22-10-2021

Rapport d'avancement

Tentative d'apprentissage des corrélations entre phases de la lune et du soleil et coefficients des marées

---

Auteurs :
Otmane EL ALOI, Abdelhafid SAOUD, Abdelhadi ZIANE

Encadrant :
Mr. Olivier ROUX

Option :
Informatique

Table des matières

1 Avancement par rapport au modèle de prédiction 2
1.1 DNN (Deep Neural Network) model 2
1.1.1 Définition 2
1.1.2 DNN pour prédire les coefficients de marrée 2
1.1.3 Préparation des données 2
1.1.4 Construction et formation du modèle 4
1.1.5 Évaluation du modèle 4
1.2 Modèle de forêt aléatoire avec mlflow Tracking 6
2 Avancement par rapport à l'interface web 7
2.1 Flask 8
2.2 Bootstrap 8
3 Planning 9
3.1 Planning de la semaine en cours 9
3.2 Planning de la semaine après les vacances 10

Introduction

Dans ce rapport, nous présenterons notre avancement par rapport à deux axes : par rapport au modèle de prédiction et par rapport à l'interface web. En effet, cette semaine, on s'est focalisé sur la construction du modèle de prédiction à savoir la manipulation, le test et l'évaluation. De plus nous avons commencé la conception de l'interface web par des outils que nous détaillerons par la suite.

1 Avancement par rapport au modèle de prédiction

1.1 DNN (Deep Neural Network) model

1.1.1 Définition

Un réseau neuronal, en général, est une technologie construite pour simuler l'activité du cerveau humain - plus précisément, la reconnaissance des formes et le passage des données d'entrée à travers différentes couches de connexions neuronales simulées.

De nombreux experts définissent les réseaux neuronaux profonds comme des réseaux comportant une couche d'entrée, une couche de sortie et au moins une couche cachée entre les deux. Chaque couche effectue des types spécifiques de tri et d'ordonnancement dans un processus que certains appellent "hiérarchie des caractéristiques". L'une des principales utilisations de ces réseaux neuronaux sophistiqués est le traitement de données non étiquetées ou non structurées. L'expression "apprentissage profond" est également utilisée pour décrire ces réseaux neuronaux profonds, car l'apprentissage profond représente une forme spécifique d'apprentissage automatique où les technologies utilisant des aspects de l'intelligence artificielle cherchent à classer et à ordonner les informations d'une manière qui va au-delà des simples protocoles d'entrée/sortie.

1.1.2 DNN pour prédire les coefficients de marrée

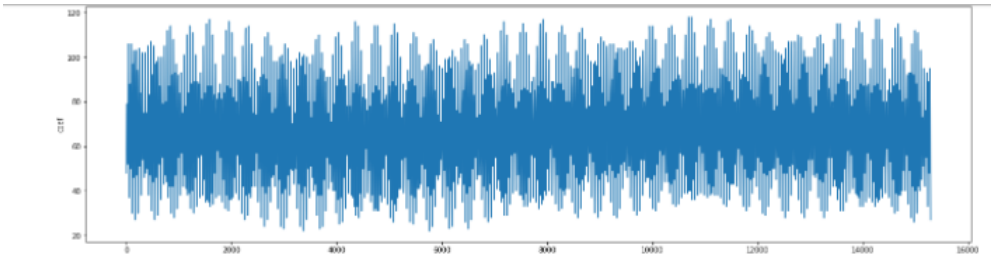
Nous avons utilisé un réseau neuronal profond personnalisé pour prévoir les coefficients de marées à l'aide de TensorFlow.

1.1.3 Préparation des données

Au cours de cette étape, nous devons transformer les données chargées et les traiter de manière à ce qu'elles puissent être transmises comme entrée à notre modèle, après ça nous pouvons commencer le processus de formation. Nous pouvons considérer la prévision de séries temporelles comme un problème de régression, et les données de séries temporelles sont converties en un ensemble de valeurs caractéristiques et la valeur vraie ou cible correspondante. La régression étant un problème d'apprentissage supervisé, nous avons besoin de la valeur cible, dans laquelle les données de la série chronologique

décalée deviennent les valeurs caractéristiques.

