Challenge Machine Learning
ECP OMA
Dream 2 - Sleep Classification

Group member:
Jed Houas
Achraf Maalej

Janvier 2020

Contents

1 Introduction

In this challenge, the objective is to predict sleep stages via first extracting the right predictive features from the provided signals and then building a machine learning model for the classification. For this challenge, we limit our selves to the models seen in the Machine Learning courses, i.e no Deep Learning models will be considered like Neural Networks or Recurrent Neural Networks.

2 Data extraction and preprocessing

Using the python **xarray** package, we load the raw data from Xtrain.h5 dataset to our environment, then we extract the 11 signals and arrange them in a dictionary of dataframes corresponding to each signal . This dictionary will be the input of all our feature extraction functions that we will further describe in the above. The raw data are organized in a 3D structure, which is why we chose the dictionary data structure: each key corresponding to a signal, for which the value is a dataframe. Each line in this dataframe is a 30 seconds observation timeseries. Above is the function we used for this extraction:

```
data = xr.open_dataset("X_train.h5")
data = data.drop(["index","index_absolute","index_window"])
dict_df ={}
for var in list(data.data_vars):
    dict_df[var] = pd.DataFrame(data[var].values)
```

Listing 1: Data extraction

3 Data exploration

We tried to navigate the data to see how it is distributed across different sleep classes. This will help us especially for choosing the features to use. First, we plotted the distribution of observation across the 5 sleep stages.
We can see that the data is rather balanced:

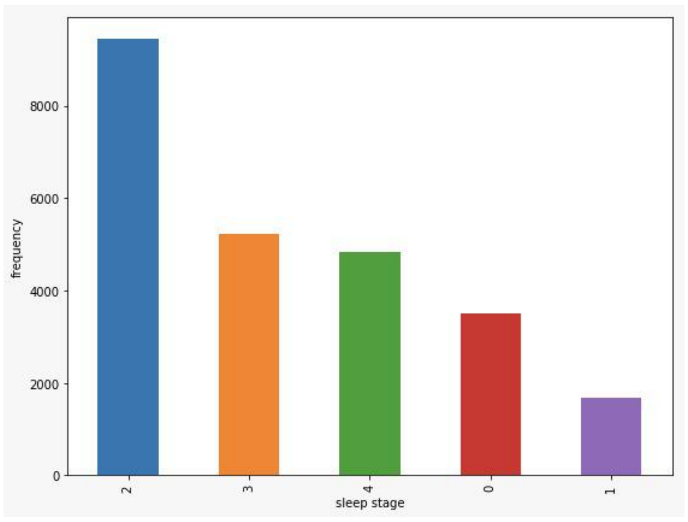


Figure 1: Distribution of observations across sleep stages

We also compared different signals paths during the same sleep stage and inversely a same signal seen in different sleep stage. This will help us identify and match signals behavior to sleep stages:

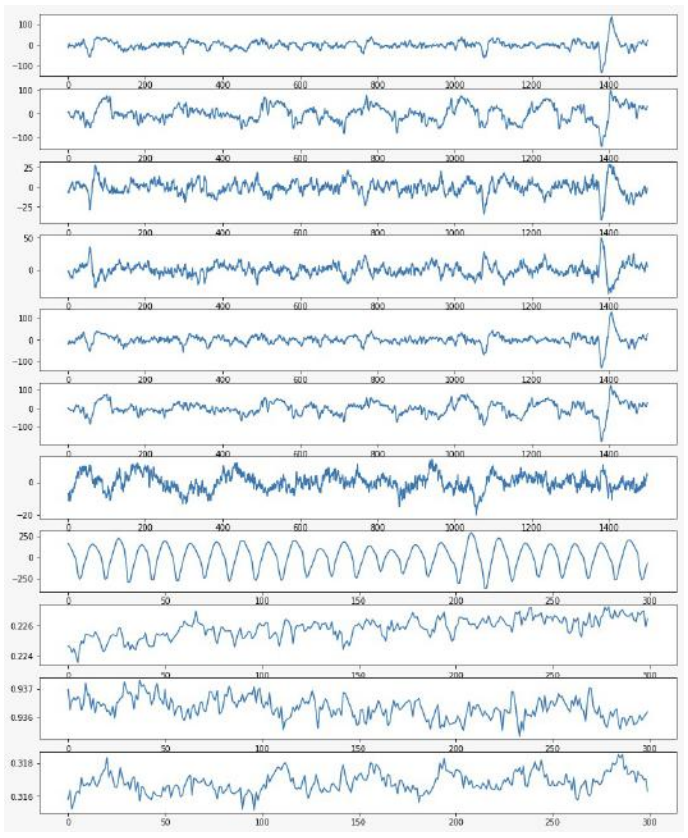


Figure 2: Signals paths in the sleep stage 2

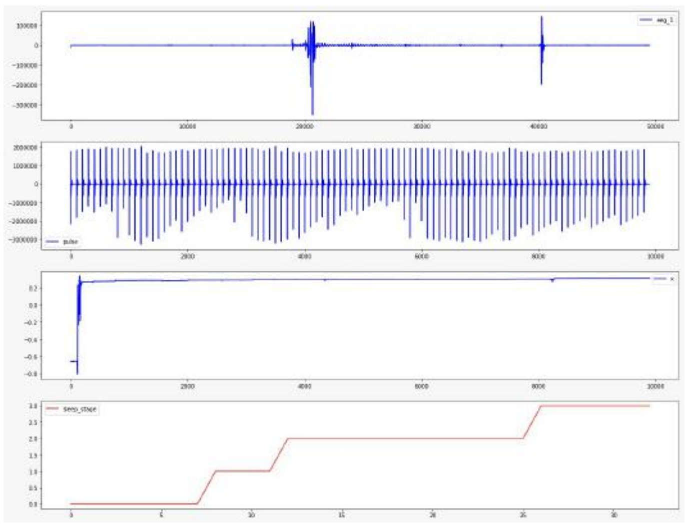


Figure 3: Some signals seen across different sleep stages

4 Features extraction

The features that we used are of two kinds:
- Features that we compute from the 11 signals by immediately applying statistical and proven scientific functions.
- Features that we compute from the frequency bands decomposition of the 11 signals. In fact, we can decompose each signal into a sum of single frequency function. For a specific frequency band ([fmin,fmax]), we filter the components that don't belong to this frequency interval and get a frequency band decomposition function on which we apply standard statistical functions.

4.1 Immediate features extraction

The idea is to apply scientific and statistical function on the 11 signals we have as they are to produce features that can be predictive for the sleep stage. For the statistical part, we tried different statistical operator like **Max-Min** and **kurtosis** . For the scientific part, we have done some research and read some articles on internet related to the subject. From this study, we identify many functions that we have tested like **Absolute Energy** and **Esis**. Note that we will perform features selection afterwards, and so it is not a problem if we include as many features as the non-predictive ones will not be selected.

Also note that we used the Python packages **tsfresh** to extract some of the features that we used (see included reference of this package).
We will present here some of the features we used:

4.1.1 Absolute Energy:

This feature returns the absolute energy of the corresponding signal which is the sum over the squared values:

$$E = \sum_{i=1,\ldots,n} x_i^2$$

4.1.2 Absolute sum of changes:

This feature calculates the sum over the absolute value of consecutive changes in the corresponding signal:

$$\sum_{i=1,\ldots,n-1} \mid x_{i+1} - x_i \mid$$

4.1.3 Aggregate autocorrelation:

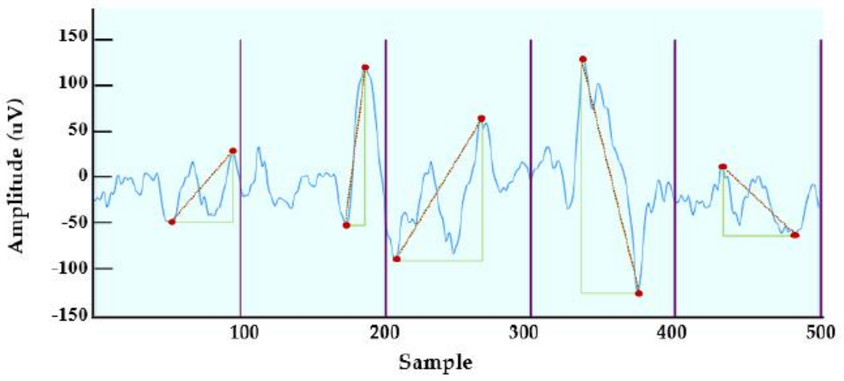
This feature calculates the value of an aggregation function f_{agg} (e.g. the variance or the mean) over the autocorrelation R(l) for different lags. The autocorrelation R(l) for lag l is defined as:

$$R(l) = \frac{1}{(n-l)\sigma^2} \sum_{t=1}^{n-l} (X_t - \mu)(X_{t+l} - \mu)$$

where X_i are the values of the corresponding signal, n its length. Finally, σ^2 and μ are estimators for its variance and mean.

4.1.4 Maximum-Minimum Distance (MMD):

We apply this feature on sub-windows of the signals' time domain. Its value corresponds to the time distance between the moments of maximal signal value and minimal signal value in the sub-window.



4.1.5 Count above/below mean:

This will return the number of values in x that are higher/lower than the mean of x

4.1.6 Kurtosis:

Like skewness, kurtosis is a statistical measure that is used to describe the distribution. Whereas skewness differentiates extreme values in one versus the other tail, kurtosis measures extreme values in either tail. Distributions with large kurtosis exhibit tail data exceeding the tails of the normal distribution (e.g., five or more standard deviations from the mean). Distributions with low kurtosis exhibit tail data that are generally less extreme than the tails of the normal distribution.

4.1.7 Esis:

The basic idea of this feature is to assume that the signal has speed and energy. The speed (velocity) of the signal can be measured by using the frequency and wavelength parameters. The frequency (f) is calculated by finding the midpoint of the pass-frequency for each band. Then, the obtained velocity (v) is multiplied by each squared amplitude (X) modulus of the sample.

$$v = f \times \lambda$$

$$Esis = \sum_{i=1}^{N} \left| X_i^2 \right| \times v$$

Where N refers to the length of the epoch

4.2 Frequency band decomposition and corresponding features extraction:

Using the Fourier decomposition, we can write the signals as sum of sinusoidal functions with frequencies f_i. We then select a frequency band and filter sinusoidals with frequencies outside of this interval to obtain a function for each band. Finally we apply the standard statistical features on each band function. This is because each band corresponds to a specific domain, and so distinguishing them may bring more predictive information. The standard features we applied to band functions are: Mean, variance, skewness, kurtosis, first quantile, median ...

