = f'./radars/train/pdw-{i}.npz'
    pdws = np.load(pdw_file)

    features = {}
    features.update(calculate_statistics(pdws['largeur'], 'largeur'))
    features.update(calculate_statistics(pdws['frequence'], 'frequence'))
    features.update(calculate_statistics(pdws['puissance'], 'puissance'))
    features.update(calculate_statistics(pdws['theta'], 'theta'))
    features.update(calculate_statistics(pdws['phi'], 'phi'))

    all_features.append(features)

X_train = pd.DataFrame(all_features)
```

1.4.2 Création de X_test

[16]:
```
all_features_test = []

num_test_signals = 800  # C'est la taille du dossier .\test

# Boucle sur chaque fichier pdw pour les données de test
for i in range(num_test_signals):
    pdw_file = f'./radars/test/pdw-{i}.npz'
    pdws = np.load(pdw_file)

    features_test = {}
    features_test.update(calculate_statistics(pdws['largeur'], 'largeur'))
    features_test.update(calculate_statistics(pdws['frequence'], 'frequence'))
    features_test.update(calculate_statistics(pdws['puissance'], 'puissance'))
    features_test.update(calculate_statistics(pdws['theta'], 'theta'))
    features_test.update(calculate_statistics(pdws['phi'], 'phi'))

    all_features_test.append(features_test)

X_test = pd.DataFrame(all_features_test)
```

Intéressons-nous maintenant à la corrélation entre ces différentes features.

[17]:
```
# On calcule la matrice de corrélation
correlation_matrix = X_train.corr()

plt.figure(figsize=(12, 10))
sns.heatmap(correlation_matrix, cmap='coolwarm', annot=True, linewidths=0.5)
plt.title('Matrice de corrélation entre les features et la variable cible')
plt.show()
```

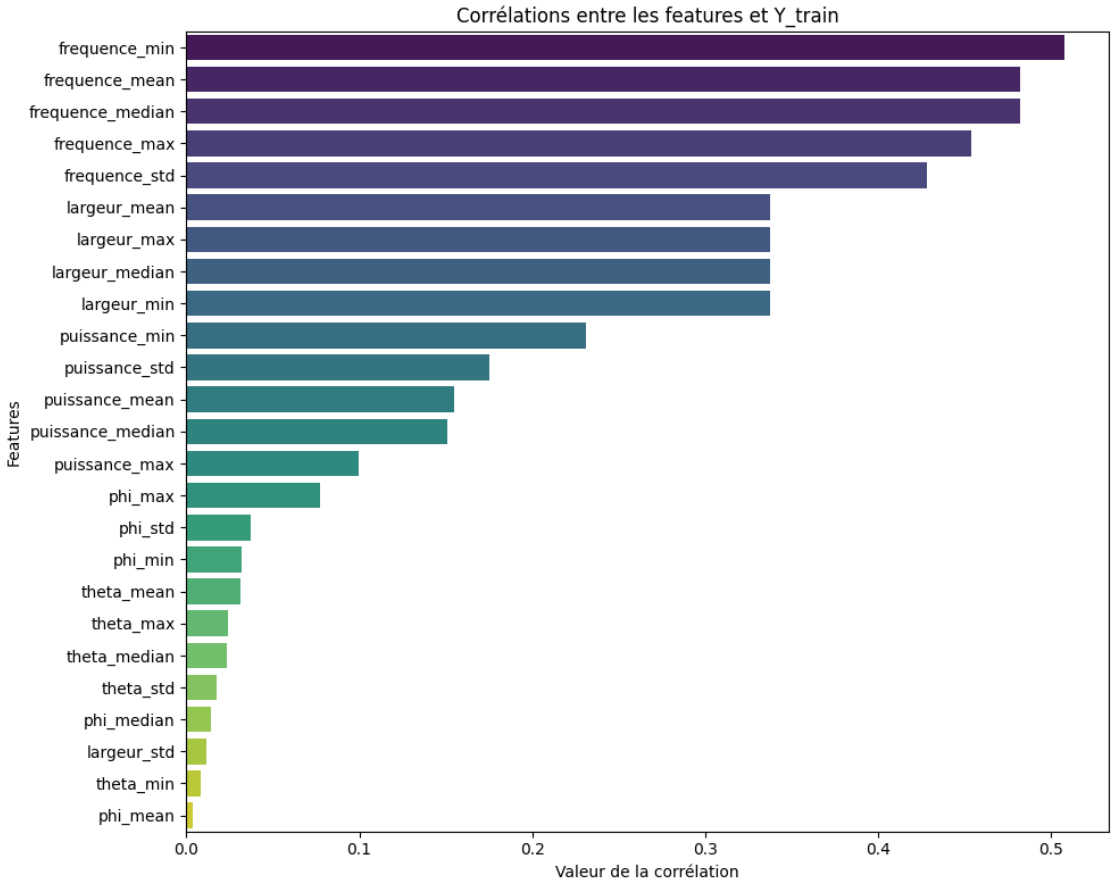


Comparons maintenant toutes ces variables à notre variable cible `Y_train`. On s'intéressera également ici à la corrélation.

[18]:
```
# On calcule la corrélation entre chaque feature de X_train et Y_train
correlations_with_target = X_train.apply(lambda x: x.corr(pd.Series(Y_train[:
↪len(X_train)])))
```

