Report Challenge 2 Team Drip Mac

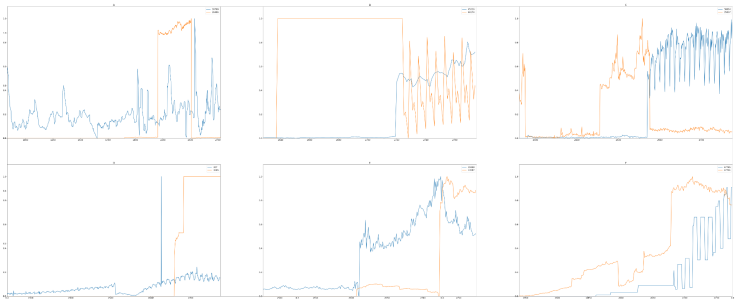
Leonardo Brusini, Marco Scarpelli, Davide Tenedini, Federico Toschi

Problem Description

The problem is a time series forecasting problem. Our objective is to train a model that can correctly predict the future *t* timesteps of the given time series. *t* equals 9 in the “Development” phase of the competition and equals 18 in the “Final” phase. Our model is evaluated based on the mean squared error between the true future time steps and the predictions.

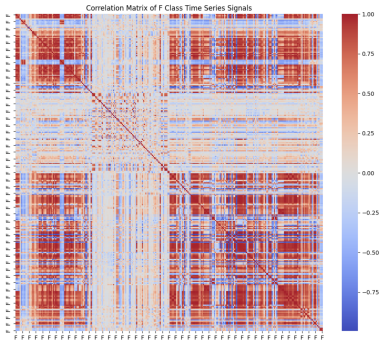
Dataset Inspection

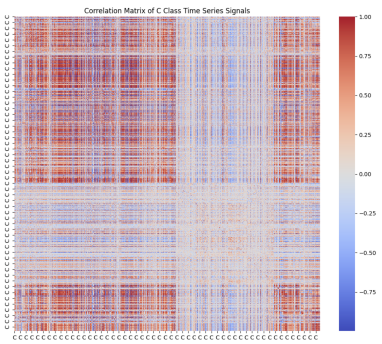
The dataset consists of 48 thousand time series of variable length, scaled between *0* and *1*. We plotted random signals for each class to see whether there were noticeable patterns, to no avail. We did notice however, that some signals seemed to clip or otherwise had random spikes that we strongly believed could be considered errors; we wrote two functions to detect signals with too low of a gradient (i.e. relatively flat) for extended periods of time and signals whose consecutive points seemed to “spike”, respectively. We also plotted these “outlier” signals by category (*see image on the left*), and found that they mostly belonged to class **B**.



In the end - and this is pretty evident in the image - we found that it was pretty hard to classify entire signals as “outliers” or not, since most of those we found only had imperfections in some portions, but where otherwise perfectly acceptable; the problem was that we had no information on how the data was sampled, the temporal scales, etc., so we had no real criterion to remove entire signals that just “looked off”.

In addition, since the original classes of the signals only indicated provenance from a specific domain and not some intrinsic time series properties, we tried plotting both inter and intra-class correlations (*see images on the left*), finding out that classes **C** and **F** had a noticeable percentage of completely unrelated signals. Any class correlated with class **F** showed the same pattern of non-correlation.

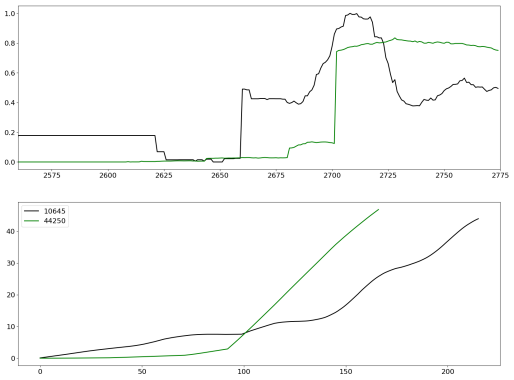




As a final step, we tried computing each signal’s autocorrelations to try and spot potential “error” points (*see image on the next page*) and we did notice that high discontinuities resulted in elbows in the autocorrelation graphs, but again defining a threshold for removal proved difficult in practice for the reasons mentioned in the plateau and spike detection step.

In the end, we settled on analyzing *windows* obtained by cutting the signals to 200 points, rather than entire signals - the rationale being that singular windows may be entirely composed of “bad” signal data.

