Machine Learning - Dreem

Driss Debbagh Nour – Mehdi Mikou

January 2019

Contents

1 Introduction

Dreem is a company that helps people improve their sleep. The first step is understanding it. This can't be done without accurately measuring key biological signals: brain activity, heart rate, respiration rate, movement. The Dreem Band's sensors track people statistics throughout the night for diagnosing sleep disorders. To this end, Dreem has to check the progress over different sleep stages.

Sleep progresses in cycles that involve multiple sleep stages : wake, light sleep, deep sleep, rem sleep. Once we got people statistics, we can couple these with artificial intelligence tools.

The Dreem headband records a lot of nights every day (A lot of Data). They labeled this data by sleep experts. And today, their teams are developing the most accurate automatic sleep staging algorithms.

Dreem came to us for this reason, providing a challenge: Developing an algorithm of sleep staging able to differentiate between Wake, N1, N2, N3 and REM on windows of 30 seconds of raw data. The raw data includes 7 eegs channels in frontal and occipital position, 1 pulse oximeter infrared channel, and 3 accelerometers channels (x, y and z).

2 Data Preprocessing

2.1 Raw Data

Dreem provided us :

- A Training set (3.5 Go)
- Sleep stages for the training set (labels)
- A Test set (3.5 Go)

The data is separated between the different signals recorded by the Dreem headband. Signal is shapes in windows of 30 seconds. The goal is to predict for each window the corresponding sleep stage.

The data is provided by three kinds of sensors: electroencephalogram (EEG), pulse oximeter and accelerometer signals.

Electroencephalogram is measured at 7 differents locations of the head (eeg 1 to 7). The sampling frequency is 50 Hz.

Pulse oximeter (pulse oximeter infrared) and acceleromter (accelerometer (x, y, z) channels are sampled at 10Hz

The files provided are "Hierarchical Data Format" (hdf5). It allows to store huge amounts of numerical data and easily manipulate that data from NumPy. It's easy to manipulate as groups work like dictionaries, and data sets work like NumPy arrays.

A first look on the training set and its labels shows that there is a considerable imbalance between different sleep stages:

We worked on rebalancing the training set. We will discuss it later.

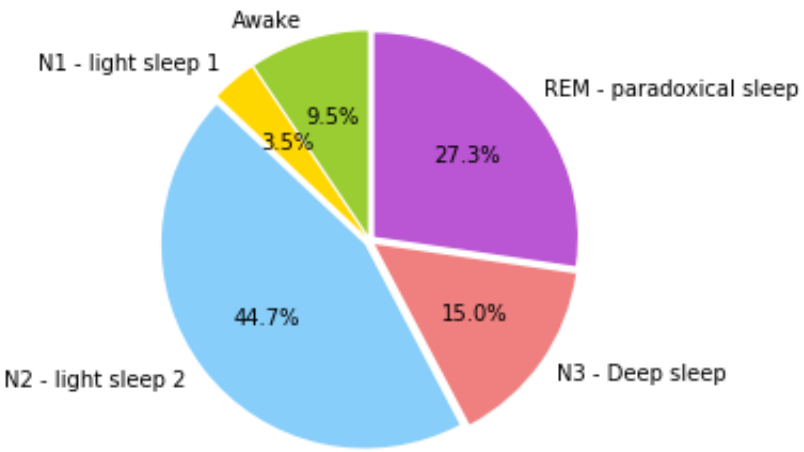

Figure 1: Distribution of sleep stages

2.2 Truncated data

Since the training file is huge (more than three Go), we first worked on a data set containing only 1000 windows of 30sec chosen randomly from the original data set.

We managed to have a similar distribution of sleep classes. in the truncated data.

This method enables fast and highly flexible generation of features and allows a first qualitative and quantitative analysis of the problem. We will talk about the extracted features later.