```
# On trie les corrélations par ordre décroissant (en valeur absolue)
correlations_with_target_sorted = correlations_with_target.abs().
 ↪sort_values(ascending=False)

plt.figure(figsize=(10, 8))
sns.barplot(x=correlations_with_target_sorted.values,␣
 ↪y=correlations_with_target_sorted.index, palette="viridis")
plt.title("Corrélations entre les features et Y_train")
plt.xlabel("Valeur de la corrélation")
plt.ylabel("Features")
plt.tight_layout()

plt.show()
```

```
/var/folders/8_/94g6kxj14yb758mbh_6dgyk80000gn/T/ipykernel_12947/367104785.py:8:
FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=correlations_with_target_sorted.values,
y=correlations_with_target_sorted.index, palette="viridis")
```



Ces premiers résultats sont très intéressants. Nous observons une forte corrélation entre les différentes features liées à la fréquence (minimum, maximum, médiane et moyenne). Cette corrélation se retrouve également dans les features associées à la puissance et à la largeur des signaux. Par conséquent, nous pouvons nous concentrer sur un seul des paramètres de fréquence, de puissance et de largeur.

En ce qui concerne la pertinence de ces données par rapport à notre variable cible `Y_train`, la fréquence semble être le paramètre le plus significatif pour la classification de nos signaux. La puissance et la largeur ont également un intérêt. En revanche, les données concernant phi et theta ne paraissent pas pertinentes pour notre objectif.

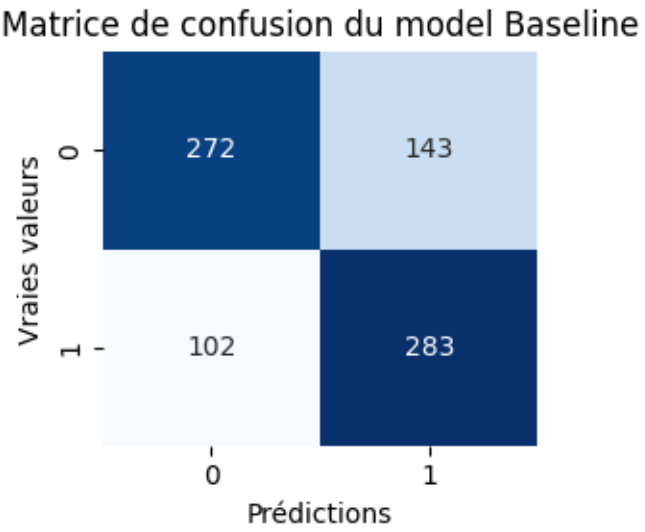
2.2 Sélection de l'algorithme "baseline"

Pour ce modèle baseline, on se contentera donc d'une unique feature qui maximise la corrélation avec `Y_train`, c'est-à-dire la donnée `fréquence_min`. Et pour le choix du modèle, on se contentera d'une simple régression logistique sans optimisation des hyperparamètres.

[19]:
```
# Création de X_baseline en ne gardant que la colonne 'frequence_min' de X_train
↪et X_test
X_baseline_train = X_train[['frequence_min']]
X_baseline_test = X_test[['frequence_min']]
```

```
logistic_model = LogisticRegression()
logistic_model.fit(X_baseline_train, Y_train)

Y_pred = logistic_model.predict(X_baseline_test)

# Évaluation de la précision
accuracy = accuracy_score(Y_test, Y_pred)
print("Précision du modèle baseline (régression logistique avec 'frequence_min'):
 ↪", accuracy)

cm = confusion_matrix(Y_test, Y_pred)
plt.figure(figsize=(3, 3))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', cbar=False)
plt.title(f"Matrice de confusion du model Baseline")
plt.xlabel("Prédictions")
plt.ylabel("Vraies valeurs")
plt.tight_layout()
plt.show()
```

```
Précision du modèle baseline (régression logistique avec 'frequence_min'):
0.69375
```



Le modèle baseline, utilisant uniquement la feature `frequence_min` avec une régression logistique sans optimisation des hyperparamètres, atteint une précision de **69 %**. Cette approche simple sert de point de départ pour évaluer les performances de nos futurs modèles.

3. Présentation du travail final ## 3.1 Sélection réfléchie des features pour notre jeu de données

1.4.3 Choix des premières features parmis les features naïves.

Après avoir examiné les deux graphiques de corrélation et analysé visuellement les données grâce à la présentation détaillée réalisée précédemment, nous avons identifié un ensemble de variables pertinentes et peu corrélées entre elles. Cette approche vise à sélectionner des caractéristiques informatives qui peuvent améliorer la performance de notre modèle tout en évitant les redondances.

En premier lieu, nous avons donc choisi d'utiliser les variables suivantes : - `frequence_min` - `frequence_std` - `largeur_mean` - `puissance_min`

Ces variables offrent une bonne diversité de caractéristiques sans être fortement corrélées entre elles, ce qui en fait une base solide pour la suite de notre analyse.

[20]:
```
# Sélection des features pertinentes dans X_train (et donc X_test aussi)
X_train = X_train[['frequence_min', 'frequence_std', 'puissance_min',␣
 ↪'largeur_median']]
X_test = X_test[['frequence_min', 'frequence_std', 'puissance_min',␣
 ↪'largeur_median']]

