Projet GML

La souris au bois dormant

Réalisé par : Tim Ernst, Justin Rausis, Grégory Rey-Mermet & Florian Conti

Table des matières

Liste de figures

Introduction

Ce projet intitulé « la souris au bois dormant » a été réalisé dans le cadre de notre cursus à la HEIG-VD dans le cours de Gestion de projet ML. Comme son nom l'indique, ce projet a pour but de nous exercer dans la gestion d'un projet de machine learning. Ce projet a été effectué par groupe de 4 est à durer 15 semaines et nous a donc permis de nous plonger au cœur d'une problématique scientifique complexe. Celle de l'analyse du cycle de sommeil chez les souris.

Comprendre le sommeil est un enjeu majeur dans le domaine médical puisque nous passons environ un tiers de notre vie à dormir. Le premier pas a effectué pour de tels projets est d'étudier le sommeil chez les animaux tels que les souris. Cela permettrait peut-être par la suite de mieux comprendre le sommeil chez l'homme et de faire des généralisations.

Pour ce faire, nous disposons de données fournis par Prof. Paul Franken comprenant les enregistrements EEG et EMG de 250 souris sur une période de 4 jours à intervalle régulier de 4 secondes. Pour chaque intervalle de 4 secondes, nous disposons de la variance EEG et EMG ainsi que 400 colonnes bins. Pour chacune de ces lignes, nous disposons également de son état. Celui-ci peut être soit REM, soit Non REM, soit réveillé. Ces états ont été déterminés par des experts pour le 3ème jour et déterminés par un modèle pour les 3 autres jours.

Le but de ce projet est donc de créer un modèle capable de déterminer l'état de la souris en fonction des données EEG et EMG. Pour cela, nous allons adopter une méthodologie agile afin de nous permettre de nous adapter aux différentes difficultés que nous pourrions rencontrer. Nous allons également utiliser des outils de gestion de projet tels que et GitHub afin de nous permettre de travailler de manière efficace et collaborative. De plus une planification initiale a été effectuée afin de nous permettre de nous organiser et de nous donner une idée de la charge de travail que nous aurons à effectuer.

Objectif

Les objectifs de ce projet ont été définis par nous-mêmes dans le cahier des charges. Les objectifs principaux sont de fournir 3 modèles de machine learning.

- Le premier modèle permettra de classifier l'état d'une souris en 2 catégories (endormie/éveillée).
- Le second modèle permettra de classifier l'état d'une souris endormie (rem/nrem).
- Le 3ème modèle plus général permettant de classifier les 3 états.

Ces 3 modèles seront d'abord entrainés et testés sur 1 souris seulement et le but sera de les généraliser à plusieurs souris d'une même souche puis si cela semble possible d'aller jusqu'à une souche de souris si cela donne des résultats acceptables.

Pour y parvenir, nous devrons tout d'abord analyser les données en effectuant une analyse profonde notamment avec la visualisation des données. Nous devrons également effectuer un pré-traitement des données (traitement des valeurs manquantes, traitement des valeurs aberrantes, suppression de certaines colonnes, normalisation, réduction de dimensionnalité).

Planification et organisation

Une planification initiale a été mise en place afin de nous permettre de nous organiser et de nous donner une idée de la charge de travail que nous aurons à effectuer. Cette planification est principalement composée de 4 étapes majeures. La première est justement la planification du projet, la seconde est le preprocessing des données, la troisième est la création des modèles et la dernière est l'évaluation des modèles et la rédaction de la documentation en parallèle. Chacune de ces tâches sont décomposées en plusieurs sous-tâches.

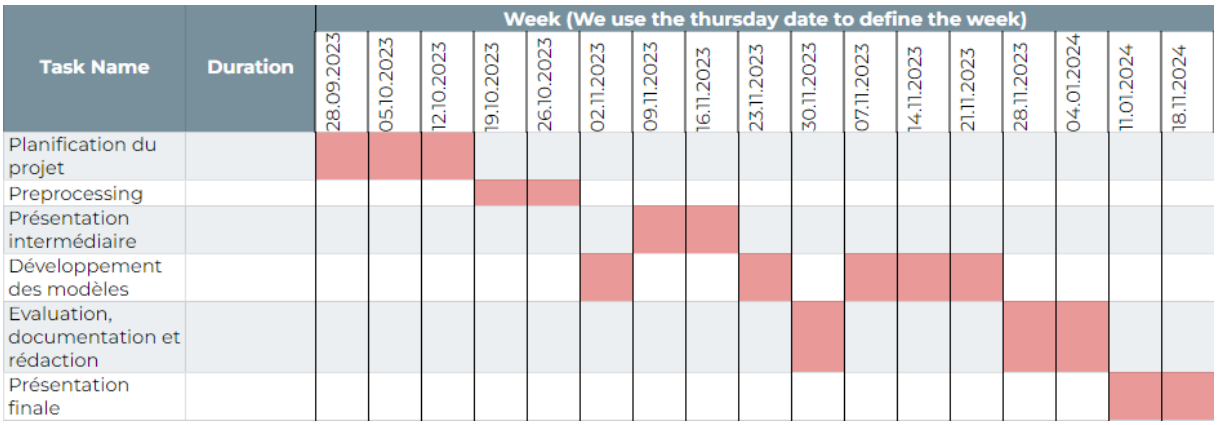

Figure 1: Planification

Pour ce qui est de l'organisation, nous avons premièrement décidé de nous séparer les tâches de cette manière : Chaque personne a développé des fonctions qui étaient utiles pour la suite du projet. Par exemple, des fonctions permettant de récupérer uniquement le 3ème jour ou encore une fonction permettant de nettoyer les données et de supprimer les valeurs aberrantes. D'un autre côté, une visualisation des données a été effectuée pour mieux comprendre les données et leurs origines.

Par la suite, par la nature complexe du problème, nous avons chacun essayé de développer des modèles différents pour voir quel modèle et quel architecture avait de meilleures performances. Une fois que nous avions trouvé un modèle relativement performant, le champ des recherches se rétrécissait et nous pouvions se focaliser sur des modèles plus similaires.

Exploration du jeu de données

L'exploration du jeu de données est une étape importante dans la création d'un modèle de machine learning. Elle permet de mieux comprendre les données et de mieux les préparer pour l'entrainement. Pour cela nous allons étudier différents graphiques nous permettant de mieux comprendre les données.

En explorant la densité de chaque catégorie, on va essayer déterminer s'il y a des classe plus représenter pour cela nous allons regarder la distribution des classes sur une des souris sur le jours trois qui a été labeliser a la main. Nous voyons que la classe REM est bien moins représenté que les deux autres. Il faudra faire attention lors de la création des modèles de prendre en compte cette disproportion.

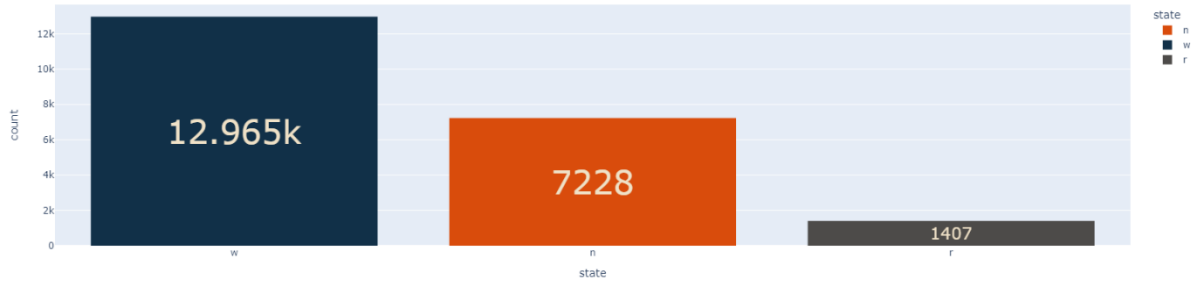

Figure 2: Répartition des états de sommeil

En faisant une analyse de l'état de transition comme nous pouvons voir sur la fig. 2. Comme nous pouvons le constater sur la fig. 2, Les changement d'état son très rare ce qui nous permet de penser qu'un modèle avec mémoire pourrait être intéressant. A noter qu'il est également très rare qu'une souris passe de l'état wake à rem. S représente l'état crise d'épilepsie est celui-ci intervient très peu.