Let us take an example of an extracted feature which is the Approximated Entropy. A technique used to quantify the amount of regularity and the unpredictability of fluctuations over time-series data. Compared to the entropy, the Approximated Entropy is easier to compute and can handle noisy data.

When plotting box plots to visualize the distribution of Entropy for the five sleep classes, and for each sensor, we notice that this feature can be very discriminant when it is calculated for some sensors:

We observe that the feature Approximated Entropy will be helpful to differentiate sleep stage, especially when using the one calculated over the signal produced by the sensors: eeg 4, eeg 5, eeg 6 and eeg 7.

This result matches with the one obtained using some algorithms for features selection. Which will be discussed later.

In the same way as for Approximated Entropy, we can make a qualitative analysis to figure out which features can improve our predictions.

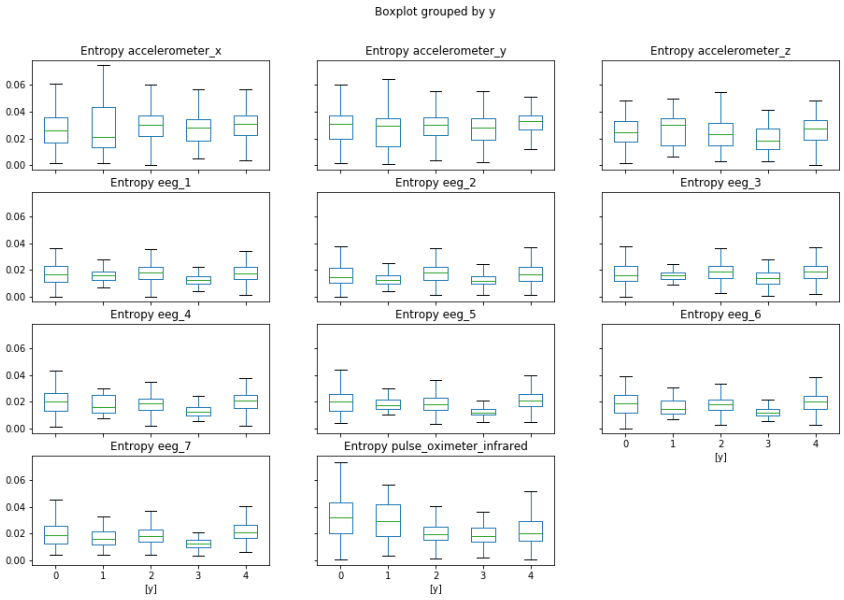

Figure 2: Distribution of sleep stages using Entropy

3 Feature Extraction

The most important part of the project is the feature extraction or the data preprocessing because of the nature of the data we had. Indeed, machine learning algorithms perform much less when they take signals as an input than when it is features extracted from them. Also, it was suggested by the teachers to extract features in the random forest solution that performed 55

We did two kinds of feature extraction in this project, one based on the original signals (from the 11 sensors) and one based on the Fourier transformation of the original signals (more explanations below).

3.1 Feature extraction based on the original signals

In this part, we have focused on extracting some data from the original time series given by the different sensors. The features extracted here are the following:

- Abs Mean: corresponding to the mean of the absolute value of the signal
- Abs Median: corresponding to the median of the absolute value of the signal
- Abs Min: corresponding to the minimum of the absolute value of the signal
- Abs Max: corresponding to the maximum of the absolute value of the signal
- Std: corresponding to the standard deviation of the signal

- Energy: corresponding to the energy of the signal (more explanations below)
- Approximate Entropy: corresponding to the entropy of the signal (more explanations below)
- Kurtosis: corresponding to the kurtosis of the signal (more explanations below)
- IQR: corresponding to the Interquartile range of the signal (more explanations below)

3.1.1 Energy

In signal processing, he energy E of a discrete-time signal x_n is defined as $E = \sum \|x_n|^2$

3.1.2 Approximate Entropy[2]

In statistics, an approximate entropy (ApEn) is a technique used to quantify the amount of regularity and the unpredictability of fluctuations over time-series data. Compared to the entropy, the ApEn is easier to compute and can handle noisy data.