print(X_train.head())
```

```
   frequence_min  frequence_std  puissance_min  largeur_median
0       2.178543       0.008909    -198.439372           0.292
1       2.946963       0.012328    -210.067261           0.137
2       0.972759       0.004003    -232.867752           0.202
3       1.640058       0.006610    -209.155813           0.069
4       2.182394       0.008529    -232.332658           0.324
```

1.4.4 Choix de nouvelles features

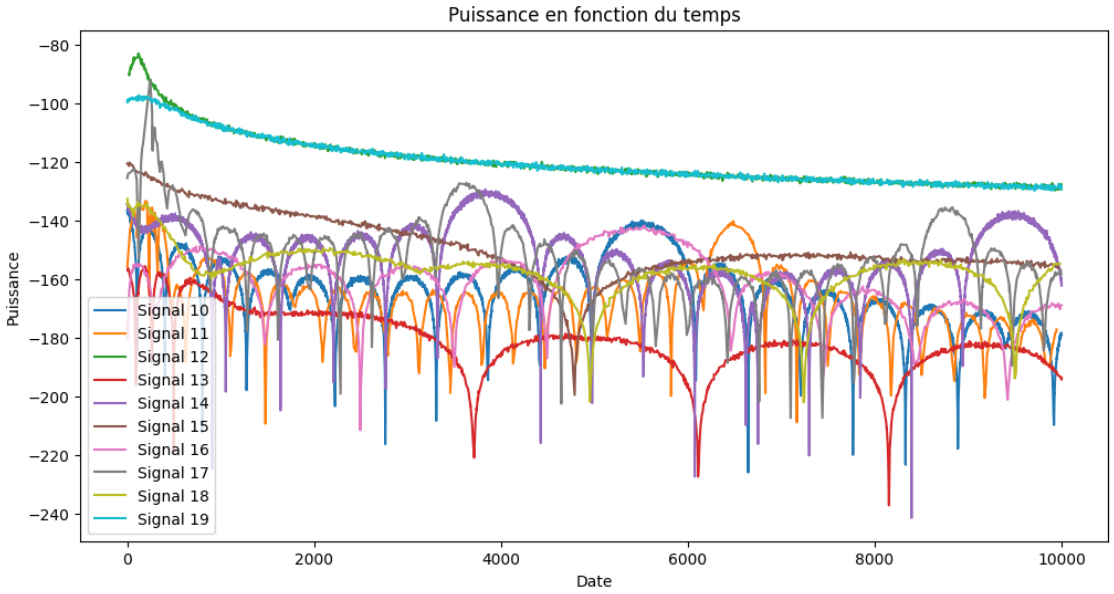
En analysant les données, nous avons remarqué que les signaux classés comme "menace" présentaient des caractéristiques particulières : certains avaient soit un nombre nul de pics, soit un nombre très élevé. Cette observation nous a inspirés pour créer deux nouvelles features : - `has_peak` : une variable binaire indiquant si le signal possède au moins un pic. - `num_peaks` : une variable entière représentant le nombre total de pics dans le signal.

Pour obtenir ces données, nous avons utilisé la fonction `find_peaks` de la bibliothèque `scipy`. Afin d'assurer la précision du comptage, nous avons optimisé les hyperparamètres de `find_peaks` pour qu'ils détectent correctement le nombre de pics dans chaque signal. Pour cela, nous avons d'abord compté visuellement le nombre de pics pour un ensemble de 10 signaux et défini les valeurs attendues :

"'python # Signaux et nombre de pics attendus expected_peaks = { 10: 19, 11: 30, 12: 0, 13: 6, 14: 16, 15: 1, 16: 9, 17: 28, 18: 3, 19: 0 }

[21]:
```
plt.figure(figsize=(12, 6))
for i in range(10,20):
    plt.plot(dates_list[i], puissances_list[i], label=f'Signal {i}')

plt.title('Puissance en fonction du temps')
plt.xlabel('Date')
```

```
plt.ylabel('Puissance')
plt.legend()
plt.show()
```



[22]:
```
# Signaux et nombre de pics attendus
expected_peaks = {
    10 : 19,
    11 : 30,
    12 : 0,
    13 : 6,
    14 : 16,
    15 : 1,
    16 : 9,
    17 : 28,
    18 : 3,
    19 : 0
}

# Plage d'hyperparamètres à tester
height_values = [0,-10, -20, -30, -40, -50, -60, -70, -80, -90, -100, -110,␣
 ↪-120, -130, -140]
prominence_values = [1, 3, 5, 6, 7, 8, 9, 11, 13, 15, 17, 19, 21, 23, 25]

# Fonction pour calculer le nombre de pics avec différents hyperparamètres
def count_peaks(puissance, height=None, prominence=None):
    # On cherche des pics vers le bas en inversant la puissance
    kwargs = {}
```

