Badreau Marie
Belnou Alexandre

Crack the neural code of the brain

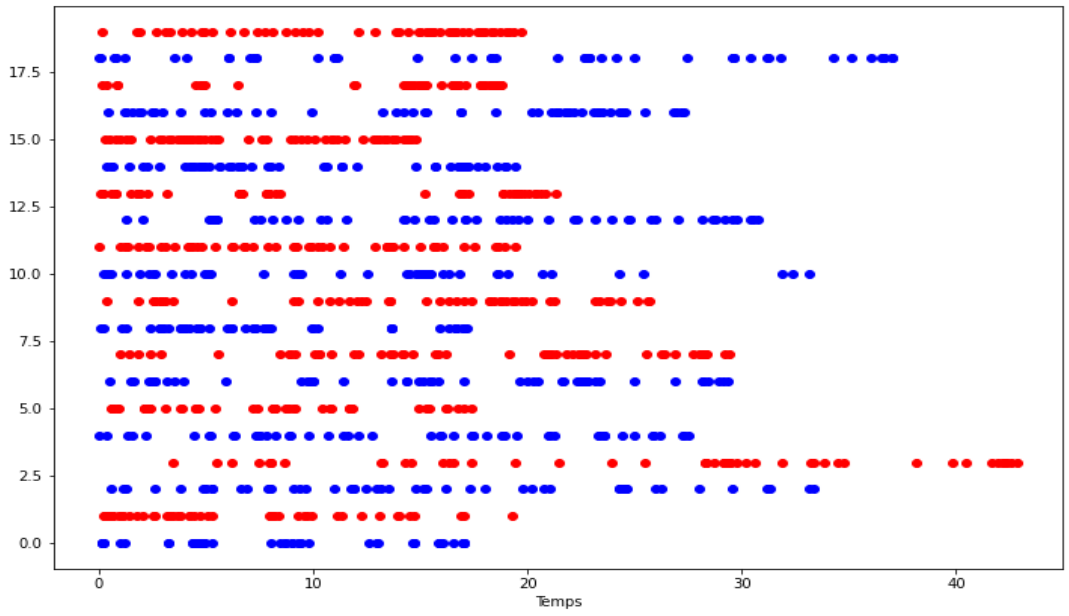
Présentation du challenge :

Description des données :

L'objectif de ce data challenge est de classifier en deux groupes une population de rats à partir de leur activité cérébrale. Pour cela nous disposons pour chacun des 28604 individus (16635 dans l'échantillon train et 11969 pour l'échantillon test) d'une série temporelle de 50 valeurs, une valeur correspondant au temps d'apparition d'un pic dans l'activité neuronal. Nous connaissons également l'identifiant du neurone sur lequel a été mesuré cette série. Cette variable ne sera cependant pas utilisée par la suite car la base de données d'entraînement et la base de données test ne comporte qu'un seul neuron_id en commun, présent dans des proportions différentes. La métrique utilisée dans ce challenge est le score Kappa de Cohen qui mesure le taux d'accord entre les valeurs prédites et les données réels.

Analyse des données :

