DATA SCIENCE: Project Report

Christophe yang
University Paris cité

Luca bresolin
University Paris cité

Amine Benmehrez
University Paris cité

April 24, 2022

I. Introduction

One of the most important fields in the fields of neural science and data science is EEG epileptic spike detection, reading EEGs takes a lot of time for specialists because they have to analyze and interpret each point which makes the cost of diagnosis high neurological diseases. The automation of epileptic spike detection becomes necessary and is very beneficial in these areas because it could respond to this problem and detect if an instance captures an epileptic spike or not. There are several methods that allow the analysis of multichannel EEG data series but not all of them have the same precision. The goal of our project is to set up an algorithm for spike detection and to compare several methods with each other in order to find the one that gives the best precision based on a series of data collected on patients which represents a normal EEG series that the we give it the value 0 or the positive instances which is a series of epileptic spikes that we attribute the value 1. The solution of this problem can bring a major help to clinical assessment of epileptic electroencephalogram (EEG) data, save time for specialists when reading these data and at the same time reduce the time taken to care for patients in countries with a number of neuroscience specialists which is very small compared to the surrounding population and improve the accuracy of models in the analysis of EEG data series in future research on the subject.

II. Problem

Set up a method which allows to detect all possible spike candidates and to have the capacity to make the difference between the latter and a normal data series in order to translate them into a series of 0 and 1 according to the data collected on patients proves to be a crucial task in the field of neuroscience and data science and not always easy to do. We have implemented spike classification methods from the SKLEARN library such as RandomForestClassifier and KNeighborsClassifier to help us in this task. Once the classification is done, we put the predictions of the model in a csv file in order to be compared with a series of 1200 data which can represent spikes or normal data series with an evaluation method regarding epilepsy spikes which is based on Precision /Recall/F1.

III. Solution

Dans l'objectif de classifier au mieux les EEG fournis par nos professeurs, nous allons utiliser principalement le langage Python ainsi que les librairies numpy, pandas, matplotlib, sci-kit learn, tensorflow et keras ainsi que des bibliothèques plus spécialisées que nous aborderons plus tard.

Avant de parvenir à notre solution définitive, nous avons expérimenté avec différentes approches, dans la partie qui suivra nous abor-

derons les différents axes et paramètres que nous avons explorés, les résultats et le choix que nous avons fait une fois ces derniers effectués :

i. PREPROCESSING

Dans cette partie nous discuterons de tous les traitements que nous avons faits ou non sur les données avant même de les transférer à un modèle de classification.

i.1 Feature extraction

Dans de nombreux cas de figures, les données ne sont pas facilement classables par un modèle quand elles sont dans leur forme brute, il faut tout d'abord extraire des caractéristiques qui sont supposément plus pures ou pertinentes afin de faciliter la tâche du modèle d'apprentissage. Dans notre cas nous avons envisagé principalement deux types de caractéristiques à extraire des signaux : linéaires et non-linéaires.[1]

Feature Extraction	
Linear Features	Variance
	Absolute Power
	Hjorth's Parameters
	Band power
	Kurtosis
	Mean
	Coherence
	Phase Lag
	Asymmetry
Nonlinear Features	Higuchi's Fractal Dimension (HFD)
	power spectral entropy
	detrended fluctuation analysis (DFA)
	Kolmogorov entropy
	Shannon entropy
	Sample Entropy (SampEn)
	correlation dimension
	co-complexity
	Lempel-Ziv complexity
	Katz Fractal dimension
	Lyapunov exponent
	Rectified Linear Unit (ReLU)

Figure 1: *features extractions*

Les premières ont étés les plus faciles à calculer puisqu'il s'agit de mesures comme la moyenne, la variance ou le maximum par exemple, les secondes un peu plus difficiles à extraire sont principalement obtenues à l'aide transformations telles que la transformée de Fourier ou la transformée en ondelettes.