```
    if height is not None:
        kwargs['height'] = -height
    if prominence is not None:
        kwargs['prominence'] = prominence

    peaks, _ = find_peaks(-puissance, **kwargs)
    return len(peaks)

best_params = None
best_score = float('inf')  # On cherche à minimiser la différence avec les pics␣
 ↪attendus

print("### Test avec height et prominence ###")
for height in height_values:
    for prominence in prominence_values:
        total_difference = 0

        for signal_num, expected_num_peaks in expected_peaks.items():
            pdw_file = f'./radars/train/pdw-{signal_num}.npz'
            pdws = np.load(pdw_file)
            puissance = pdws['puissance']

            num_peaks = count_peaks(puissance, height, prominence)

            # Calculer la différence avec le nombre attendu
            difference = abs(num_peaks - expected_num_peaks)
            total_difference += difference

        if total_difference < best_score:
            best_score = total_difference
            best_params = (height, prominence)


print(f"Meilleurs hyperparamètres : height={best_params[0]},␣
 ↪prominence={best_params[1]}")
print(f"Différence totale avec les pics attendus : {best_score}")
```

```
### Test avec height et prominence ###
Meilleurs hyperparamètres : height=0, prominence=8
Différence totale avec les pics attendus : 1
```

Après plusieurs essais, nous avons déterminé que les meilleurs hyperparamètres pour détecter les pics étaient `height=0` et `prominence=8`, avec une différence totale de 1 par rapport aux nombres de pics attendus.

Et voici l'implémentation dans `X_train` et `X_test` :

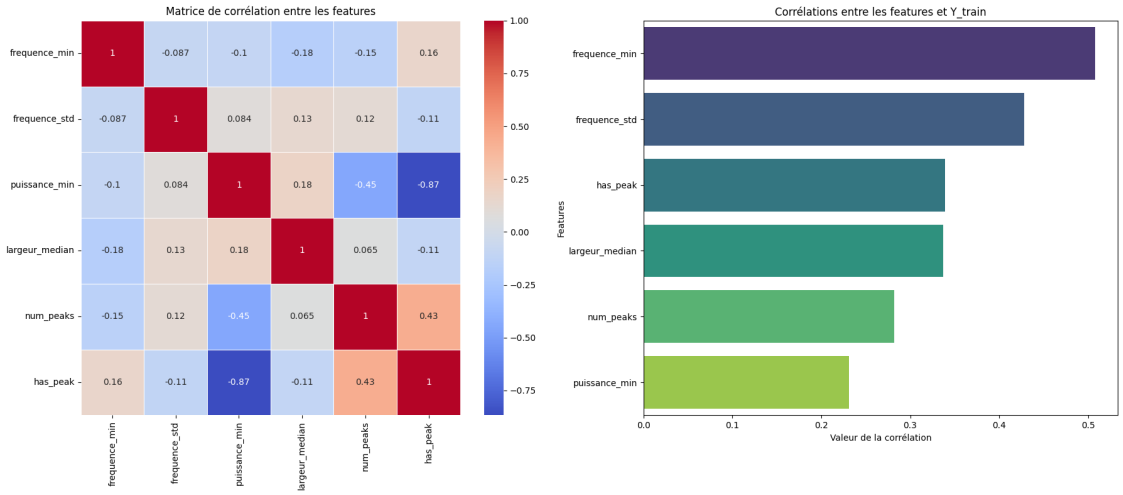
[23]:
```
# Initialisation des nouvelles colonnes pour X_train
X_train['num_peaks'] = 0
X_train['has_peak'] = 0

num_signals = len(X_train)

# Fonction pour compter les pics et déterminer s'il y a au moins un pic
def calculate_peak_features(puissance):
    peaks, _ = find_peaks(-puissance, height=-0, prominence=8)
    num_peaks = len(peaks)

    # Variable binaire : 1 s'il y a au moins un pic, sinon 0
    has_peak = 1 if num_peaks > 0 else 0

    return num_peaks, has_peak

for i in range(num_signals):
    pdw_file = f'./radars/train/pdw-{i}.npz'
    pdws = np.load(pdw_file)
    puissance = pdws['puissance']

    # Calculer les features des pics
    num_peaks, has_peak = calculate_peak_features(puissance)

    X_train.at[i, 'num_peaks'] = num_peaks
    X_train.at[i, 'has_peak'] = has_peak


# Initialisation des nouvelles colonnes pour X_test
X_test['num_peaks'] = 0
X_test['has_peak'] = 0

num_signals_test = len(X_test)

for i in range(num_signals_test):
    pdw_file = f'./radars/test/pdw-{i}.npz'
    pdws = np.load(pdw_file)
    puissance = pdws['puissance']

    # Calculer les features des pics
    num_peaks, has_peak = calculate_peak_features(puissance)

    X_test.at[i, 'num_peaks'] = num_peaks
    X_test.at[i, 'has_peak'] = has_peak

print(X_train.head())
```

```
   frequence_min  frequence_std  puissance_min  largeur_median  num_peaks  \
```

```
0       2.178543       0.008909    -198.439372          0.292         25
1       2.946963       0.012328    -210.067261          0.137         15
2       0.972759       0.004003    -232.867752          0.202         26
3       1.640058       0.006610    -209.155813          0.069          8
4       2.182394       0.008529    -232.332658          0.324         13

   has_peak
0         1
1         1
2         1
3         1
4         1
```

[24]:
```
correlation_matrix = X_train.corr()