Data Preprocessing



After testing, we concluded that the only windows worth removing were the ones containing long flat trends. About spikes in the series, removing these “outliers” may intuitively be beneficial to train the model on the best possible data, but this wasn’t the case, possibly because the test signals also exhibit these imperfections; also, subjecting the model to this transient noise could make it more robust to similar patterns in the test set, and help it “smooth out” sudden spikes in an otherwise pretty flat signal.

We tried different techniques to further normalize the time series. We tried removing the median of the signal in different ways; for example, removing the median of the window from the window and the telescope during training, removing the median computed on the union of both window and telescope, etc. The method that improved the training the most was removing the median of the window from the window and the median of the telescope from the telescope separately. We tried also normalizing with IQR and using the mean but without significant improvement to the before mentioned method.

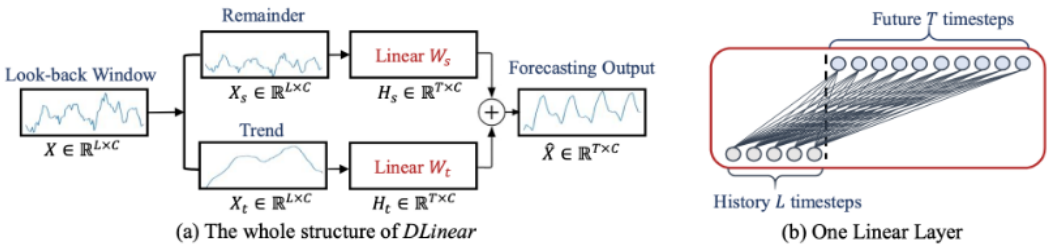
Median Regressor: Of course, normalizing the windows by removing the median meant that we had to build a regressor model that predicted the median of the telescope so that we could add it to the prediction of the model to get the final prediction. By doing so the error of the forecasting model and the median regressor are accumulating, but in most cases, we found it improving the overall performance. We tried multiple architectures for the median predictor as improving it would improve the final prediction greatly. The best architecture was a simple DNN with 4 layers of 200, 400, 400, and 200 neurons each, which had (locally) marginally worse performance than a RNN architecture composed of 2 GRU layers of 200 units each with a 20% dropout between the two layers.

Notable Attempts - LSTMs/GRUs, DLinear, PatchTST

LSTMs/GRUs: We tried various RNN architectures. Using 1, 2, and 3 LSTM [4] layers with different numbers of neurons each. We tried using bidirectional layers for some, or all layers. We also tried using GRU [1] layers. Ultimately, the best architecture turned out to be a Fully-RNN with 2 GRU layers with 200 units each. Adding an extra layer did not bring any significant benefits.

We tried to add dense layers and/or convolutional layers after the recurrent layers but this also did not improve the performance with respect to the FRNN architecture.

Dlinear [7]: We implemented this model because it offers decent performance while being very lightweight and fast to train. The model splits the signal in its rolling mean (trend) and in its residual (remainder), it applies a dense layer to each of them separately and then combines them for the final prediction (*see image below*). This allows the model to have a very low number of parameters, while being very accurate. In fact, this model turned out to be one of the best performing.



(a) The whole structure of *DLinear* (b) One Linear Layer

PatchTST [3]: PatchTST is a SOTA transformer model for time series forecasting. It works by generating small "patches" of the window, generating embeddings for those patches with 2 learnable weight matrices, which passes through a stack of encoder modules, each made of one Multi Head Attention layer and a FCN, both with skip connections and batch normalization, and finally the dense output layer at the end. The model managed to reach similar results to our final model.

Unsuccessful Attempts: Clustering, ResNets, T2V, (Transformers)

Clustering: Having noticed the correlation issue previously mentioned, we tried grouping the signals using Dynamic Time Warping [6] for clustering, but the algorithm never managed to group the signals in a meaningful way, with our data preprocessing.

ResNet: One of our attempts in building a TS forecasting model was through a custom ResNet, using 1-Dimensional convolutions: This was initially a promising model, though adding layers and regularization only contributed to worsening performance. To counter this we tried using an attention mechanism that slightly improved the model, reaching a validation loss near 0.01 with a telescope of 18: still, a poor performance when compared to our best-performing models.