Après de multiples expérimentations sur les différentes caractéristiques que nous avons pu extraire, aucune d'elles ne rivalisaient avec les données brutes en termes de précision, toutefois certaines apportaient tout de même une plus-value et nous les avons ainsi conservées. Il s'agit de mesures de l'écart-type, du maximum, du minimum et de coefficients issus de la transformée en ondelettes.

```
std_ds =  np.std(train_ds, axis=1).reshape(4800,1,768)
max_ch12_ds = np.max(train_ds,axis=1).reshape(4800,1,768)
min_ch12_ds = np.min(train_ds,axis=1).reshape(4800,1,768)

new_ds = np.concatenate((train_ds,std_ds,max_ch12_ds,min_ch12_ds), axis=1)
```

Figure 2: *Exemple de feature extraction sur des caractéristiques linéaires simples*

i.2 Outlier detection and removal

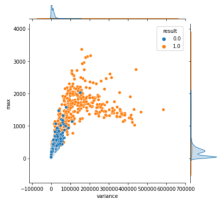

Figure 3: *Affichage des outliers en fonction de deux axes: variance et max (canal 2 uniquement)*

Après avoir observé que certains échantillons juraient avec le reste des membres de sa classe, nous avons imaginés que tenter de détecter et de retirer des échantillons pourrait améliorer la précision de nos modèles. Nous avons alors utilisé deux manières principales: une manière «naïve» où nous repérions nous mêmes les outliers à l'aide de différentes visualisations et des traitements faits à la main et une manière algorithmique à l'aide de modèle comme l'IsoForest étant conçu spécifiquement pour détecter les anomalies dans un ensemble de données.

Les résultats obtenus ont été différents d'une phase à l'autre. Là où ce traitement semblait améliorer la précision lors de la première phase, il n'apportait aucune amélioration (ou sabotait) la précision des modèles lors de la seconde phase.

i.3 Data augmentation

Les techniques de data augmentation nous ont semblées être particulièrement adaptées ici puisque les deux classes (positif/négatif) étaient déséquilibrées avec 5 fois

plus d'éléments négatifs que d'éléments positifs ce qui empêche certains modèles de fonctionner aussi bien que dans des conditions plus équilibrées. Afin de vérifier notre hypothèse, nous avons utilisé imblearn qui permet d'utiliser des fonctions d'oversampling et d'undersampling sur nos données avant de les fournir aux modèles.

Nous avons observé que l'undersampling fournissait des résultats insatisfaisants là où l'oversampling (algorithme SMOTE) semblait apporter une amélioration significative à notre précision. Toutefois cette amélioration a disparue lors de la deuxième phase du projet.

i.4 Données brutes

Les échantillons étant composées de 5 signaux chacun disposant de 768 points, nous disposons de plusieurs manières de classer ces derniers. Nous pouvions soit donner tout le signal aux modèles, une sous-partie des signaux (un ou plusieurs canaux, une seule partie de chaque canal par exemple), ou alors entraîner un modèle pour chaque canal et regrouper leurs prédictions dans un deuxième temps.

Cette dernière méthode nous a paru incontestablement supérieure aux autres et nous l'avons donc conservée tout au long du projet.

Afin de regrouper les modèles nous pouvions opter pour différentes solutions mais nous avons décidé d'utiliser un dernier modèle qui s'entraînerait sur les prédictions des précédents modèles afin de les combiner au mieux.

i.5 Fonctions natives dans sklearn

Nous avons essayé d'utiliser quelques fonctions intégrées dans sklearn étant habituellement utilisées par les data scientists telles que des fonctions de normalisation. Malheureusement aucunes d'entre elle n'a été d'une grande aide dans le cadre de ce projet

ii. MACHINE LEARNING

ii.1 Modèle

Nous avons premièrement décider de comparer de multiples modèles et leurs différentes performances (précision et rappel sur les deux classes, f1-score) sur nos données d'entraînement. Tout comme pour l'implémentation du pre-processing, les résultats n'ont pas été la même dans les deux phases.

Phase 1:

Dans cette phase les modèles les plus prometteurs étaient les Support Vector Machines, les Random Forest et les Perceptron. Les meilleurs résultats ont été obtenus avec le modèle SVC() de sklearn (classificateur basé sur les SVM).

Phase 2:

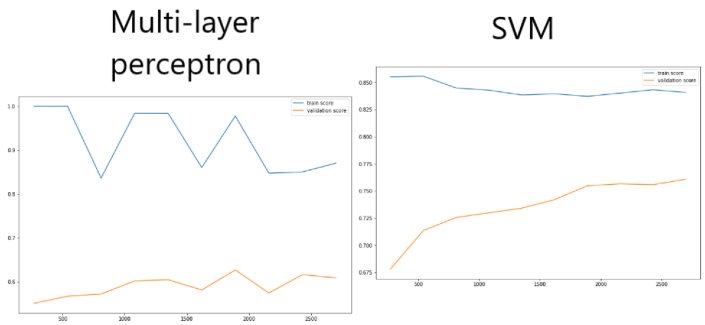

Figure 4: *Comparaison de deux grapiques : le SVM semble un meilleur choix*

Dans cette phase les modèles les plus prometteurs étaient les Random Forest, les Recurrent Neural Network et les SVM. Les meilleurs résultats ont été obtenus avec à la fois RandomForestClassifier() de sklearn et un RNN de keras.