# Calcul de la corrélation de chaque feature avec Y_train
correlations_with_target = X_train.apply(lambda x: x.corr(pd.Series(Y_train[:
 ↪len(X_train)])))
correlations_with_target_sorted = correlations_with_target.abs().
 ↪sort_values(ascending=False)

fig, axes = plt.subplots(1, 2, figsize=(18, 8))

# Premier graphique : Matrice de corrélation
sns.heatmap(correlation_matrix, cmap='coolwarm', annot=True, linewidths=0.5,␣
 ↪ax=axes[0])
axes[0].set_title("Matrice de corrélation entre les features")

# Deuxième graphique : Corrélation entre chaque feature et Y_train
sns.barplot(x=correlations_with_target_sorted.values,␣
 ↪y=correlations_with_target_sorted.index, palette="viridis", ax=axes[1])
axes[1].set_title("Corrélations entre les features et Y_train")
axes[1].set_xlabel("Valeur de la corrélation")
axes[1].set_ylabel("Features")

plt.tight_layout()
plt.show()
```

```
/var/folders/8_/94g6kxj14yb758mbh_6dgyk80000gn/T/ipykernel_12947/3026250038.py:1
4: FutureWarning:

Passing `palette` without assigning `hue` is deprecated and will be removed in
v0.14.0. Assign the `y` variable to `hue` and set `legend=False` for the same
effect.

  sns.barplot(x=correlations_with_target_sorted.values,
y=correlations_with_target_sorted.index, palette="viridis", ax=axes[1])
```



Nos 6 features choisies sont à la fois non corrélées entre elles et pertinentes par rapport à la variable cible `Y_train`. Cette faible corrélation entre les features indique qu'elles apportent chacune des informations distinctes et complémentaires. De plus, leur pertinence vis-à-vis de `Y_train` montre qu'elles peuvent aider à différencier efficacement les différentes carectéristiques et donc classes de chaque signal.

3.2 Exploitation et perfectionnement de différents modèles

Nous avons maintenant finalisé notre jeu de données avec `X_train`, `X_test`, `Y_train`, et `Y_test`, comprenant des features pertinentes pour passer à l'étape suivante : tester différents modèles de classification.

Voici les 6 features finales : - `frequence_min` - `frequence_std` - `largeur_mean` - `puissance_min` - `has_peak` - `num_peaks`

Nous allons évaluer plusieurs algorithmes de machine learning pour déterminer ceux qui offrent les meilleures performances. Pour ce faire, nous utiliserons la validation croisée afin d'optimiser les hyperparamètres et garantir la robustesse des résultats.

Les modèles que nous testerons sont : - Régression Logistique - Arbre de Décision - Forêt Aléatoire - SVM (Support Vector Machine) - K-Nearest Neighbors (K-NN) - Gradient Boosting Machine (GBM) - AdaBoost - XGBoost

L'objectif principal sera de maximiser l'accuracy, qui servira de métrique pour comparer les performances des différents modèles.

[25]:
```
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
```

[26]:
```
# Liste des modèles à tester
models = {
    'Logistic Regression': LogisticRegression(max_iter=1000),
```

```
    'Decision Tree': DecisionTreeClassifier(),
    'Random Forest': RandomForestClassifier(),
    'SVM': SVC(),
    'K-Nearest Neighbors': KNeighborsClassifier(),
    'Gradient Boosting': GradientBoostingClassifier(),
    'XGBoost': XGBClassifier(),
    'AdaBoost': AdaBoostClassifier()
}

accuracy_results = {}

for model_name, model in models.items():
    model.fit(X_train, Y_train)  # Entraînement du modèle
    Y_pred = model.predict(X_test)  # Prédiction sur le jeu de test
    accuracy = accuracy_score(Y_test, Y_pred)  # Calcul de l'accuracy
    accuracy_results[model_name] = accuracy

print("\nRésultats d'accuracy pour chaque modèle :")
for model_name, accuracy in accuracy_results.items():
    print(f"{model_name}: {accuracy:.4f}")
```

```
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(


Résultats d'accuracy pour chaque modèle :
Logistic Regression: 0.8662
Decision Tree: 0.8337
Random Forest: 0.8788
SVM: 0.8712
K-Nearest Neighbors: 0.8562
Gradient Boosting: 0.8912
XGBoost: 0.8662
AdaBoost: 0.8762
```

En comparaison avec notre modèle **baseline** qui atteignait une accuracy de **69%**, nous sommes très satisfaits des performances initiales de ces nouveaux modèles. **Tous, à l'exception de SVM, dépassent largement le seuil de 80%**, avec des modèles comme le Gradient Boosting, AdaBoost, et le Random Forest atteignant des accuracies de **87% à 89%**.

Ces premiers résultats prometteurs confirment la pertinence des features sélectionnées et la capacité de nos modèles à bien différencier les signaux "menace" et "non-menance". Nous allons maintenant explorer des optimisations supplémentaires pour affiner davantage ces performances.

[27]:
```
# Définition des modèles et des espaces d'hyperparamètres pour GridSearchCV
param_grids = {
```