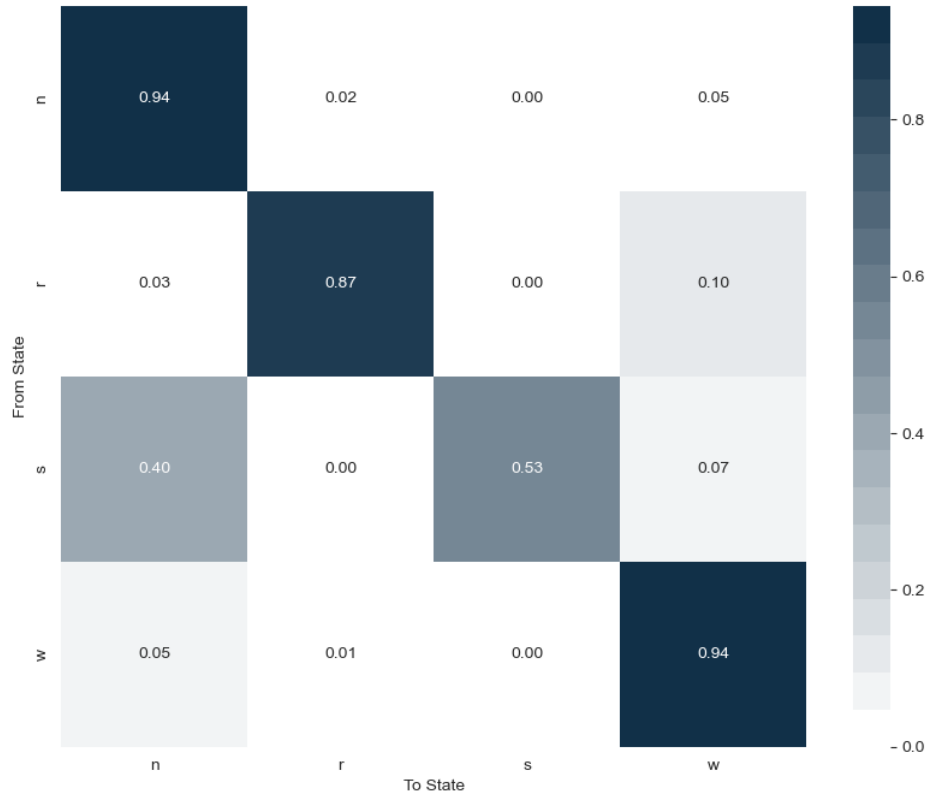

Figure 3: Transitions des états de sommeil

Umap nous permet de constater que 2 classes semblent bien séparables (wake et nrem) mais que rem semble assez difficile à prédire.

Figure 4: UMAP projection

Ce graphique permet de visualiser la perte d'information en appliquant la PCA sur notre jeu de données. Etant donné que notre jeu de données dispose de plus de 400 features, il est primordial d'utiliser la PCA afin de réduire la complexité de notre jeu de données et d'accélérer le temps d'entraiment. On peut voir qu'avec simplement 5 composantes, le pourcentage de variance expliquée est d'environ 78%. Nous avons pris le choix d'utiliser majoritairement 50 composantes afin d'avoir un pourcentage supérieur à 90%. Cela permet d'avoir un bon compromis entre complexité et information.

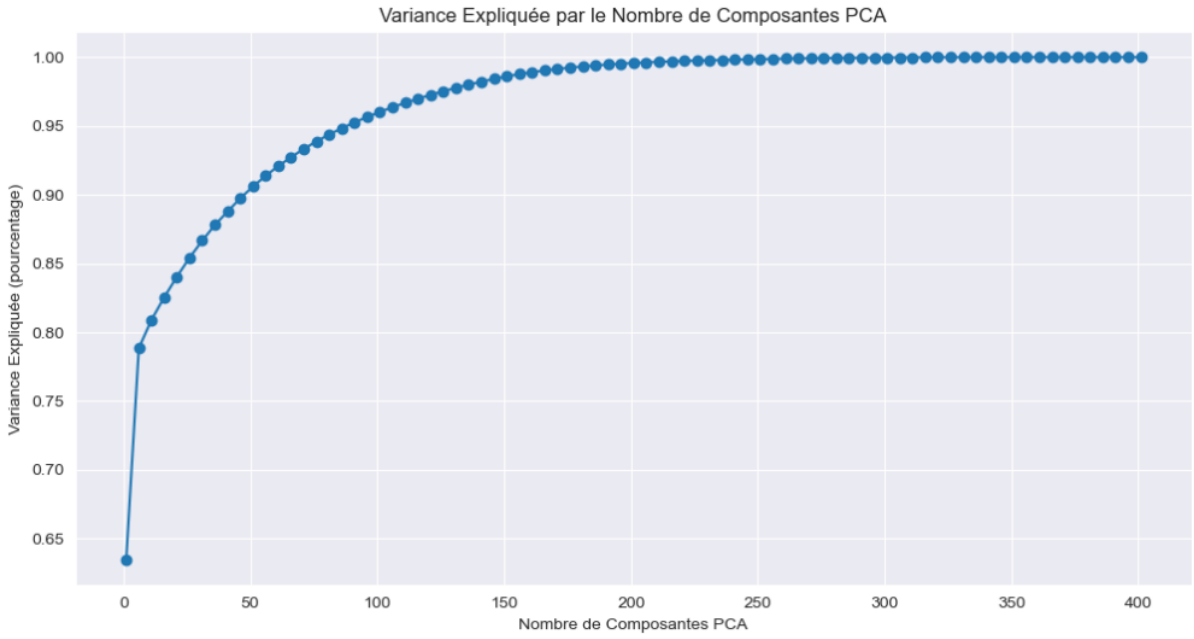

Figure 5: PCA variance

Dernièrement, ces 2 graphiques permettent de comparer la variances des activités électriques cérébrales et musculaires par rapport à leurs états de sommeils. On remarque sur le premier graphique que l'activité électrique cérébrale est très similaire entre les états éveillé et rem. Ce dernier étant souvent associé à des périodes de rêves intenses, cela oblige les régions du cerveau impliquées dans le traitement des informations et la mémoire a travaillé similairement à lorsque la souris est éveillée. A contrario, cette cet état est caractérisé par une activité musculaire presque nulle comme dans l'état non-rem. Nous pouvons donc constater que nous pourrions classifier presque classifier les données avec uniquement ces 2 informations. Cependant, les outliers ont été retiré de ce graphique pour permettre une meilleur visualisation mais ceux-ci complexifient nettement la tâche de classification

