ARTIFICIAL NEURAL NETWORK

HOMEWORK 2

Annarita Barone - Cristian C. Spagnuolo

Person Code: 10615770 - 10745353

Team BS

Computer Science and Engineering

Politecnico di Milano

January 21, 2022

Contents

1 Task Description

In this homework, we are required to predict future samples of a multivariate time series. The goal is to design and implement forecasting model to learn how to exploit past observations in the input sequence to correctly predict the future.

2 Loading Data

To begin, we load the dataset and we perform perform the **Exploring Data Analysis**. To this end, we defined some ad-hoc functions. In Figure 1 we can see a visual representation of the dataset.

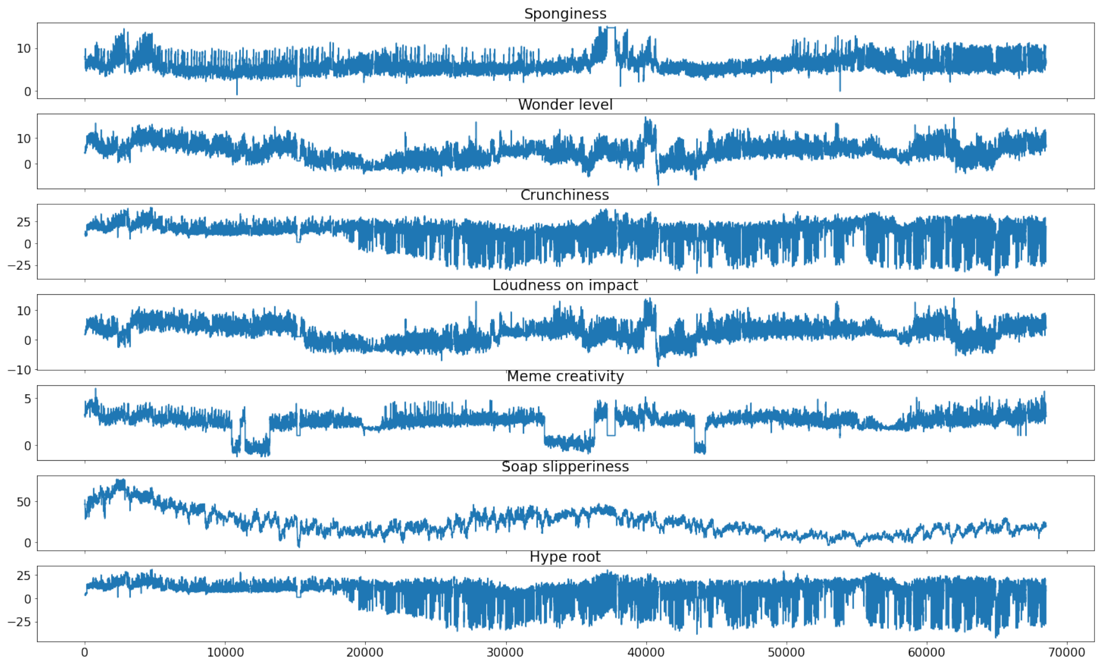


Figure 1: Dataset's trend visualization.

3 Pre-processing

Analyzing the dataset's trends, we noticed some plateau of values, we tried to remove them in different ways:

1. Remove these values, and all other values of a different feature and the same timestamp.
2. Replace these values using a model, trained on the original dataset.

However, we noticed that we didn't improve performances, therefore, we reached the conclusion that our models were robust to this small number of anomalies.

As about dataset splits, we used **hold out** cross validation (0.9 - 0.1), considering the last samples of the dataset as validation. After choosing the best model we retrained it with all data, using the same number of epochs suggested by early stopping, to avoid overfitting. Another important thing to be pointed out is that, when dealing with time series is important to choose the model focusing on most recent data, preserving the chronological order. In other words, data shuffle must be avoided.

Finally, we choose **min-max scaling** as normalization.

4 Timeseries Forecasting

Our first step was to define a baseline model to compare performance with. We used a very simple RNN, based on two **Bidirectional LSTMs**, like the one seen during lessons. As about the output shape, we are dealing with a multivariate forecasting problem, hence, we want to predict several future values for each class. To this end, we used a Dense layer, of $telescope * \#channels$ neurons, and then a reshape one to get a $(telescope, \#channels)$ output. Finally, it is performed a **1D convolution** with #channels filters $(1, 1, \#channels)$ to smooth results. We did not submit the model, since, on our validation score it performs very bad, predicting the mean value.