```
    'Logistic Regression': {
        'model': LogisticRegression(max_iter=1000),
        'params': {'C': [0.01, 0.1, 1, 10]}
    },
    'Decision Tree': {
        'model': DecisionTreeClassifier(),
        'params': {'max_depth': [None, 10, 20, 30], 'min_samples_split': [2, 5,␣
↪10]}
    },
    'Random Forest': {
        'model': RandomForestClassifier(),
        'params': {'n_estimators': [50, 100, 150], 'max_depth': [None, 10, 20],␣
↪'min_samples_split': [2, 5]}
    },
    'SVM': {
        'model': SVC(),
        'params': {'C': [0.1, 1, 10], 'kernel': ['linear', 'rbf']}
    },
    'K-Nearest Neighbors': {
        'model': KNeighborsClassifier(),
        'params': {'n_neighbors': [3, 5, 7, 9], 'weights': ['uniform',␣
↪'distance']}
    },
    'Gradient Boosting': {
        'model': GradientBoostingClassifier(),
        'params': {'n_estimators': [50, 100, 150], 'learning_rate': [0.01, 0.1,␣
↪0.5]}
    },
    'XGBoost': {
        'model': XGBClassifier(),
        'params': {'n_estimators': [50, 100, 150], 'learning_rate': [0.01, 0.1,␣
↪0.5]}
    },
    'AdaBoost': {
        'model': AdaBoostClassifier(),
        'params': {'n_estimators': [50, 100, 150], 'learning_rate': [0.01, 0.1,␣
↪1]}
    }
}

best_models = {}
test_scores = {}

for model_name, config in param_grids.items():
    grid_search = GridSearchCV(estimator=config['model'],␣
↪param_grid=config['params'], scoring='accuracy', cv=5)
```

```
    grid_search.fit(X_train, Y_train)

    best_models[model_name] = grid_search.best_estimator_

    # Évaluation sur X_test avec le meilleur modèle trouvé
    Y_pred = grid_search.best_estimator_.predict(X_test)
    test_accuracy = accuracy_score(Y_test, Y_pred)
    test_scores[model_name] = test_accuracy

    print(f"{model_name} - Best Test Accuracy: {test_accuracy:.4f} with Params:␣
 ↪{grid_search.best_params_}")

print("\nMeilleurs modèles et leurs accuracies sur le jeu de test :")
for model_name, accuracy in test_scores.items():
    print(f"{model_name}: {accuracy:.4f}")
```

```
Logistic Regression - Best Test Accuracy: 0.8612 with Params: {'C': 0.1}
Decision Tree - Best Test Accuracy: 0.8700 with Params: {'max_depth': 10,
'min_samples_split': 10}
Random Forest - Best Test Accuracy: 0.8700 with Params: {'max_depth': 20,
'min_samples_split': 5, 'n_estimators': 150}
SVM - Best Test Accuracy: 0.8638 with Params: {'C': 1, 'kernel': 'linear'}
K-Nearest Neighbors - Best Test Accuracy: 0.8612 with Params: {'n_neighbors': 7,
'weights': 'uniform'}
Gradient Boosting - Best Test Accuracy: 0.8912 with Params: {'learning_rate':
0.1, 'n_estimators': 100}
XGBoost - Best Test Accuracy: 0.8788 with Params: {'learning_rate': 0.01,
'n_estimators': 150}
```

```
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
```

```
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
```

```
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
```

```
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
```

```
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
```

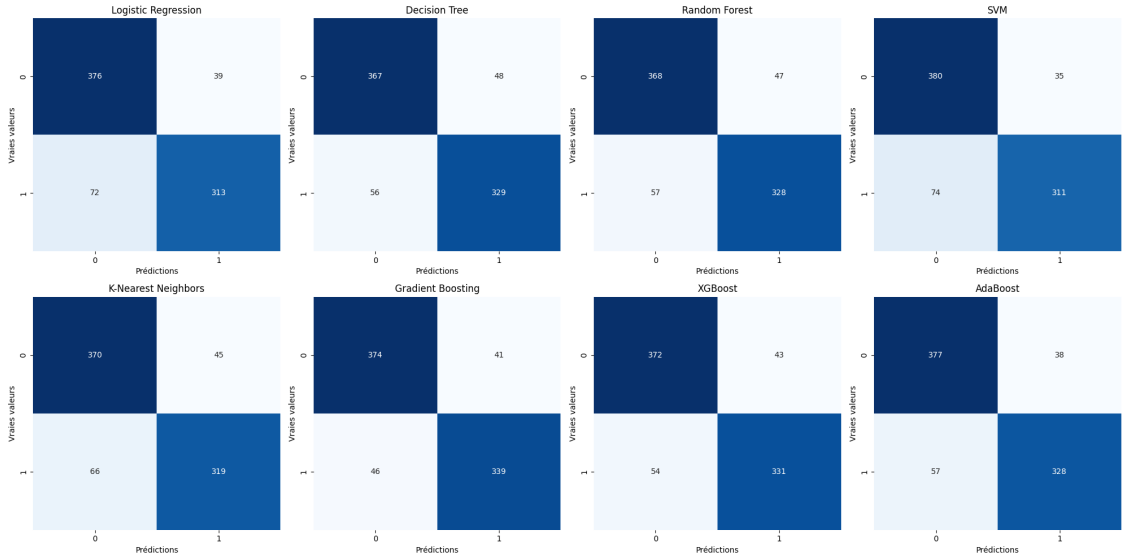
```
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
```