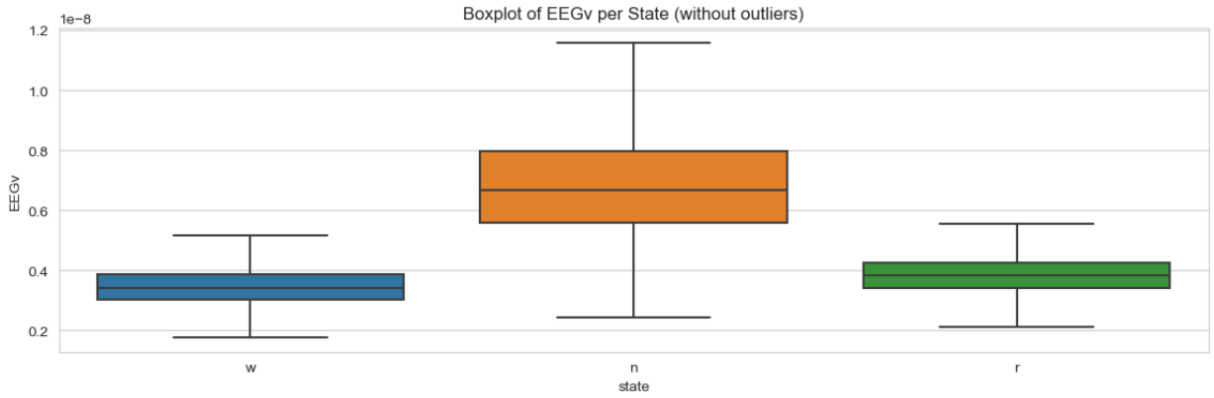

Figure 6: Plot variance EEG

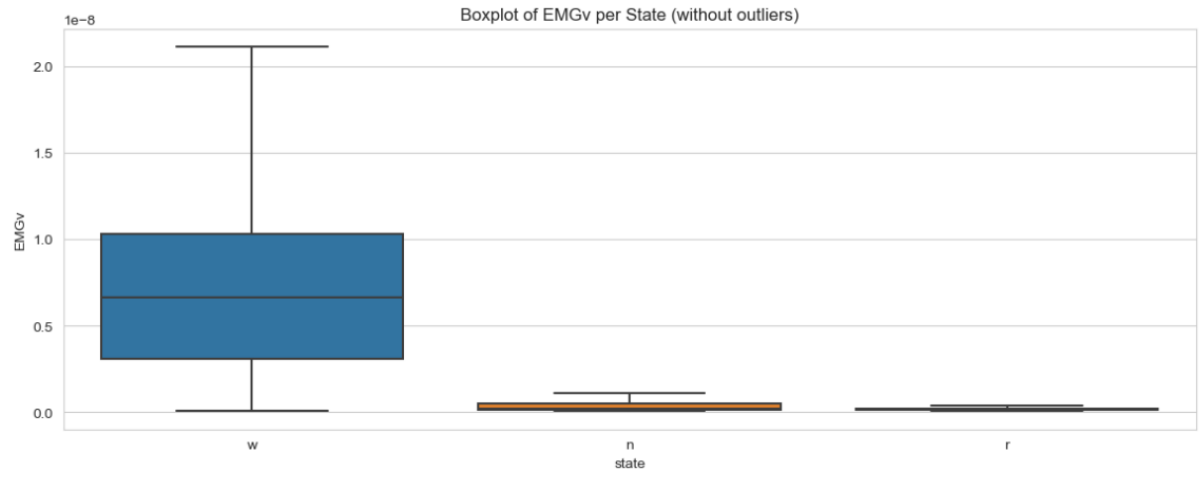

Figure 7: Plot variance EMG

Développement

Prétraitement

Pour le prétraitement des données, la première chose que l'on fait est de garder uniquement le jour 3 puisqu'uniquement celui-ci a été annoté à la main par un expert.

Ensuite, nous supprimons premièrement les colonnes « temp » sur laquelle nous n'avons pratiquement aucune donnée et « rawstate » car elle est l'équivalent de la colonne « state » mais avec plus de détail, il n'est donc pas pertinent dans la conservé dans notre cas. Nous avons pu le constater car en remplaçant simplement les chiffres correspondent à l'état 1->w, 2->n et 3->r dans la colonne « rawstate » nous obtenons exactement la même chose que la colonne « state ».

Nous avons également constaté que nous n'avions pas de valeurs manquantes nous n'avons donc pas eu besoin de traiter ce problème.

Ensuite nous supprimons les outliers avec l'algorithme « IsolationForest ».

Après cela, nous encodons les labels « state » et nous standardisons les colonnes utiles pour l'entrainement (EEGv, EMGv, et bin0-bin400). Comme on a pu le constater dans la visualisation des données, nous appliquons la PCA pour réduire le nombre de composante à 50 pour garder un pourcentage supérieur à 90%.

Exemple d'un notebook type

Voici la structure des 3 notebook comportant les 3 modèles les plus performant (Generalize_3_State_Classifications_LSTM.ipynb, Generalize_REM_NREM_LSTM.ipynb, Generalize_Wake_Sleep_LSTM.ipynb).

Import des fonctions utiles pour le preprocessing, la gestion des dataframes et l'affichage des résultats que nous avons créé :

```
import importlib
spec = importlib.util.spec_from_file_location("preprocessing", "..\\utils\\preprocessing.py")
preprocessing = importlib.util.module_from_spec(spec)
spec.loader.exec_module(preprocessing)

spec = importlib.util.spec_from_file_location("fspliter", "..\\utils\\files_spliter.py")
fspliter = importlib.util.module_from_spec(spec)
spec.loader.exec_module(fspliter)

spec = importlib.util.spec_from_file_location("results", "..\\utils\\results.py")
results = importlib.util.module_from_spec(spec)
spec.loader.exec_module(results)
```

Figure 8: Import for preprocessing

Ensuite, nous avons notre fonction « train_test_validation_split_on_day_3 » qui s'occupe de charger un dataset en fonction du numéro de souris, puis garde uniquement le 3ème jour et supprime les 6 premières heures pour ne pas avoir un dataset trop déséquilibré puisque les souris sont restées éveillées à ce moment. Après cela, nous appelons la fonction « do_preprocessing » qui se charge de nettoyer les valeurs manquantes et gérer les valeurs aberrantes ainsi que transformer les labels en REM/NREM ou Wake/Sleep dépendamment de quel modèle nous souhaitons entrainer. Après cela nous encodons nos labels, sélectionnons les colonnes bins et les 2 colonnes de variance puis nous normalisons les données avant d'appliquer la pca. Finalement la fonction crée des jeu d'entrainement, de test et de validation.

```python
def train_test_validation_split_on_day_3(num_mice):

    # Loading day 3 without first 6 hours
    data = fspliter.get_mice(num_mice)
    day3 = fspliter.retrieve_day(data, 3)
    day3_without_first_6_hours = day3.iloc[5400:]

    # preprocessing and encoding
    data_processed = preprocessing.do_preprocessing(day3_without_first_6_hours)
    data_processed['state_encoded'] = label_encoder.fit_transform(data_processed['state'])

    # Feature selection
    feature_columns = data_processed.columns[1:-2]

    # Scaling
    data_processed[feature_columns] = StandardScaler().fit_transform(data_processed[feature_columns])

    # PCA
    pca = PCA(n_components=50)
    pca = pca.fit_transform(data_processed[feature_columns])

    # Train test val split
    X_train, X_temp, y_train, y_temp = train_test_split(pca, data_processed['state_encoded'], test_size=0.3, shuffle = False, stratify = None)
    X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, shuffle = False, stratify = None)

    return X_train, X_val, X_test, y_train, y_val, y_test
```

Figure 9: Train_test_validation_split

Après cela, nous avons la fonction « create_bidirectional_sequences » qui crée des séquences de n-10 à n+10 observations pour nos modèles récurrent.

```python
def create_bidirectional_sequences(data, n):
    sequences = []
    data_length = len(data)

    for i in range(n, data_length - n):
        seq = data[i - n: i + n + 1]
        sequences.append(seq)

    return np.array(sequences)
```

Figure 10: Séquences bidirectionnel

Une fois cela fait nous pouvons entrainer notre modèle. Pour afficher les résultats, nous utilisons la fonction inverse_transorm pour désencoder nos labels.