3.1.3 Kurtosis

In probability theory, kurtosis is a measure of the of the "tailedness" of the probability distribution of a real-valued random variable. If X is a random variable, it is given by: $Kurt(X) = E\left[(\frac{X-\mu}{\sigma})^4\right]$ where μ is the expectation of X and σ its standard deviation.

3.1.4 IQR

The interquartile range is a measure of the dispersion of a random variable given by the difference between the third and the first quartile.

3.2 Feature extraction based on the transformed signals

Signal analysis techniques are very helpful to analyze, model and classify signals. Dreem provided us discrete-time domain signals. The mathematical function which transform a signal from the time-domain to the frequency-domain is called the Fourier Transform.

We made three functions to produce three new signals from the original one: The Fast Fourier Transform (FFT) to calculate the Discrete Fourier Transform. The Power Spectral Density, which takes (in addition of the FFT) the power distribution at each frequency. And finally the Auto-Correlation, which calculates the correlation of the signal with a time-delayed version of itself.

Once the three signals generated for each sensor. We detect the first peaks [1] of each of them (5 first peaks). We get then the coordinates of each one. Finally we obtain ten features for each new signal for each sensor. In total we get 330 features.

Given the large amount of data generated here (429 features in total) , we think it is important to perform a feature selection to reduce the number of features in order to improve the performance of the algorithms used here and also to speed them. In the next part, we will discuss more heavily on how we reduced the number of features in this project.

Given the large amount of data generated here (429 features in total) , we think it is important to perform a feature selection to reduce the number of features in order to improve the performance of the algorithms used here and also to speed them. In the next part, we will discuss more heavily on how we reduced the number of features in this project.

4 Theoretical Description of the Models

4.1 Feature Selection

4.1.1 Removing correlated features

The first part of our feature selection consisted of removing correlated features among the 429 features generated in total. Indeed, leaving correlated features can harm the performance of the models we use generating poor results as seen during the course. Hence, we have removed features with a coefficient of correlation higher than 95% .

For example, if we have 3 features X, Y, Z that are strongly correlated, meaning that r(X,Y) > 0.95 and r(X,Z) > 0.95 and r(Y,Z) > 0.95, then we randomly remove 2 of them out of the 3. In our case, this operation made us to remove 67 features which allowed us to increase the performance of our models and to decrease the time of the learning.

4.1.2 Keeping most relevant features

Feature selection consists on finding features that contribute the most to the overall model's predictive performance. We will use 3 different methods to perform this operation:

- Random Forest Feature Importance: Scikit Learn performs it by randomly permuting a given feature and observe how the classification changes
- ExtraTreeClassifier Feature Importance: same as random forest
- Recursive feature elimination : which consists on performing multiple feature importance and removing at each step the most important feature.

The next question is to determine how many features we should keep in order to have the best performance. To answer this question, we have tried different number of features to keep and stored the number which gave the better prediction. After determining this number, we union the features from the different methods and then we remove duplicates.

4.2 Choice of models

This part is quite crucial in this Kaggle competition. Indeed, choosing the appropriate model can lead to way better results. After reviewing the literature on Kaggle Challenges, we found out that the most efficient models are ensemble models : the ones that are built by mixing many models.

Since the purpose of this project is to perform a classification, we thought about many classification algorithms such as logistic regression, logistic regression with penalization, support vector machine, random forest and boosting methods. We tested all of these methods but we quickly

came to the conclusion that random forest and boosting methods are the ones which perform the best.

Also, it is well known in the Kaggle community that these methods are the most efficient. From now on, we will focus on random forest and boosting methods.

4.2.1 Random forest

Random Forest is a supervised learning algorithm that builds multiple decision trees and merges them together to get a more accurate and stable prediction for regression or classification problems.