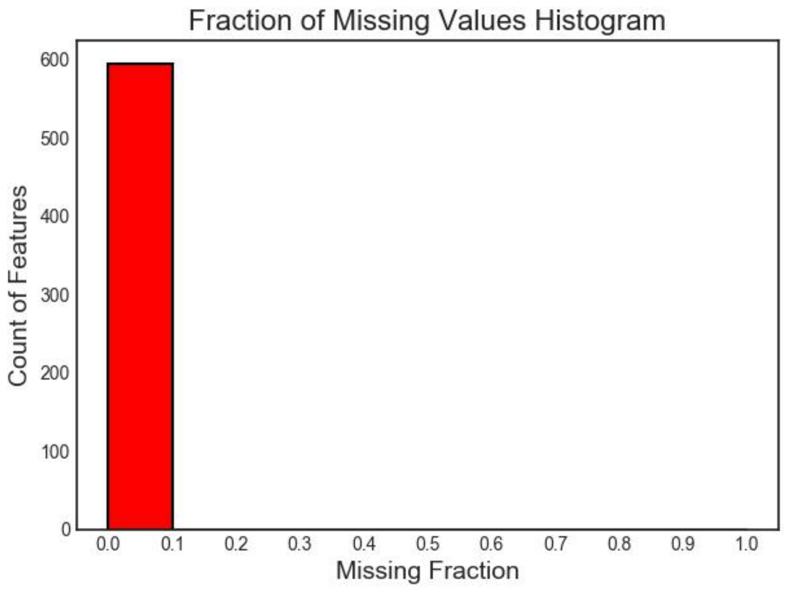
5 Feature selection

At the end of the feature extraction step, we obtained a total of 549 features. The next step is to perform a feature selection to have a better performance and avoid overfitting the train data. Feature selection, the process of finding and selecting the most useful features in a dataset, is a crucial step of the machine learning pipeline. Unnecessary features decrease training speed, decrease model interpretability, and, most importantly, decrease generalization performance on the test set.

we will walk through using the FeatureSelector python package which implements the principal feature selection techniques, we will go through all the following steps :

5.1 Missing values

We want to be sure that there is not missing values in our data and that all the statistical and scientific features were well computed on our time series.



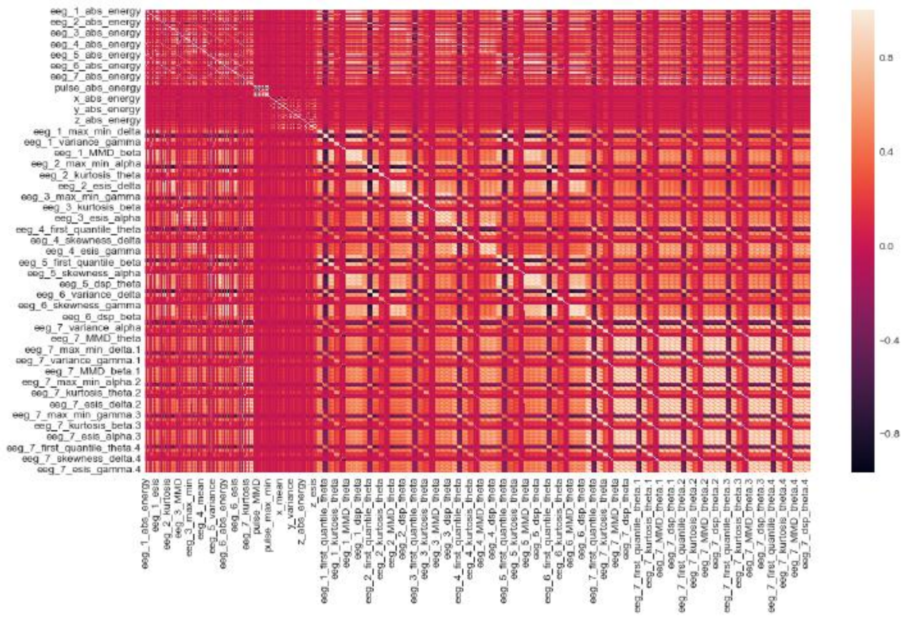
No missing values in our data.

5.2 Collinear Features

As we applied common features on the immediate signals and the frequency bands filtered signals we will check if there are some highly correlated columns in our dataset. The presence of multicolinearity leads to decreased generalization performance on the test set due to high variance and less model interpretability.

In [1]:
```
fs.identify_collinear(correlation_threshold = 0.99)
```

257 features with a correlation magnitude greater than 0.99. Here we show the matrix of correlation between the columns :