Dernièrement, nous avons la fonction « test_model_on_other_mice » qui permet comme son nom l'indique de tester notre modèle sur une autre souris en appelant la fonction « train_test_validation_split_on_day_3 », crée les séquences, prédit le résultat, transforme les labels et affiche le résultat.

```
def test_model_on_other_mice(model, num_mice):
    X_testmouse, _, _, y_testmouse, _, _ = train_test_validation_split_on_day_3(num_mice)

    X_test_sequences = create_bidirectional_sequences(X_testmouse, 10)

    y_test_adjusted = y_testmouse[20:]

    X_test_pred = model.predict(X_test_sequences)
    predicted_labels = X_test_pred.argmax(axis=1)
    y_test_original = label_encoder.inverse_transform(y_test_adjusted)
    y_pred_original = label_encoder.inverse_transform(predicted_labels)
    results.scores(y_test_original, y_pred_original, ('n', 'r', 'w'))
```

Figure 11: test_model_on_other_mice

Outils d'analyses

Pour l'analyses des résultats obtenus nous avons décidé d'utiliser différents indicateurs pour vérifier la qualité de nos modèles.

Dans un premier temps nous imprimons un rapport de classification qui nous fournit des informations détaillées sur les résultats du model tel que :

- Précision (Precision) : Cela correspond au nombre de vrais positifs divisé par la somme des vrais positifs et des faux positifs. Elle mesure la capacité du modèle à ne prédire une classe que lorsqu'elle est réellement présente.
- Rappel (Recall) : Cela correspond au nombre de vrais positifs divisé par la somme des vrais positifs et des faux négatifs. Il mesure la capacité du modèle à identifier toutes les instances positives.
- F-mesure (F1-score) : Cela correspond à la moyenne harmonique de la précision et du rappel, cela nous fournit une mesure équilibrée entre la précision et le rappel.
- Support : Cela correspond au nombre d'occurrences réelles de chaque classe dans l'ensemble de données.
- Exactitude (Accuracy) : Cela correspond au nombre total de prédictions correctes divisé par le nombre total d'échantillons. Cela nous donne un aperçu de la fiabilité de notre model.
- Moyenne/macro (macro avg) : C'est une moyenne dans laquelle toutes les classes contribuent de façon équivalente.
- Moyenne/pondérée (weighted avg) : C'est une moyenne dans laquelle toutes les classes contribuent proportionnellement à leur taille.

Nous utilisons également les matrices de confusion qui sont un très bon moyen de se rendre compte visuellement du résultat de notre model.

Et nous avons également utilisé le coefficient de cohen's Kappa comme indicateur qui nous fournit une information sur la proportion d'accord entre 2 jugements différents soit ici les valeurs que nous connaissons être correct avec les valeurs trouvées par notre model.

Modèles

Support Vector Machine-SVC

Nous avons donc commencé par essayer de catégoriser nos données en 2 catégories de 2 façons différentes. Soit en faisant dans un premier temps une séparation « wake » / « sleep » puis également une séparation « REM » / « NREM ».

Il est également important de préciser que nous n'avons avons dans un premier temps expérimenté l'entrainement et les tests sur les données d'une seule souris avant de voir s'il était possible de s'entrainer sur une souris et de tester sur une autre souris de la même souche afin de savoir si le modèle était capable de généraliser sur plusieurs souris différentes bien que de la même souche.

Pour cette partie la séparation des données en sets de tests et d'entrainements est fait de façon aléatoire, nous avons également vérifié si le fait de ne pas faire cette séparation de façon aléatoire exerçait une influence sur les résultats obtenus et il ne semble pas que ce soit le cas.

« wake » / « sleep »

Pour cette première classification les résultats obtenus sont relativement bons et nous avons donc également essayer de voir si le nombre de composantes pouvaient encore être réduites.

Voici les résultats :

- Normal (Mouse_Wake_Sleep.ipynb)

```
Rapport de classification :
              precision    recall  f1-score   support

           s       0.91      0.86      0.88      1672
           w       0.91      0.94      0.93      2540

    accuracy                           0.91      4212
   macro avg       0.91      0.90      0.91      4212
weighted avg       0.91      0.91      0.91      4212

Coefficient de cohen's kappa : 0.8104095259133632
```

Figure 13: Résultats Mouse_wake_sleep

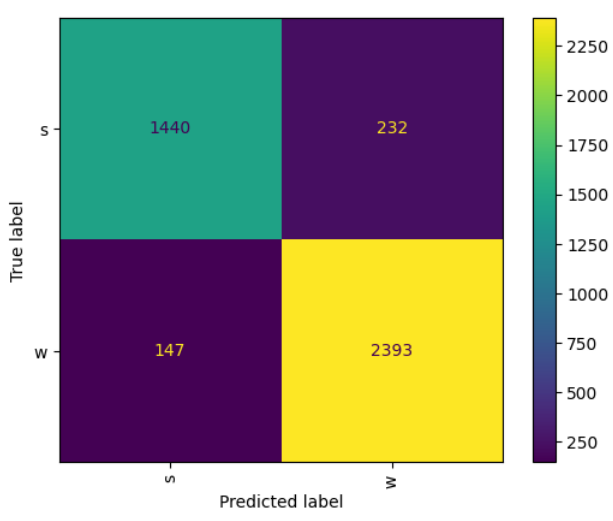

Figure 12: Résultats Mouse_wake_sleep

- PCA 5 composantes (Mouse_Wake_Sleep_pca.ipynb)

```
Rapport de classification :
              precision    recall  f1-score   support

           s       0.87      0.79      0.83      1727
           w       0.86      0.92      0.89      2535

    accuracy                           0.87      4262
   macro avg       0.87      0.86      0.86      4262
weighted avg       0.87      0.87      0.87      4262

Coefficient de cohen's kappa : 0.7220299517901653
```

Figure 15: Résultats Mouse_wake_sleep_pca

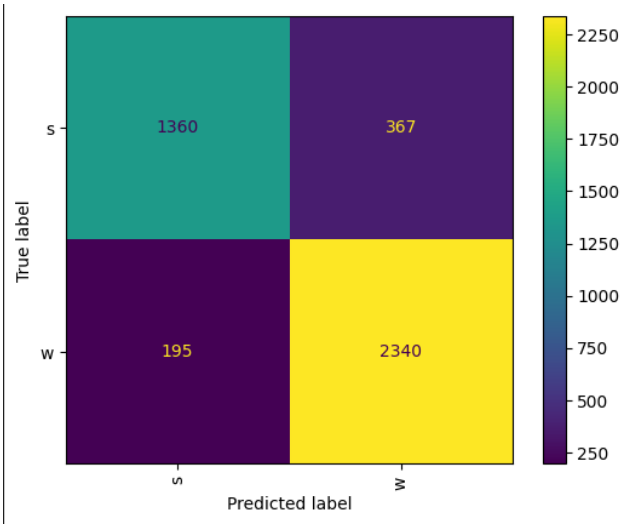

Figure 14: Résultats Mouse_wake_sleep_pca

On constate donc ici que même en descendant à 5 composantes uniquement, on obtient des résultats quasiment similaires pour cette classification en 2 états bien que légèrement plus faible. Cela nous laisse donc penser que seulement quelques-unes de nos features sont responsable de la différence entre ces 2 classes. Nous verrons également que cela semble encore plus vrai pour la classification en « REM » / « NREM » qui elle n'a nécessité que 2 composantes pour obtenir des résultats similaires.