Random Forest, compared to decision trees, adds some randomness to the model and tends to be more robust. Indeed, while deep decision trees tend to overfit, Random Forest doesn't because it builds smaller trees from a smaller subset of features.

Like most machine learning algorithms, Random Forest comes with hyper parameters: for us the most important ones are the number of trees we build (n-estimators in sklearn) which helps improving the power of the prediction and the maximum number of estimators to use in each tree.

4.2.2 Boosting Trees

Boosting Trees methods are also based on trees and are quite similar to Random Forest, the only difference is in the way the trees are built.

Gradient Boosting build trees one at a time, where each new tree helps to correct errors made by previously trained tree. Random Forest trains each tree independently, using a random sample of the data. This randomness helps to make the model more robust than a single decision tree, and less likely to overfit on the training data.

In the ensemble model, we will use the gradient boosting method found in the package Scikit Learn and the XGboost package which is very famous for outperforming other methods.

5 Tuning the models

Once the three models chosen. We need to find out their hyperparameters that fit the most with our data sets.

5.1 Grid Search Randomized Search

For this end we will use two different methods which explore exactly the same space of parameters: Randomized search and Grid search.

The results in parameter settings are quite similar, while the run time for randomized search is drastically lower. Usually, most parameters that influence the learning are searched simultaneously (except for the number of estimators, which poses a time / quality tradeoff).

The performance is slightly worse for the randomized search. Since we had the time we moved forward with the GridSearch method.

We used the function GridSearchCV which takes as inputs the grid of parameter values as space where to look for the best hyperparameters. The function explores all possible combinations and test them all. The Randomized Search explores randomly a number of combinations specified by the user. If the number is high enough the results are the quite the same as for the grid search.

5.2 Cross Validation

When testing the combinations, the two functions use a Cross Validation. Indeed, trying to find the best hyperparameters over the same test set leads to overfitting. Therefore, to avoid it, the training set is split into k smaller sets. A model is trained using (k - 1) of the folds as training data, the resulting model is validated on the remaining part of the data (used as a test set), and that for each of the k folds. The performance measure reported by k-fold cross-validation is then the average of the values computed in the loop.

5.3 Voting Classification

We will talk about the results of our predictions later. But since the results on the test set are quite low (around 60%), we decided to use a method called voting classifier, as mentionned before. The goal behind that is to try different models and different hyperparameters, then compare the results obtained for each model, to finally keep the sleep stage with the most "votes" (the one that was the most predicted over the different models).

6 Predictions

Once the models chosen, the hyperparameters determined (Cross Validation), the features selected, we move to the next step: classify the windows of the test set.

6.1 Unbalanced Data

We group our results in the following table:

Model	Train/Test split	Cross Validation	Test
Random Forest Classifier	61.67 %	67.23 %	56.38 %
Gradient Boosting	60.52 %	66.86 %	56.22 %
XG Boost	65.32 %	69.49 %	58.42 %

Other models were tried but we quickly decided not to use them anymore, because of what we read on several articles about their efficiency in this type of problems, and because of the results we got by ourselves.

Let us describe the performance of our classification model (Random Forest) by plotting the confusion matrix of the predictions made:

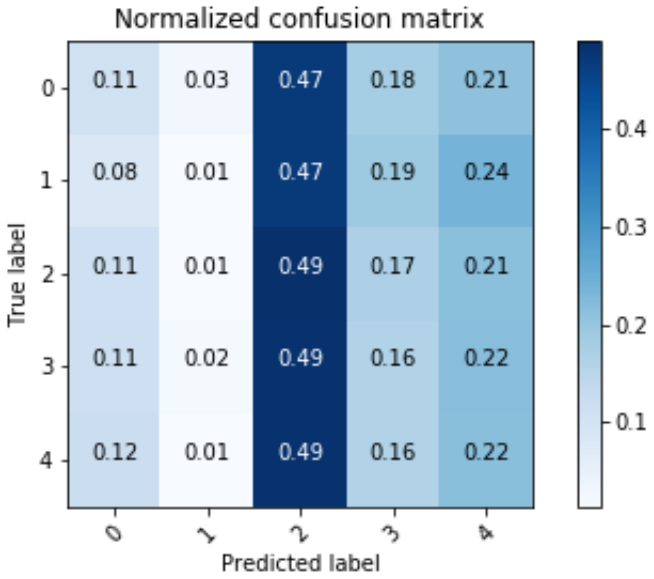

Figure 3: Confusion matrix

We observe that the predictions are bad for the classes that are the less represented on the training data set. Indeed, as seen before data we have is out of balance.

If we take for example class 1 ("N1 - light sleep 1"), the distribution pie shows that it represents only 3.5% of the data. The confusion matrix shows that we only predict correctly 0.01% of it. Which is extremely low. And half of the windows that should have been predicted as class 1 were predicted as class 2, which is the most represented class in the original data set.

6.2 Rebalancing Data

We decided then to rebalance our training data set. And make new predictions. To this end, we separated the different classes, then we duplicated twice or more the ones with low appearance ratio, and truncated the ones with high appearance ratio in the data set.

The results obtained weren't satisfying. We consider that it is because of duplicating the data many times. We decided then to not duplicate, by to truncate more classes 2 and 4, making them as much represented as the other classes. Unfortunately the results were worse. The training set was obviously insufficient to allow good predictions.

Finally we decided to generate news data, especially for classes that are less represented. We took the exsisting data and added some gaussian noise on it. That allowed us to generate 9000 new windows and rebalance our training dataset. The predictions were clearly better: we observe a better distribution of predictions and better scores on the diagonal of the confusion matrix.

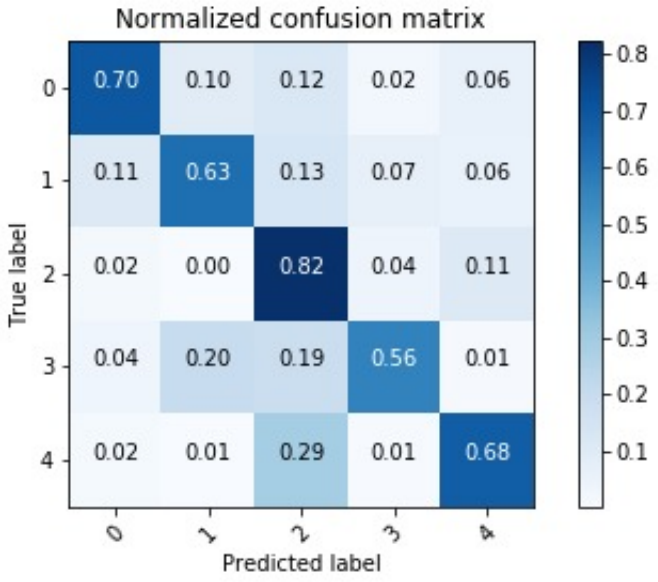

Figure 4: Confusion matrix of rebalanced data

Bellow are represented our results:

Model	Train/Test split	Cross Validation	Test
XG Boost	69.32 %	74.49 %	62.73 %

7 Conclusion

Unfortunately the results were not up to the work done. An improvement of what we have done would have been to generate new signals of the classes less represented in a more sophisticated way. That would have given us a better balance into our data, hence, more accuracy.

We still are grateful to participate to such a competition. We think that it was the best way to apply what we learned during class. We had the chance to work on a complex data, and work on both data engineering and data processing. We also had the chance during holidays to apply what we learned while working on this Kaggle competition during some interviews.

References

[1] Marcos Duarte. Detection of peaks in data.

[2] I. M.; Ehrenkranz Pincus, S. M.; Gladstone. A regularity statistic for medical data analysis. 1991.