4.1 Autoregression

The first improvement was to change the telescope of our model and use an autoregressive approach. We set $telescope = 48$ and we reached the desired one (864) with an iterative approach. We used a shifting window on last 200 samples, moving telescope samples at each iteration. The results of this model were very promising, reaching *3.912* on CodaLab. After this results, we tried different configurations of telescope, window, stride, units over the baseline model, without any improvements.

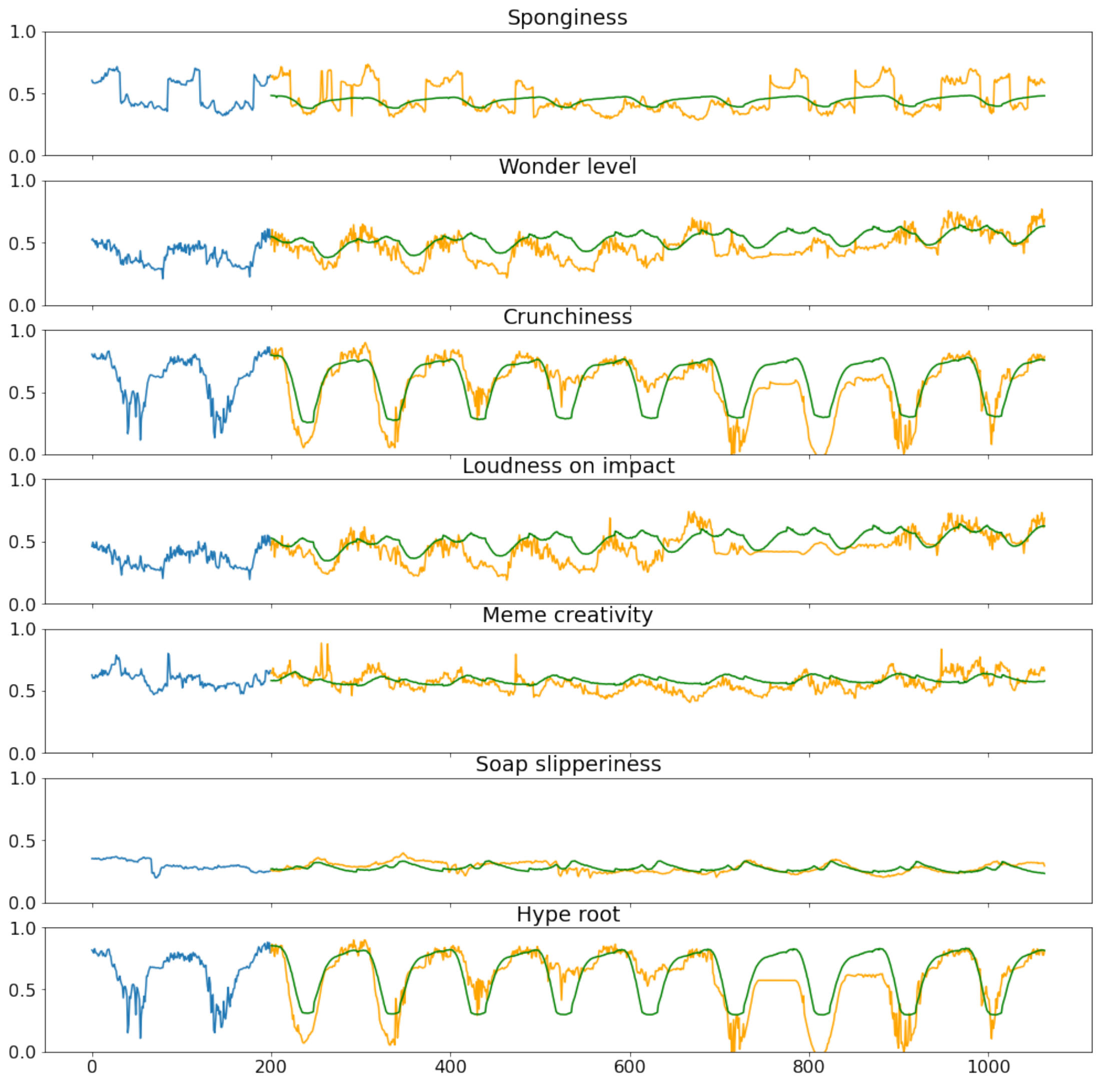


Figure 2: Prediction on the validation set for the best baseline model.