- Entrainement sur 1 souris puis test sur une autre de la même souche (Mouse1_2_Wake_Sleep_pca.ipynb)

```
Rapport de classification :
              precision    recall  f1-score   support

           s       0.90      0.84      0.87      9302
           w       0.88      0.93      0.90     12022

    accuracy                           0.89     21324
   macro avg       0.89      0.88      0.89     21324
weighted avg       0.89      0.89      0.89     21324

Coefficient de cohen's kappa : 0.7724264679250269
```

Figure 17: Résultats Mouse1_2_wake_sleep_pca

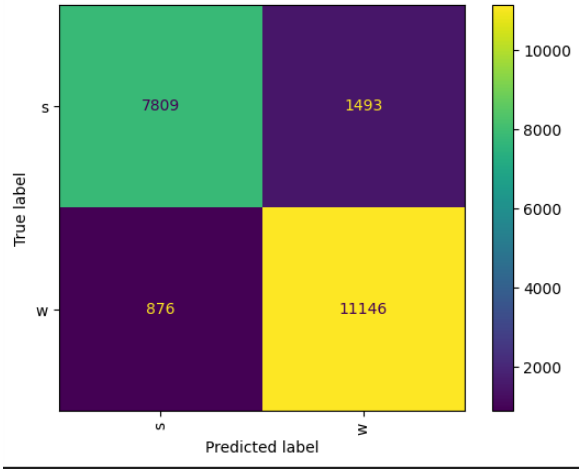

Figure 16: Résultats Mouse1_2_wake_sleep_pca

Les résultats ne sont pas aussi bons que précédemment, ils restent cependant très bon et ils sont même meilleur que ce à quoi on s'attendait dans un premier temps ce qui est une bonne chose car cela veut dire qu'il existe une certaine similarité dans les phases de sommeil de 2 souris différentes.

« REM » / « NREM ».

Pour cette séparation nous obtenons également de très bons résultats que nous pouvons retrouver ci-dessous :

- Normal (Mouse_REM_NREM.ipynb)

```
Rapport de classification :
              precision    recall  f1-score   support

           n       0.95      0.97      0.96      1423
           r       0.83      0.76      0.79       271

    accuracy                           0.94      1694
   macro avg       0.89      0.86      0.88      1694
weighted avg       0.94      0.94      0.94      1694

Coefficient de cohen's kappa : 0.7558703024784974
```

Figure 19: Résultats Mouse_REM_NREM

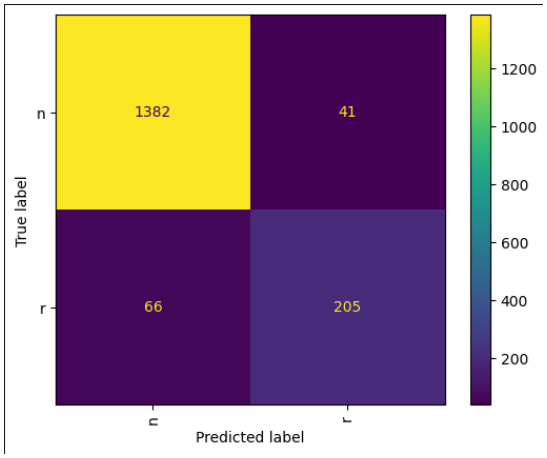

Figure 18: Résultats Mouse_REM_NREM

- PCA 2 composantes (Mouse_REM_NREM_pca.ipynb)

```
Rapport de classification :
              precision    recall  f1-score   support

           n       0.94      0.97      0.96      1382
           r       0.82      0.68      0.75       260

    accuracy                           0.93      1642
   macro avg       0.88      0.83      0.85      1642
weighted avg       0.92      0.93      0.92      1642

Coefficient de cohen's kappa : 0.7036339824043771
```

Figure 21: Résultats Mouse_REM_NREM_pca

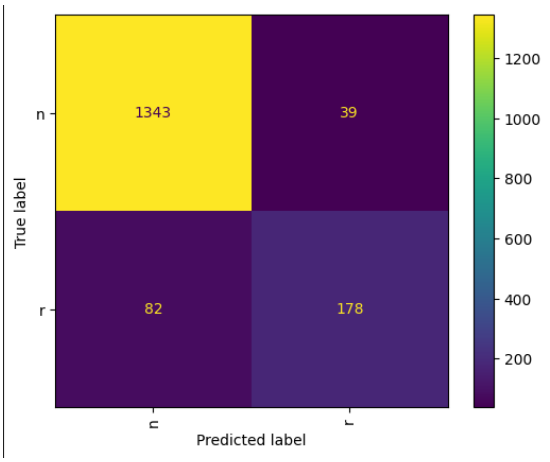

Figure 20: Résultats Mouse_REM_NREM_pca

Comme on le voit les résultats sont toujours très bons et pratiquement identique malgré le faible nombre de composantes comme précisé précédemment.

- Entrainement sur 1 souris puis test sur une autre de la même souche (Mouse1_2_REM_NREM_pca.ipynb)

Rapport de classification :

	precision	recall	f1-score	support
n	0.96	0.96	0.96	7931
r	0.76	0.76	0.76	1251
accuracy			0.93	9182
macro avg	0.86	0.86	0.86	9182
weighted avg	0.93	0.93	0.93	9182

Coefficient de cohen's kappa : 0.7201378518116237

Figure 23: Résultats Mouse1_2_REM_NREM

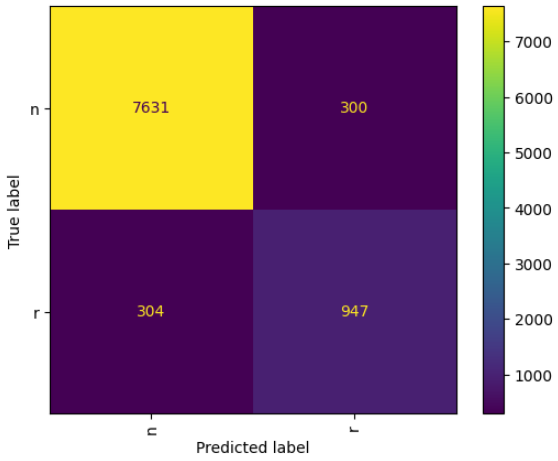

Figure 22: Résultats Mouse1_2_REM_NREM

Comme pour la séparation précédente, on constate que pour celle-ci les résultats sont également très bons lorsqu'on passe d'une souris à une autre.

Catégorisation hiérarchique pour les 3 classes (Mouse_Hierarchical.ipynb)

En se basant sur nos 2 modèles obtenus précédemment, l'objectif était de faire une classification en 2 temps afin d'essayer d'obtenir des résultats adéquats. En données dans un premier temps les données au model pour effectuer une classification en « wake » / « sleep » puis une fois cela fait on passer les données catégorisées comme « sleep » au deuxième model afin qu'il puisse s'occuper de faire la classification en « REM » / « NREM ».

Cependant, cela ne s'est pas avéré si efficace que ce qu'on pouvait penser initialement. Voici les résultats obtenus :

Rapport de classification :

	precision	recall	f1-score	support
n	0.89	0.88	0.89	1369
r	0.70	0.47	0.56	273
w	0.91	0.95	0.93	2530
accuracy			0.90	4172
macro avg	0.83	0.77	0.79	4172
weighted avg	0.89	0.90	0.89	4172

Figure 25: Résultats Mouse_hierarchical

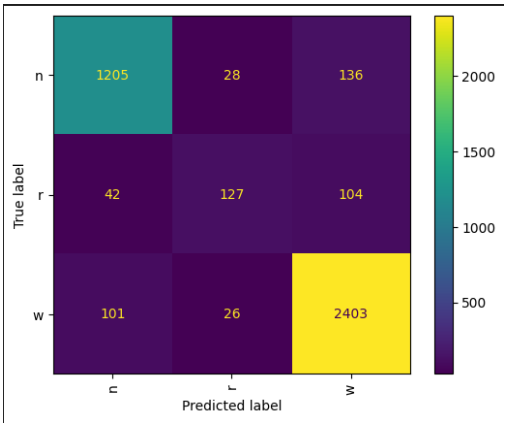

Figure 24: Résultats Mouse_hierarchical

On constate que les résultats obtenus pour les classes « NREM » et « wake » sont relativement bon cependant c'est assez compliqué de bien classé « REM » même si les résultats obtenus ne sont de loin pas proche de l'aléatoire.