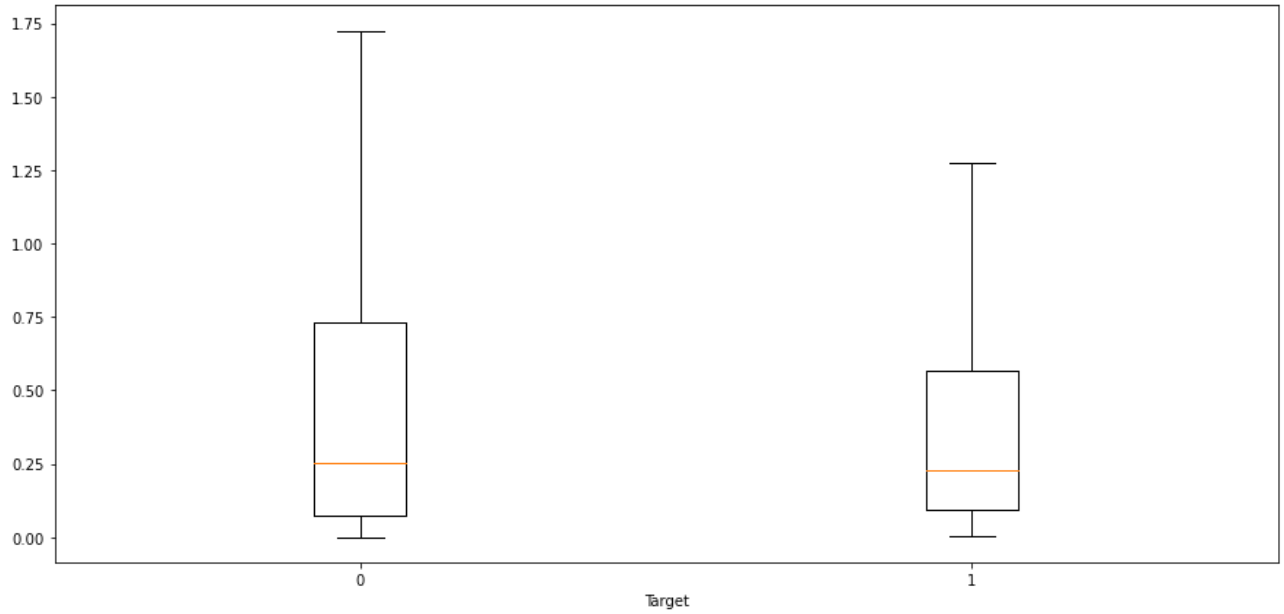
Dans un premier temps, nous constatons que nos données ne sont pas équilibrées. Effectivement 82 % des individus appartiennent au groupe 0.
Afin de déterminer s'il existe des différences significatives dans les temps d'apparition des pics d'activité cérébrale en fonction de la classe d'appartenance des individus, nous commençons par représenter graphiquement ces séries avec en bleu les individus classés 1 et en rouge les individus classés 0.



Avec ici 10 observations du groupe 1 et 10 observations du groupe 0, il semble difficile d'établir des différences notables dans les séries qui permettraient la classification.

Prétraitement des données :

Nous effectuons une différenciation à l'ordre 1 sur notre série temporelle pour avoir pour chacun des individus les temps entre l'apparition de deux pics successifs. Au sein de chacune des deux classes, nous réalisons un boxplot afin d'étudier la répartition de nos données.



Ce boxplot nous montre que les individus appartenant au groupe 0 vont avoir tendance à avoir des temps inter-pics plus élevés que ceux du groupe 1.
Nous créons donc un dataframe contenant pour chacun des individus la valeur de ses quantiles d'ordres 5 %, 15 %, 25 %,50 %,75 %,85 %,95 %, ainsi que la valeur minimale, moyenne, maximale et l'écart type de la série différenciée.

ID	Min	Mean	Max	Std	Qi_0.05	Qi_0.15	Qi_0.25	Qi_0.5	Qi_0.75	Qi_0.85	Qi_0.95
0	0.004133	0.341128	2.838052	0.654644	0.005319	0.007860	0.014097	0.065681	0.328098	0.460422	1.856335
1	0.008825	0.402474	6.492402	0.944283	0.012355	0.025339	0.053396	0.186008	0.365118	0.490178	1.183569
2	0.014021	0.667219	3.185310	0.666855	0.027036	0.063152	0.165435	0.359982	1.090869	1.325181	1.765076
3	0.004493	0.345192	1.077179	0.299719	0.008717	0.015407	0.099267	0.300643	0.515438	0.705640	0.926319
4	0.012688	0.385798	2.653620	0.571187	0.017000	0.041005	0.078566	0.192945	0.339710	0.737590	1.729201

Nous comparons également ces valeurs à celles des quantiles de chacun des deux groupes, en comptant pour chaque entrée le nombre de valeurs supérieures aux quantiles d'ordre 10 %, 25 %, 50 %, 75 %, 90 %.

ID	Q0_0.1	Q1_0.1	Q0_0.25	Q1_0.25	Q0_0.75	Q1_0.75	Q0_0.9	Q1_0.9
0	35	29	23	21	7	7	4	4
1	45	40	34	30	6	7	2	2
2	47	47	41	40	18	22	6	11
3	41	40	39	38	8	12	0	0
4	46	43	38	35	8	8	3	3

Nous ajoutons également les statistiques descriptives pour des différenciations d'ordre 2, 5 et 10.