Time2Vec Embeddings [2]: In our approach, we utilized the Time2Vec model to transform time series data into embeddings. These embeddings are generated using a periodic function equipped with learnable parameters. This unique feature of T2V allows us to manipulate and analyze the time series data at multiple scales of time. To exploit this, we tried integrating them into subsequent layers of GRUs and DNNs, and in both cases we ended up with an extremely time-efficient model, but with average performance, with the DNN being the slight favorite.

Final model - Ensemble model - GRU/DLinear

During the development phase, we managed to lower the test set's *mse* by averaging the telescope 9 predictions of the model based on GRU layers and DLinear before adding the predicted median. We believe that the different architectures are able to work well together. By changing the telescope to 18, the results of the ensemble model were only slightly better than the GRU model alone, that is due to DLinear not being able to scale its prediction capability on a longer telescope, since the complexity of the problem increases while the number of its parameters is still very low. The *mse* on the test set went from 0.00469983 in the development phase to 0.01051551, a 2.34x increment.

Final Considerations and Possible Improvements

On all the trials we did, during both phases of the competitions, making the predictions on the full telescope has worked out better than making an autoregressive version of the same model.

Regarding the building of the dataset, we have built windows of length 200, corresponding to the length of the signal in the test set, with a stride of 10, then randomly splitted the windows into train and validation sets. With a telescope of 18, though, we have an almost 50% overlap between couples of telescopes of different samples, and by randomly splitting the dataset, this would probably make the validation metrics not as effective on indicating the generalization of the model. Finally, we could have tried out different lengths for the windows (instead of only 200, which we choose considering the length of the test set's samples), and, considering the huge increment of the *mse*, focusing only on developing the best possible model on a telescope of 18 looking forward to the final phase of the competition, by cutting the last 9 time steps of each predictions during the development phase.

Contributions

Leonardo Brusini (*LeonardoBrusini*):

- PatchTST
- DLinear
- Full LTSM/GRU
- DLinear - GRU ensemble
- Basic Transformer model
- Median rescaling, FCNN and GRU model for target's median regression

Marco Scarpelli (*MarcSka*):

- Dataset inspection and outlier detection
- Functions to pipeline data extraction with *tf.Data*
- Performance analysis with TensorBoard and optimizations
- PatchTST
- Tests with GRU and median regressor

Davide Tenedini (*DavideTenedini*):

- Dataset Inspection
- Class predictor (before realizing that the classes were provided during testing)
- Median Regressor (FCNN, DNN, LSTM, CNN)
- LSTM/Bi-LSTM
- DLinear

Federico Toschi (*ftoschi*):

- Correlation analyses
- Clustering
- ResNet 1D
- Tests on RNNs, LSTM/Bi-LSTM nets
- DLinear w/Attention
- Custom CNN + DLinear net
- Experiments with T2V Embeddings

References

[1] Cho, K., van Merrienboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., & Bengio, Y. (2014). Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. arXiv [Cs.CL]. Retrieved from http://arxiv.org/abs/1406.1078

[2] Kazemi, S. M., Goel, R., Eghbali, S., Ramanan, J., Sahota, J., Thakur, S., … Brubaker, M. (2019). Time2Vec: Learning a Vector Representation of Time. arXiv [Cs.LG]. Retrieved from http://arxiv.org/abs/1907.05321

[3] Nie, Y., Nguyen, N. H., Sinthong, P., & Kalagnanam, J. (2023). A Time Series is Worth 64 Words: Long-term Forecasting with Transformers. arXiv [Cs.LG]. Retrieved from http://arxiv.org/abs/2211.14730

[4] Sepp Hochreiter, Jürgen Schmidhuber; Long Short-Term Memory. Neural Comput 1997; 9 (8): 1735–1780. doi: https://doi.org/10.1162/neco.1997.9.8.1735

[5] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., … Polosukhin, I. (2023). Attention Is All You Need. arXiv [Cs.CL]. Retrieved from http://arxiv.org/abs/1706.03762

[6] Wang, W., Lyu, G., Shi, Y., & Liang, X. (2018). Time Series Clustering Based on Dynamic Time Warping. 2018 IEEE 9th International Conference on Software Engineering and Service Science (ICSESS), 487–490. doi:10.1109/ICSESS.2018.8663857

[7] Zeng, A., Chen, M., Zhang, L., & Xu, Q. (2022). Are Transformers Effective for Time Series Forecasting? arXiv [Cs.AI]. Retrieved from http://arxiv.org/abs/2205.13504