4.2 Encoder-Decoder

Another architecture we tried was the encoder-decoder one. At the beginning, we used a small window size, comparable with the best one of the baseline model. However, in our task, the goodness of future predictions, depends on the goodness of previous ones. Increasing the window size, we can reduce the weight of predicted values, since we are considering a larger amount of real samples. To make the model more robust, w.r.t. input variations, we added two **Gaussian Noise** layers, whereas, the dense part was left unchanged. The best performance on Codalab was *3.892*, achieved using $telescope = 108, window = 3900, stride = 30$.

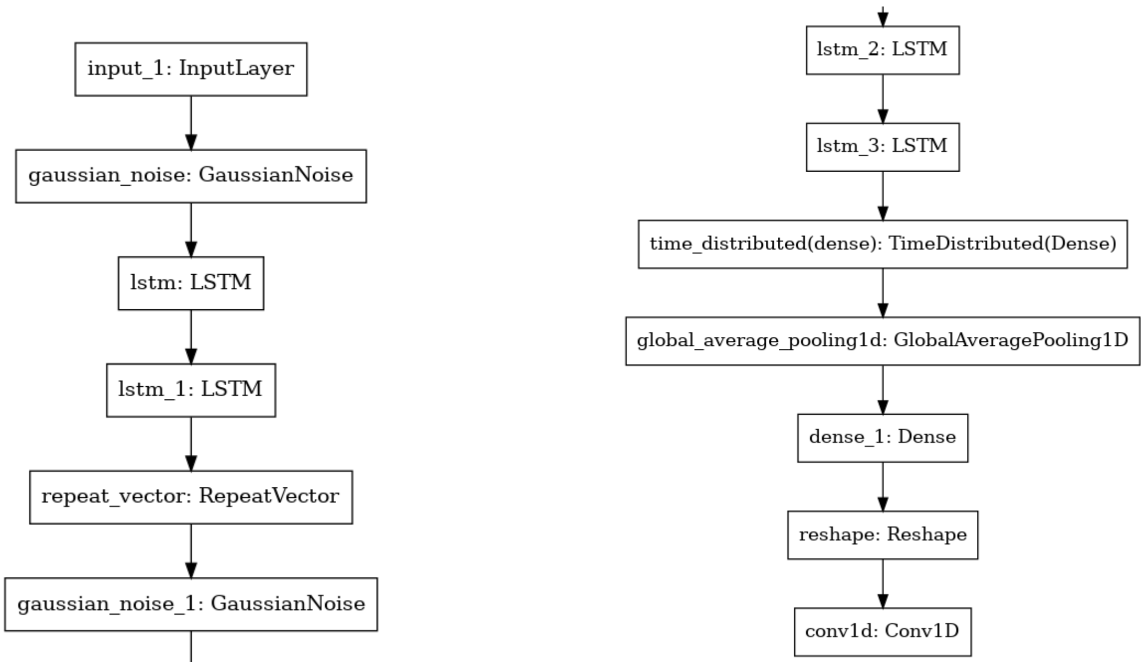
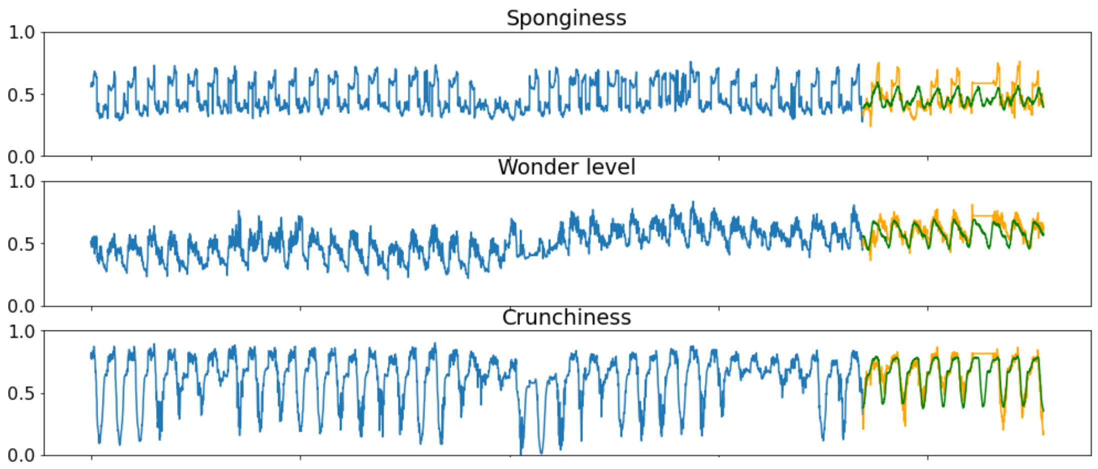