Nous suivrons une approche de fenêtre ou de tampon dans laquelle nous devrons considérer une taille de fenêtre appropriée. Ensuite, nous déplacerons la fenêtre de gauche à droite de la séquence ou de la série de données. Nous considérerons la valeur immédiatement à droite de la fenêtre comme la valeur cible ou la valeur réelle. Ainsi, à chaque pas de temps, nous déplaçons la fenêtre de manière à obtenir une nouvelle rangée de paires de valeurs caractéristiques et de valeurs cibles. De cette façon, nous formons les données d'apprentissage et les étiquettes d'apprentissage. De la même manière, nous formons l'ensemble de données de test et de validation, qui est généralement requis pour un modèle de prédiction d'apprentissage automatique. Ensuite, pour le rapport de division formation-test-validation, nous devrons le déterminer en fonction de la taille des données. Pour notre modèle, nous avons utilisé un rapport de division de 0,8 et, en fonction de la saisonnalité des données, nous avons pris une taille de fenêtre de 14. Mais ces variables sont toutes des hyper-paramètres, ce qui nécessite un certain réglage pour obtenir les meilleurs résultats possibles.



[4]:
```
# Convert the data values to numpy for better and faster processing
time_index = np.array(mare['date'])
data = np.array(mare['coef'])
# Certain Hyper-parameters to tune
SPLIT_RATIO = 0.8
WINDOW_SIZE = 14
BATCH_SIZE = 32
SHUFFLE_BUFFER = 1000
# Dividing into train-test split
split_index = int(SPLIT_RATIO * data.shape[0])
# Train-Test Split
train_data = data[:split_index]
train_time = time_index[:split_index]
test_data = data[split_index:]
test_time = time_index[split_index:]
```

Ensuite, nous avons préparé un générateur de données qui prépare les données d'entraînement et de test pour nous.

[5]:
```
def ts_data_generator(data, window_size, batch_size, shuffle_buffer):
  '''
  Utility function for time series data generation in batches
  '''
  ts_data = tf.data.Dataset.from_tensor_slices(data)
  ts_data = ts_data.window(window_size , shift=1, drop_remainder=True)
  ts_data = ts_data.flat_map(lambda window: window.batch(window_size))
  ts_data = ts_data.map(lambda window: (window[:-7], window[-7:]))
  ts_data = ts_data.batch(batch_size).prefetch(1)
  return ts_data
```

[6]:
```
train_dataset = ts_data_generator(train_data, WINDOW_SIZE, BATCH_SIZE, SHUFFLE_BUFFER)
test_dataset = ts_data_generator(test_data, WINDOW_SIZE, BATCH_SIZE, SHUFFLE_BUFFER)
```

1.1.4 Construction et formation du modèle

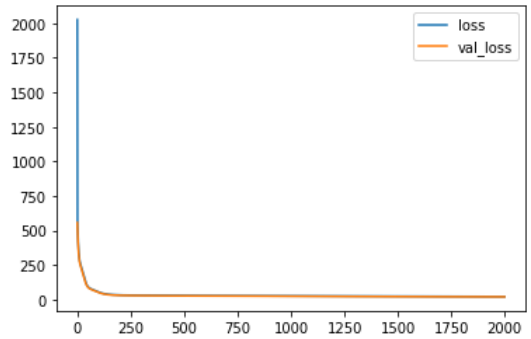
Le code de l'architecture de notre modèle ressemble à ceci :

```
model = tf.keras.models.Sequential([
    tf.keras.layers.Dense(20, input_shape=[7], activation="relu"),
    tf.keras.layers.Dense(10, activation="relu"),
    tf.keras.layers.Dense(7)
])
model.compile(loss="mse", optimizer=tf.keras.optimizers.SGD(lr=1e-7, momentum=0.9))
h = model.fit(train_dataset, epochs=2000, validation_data = test_dataset)
```