Nous pouvons expliquer cela car après analyses nous avons pu constater que les états « REM » et « wake » étaient relativement proche. Le problème que nous avons ici viens donc du fait que lors de la première classification « wake » / « sleep » notre modèle s'est déjà trompé sur une quasi moitié des états « REM » qu'il a classifié comme « wake », cela se confirme bien en regardant la matrice de confusion ci-dessus. Et lors du passage au deuxième model, il va également y avoir quelques erreurs de classifications qui vont nous amener à ce f1-score de 0.56 qui n'est pas très bon. À partir de là il a donc fallu essayer d'autres méthodes qui nous permettraient d'obtenir une meilleure classification.

Catégorisation pour les 3 classes (Mouse_3_State_NonHierarchical.ipynb)

Dans ce test nous avons simplement essayé de prédire les 3 classes sans procédé de façon hiérarchique et les résultats sont les suivants :

Rapport de classification :

	precision	recall	f1-score	support
n	0.89	0.89	0.89	1397
w	0.76	0.32	0.45	263
r	0.90	0.95	0.93	2567
accuracy			0.89	4227
macro avg	0.85	0.72	0.76	4227
weighted avg	0.89	0.89	0.88	4227

Coefficient de cohen's kappa : 0.7858891543322896

Figure 27: Résultats Mouse_3_state_nonhierarchical

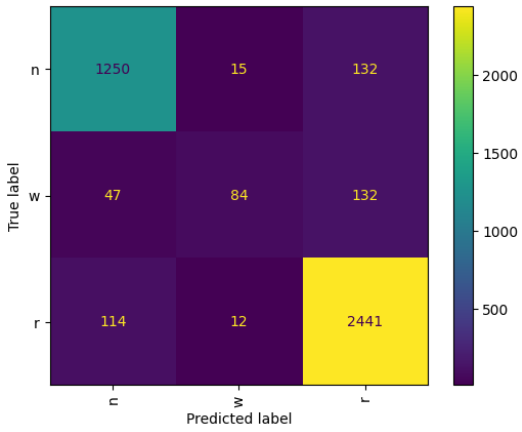

Figure 26: Résultats Mouse_3_state_nonhierarchical

Cela est intéressant car ça nous montre que l'approche hiérarchique est légèrement meilleure qu'une simple approche avec un model qui essaye de prédire les 3 classes par lui-même. Les résultats ne sont pas beaucoup moins bons que précédemment mais il existe quand même une différence.

Multilayer Perceptron

Premièrement, nous avons essayé de faire un réseau de neurones pour tester l'efficacité d'une telle architecture. Pour ce modèle et tous les suivants, nous utiliserons « class_weight » pour pondérer chaque classe lors de l'entrainement.

```
class_weights = class_weight.compute_class_weight(class_weight = "balanced", classes= np.unique(y_train_adjusted), y= y_train_adjusted)
```

Figure 28: class_weights

En regardant les résultats des poids du model on peut constater que la classe REM à un poids bien plus haut car bien moi représenté. Avec 0 = Non REM, 1 = REM, 2 = réveillé.

{0: 0.6772082878953108, 1: 4.535856573705179, 2: 0.7675280898876404}

Après avoir effectué quelques expériences, nous avons obtenus un f1-score d'environ 88% comme meilleur résultat pour prédire les 3 classes.

Multilayer Perceptron with memory

Les réseaux de neurones récurrents permettent de capturer des motifs dans les séquences de données comme dans les séries temporelles. Ces réseaux de neurones sont particulièrement importants lorsque la séquence des données est importante. Dans notre cas, si on sait que la souris dormait 30 secondes avant et 30 secondes après un moment t, il est fortement probable que la souris dort encore au moment t.

Les réseaux de neurones récurrents fonctionnent de cette manière. Contrairement aux réseaux de neurones traditionnels qui traitent chaque entrée indépendamment, les RNN ont une sorte de « mémoire » leurs permettant de prendre en compte les précédentes informations.

Le principal problème d'un réseau de neurones récurrent et le problème de gradient qui disparaît. Pour cela, les 2 architectures ci-dessous permettent d'atténuer ce problème.

LSTM (Generalize_3_State_Classifications_LSTM.ipynb)

LSTM (Long Short Term Memory) est un réseau de neurones récurrent très performant pour capturer des dépendances à long termes dans les données, le rendant très bon dans des tâches de prédictions de séries temporelles.

Dans notre cas, nous avons décidé d'utiliser une séquence de 10 valeurs bi-directionnellement.

Ce qui signifie que pour chaque valeur au moment t, nous disposons de ces 10 valeurs précédentes et 10 valeurs suivantes. Voici l'architecture de notre modèle :

```
model = Sequential([
    Bidirectional(LSTM(100, return_sequences=True, input_shape=(sequence_size, X_train.shape[1]))),
    Dropout(0.3),
    Bidirectional(LSTM(100)),
    Dropout(0.3),
    Dense(50, activation='relu'),
    Dense(3, activation='softmax')
])
```

Figure 29: Model LSTM

Et son résumé :

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 bidirectional (Bidirectiona  (None, 21, 200)          120800
 l)

 dropout (Dropout)           (None, 21, 200)           0

 bidirectional_1 (Bidirectio  (None, 200)              240800
 nal)