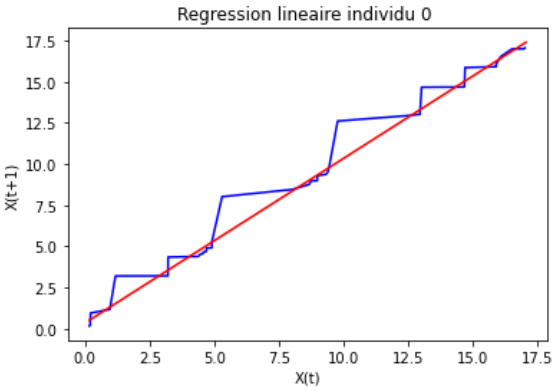
Autres variables introduites :

Le rapport nombre de saut en fonction du temps est également un bon moyen de caractériser ce type de série temporelle. Plus le rapport est élevé ,plus on va avoir de pic sur une courte période de temps. Au contraire, plus ce rapport est faible, plus on va avoir de longues périodes avec une faible activité cérébrale.
Une fois ces rapports obtenus, nous calculons les statistiques descriptives de chacune des nouvelles séries formées afin d'avoir le dataframe ci-dessous (à gauche). Nous déterminons également le nombre moyen, le nombre maximum et l'écart-type du nombre de saut pour une unité de temps (tableau de droite)

ID	freq_min	freq_q1	freq_mean	freq_q3	freq_max	freq_std
0	0.352354	2.241230	7.623908	3.979555	241.971649	19.423931
1	0.154026	2.111140	4.511283	4.645994	113.319481	6.686931
2	0.313941	1.387725	2.021894	1.802788	71.321277	3.627910
3	0.928350	2.737550	4.704633	4.330145	222.564462	11.388472
4	0.376844	2.354079	4.318199	4.214654	78.811648	5.414014

ID	Mean_ut	Max_ut	Std_ut
0	14.255814	37.0	10.186843
1	14.325581	39.0	10.075808
2	18.581395	39.0	10.914536
3	14.604651	40.0	10.483962
4	16.720930	39.0	10.901943

Les dernières variables que nous introduisons correspondent aux coefficients et au score issu de la régression linéaire de X(t+1) en fonction de X(t). Cela permet d'établir s'il existe ou non une relation de linéarité entre deux temps de saut successifs.



ID	c1	c0	LR_score
0	0.998964	0.352888	0.985349
1	1.025497	0.241984	0.977476
2	1.007058	0.559692	0.994857
3	0.981200	0.526766	0.995588
4	1.048270	0.097405	0.990297

Nous joignons l'ensemble des dataframes créés durant cette étape de prétraitement des données, ce qui nous donne deux matrices, xtrain de taille (16635 x 49) et xtest de taille (11969 x 49). Nous standardisons ensuite ces deux datatframes à l'aide de la fonction StandardScaler du module preprocessing de sklearn.

Les meilleurs modèles :

Dans cette partie du rapport nous présenterons les meilleurs modèles utilisés, l'ensemble des autres méthodes sont présentes dans le jupyter notebook.
Afin d'éviter le surapprentissage, nous séparons notre jeu de données d'entraînement en deux parties, la partie x_train comportant 80 % des individus sur laquelle nous allons optimiser nos modèles et la partie x_test sur laquelle nous testerons nos classifiers. Dans chacun de ces deux sous ensembles nous gardons le même rapport d'individu de classe 0 et 1 que dans l'ensemble de l'échantillon d'entraînement.

Random Forest

Le meilleur score à été obtenu à l'aide de la fonction RandomForestClassifier du module ensemble de sklearn.

Ce modèle va construire un ensemble d'arbres de décision basé sur des échantillons bootstrap issue des données d'entraînement. Pour chacun des sous-arbres, le classifier va tirer au sort un ensemble de taille égale à la racine de p (soit 7 variables parmi les 49 possibles) et séparer les données selon les variables qui discriminent au mieux les deux groupes, par un critère choisi en paramètres.