ii.2 Hyper-paramètres

Pour pousser nos meilleurs modèles encore plus loin, nous avons pensé à faire de l'optimisation d'hyperparamètres notamment à l'aide des fonctions GridSearchCV() et RandomizedSearchCV() qui permettait, en plus de déterminer quelles valeurs sont les meilleures

pour les hyperparamètres de nos différents modèles, d'effectuer de la cross validation et ainsi éviter le plus possible le sur apprentissage.

Cette étape n'est néanmoins pas des plus cruciales puisqu'elle n'a jamais permis de gagner plus d'un pourcent de précision.

ii.3 Modèle de regroupement

Afin de regrouper, les différents modèles précédemment entraînés, il nous faut un dernier modèle qui travaillera sur les prédictions de tous ses prédécesseurs avant d'effectuer une prédiction finale plus précise que celles de chacun de ces modèles pris individuellement, ici c'est les SVM, les Perceptron et un RNN qui nous ont apporté les meilleures performances, le modèle final est donc la combinaison de ces derniers.

IV. Expérimentations

Le code a été exécuté sur Google Colab (version gratuite) et sur nos ordinateurs personnels (Luca : i7-8750h, Nvidia GeForce GTX 1050, Christophe : i5, GTX 1050, Amine: i7, Nvidia GeForce GTX 1050). Pour chaque point abordé précédemment et afin de comparer des solutions alternatives, nous avons utilisé quatre méthodes différentes :

i. Le preprocessing:

- partir d'un code complet fonctionnel contenant un modèle classifiant les EEG
- copier ce code autant de fois qu'il y a d'alternatives
- pour chaque copie implémenter une des alternatives en effectuant le traitement avant d'entraîner le modèle
- comparer les performances à différents stades (avant/après assemblage), sur différentes données (set de validation, de test et sur kaggle) et avec différentes métriques (précision, rappel, f1-score à l'aide d'une matrice de confusion par exemple)

Pour illustrer notre méthode, prenons l'exemple de la Data Augmentation :

```
#grid normale avec peu de paramètres pour trouver les meilleurs
grid = GridSearchCV(model, {"n_estimators":[50,75,100], "max_features":[0.5,0.75,'auto']})
smote = SMOTE(random_state=0)
nearmiss = NearMiss(version=3)
for i in range(6):
    #X_train_nearmiss, y_train_nearmiss= nearmiss.fit_resample(X=X[:,i], y=train_labels)
    #X_train_smote, y_train_smote= smote.fit_resample(X[:,i], Y)
    grid.fit(X[:,i],Y)
    splitted_models.append(grid.best_estimator_)

    #Evaluation du modèle avec differentes métriques
    print(f"Performance sur le canal {i+1} : {splitted_models[i].score(X_test[:,i], Y_test)}")
    print("infos : ",grid.best_params_, grid.best_score_)
```

Figure 5:

Extrait de la fonction model_prediction, pour simplifier codé en une partie où en décommentant soit nearmiss, soit smote ou en les laissant commenté nous pourrions tester les performances des modèles soit sans rien, soit avec oversampling, soit avec undersampling

ii. Les modèles :

Nous avons programmé une fonction permettant d'évaluer un modèle sur un certain ensemble d'apprentissage.

```
#Fonction permettant d'évaluer les différents modèles rapidement
def evaluation(model, X, Y):
    X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size=0.25)
    model.fit(X_train, y_train)

    #Affichage de différentes métriques
    ypred = model.predict(X_test)
    print(model.score(X_test,y_test))
    print(confusion_matrix(y_test, ypred))
    print(classification_report(y_test, ypred))

    #Affichage de la courbe "d'apprentissage"
    N, train_score, val_score = learning_curve(model, X_train, y_train,
                                               cv=4, scoring='f1',
                                               train_sizes=np.linspace(0.1, 1, 10))
    plt.figure(figsize=(12, 8))
    plt.plot(N, train_score.mean(axis=1), label='train score')
    plt.plot(N, val_score.mean(axis=1), label='validation score')
    plt.legend()
    return model
```

Figure 6:

Elle nous permet de rapidement l'utiliser avec différents modèles pour les comparer entre eux en utilisant différentes métriques et même une courbe nous montrant l'évolution de la précision (sur validation set et training set) en fonction de la quantité de données auxquelles le modèle à accès à l'entraînement, permettant ainsi d'avoir une idée de la performance du modèle (overfitting ou non ? Scalable?)

iii. Le choix des hyper-paramètres:

Fonction: Model_prediction() Nous avons principalement utilisé GridSearchCV avec des choix de paramètres qui nous paraissait pertinents pour chaque modèle (en nous aidant de la documentation et de la communauté)

iv. Le modèle d'assemblage:

Fonction: merge_channels() On choisit un certain nombres de modèles à tester pour qu'ils assemblent les prédictions de tous les modèles précèdents, et on compare leurs performances.

v. Les canaux:

Fonction: canal() Même principe que pour le modèle d'assemblage mais se focalisant sur un seul des canaux des données brutes afin de vérifier si les canaux réagissent différemment aux modèles et ainsi optimiser l'apprentissage sur chaque canal.

vi. Modele final (1.0):

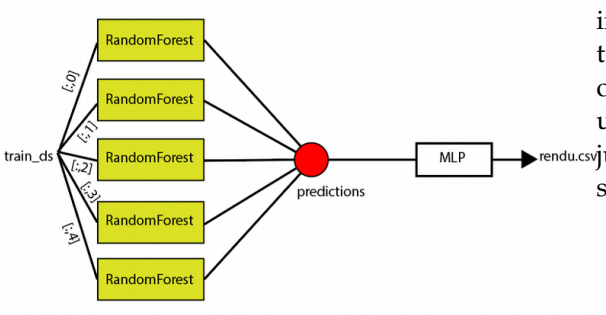

Figure 7:

Comme nous pouvons le voir sur ce schéma, nous avons utilisé au total six différents modèles, cinq pour la première étape de prédiction sur les différents canaux et un seul dans l'optique de synthétiser les prédictions des cinq précédents dans un dernier temps.
Au fur et à mesure pour augmenter notre précision, nous avons rajouté un canal écart-type comme montré dans la sous-partie preprocessing qui rajoute donc un sixième mod-

```
infos :  {} 1.0
Performance sur l'assembleur : 1.0
[[4 0]
 [0 1]]
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00         4
         1.0       1.00      1.00      1.00         1

    accuracy                           1.00         5
   macro avg       1.00      1.00      1.00         5
weighted avg       1.00      1.00      1.00         5
```

Figure 8:

èle. Nous avons également multipliés les modèles d'assemblage et utiliser une fonction codées par nos soins pour maximiser le nombre d'éléments de classe positif car nos modèles n'en fournissait jamais assez.

En dernier lieu nous avons ajouté expérimentalement sept canaux supplémentaires (les six actuels copiés et un de plus sur le maximum) que nous avons fournis à des RNN pour finalement tout combiner, mais nous n'avons pas eu le temps de bien comprendre, optimiser et commenter notre méthode donc nous avons choisi de ne pas en parler mais elle reste disponible dans les sources. Notre meilleur score provient de cette dernière implémentation et a pris environ six heures à s'exécuter (à cause des RNN), toutefois un score relativement proche était obtenable en bien moins de temps (10 minutes) avec la version 1.0 qui pouvait montait jusqu'à 96.7% contre 97.5% pour la deuxième sus ses meilleures prédictions.

V. Conclusion

To sum up, we found that preprocessing (at least how we tried to do it) didn't seem to be very useful in this problem whether it was linear feature or non-linear ones, we haven't found a single feature that outperformed the raw signal when it came to make a model classify it with the most accuracy. Despite this, some features did add some value to our analysis and we decided to keep some of them to stack them with the raw signal. Our best finding was to learn on all the channels separately and even on some features, then combine all

of them thanks to a last model from which we use the predictions to send at Kaggle.

Overall, this project was very interesting and rewarding and we had to learn a lot of machine learning and data science methods and techniques.

References

[1] Antora Dev, Nipa Roy, Md. Kafiul Islam, Chiranjeeb Biswas, Helal Uddin Ahmed, Md. Ashraful Amin, Farhana Sarker, Ravi Vaidyanathan, and Khondaker A. Mamun. Exploration of eeg-based depression biomarkers identification techniques and their applications: A systematic review. *IEEE Access*, 10:16756–16781, 2022.