input_1: InputLayer
gaussian_noise: GaussianNoise
lstm: LSTM
lstm_1: LSTM
repeat_vector: RepeatVector
gaussian_noise_1: GaussianNoise
lstm_2: LSTM
lstm_3: LSTM
time_distributed(dense): TimeDistributed(Dense)
global_average_pooling1d: GlobalAveragePooling1D
dense_1: Dense
reshape: Reshape
conv1d: Conv1D

Figure 3: Encoder-Decoder architecture



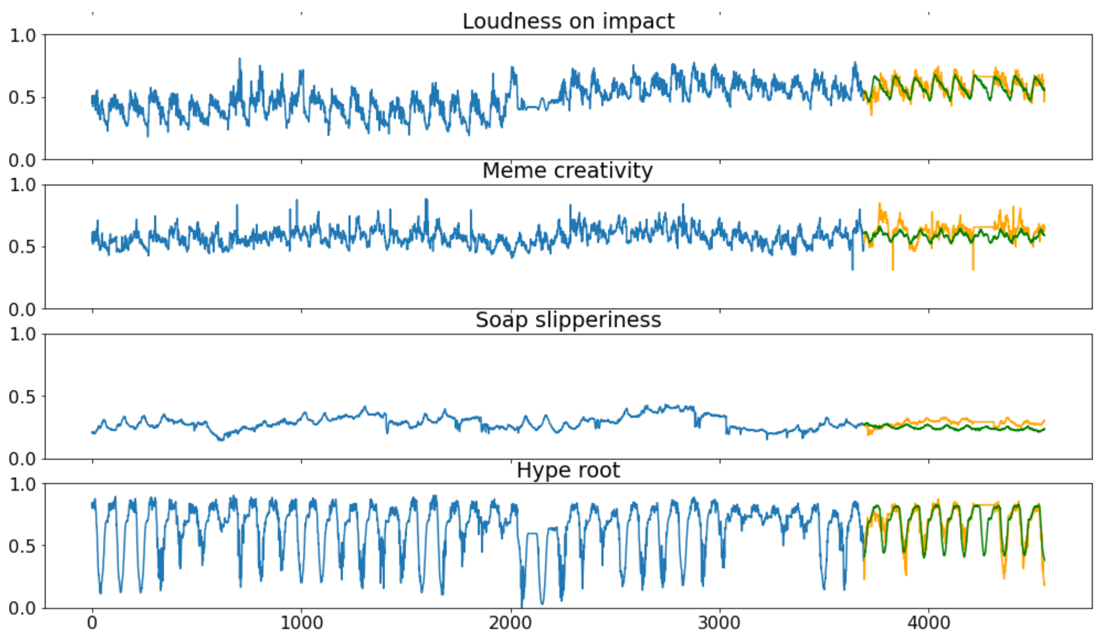


Figure 4: Prediction on the validation set for the best Encoder-Decoder model.

4.3 Ensemble

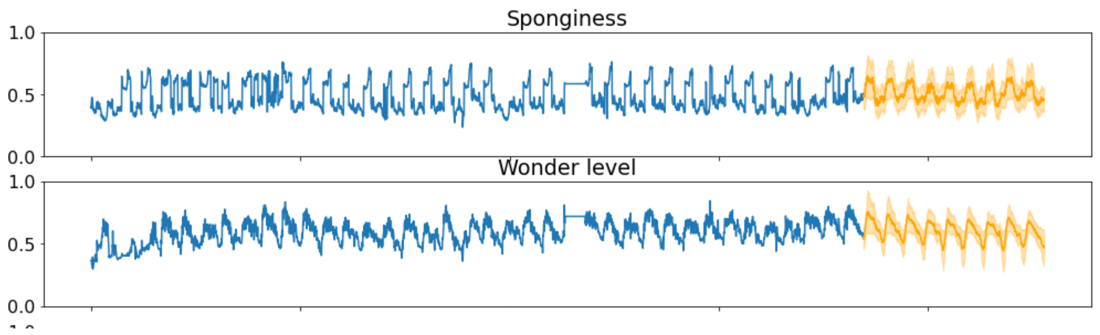
To improve the test performance, we used also some ensemble techniques. The basic idea was to mix the outcome of several models in to get better results. We tried the following strategies, considering our best models:

1. **Average**: perform a simple average among the values predicted by each model, *3.836* on Codalab.