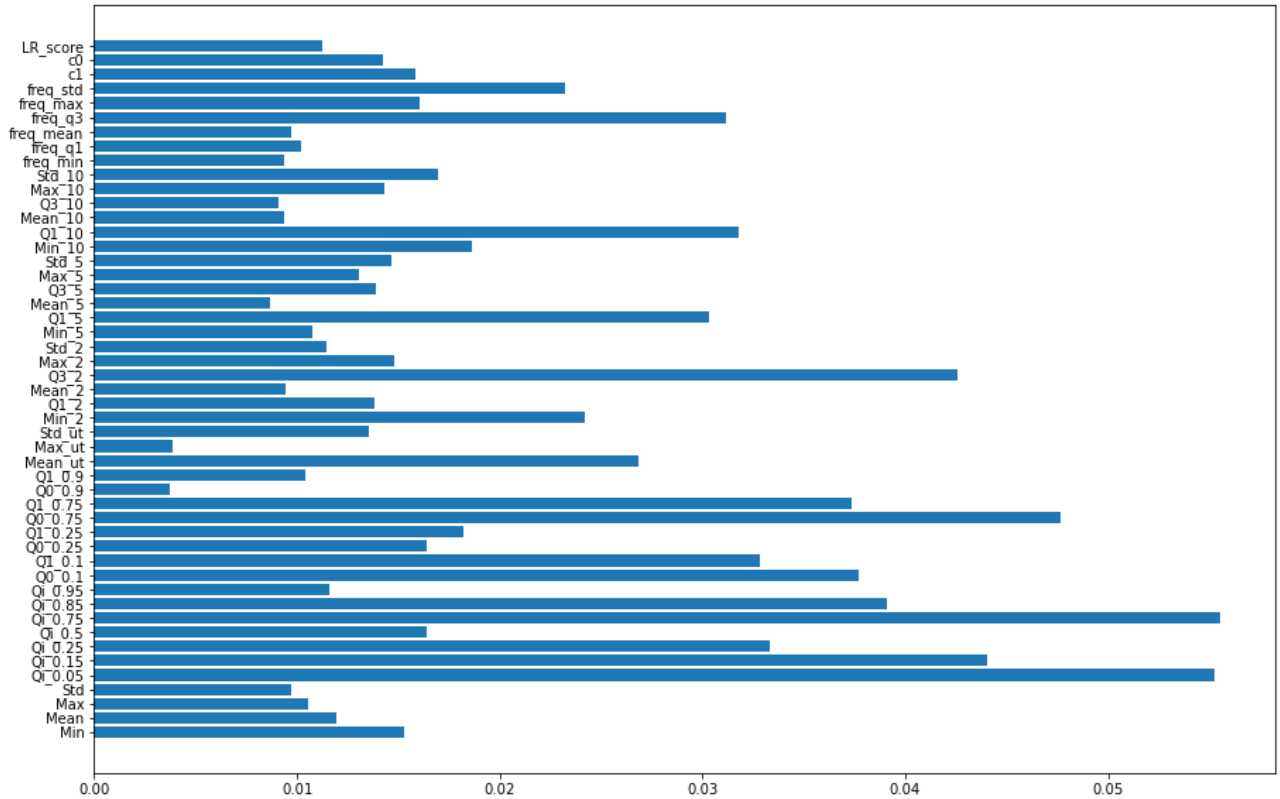
Afin d'optimiser notre classifier, nous cherchons par validation croisée sur x_train avec 5 folds les paramètres qui fournissent le meilleur score kappa de Cohen. Nous introduisons également un poids supérieur aux individus du groupe afin de prendre en compte le fait que nos données soient déséquilibrées.

Nous obtenons les paramètres suivants : n_estimators = 150, max_depth = 8, min_samples_split = 2, criterion='gini', class_weight={0:1,1:3}

n_estimators correspond au nombre d'arbre construit par le modèle. La profondeur maximale des arbres est donnée par *max_depth*. Chaque division de nœud est possible lorsqu'il y a au minimum 2 observations dans le nœud (*min_samples_split)*, par minimisation du critère d'hétérogénéité *gini*. Le dernier paramètre permet de prendre en compte le déséquilibre entre les deux classes en donnant un poids plus élevé (3) à la classe 1.

Le score kappa de Cohen sur x_test est de 0.342.