 dropout_1 (Dropout)         (None, 200)               0

 dense (Dense)               (None, 50)                10050

 dense_1 (Dense)             (None, 3)                 153

=================================================================
Total params: 371,803
Trainable params: 371,803
Non-trainable params: 0
_________________________________________________________________
```

Figure 30: Model LSTM résumé

Ce modèle (3 classes) est relativement très performant puisque nous obtenons un f1-score de 0.95 et un Coefficient de cohen's kappa de 0.914 lorsque nous testons sur la même souris. Cependant, il n'est plus du tout performant si nous testons le modèle sur une souris sur laquelle il ne s'est pas entrainé mais cela sera plus approfondi dans le chapitre « Généralisation des modèles aux souches de souris ».

Rapport de classification :

	precision	recall	f1-score	support
n	0.95	0.97	0.96	1216
r	0.90	0.91	0.90	162
w	0.97	0.94	0.95	980
accuracy			0.95	2358
macro avg	0.94	0.94	0.94	2358
weighted avg	0.95	0.95	0.95	2358

Coefficient de cohen's kappa : 0.9137101208785017

Matrice de confusion :

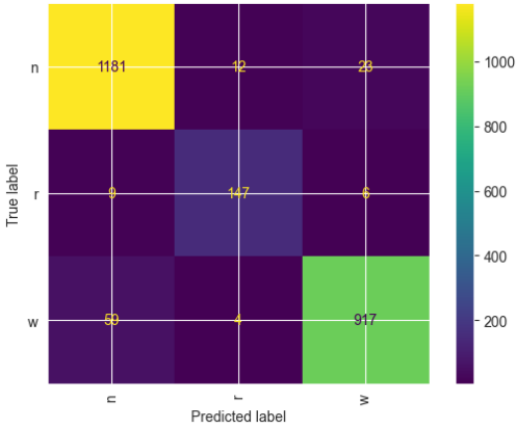

Figure 31: Résultats LSTM_3_state

Pour le modèle prédisant si la souris est éveillée ou endormie, le modèle a des résultats similaires au précédent :

	precision	recall	f1-score	support
W	0.96	0.97	0.97	1369
S	0.96	0.94	0.95	1014
accuracy			0.96	2383
macro avg	0.96	0.96	0.96	2383
weighted avg	0.96	0.96	0.96	2383

Coefficient de cohen's kappa : 0.9173681841791397

Figure 32: Résultats LSTM_wake_sleep

Dernièrement, le modèle prédisant simplement entre l'état rem et nrem obtient légèrement de moins bons résultats. Cela peut s'expliquer par le fait que la classe rem est très peu représentée et que même en utilisant class_weight lors de l'entrainement, cela ne suffit pas.

	precision	recall	f1-score	support
N	0.99	0.93	0.96	1197
R	0.64	0.94	0.76	149
accuracy			0.93	1346
macro avg	0.81	0.94	0.86	1346
weighted avg	0.95	0.93	0.94	1346

Coefficient de cohen's kappa : 0.7221291816085769

Figure 33: Résultats LSTM_rem_nrem

GRU (Generalize_3_State_Classifications_GRU.ipynb)

GRU (Gated recurrent unit) est, pour faire simple, une version simplifiée et moins complexe de LSTM.

Nous avons testé ce modèle uniquement sur la prédiction des 3 états avec la même architecture que LSTM et nous n'avons pas remarqué de différence en termes de temps d'entrainement et celui-ci semble être un peu moins performant que LSTM. Voici le meilleur résultat obtenu :

Rapport de classification :

	precision	recall	f1-score	support
n	0.94	0.97	0.96	1208
r	0.94	0.75	0.83	164
w	0.94	0.94	0.94	986
accuracy			0.94	2358
macro avg	0.94	0.89	0.91	2358
weighted avg	0.94	0.94	0.94	2358

Coefficient de cohen's kappa : 0.8969813513201575

Matrice de confusion :

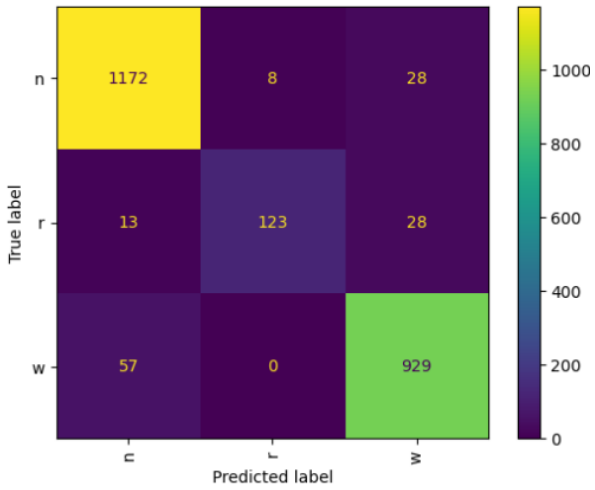

Figure 34: Résultats GRU

Généralisation des modèles aux souches de souris

La généralisation des modèles aux souches de souris est une tâche qui semblaient très facile mais qui s'est avérée bien plus compliqué que cela ne laissait penser.

Premièrement, nous avons dû réfléchir à comment s'y prendre pour entrainer un modèle sur plusieurs souris. Nous avions d'abord testé de concaténer plusieurs jeux de données de souris et d'entrainer le modèle. Les données n'étant pas mélangées pour gardées la séquence temporelle, le modèle apprenait principalement que sur la première souris. Nous avons donc modifié le code pour créer des « blocs » de données. Par exemple, on prenait les 100 premières lignes d'une souris, les 100 premières d'une autres, les 100 suivantes de la première souris et ainsi de suite.

Cependant, cela fonctionnait mais brisait la séquentialité des données puisque nous pouvions passer d'une souris qui était endormie à une sourie réveillée.

Nous avons donc finalement opté pour une méthode pas très automatisée mais qui fonctionne. Il s'agit simplement de charger le jeu de données d'une souris, d'entrainer le modèle, de charger un autre jeu de donnée et d'entrainer à nouveau le modèle.

Entrainement sur 1 souris (Generalize_3_State_Classifications_LSTM.ipynb)

Premièrement, nous avons testé d'entrainer le modèle LSTM car celui-ci est notre meilleur modèle sur une seule souris de la souche et de tester sur les 7 autres souris. Avec cela, nous obtenons des résultats assez variables mais obtenons un f1-score macro d'environ 0.75. On peut voir ci-dessous un résultat concluant de l'un de ces tests :

```
Rapport de classification :
              precision    recall  f1-score   support

           n       0.86      0.93      0.89      5508
           r       0.56      0.79      0.65       800
           w       0.96      0.80      0.87      4816

    accuracy                           0.86     11124
   macro avg       0.79      0.84      0.81     11124
weighted avg       0.88      0.86      0.87     11124