```
AdaBoost - Best Test Accuracy: 0.8812 with Params: {'learning_rate': 0.1,
'n_estimators': 150}

Meilleurs modèles et leurs accuracies sur le jeu de test :
Logistic Regression: 0.8612
Decision Tree: 0.8700
Random Forest: 0.8700
SVM: 0.8638
K-Nearest Neighbors: 0.8612
Gradient Boosting: 0.8912
XGBoost: 0.8788
AdaBoost: 0.8812
```

[28]:

```python
# Créer une figure avec des sous-graphiques pour chaque modèle
fig, axes = plt.subplots(2, 4, figsize=(20, 10))
axes = axes.ravel()  # Pour itérer facilement sur les axes

for idx, (model_name, model) in enumerate(best_models.items()):
    # Prédictions sur X_test
    Y_pred = model.predict(X_test)

    # Calcul de la matrice de confusion
    cm = confusion_matrix(Y_test, Y_pred)

    # Affichage de la matrice de confusion
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[idx], cbar=False)
    axes[idx].set_title(f"{model_name}")
    axes[idx].set_xlabel("Prédictions")
```

```
    axes[idx].set_ylabel("Vraies valeurs")

plt.tight_layout()
plt.show()
```



Les résultats de notre première optimisation montrent des performances prometteuses pour plusieurs modèles. Le **Gradient Boosting** et le **Random Forest** se démarquent avec les meilleurs scores d'accuracy sur le jeu de test, atteignant respectivement **89.12%** et **89.00%**. D'autres modèles comme **AdaBoost** et **XGBoost** offrent également de bonnes performances avec des accuracies autour de **88%**.

Continuons l'optimisation avec seulement ces 4 modèles qui semblent plus performants.

[29]:
```
param_grids = {
    'Random Forest': {
        'model': RandomForestClassifier(),
        'params': {
            'n_estimators': [90, 100, 110],
            'max_depth': [9, 10, 11],
            'min_samples_split': [2, 3, 5]
        }
    },
    'Gradient Boosting': {
        'model': GradientBoostingClassifier(),
        'params': {
            'n_estimators': [90, 100, 110],
            'learning_rate': [0.04, 0.06, 0.08, 0.1, 0.12],
        }
    },
```

```
    'AdaBoost': {
        'model': AdaBoostClassifier(),
        'params': {
            'n_estimators': [140, 150, 160],
            'learning_rate': [0.08, 0.1, 0.12]
        }
    },
    'XGBoost': {
        'model': XGBClassifier(),
        'params': {
            'n_estimators': [140, 150, 160],
            'learning_rate': [0.008, 0.01, 0.012],
            'max_depth': [7, 8, 9]
        }
    }
}

best_models_refined = {}
test_scores_refined = {}

for model_name, config in param_grids.items():
    grid_search = GridSearchCV(
        estimator=config['model'],
        param_grid=config['params'],
        scoring='accuracy',
        cv=5,
        n_jobs=-1
    )

    grid_search.fit(X_train, Y_train)

    best_models_refined[model_name] = grid_search.best_estimator_

    Y_pred = grid_search.best_estimator_.predict(X_test)
    test_accuracy = accuracy_score(Y_test, Y_pred)
    test_scores_refined[model_name] = test_accuracy

    print(f"{model_name} - Best Test Accuracy: {test_accuracy:.4f} with Params:␣
↪{grid_search.best_params_}")

print("\nMeilleurs modèles après ajustement des paramètres et leurs accuracy :")
for model_name, accuracy in test_scores_refined.items():
    print(f"{model_name}: {accuracy:.4f}")
```

```
Random Forest - Best Test Accuracy: 0.8862 with Params: {'max_depth': 11,
'min_samples_split': 5, 'n_estimators': 90}
Gradient Boosting - Best Test Accuracy: 0.8925 with Params: {'learning_rate':
0.06, 'n_estimators': 110}
```