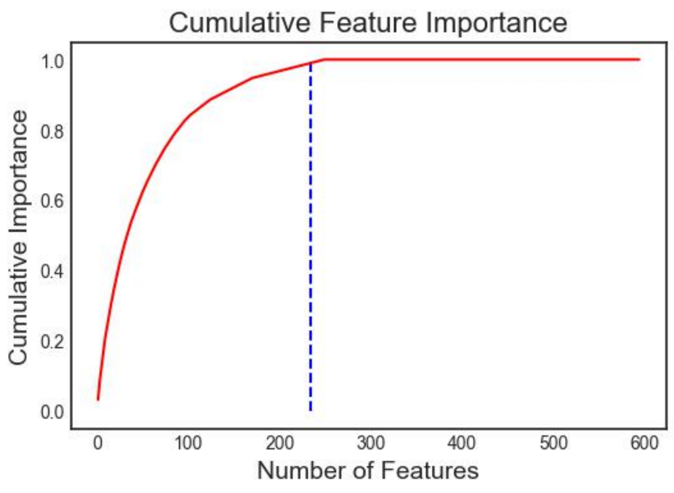


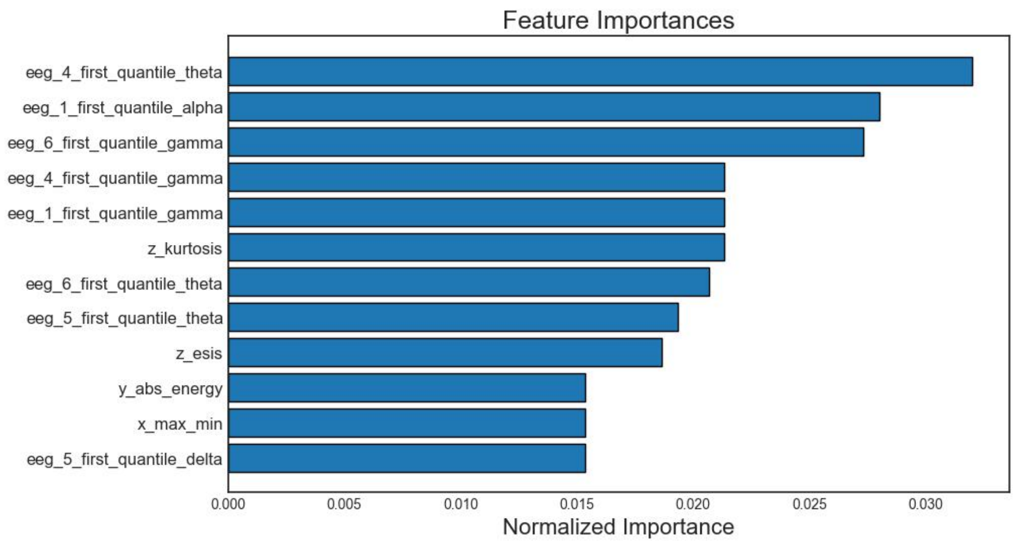
5.3 Zero importance Features

We will remove features that have zero importance according to a gradient boosting machine (GBM) learning model. With tree-based machine learning models, such as a boosting ensemble, we can find feature importances. In a tree-based model, the features with zero importance are not used to split any nodes, and so we can remove them without affecting model performance. We will keep top features contributing to 0.99 of cumulative importance.

In [2]:
```
fs.identify_zero_importance(task = 'classification',
    eval_metric = 'auc', n_iterations = 10, early_stopping =
    True)
```

345 features with zero importance after one-hot encoding.





We notice that first quantile feature is the most important feature for the classification. Combining these steps of feature selection, we kept 199 final features for the classification task.

6 Model Description : Extreme Gradien boosting (XG-Boost)

In the class we saw a the boosting method which refers to a family of algorithms which converts weak learner to strong learners. The idea of boosting is to train weak learners sequentially, each trying to correct its predecessor. The Adaboost is one of the implementations of the boosting method (that we saw in class) which uses adaptive boosting in such a way that at every step the sample distribution are adapted to put more weight on misclassified samples and less weight on correctly classified samples. The final prediction is a weighted average of all the weak learners, where more weight is placed on stronger learners.

The gradient boosting was introduced to build the model in a stage-wise fashion like other boosting methods do, and it generalizes them by allowing optimization of an arbitrary differentiable loss function using gradient descent.

Input: training set $\{(x_i, y_i)\}_{i=1}^{n}$, a differentiable loss function $L(y, F(x))$, number of iterations M.
Algorithm:

1. Initialize model with a constant value: $F_0(x) = \arg\min_{\gamma} \sum_{i=1}^{n} L(y_i, \gamma)$.

2. For m = 1 to M:

   (a) Compute so-called pseudo-residuals:
   $r_{im} = -\left[\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}\right]_{F(x)=F_{m-1}(x)}$ for $i = 1, \ldots, n$.