Coefficient de cohen's kappa : 0.7612720157088367
```

Figure 35: Résultats LSTM_3_state_on_bread

Cependant sur 2 souris de la souche et plus particulièrement 1, le modèle n'arrive pas du tout à détecter le classe rem. On peut voir ci-dessous que la classe rem obtient un f1-score de seulement 0.12. Cela pourrait être dû à la répartition des classes mais ce qui est étrange c'est que cette répartition est presque la même que sur les autres souris comme sur le résultat précédent. Il pourrait donc s'agir de signaux rem spécifique à cette souris et que le modèle n'arrive donc pas à prédire cette classe. On remarque sur la matrice de confusion que le modèle à tendance à prédire la classe wake lorsqu'il s'agit de la classe rem très fréquemment.

Rapport de classification :

	precision	recall	f1-score	support
n	0.86	0.95	0.90	4576
r	0.12	0.13	0.12	766
w	0.89	0.80	0.85	5765
accuracy			0.82	11107
macro avg	0.62	0.63	0.62	11107
weighted avg	0.82	0.82	0.82	11107

Coefficient de cohen's kappa : 0.6763444073481908

Figure 37: Résultats LSTM_3_state_on_bread

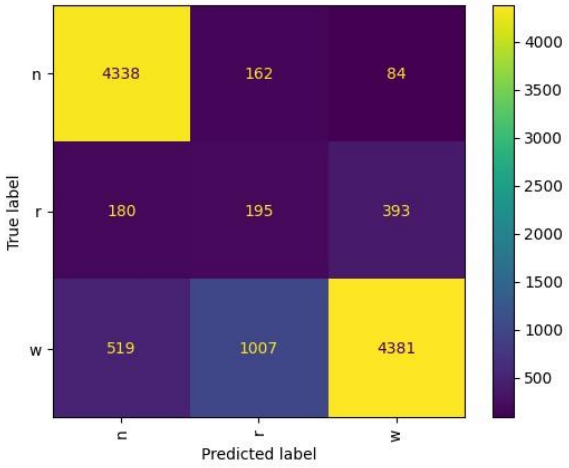

Figure 36: Résultats LSTM_3_state_on_bread

L'entrainement sur une seule souris pour prédire la souche est donc évidemment moins performant que de tester sur la même souris mais le résultat est quand même acceptable. Cependant comme vu ci-dessus, dans certains cas, le modèle n'arrive pas du tout à prédire la classe Rem.

Entrainement sur la moitié des souris de la souche (Generalize_3_State_Classifications_LSTM.ipynb)

Ensuite, nous avons tenté de généraliser notre modèle en l'entrainant sur la moitié de la souche. Cela nous permettra de constater si le modèle a une faculté de généralisation et pourra ainsi mieux prédire les autres souris de la souche. Comme précédemment, nous utilisons une souche de 8 souris et nous entrainons notre modèle sur les 4 premières souris. Voici les résultats obtenus sur les 4 dernières souris :

Rapport de classification :

	precision	recall	f1-score	support
n	0.93	0.91	0.92	4931
r	0.54	0.62	0.57	847
w	0.93	0.92	0.93	5325
accuracy			0.89	11103
macro avg	0.80	0.82	0.81	11103
weighted avg	0.90	0.89	0.90	11103

Coefficient de cohen's kappa : 0.8162157783621653

Rapport de classification :

	precision	recall	f1-score	support
n	0.88	0.96	0.92	4716
r	0.84	0.62	0.71	718
w	0.94	0.90	0.92	5296
accuracy			0.91	10730
macro avg	0.89	0.83	0.85	10730
weighted avg	0.91	0.91	0.91	10730

Coefficient de cohen's kappa : 0.8341526437471903

Rapport de classification :

	precision	recall	f1-score	support
n	0.91	0.97	0.94	5563
r	0.83	0.75	0.79	988
w	0.97	0.90	0.94	4437
accuracy			0.93	10988
macro avg	0.90	0.88	0.89	10988
weighted avg	0.93	0.93	0.93	10988

Coefficient de cohen's kappa : 0.8689934017744924

Rapport de classification :

	precision	recall	f1-score	support
n	0.91	0.96	0.93	5073
r	0.76	0.68	0.71	994
w	0.94	0.90	0.92	4950
accuracy			0.91	11017
macro avg	0.87	0.85	0.85	11017
weighted avg	0.91	0.91	0.91	11017

Coefficient de cohen's kappa : 0.8397043252794998

Figure 38: Résultats LSTM_3_state_on_bread

Comme d'habitude, la classe rem est moins bien prédit que les 2 autres mais l'amélioration par rapport à l'entrainement sur une seule souris est notable. Cette fois-ci, il n'y a pas de cas où la classe rem n'est pas bien classifié du tout et le f1-score macro passe environ de 0.75 à 0.85 en moyenne. Cela démontre donc que notre modèle est capable d'apprendre plusieurs activités électriques cérébrales et de généraliser sur d'autres souris. Cela est très encourageant car le but n'est évidemment pas de devoir entrainer un modèle sur une souris avant de vouloir effectuer une prédiction.

De plus, nous avons utilisé ce modèle pour tester de prédire les classes de souris de souches aléatoires pour tester notre réseau de neurones LSTM dans des cas encore plus complexes. Cette fois-ci, celui-ci fonctionne à nouveau un petit peu plus aléatoirement avec de très bons résultats par moment et de très mauvais (dû à la classe Rem) dans d'autres cas.

Entrainement sur toute la souche (Generalize_3_State_Classifications_LSTM.ipynb)

L'entrainement de notre modèle sur toute une souche de souris fourni des résultats un peu biaisés puisque nous testons notre modèle sur des souris sur lesquelles il s'est entrainé contrairement au 2 cas précédent. De plus, comme expliqué précédemment, le but n'est pas de devoir entrainer notre modèle sur toute les souris mais plutôt sur quelques-unes d'entre elles afin que le modèle puisse ensuite généraliser sur des souris qu'il n'aurait jamais rencontrée.

Nous avons quand même tenté l'expérience et les résultats sont comme attendus relativement bons. Comme on peut le constater ci-dessous, les résultats sont meilleurs que lors de l'entrainement avec 1 ou la moitié des souris mais un peu moins performant que lorsqu'on entrainait et testait notre modèle sur une seule et même souris.

```
Rapport de classification :
              precision    recall  f1-score   support

           n       0.92      0.94      0.93      5383
           r       0.69      0.80      0.74       780
           w       0.96      0.91      0.93      4767

    accuracy                           0.92     10930
   macro avg       0.86      0.88      0.87     10930
weighted avg       0.92      0.92      0.92     10930

Coefficient de cohen's kappa : 0.8549033213448611
```

Figure 39: Résultats LSTM_3_state_on_bread

Notre modèle final sera donc celui entrainé sur la moitié des souris de la souche afin de pouvoir le tester sur les autres souris.

Généralisation des modèles pour toutes les souris

Etant donné la taille des jeux de données, la puissance de nos machines ainsi que ce que nous avions prévu dans le cahier des charges, nous n'avons pas essayé de généraliser le modèle pour toutes les souris. Cela aurait pris beaucoup de temps et de mémoire. Il pourrait cependant être intéressant dans une continuité future du projet d'essayer de réaliser cette partie afin d'essayer de savoir jusqu'à quel point il est possible de généraliser le modèle.

Cependant, comme vu dans le chapitre précédent, la généralisation du modèle semble possible lorsque nous l'entrainons sur plusieurs souris différentes. Dans quelques cas, la prédiction est altérée par la classe rem qui n'est pas prédite correctement.

Pour entrainer un modèle complet capable de généraliser au maximum l'activité cérébrale des souris, il pourrait donc être intéressant d'entrainer notre modèle sur 1 ou 2 souris d'un maximum de souche.

Conclusion

En résumé, les analyses effectuées dans le cadre de ce projet démontrent la possibilité de prédire les états de sommeils des souris de manière très efficace avec une possibilité de généralisation des modèles aux souches voir à toutes les souris de notre jeu de données.

Notre objectif était de réaliser 3 modèles différents permettant de classifier les états de 3 manière différentes (Rem/Nrem, Wake/Sleep, et Rem/Nrem/Wake) et cela a été réalisé avec succès.

Pour cela, nous avons réalisés plusieurs modèles de machine learning se basant sur des architectures différentes. Nous avons par exemple testé des modèle de type SVC, réseaux de neurones simples et des réseaux de neurones récurrents comme les LSTM et GRU. Pour chacun de ces modèle nous avons testé plusieurs hyperparamètres afin de trouver les meilleurs modèles possibles. Pour optimiser le code, nous avons implémenté des fonctions de traitement des jeux de données, de prétraitement des données ainsi que pour l'affichage des résultats

Le meilleur modèle que nous avons développé est un modèle LSTM permettant de classifier les 3 états avec un score d'environ :

- 0.95 lorsque nous entrainons et testons le modèle sur la même souris
- 0.75 lorsque nous entrainons le modèle sur une souris de la souche et testons sur d'autres souris de la même souche
- 0.85 lorsque nous entrainons le modèle sur la moitié des souris de la souche et testons sur l'autre moitié.

En outre, la méthodologie adoptée, combinant l'agilité et des outils de gestion de projet comme GitHub, nous a permis de collaborer efficacement et de nous adapter aux divers défis rencontrés. La réussite de ce projet ouvre des perspectives pour des recherches futures, notamment en ce qui concerne la compréhension du sommeil chez l'homme.

Sources

https://towardsdatascience.com/understanding-gru-networks-2ef37df6c9be

https://datascientest.com/long-short-term-memory-tout-savoir#:~:text=Les%20LSTM%20introduisent%20une%20nouvelle,informations%20sur%20une%20p%C3%A9riode%20%C3%A9tendue.&text=La%20cellule%20m%C3%A9moire%20d'un,et%20une%20porte%20d'oubli.

https://www.analyticsvidhya.com/blog/2021/03/introduction-to-long-short-term-memory-lstm/

https://www.hug.ch/laboratoire-du-sommeil/structure-du-sommeil

https://www.ncbi.nlm.nih.gov/pmc/articles/PMC3228710/

https://www.sciencedirect.com/science/article/abs/pii/S0168010221001735

https://keras.io/guides/

https://scikit-learn.org/stable/index.html

Annexes

- Notebooks (code source)
- Modèles
- Marche à suivre
- Cahier des charges
- Planification et journal de travail