1.1.5 Évaluation du modèle

Dans un premier temps, nous comparé la fonction de perte des données formation et de validation. Cette fonction est l'un des composants importants des réseaux de neurones. La perte n'est rien d'autre qu'une erreur de prédiction de ces réseaux.
La perte est utilisée pour calculer les gradients. Et les gradients sont utilisés pour mettre à jour les poids des réseaux de neurones. C'est ainsi qu'un réseau neuronal est formé.

```
history = pd.DataFrame(h.history).plot()
```



Cette résultat signifie que notre modèle est dans le sous-apprentissage qui est un scénario en science des données où un modèle de données est incapable de capturer avec précision la relation entre les variables d'entrée et de sortie, générant un taux d'erreur élevé à la fois sur l'ensemble d'apprentissage et sur les données invisibles.

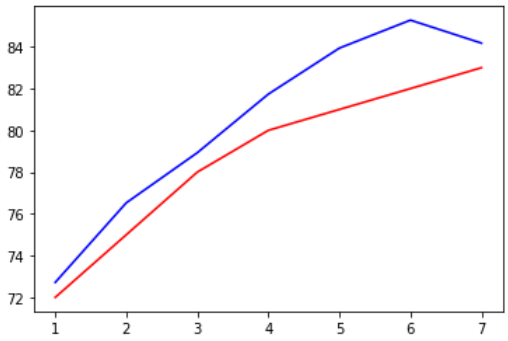
Alors, nous devons ajouter d'autres couches dans modèle.
Cependant, nous avons fait une prédiction d'une semaine, tout en donnant les données du semaine qu'elle a précède.

```
for elem in test_dataset.as_numpy_iterator():
    X0 = elem[0][4]
    Y0 = elem[1][4]
    break
```

```
y_predicted6 = model.predict(np.reshape(X0, newshape = (1,7)))
```

```
plt.plot(np.arange(7)+1, y_predicted6[0], color= "blue")
plt.plot(np.arange(7)+1, Y0, color = "red")
```

```
[<matplotlib.lines.Line2D at 0x2951f59d5e0>]
```



Ensuite, nous avons mesuré l'erreur de notre modèle, tout en utilisant le RMSE(Root Mean Square Error), pour une seule prédiction.

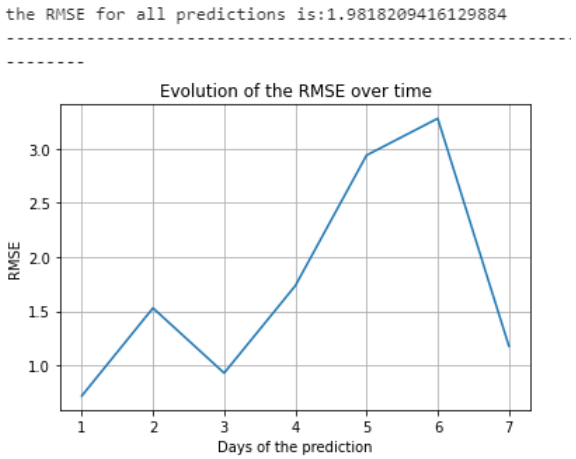
```
def evaluate_forecasts(actual, predicted):
    scores = list()
    # calculate an RMSE score for each day
    for i in range(7):
        # calculate mse
        mse = mean_squared_error(actual[:,i], predicted[:,i])
        # calculate rmse
        rmse = sqrt(mse)
        # store
        scores.append(rmse)
    # calculate overall RMSE
    s = 0
    for row in range(actual.shape[0]):
        for col in range(actual.shape[1]):
            s += (actual[row, col] - predicted[row, col])**2
    score = sqrt(s / (actual.shape[0] * actual.shape[1]))
    return score, scores
```

```
score, scores = evaluate_forecasts(Y0, y_predicted6)
```

```
print("the RMSE for all predictions is:{}".format(score))

print("-"*100)

plt.plot(np.arange(7)+1, scores)
plt.xlabel("Days of the prediction")
plt.ylabel("RMSE")
plt.grid()
plt.title("Evolution of the RMSE over time");
```