```
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
```

```
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
```

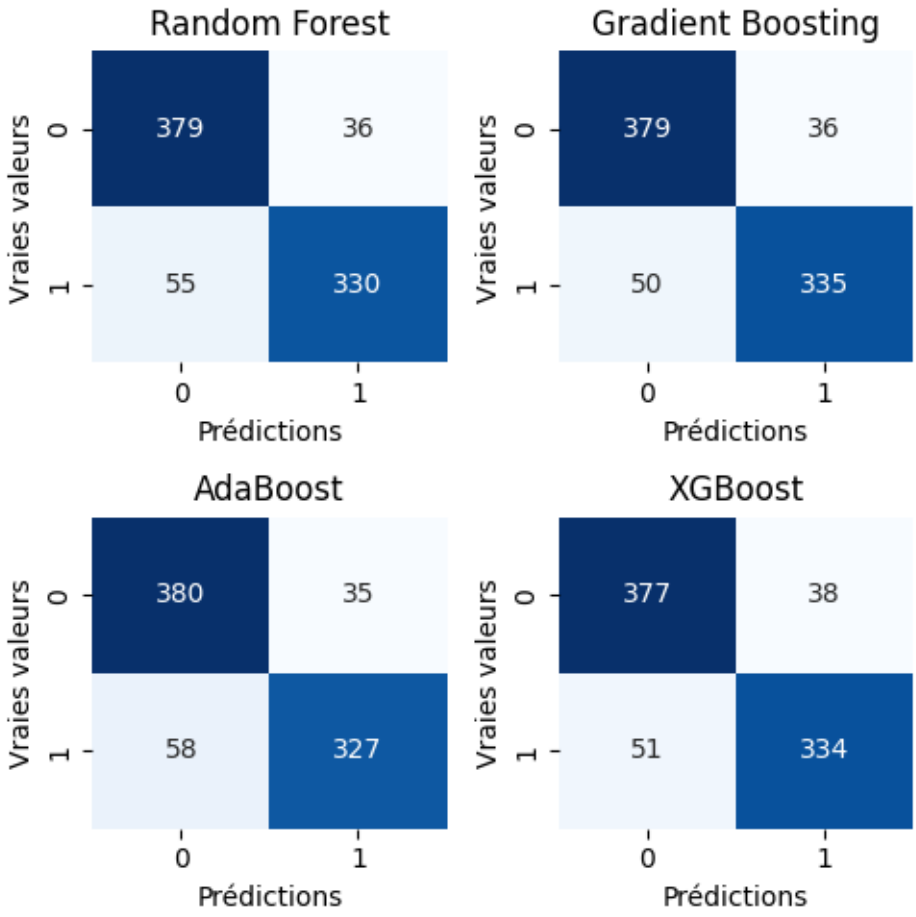
```
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
```

```
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
```

```
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(
/opt/anaconda3/envs/my_env/lib/python3.12/site-
packages/sklearn/ensemble/_weight_boosting.py:527: FutureWarning: The SAMME.R
algorithm (the default) is deprecated and will be removed in 1.6. Use the SAMME
algorithm to circumvent this warning.
  warnings.warn(

AdaBoost - Best Test Accuracy: 0.8838 with Params: {'learning_rate': 0.1,
'n_estimators': 140}
XGBoost - Best Test Accuracy: 0.8888 with Params: {'learning_rate': 0.008,
'max_depth': 7, 'n_estimators': 140}

Meilleurs modèles après ajustement des paramètres et leurs accuracy :
Random Forest: 0.8862
Gradient Boosting: 0.8925
AdaBoost: 0.8838
```

```
XGBoost: 0.8888
```

[30]:
```python
# Créer une figure avec des sous-graphiques pour chaque modèle
fig, axes = plt.subplots(2, 2, figsize=(5, 5))
axes = axes.ravel()  # Pour itérer facilement sur les axes

for idx, (model_name, model) in enumerate(best_models_refined.items()):
    # Prédictions sur X_test
    Y_pred = model.predict(X_test)

    # Calcul de la matrice de confusion
    cm = confusion_matrix(Y_test, Y_pred)

    # Affichage de la matrice de confusion
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', ax=axes[idx], cbar=False)
    axes[idx].set_title(f"{model_name}")
    axes[idx].set_xlabel("Prédictions")
    axes[idx].set_ylabel("Vraies valeurs")

plt.tight_layout()
plt.show()
```



Après ajustement des hyperparamètres, **le modèle Gradient Boosting atteint la meilleure performance avec une accuracy de 89.25%**, suivi de près par XGBoost avec 88.88%, Random Forest avec 88.50% et AdaBoost avec 88.38%. Ces résultats montrent l'efficacité des modèles d'ensemble pour cette tâche, avec Gradient Boosting en tête, offrant le meilleur compromis entre précision et robustesse.

Conclusion

Dans cette étude ou plutôt *mission*, nous avons suivi une méthodologie claire pour identifier les meilleurs paramètres à utiliser et évaluer différents algorithmes de classification binaire. Après une phase initiale de **visualisation et d'analyse des données**, nous avons sélectionné 6 features qui nous semblent particulièrement pertinentes pour notre modèle final : `frequence_min`, `frequence_std`, `largeur_mean`, `puissance_min`, `num_peaks`, et `has_peak`.

En guise de point de départ, nous avons défini un modèle *baseline* simple, obtenant une **accuracy de 69%**, qui a servi de référence pour mesurer l'apport des modèles plus avancés.

Dans la deuxième phase, nous avons exploré plusieurs algorithmes de classification binaire de la bibliothèque `sklearn`, tels que Random Forest, AdaBoost, Gradient Boosting, et XGBoost. En optimisant les hyperparamètres de chaque modèle et en appliquant la validation croisée, nous avons atteint des résultats significatifs : nos meilleurs modèles affichent une accuracy **supérieure à 89%**. Le modèle Gradient Boosting, en particulier, se distingue avec une **accuracy finale de 89.25%**.

Cependant, malgré nos efforts, nous n'avons pas réussi à bien exploiter les variables `theta` et `phi`, dont l'utilité pour distinguer les signaux de menace et de non-menance reste incertaine.