Team: ***bisogna_tenersi_idratati_nelle_ore_più_calde***

Time Series Classification Report

This document will explain the pipeline we used to process the **multivariate time series** given as input and subsequently train a model to classify them.

NOTE: No outlier detection and subsequent cleaning were applied since we didn't know details about the meaning (domain) of the dataset.

1. Pre-processing

Differently from the previous challenge, this time we focused heavily on the pre-processing of the data. We tried several normalization, standardization, and feature-crafting techniques. All implemented preprocessing methods can be found in the *libreriabella.py* file present in the same folder as this file.

Feature Building

Note: we are improperly referring to normalization and standardization as features because we think it makes sense in this context.

Techniques we used to preprocess the data / create new features include:

- **standardization with z-score** (global and sample-wise): we tried it both by using the global mean and std of each of the 6 dimensions of the dataset and also by computing it "locally" for each 36 sample group. Global standardization seemed to generally perform better.
- **normalization** (global and sample-wise): again was computed both "globally" and "locally".The sample-wise (local) version performed better in our tests.
- **log**: we simply applied a logarithm to all datapoints in order to rescale values in a more manageable range and give less importance to very high ones.
- **lognorm:** we also did a combination of logarithm and normalization in order to have a more usable feature alongside the others already normalized.
- **logstd:** same but with standardization.
- **Robust scaling**: this Scaler removes the median and scales the data according to the Interquartile Range (IQR). The IQR is the range between the 1st quartile (25th quantile) and the 3rd quartile (75th quantile).
- **FFT**: we tried using the fast Fourier transform as a feature. The network managed to train successfully also when using this feature by itself. Merged with other features it apparently gave no advantage and we, therefore, discarded it.

All these techniques were not simply applied one by one separately. We used multiple possible combinations of them as input to the model until we found the best one.
To clarify: the input to our model was not just an **NxTxF** array (N=number of samples, T=number of timestamps and F=number of features) but instead, F was a much larger number like for example F=6*4=24 if we decided to combine 4 of the features presented above.

The best combination of features we found was [standardization, norm, sample-wise_norm, std(log)] and so that is what we used for 2 of the 3 models present in the ensemble of the final, best, submission (see Section 3 - SULAUZYTAS).

Data augmentation

In order to have more data to train the model and as a consequence have less overfitting we tried several data augmentation techniques. Here is a comprehensive list of the most interesting ones:

- **value interpolation**: we created new time series by sampling the mean value between each couple of points of the original time series.
- **window reduction**: we reduced the window in the samples e.g. 34 and created new samples with this smaller window. We then created multiple predictions for the different versions and voted between them.
- **time series resampling**: we tried guessing that the samples of the same classes were simply separated using a window of 36 from a bigger one before being delivered to us. We therefore resampled the whole dataset with again a window of 36 but creating way more samples by using a hopping window with small stride instead of the tumbling window that was probably used to create the samples given to us. The results were not that good and we had no evidence that the approach actually made sense for the data that we had. We discarded the approach.
- **under-sampling of class 9**: using a data generator we gave the *model.fit( )* method less class 9 samples in each epoch. That is because class 9 had way more samples compared to the other classes.
- **gaussian dropout** (not properly data augmentation but still…)**:** we used the gaussian dropout layer to add noise to the input, it gave a slight enhancement to the results.

---

2. Models

Different kinds of models were tried. All things considered, convolutional ones were definitely the better ones.

Since the dataset was heavily unbalanced towards some classes we applied **class weights** to some models during training. This yielded some small improvements but nothing too relevant.

Dense Models

To see if there was potential we tried a few models composed of **only dense layers**. The idea was that maybe we could have combined their features with that of convolutional models when ensembling in order to have more diversity in the extracted features.
Results when ensembling with feature concatenation were poor and so these models were discarded.

Recurrent Models

Both standard LSTMs and bi-directional LSTMs were tried but results were disappointing to say the least.
No matter how we assembled the layers they seemed to perform worse than their convolutional counterparts.

Convolutional Models

As previously stated convolutional models were definitely the better ones according to our experiments.

Other than standard Conv1D networks we tried some fancier architectures including skip connections, processing the input with a Conv1D and Conv2D pipeline in parallel, ResNet1D and Inception1D.

The simpler architecture without anything fancy worked the best and interestingly, in contrast to what happens with image classification, **shallower convolutional models seemed to perform better than deep ones**.

---

3. Model Ensembles

Mainly two ensembling techniques were tried and one, voting, worked really well, increasing performance on the codalab dataset by around 2%

- **feature concatenation**: we concatenated the features extracted by 3 different networks and attached a common dense layer at the end to perform the final classification. Performances were surprisingly worse.
- **voting**: we run the three models separately and used a simple algorithm to predict the most voted class between the 3.

SULAUZYTAS

Ensemble model that does **voting** between 3 convolutional models. This was the best model we tried.
By using this model we obtained an accuracy score of **0.73** on Codalab 2nd phase.