1.2 Modèle de forêt aléatoire avec mlflow Tracking

Un projet de Machine Learning consiste, en étant très générique, en l'application d'une méthode théorique à un jeu de données particulier. Le travail est très schématiquement, de sélectionner la méthode la plus performante avec les paramètres qui lui permettront de s'adapter au mieux aux données en présence. Et pour se faire, les connaissances théoriques et l'intuition, aussi géniale puisse-t-elle être, ne suffisent pas. Il faut faire des essais, des tests, des expériences. Une des difficultés majeures rencontrée dans tous les projets de Machine Learning est de conserver une trace de chaque expérience afin de reproduire celles qui ont fourni les meilleurs résultats. Pour surpacer ce problème, nous utilisons une platforme open source mlflow tracking qui permet Le suivi des expériences ; les algorithmes de Machine Learning possèdent des dizaines de paramètres configurables et il est absolument vital de savoir quel paramètre fournit quel résultat.

FIGURE 1 – Logo de la platforme mlflow

Nous avons choisi mlflow parce que mlflow Tracking est simple. Simple à installer, simple à intégrer dans le code, simple à utiliser en interface graphique. La figure ci-dessus montre l'interface graphique qui permet de visualiser les informations des différentes expériences faites.

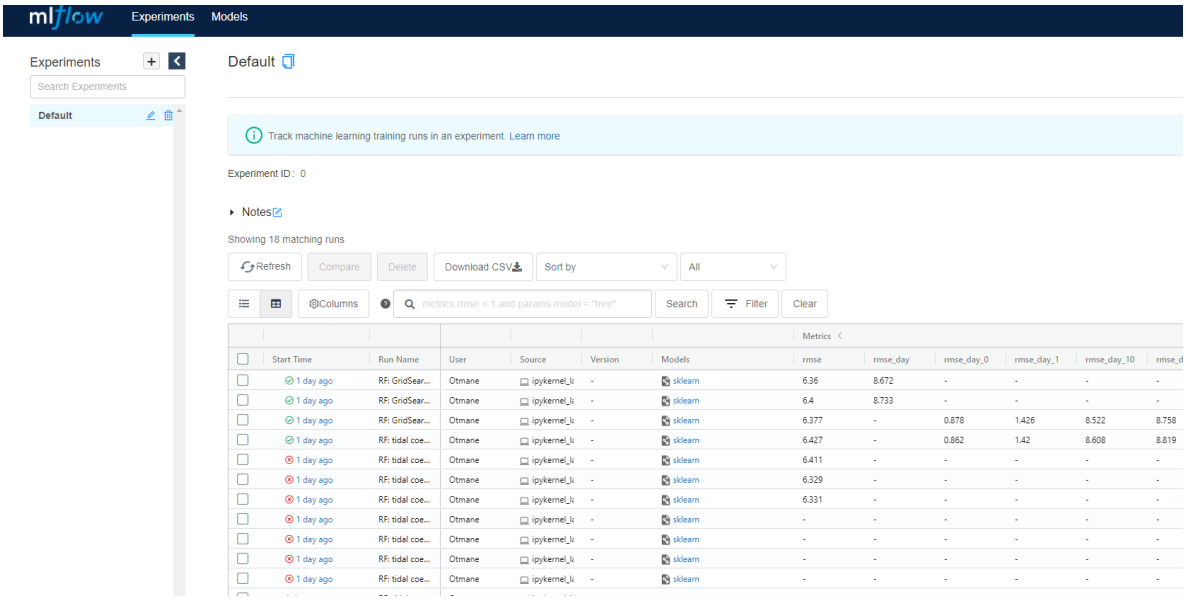


FIGURE 2 – Illustration de l'interface graphique de mlflow tracking

L'interface grapique de mlflow tracking permet la comparaison entre deux expérience tant qu'elle dispose des mêmes métriques. La figure ci-dessous montre un exmple de comparaison entre deux modèle de forêt aléatoire.