Il également possible d'étudier l'importance de chacune de nos variables dans la discrimination des classes.



La classification des individus s'effectue donc principalement grâce à l'étude des valeurs les plus grandes et les plus faibles de la série. En revanche l'étude du temps moyen ne semble pas fournir d'informations intéressantes.

Extremely Randomized Trees

La seconde méthode ayant permis d'obtenir de bons résultats est la fonction ExtraTreesClassifier également présente dans le module ensemble de sklearn. Comme pour les randoms forests, les extremely randomized trees utilisent un sous-ensemble de variables sélectionnées aléatoirement pour séparer les individus au sein de chaque nœud. Une fois ce sous ensemble sélectionné, le seuil de discrimination est lui aussi tiré aléatoirement pour chacune des variables. Les observations du nœud seront alors réparties en fonction du critère qui discrimine le mieux les données. Contrairement à la fonction Random Forest Classifier, Extra Trees Classifier n'utilise pas d'échantillon bootstrap par défaut et construit ses arbres de décision sur l'ensemble du jeu de données.

La paramétrisation ayant permis d'obtenir le meilleur score sur Xtest (0,3247) est : n_estimators = 50, max_depth = 12, min_samples_split = 10, criterion='gini', class_weight={0:1,1:3}

Nous avons également essayé de classifier les individus avec des méthodes utilisant le boosting à savoir Adaboost et XGBoost, mais ces dernières ne fournissent pas de bons scores. Nous ne les avons donc pas utilisées pour estimer y_test.

Réseau de neurones :

Nous avons également testé un réseau de neurones à une couche cachée, avec une fonction d'activation sigmoïde pour la couche de sortie et un dropout à 0.5 pour désactiver 50 % des neurones de manière aléatoire. La métrique utilisée pour la compilation du modèle est le Cohen Kappa score.

Comme nous obtenons en sortie la probabilité d'appartenir à la classe 1, la fonction de décision pour la classification sera de la forme suivante : si la probabilité d'appartenir à la classe 1 est supérieure à un seuil s, alors on classe en 1, sinon on classe en 0. Un seuil naturel est 0.5 mais ce n'est pas toujours le meilleur choix. Nous avons donc créé une fonction permettant de visualiser l'évolution du Cohen Kappa score et autres résultats (taux de vrai positif, négatif, …) en fonction du seuil utilisé.

En décidant s = 0.7 et en utilisant un poids plus élevé (5 contre 1) pour la classe 1, nous obtenons un score de 0.31 pour X_test.

Compensations des données déséquilibrées

Au lieu d'utiliser une pénalisation du poids de chaque classe pour pallier au fait que nos données ne soiten pas équilibrées, nous allons créer de manière artificielle des individus de la classe 1 à l'aide des fonctions SMOTE et SVMSMOTE du module over_sampling d'imblearn. Nous appliquerons ensuite un random forest sur ces nouveaux jeux de données.

SMOTE

Cette méthode sélectionne aléatoirement un point dans l'espace engendré par nos variables, puis classe ce point en fonction de ses plus proches voisins (5-ppv par défaut). Si le point tiré au sort est classé 1 (notre classe sous-représentée), l'algorithme ajoute cette observation à notre jeu de données, et ainsi de suite jusqu'à ce que les tailles des deux groupes soient équivalentes.

Après validation croisée des paramètres d'une forêt aléatoire sur cet échantillon, nous estimons Y_test avec : *n_estimators* = 100, *max_depth* = 10, *min_samples_split* = 2, *criterion* = 'gini'

Cela nous fournit un kappa score de 0,248. Ce score est moins bon qu'avec l'ajout de poids sur nos classes. La création d'individu entraîne probablement du sur-apprentissage.

SVMSMOTE

Cette méthode commence par définir les vecteurs supports de l'hyperplan séparateur qui discrimine au mieux nos données. L'algorithme va ensuite générer de façon aléatoire des individus du groupe 1 en fonction de ces hyperplans.

En appliquant un RandomForestClassifier sur ces données nous obtenons un kappa score de 0,307 sur l'échantillon X_test. C'est mieux qu'avec la fonction SMOTE classique mais cela reste moins performant que l'introduction de poids sur les classes.