2. **Weighted Average**: weighting the contribution of each model, on each feature, according to the score that the model achieved on them, getting a weighted average of results, *3.737* on Codalab.

3. **Feature Replacement**: the intuition was that some models perform better than other models in predicting a specific feature, even if the overall performances were lower. To this end, we used our best models to perform a normal prediction and then we computed the final output by replacing feature predictions with the output of the best model w.r.t. that specific feature, *3.364* on Codalab.

All these three approaches lead to some improvements, w.r.t. the single model, however, the third one was the most promising. Predicted values are shown in Figure 5.



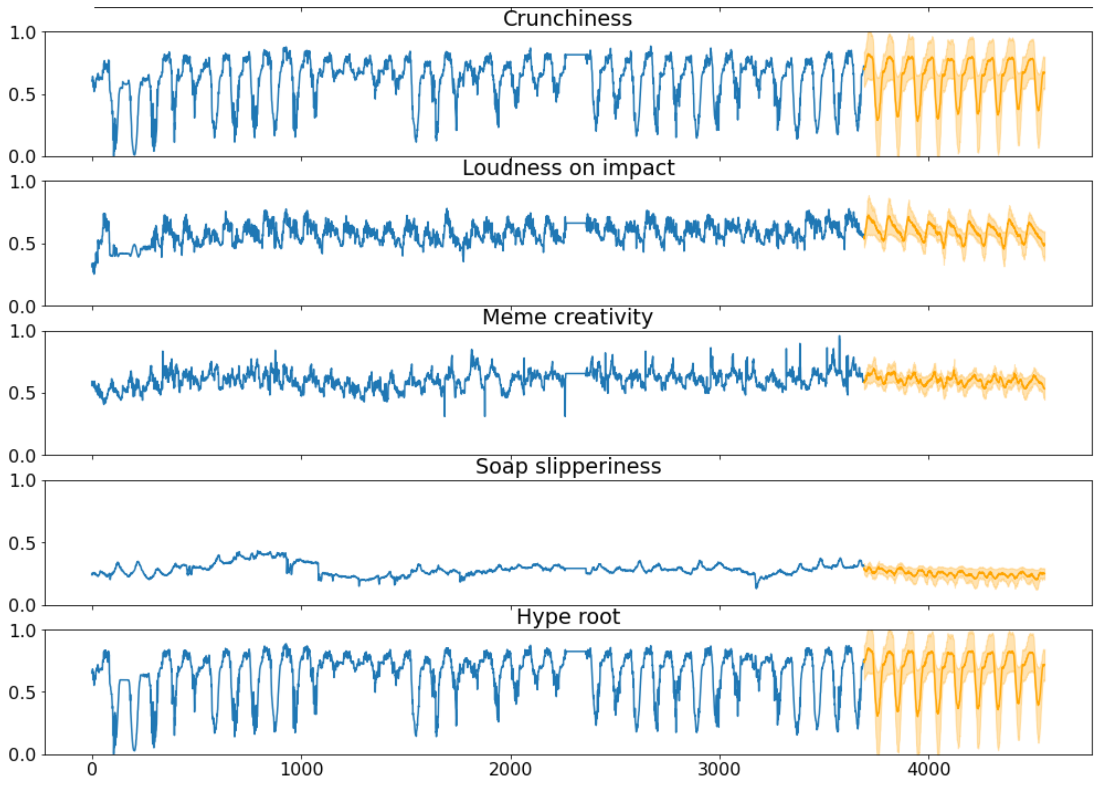


Figure 5: Future predictions of the best model after ensemble.

5 Hyper-parameter tuning

The hyper-parameter selection has been done using the **BayesianOptimization**[1]. To this end, we defined a function to build a model which receives a pool of parameters to be chosen, in order to use hold out cross validation for hyper-parameter tuning. This is a quite long procedure, hence, we used it only when significant changes where done.

6 Conclusions

To sum up, several models have been implemented. Throughout the process we applied several theoretical notions learned in class, and we tried different approaches to achieve better performances. Following a trial and error approach, we tryied to implement an attention mechanism, however, it seems that performances were not improved. As last option, a transformer model was investigated, but, we didn't managed the computational complexity (Kaggle went out of memory). We have still have a lot to learn but we are on the right path and we are eagerly waiting to test our skills again.

[1]From keras_tuner.tuners