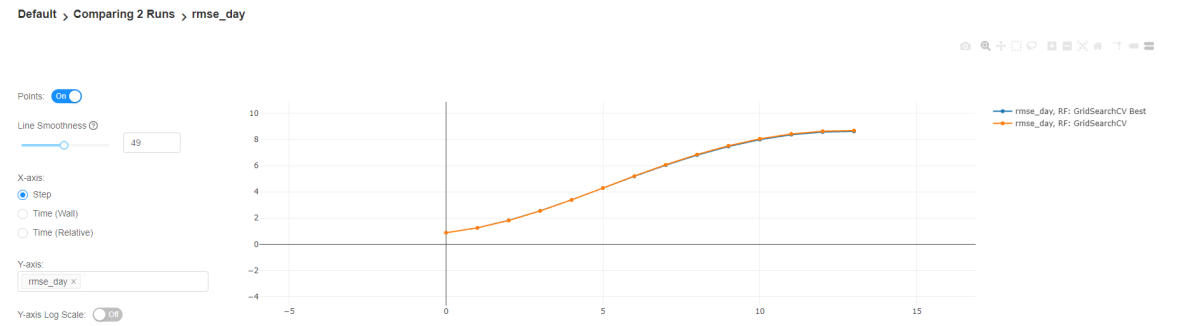


FIGURE 3 – Illustration de la comparaison entre les erreurs de deux modèles

2 Avancement par rapport à l'interface web

En parallèle avec le développement du modèle de prédiction, nous avons commencé le travail sur l'interface web avec lequel l'utilisateur pourra prédire et visualiser les coefficients des marées. Comme mentionné dans le cahier dès charges, cette interface vat être développer sous HTML/CSS et du JavaScript. Ainsi, nous avons commencé par la prise en main des frameworks que nous allons utiliser par la suite pour concevoir notre interface web :

2.1 Flask

Flask est un framework d'application Web en python. Il est conçu pour rendre le démarrage rapide et facile, avec la possibilité d'évoluer vers des applications complexes. Il a commencé comme un simple wrapper autour de Werkzeug et Jinja et est devenu l'un des frameworks d'applications Web Python les plus populaires. Flask propose des suggestions, mais n'impose aucune dépendance ni disposition de projet. C'est au développeur de choisir les outils et les bibliothèques qu'il souhaite utiliser. Il existe de nombreuses extensions fournies par la communauté qui facilitent l'ajout de nouvelles fonctionnalités.

2.2 Bootstrap

Bootstrap est une collection d'outils utiles à la création du design (graphisme, animation et interactions avec la page dans le navigateur, etc.) de sites et d'applications web. C'est un ensemble qui contient des codes HTML et CSS, des formulaires, boutons, outils de navigation et autres éléments interactifs, ainsi que des extensions JavaScript en option. Cette outil va servir à bien construire l'interface web soit du côté fonctionnel ou du côté esthétique.