   (b) Fit a base learner (or weak learner, e.g. tree) $h_m(x)$ to pseudo-residuals, i.e. train it using the training set $\{(x_i, r_{im})\}_{i=1}^{n}$.

   (c) Compute multiplier γ_m by solving the following one-dimensional optimization problem:
   $\gamma_m = \arg\min_{\gamma} \sum_{i=1}^{n} L\left(y_i, F_{m-1}(x_i) + \gamma h_m(x_i)\right).$

   (d) Update the model:
   $F_m(x) = F_{m-1}(x) + \gamma_m h_m(x)$.

3. Output $F_M(x)$.

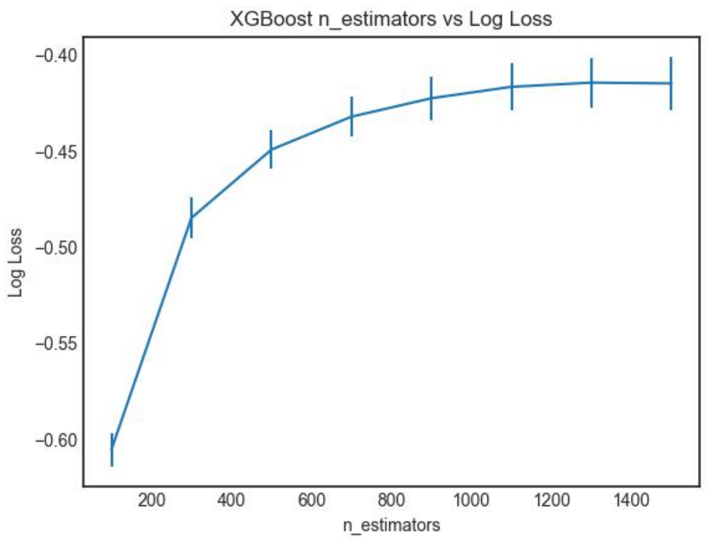
The weak learner models can be either linear models like perceptron, logistic regression or tree based models (CART). Gradient boosting is typically used with decision trees of a fixed size as base learners.

The basic Gradient boosting classifier is known as the LGBMClassifier. The extreme gradient boosting XGBoost classifier implements the same algorithm of gradient boosting but uses a more regularized model formalization to control over-fitting, which gives it better performance.

7 Grid search for parameters tuning

7.1 Number of trees

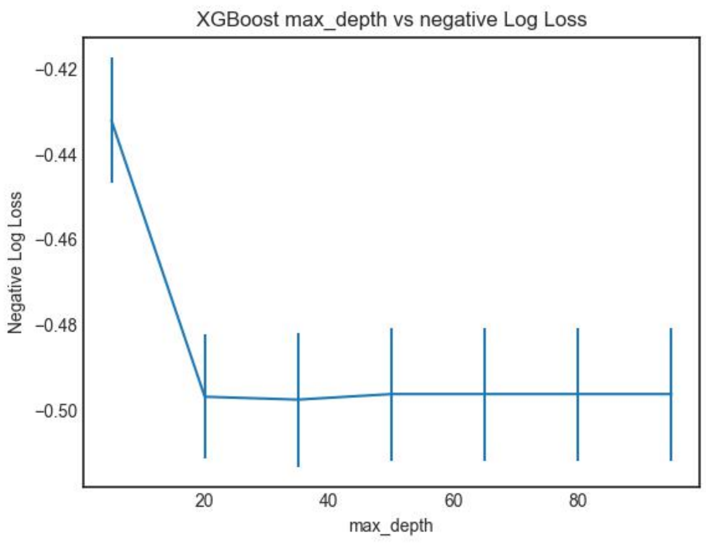
The number of estimators corresponds to the number of iterations in the gradient boosting algorithm. Most implementations of gradient boosting are configured by default with a relatively small number of trees, which is 100 by default. The general reason is that on most problems, adding more trees beyond a limit does not improve the performance of the model. We will fix all the athoer parameters as default parameters and we will vary the number of trees from 100 to 1500. We use negative log loss as an evaluation metric.