3 Planning

3.1 Planning de la semaine en cours

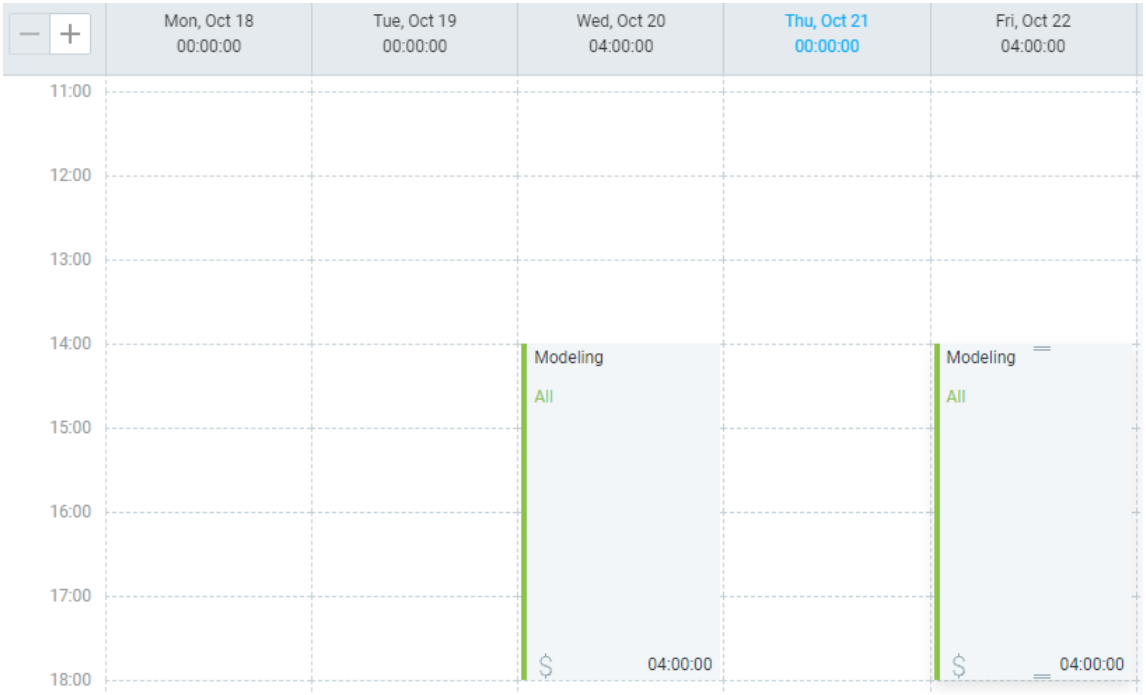


FIGURE 4 – Planning de la semaine en cours

3.2 Planning de la semaine après les vacances

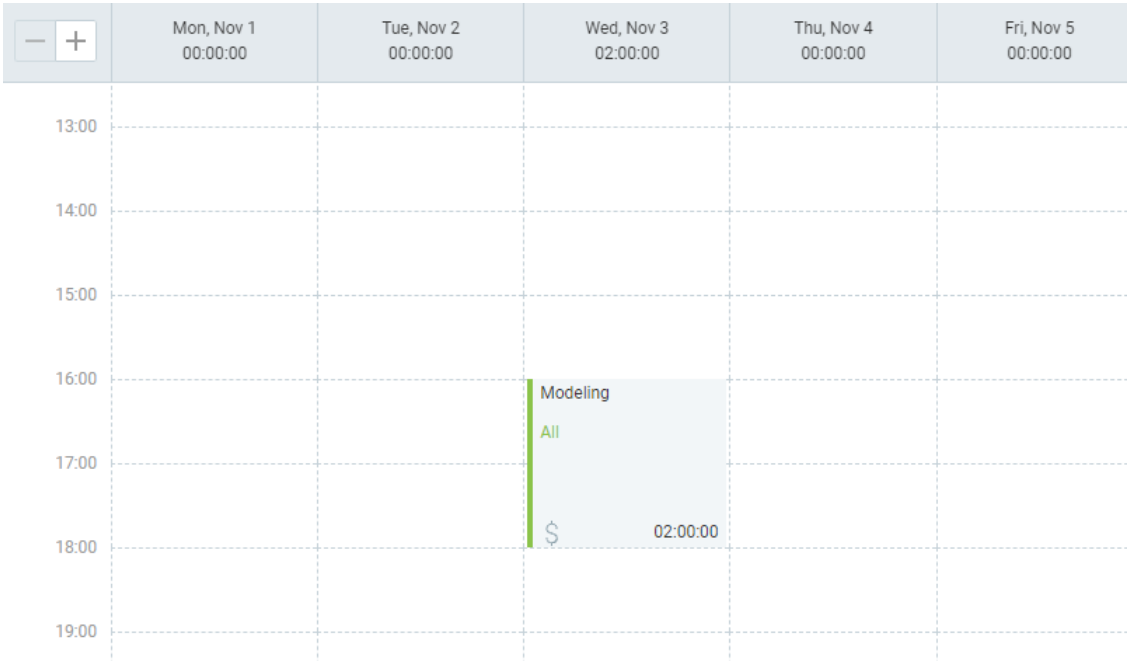


FIGURE 5 – Planning de la semaine après les vacances