Best n_estimator found is 1300.

7.2 Size of trees

In gradient boosting, we can control the size of decision trees, also called the number of layers or the depth. Shallow trees are expected to have poor performance because they capture few details of the problem and are generally referred to as weak learners. Deeper trees generally capture too many details of the problem and overfit the training dataset, limiting the ability to make good predictions on new data. We will fix the number of trees to the best one obtained previously and we will vary the number of trees from 5 to 95 with a step of 15 to find the best value.



We did also a fine tuning of the random forest classifier hyperparameters in order to compare it later with XGBoost within the final results. (Results of the grid search are present in the notebook).

8 Results

In this table, we see the comparison of validation set results between the two models we implemented using the best parameters found in the grid search optimisation:

Model	XGBoost	Random Forest
Accuracy	0.8597	0.8288
F1 score	0.8524	0.8151

We decided then the select the XGBoost model for which we have the following results:

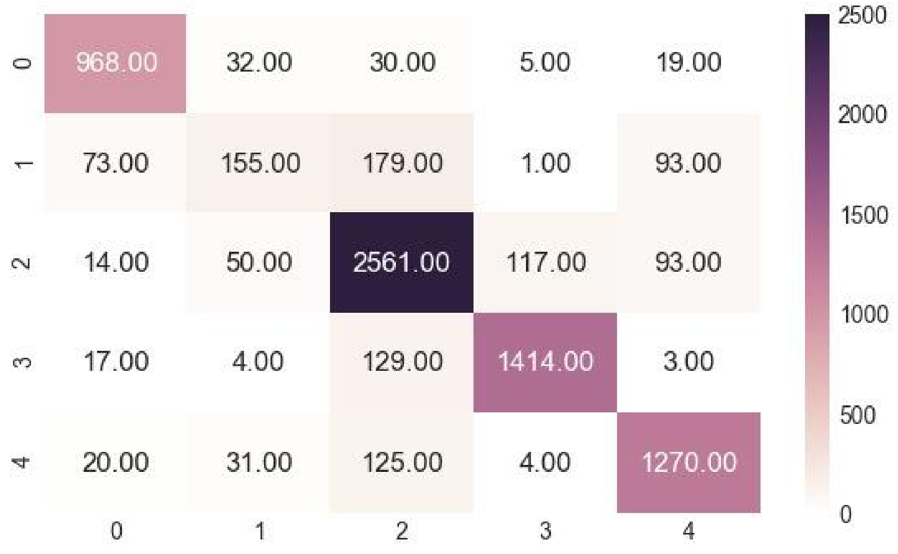


Figure 4: Confusion Matrix - XGBoost model

And also the classification report for the XGBoost model:

Class	Precision	Recall	F1 score	Support
0	0.89	0.92	0.90	1054
1	0.57	0.31	0.40	501
2	0.85	0.90	0.87	2835
3	0.92	0.90	0.91	1567
4	0.86	0.88	0.87	1450
Avg / Total	0.85	0.86	0.85	7407

The overall classification F1 scores are good and around 0.9 for the classes 0,2,3 and 4 except for the class 1 which is worse. The low F1 score on this class is certainly due to a lower number of train samples corresponding to this class. We could explore some oversampling methods to generate new data for this class and see its impact on the classification.

9 References

- Sleep Stage Classification Using EEG Signal Analysis: A Comprehensive Survey and New Investigation (Khald Ali I. Aboalayon, Miad Faezipour,*, Wafaa S. Almuhammadi and Saeid Moslehpour)
- https://tsfresh.readthedocs.io/en/latest/text/list_of_features.html
- Cs229.stanford.edu/extra-notes/boosting.pdf
- Machinelearningmastery.com/tune-number-size-decision-trees-xgboost-python/