Artificial Neural Networks and Deep Learning - Challenge 2
Time Series Classification

December 23, 2022

Riccardo Luigi Aielli
Salvatore Buono
Simone Buranti

POLITECNICO
MILANO 1863

Contents

1 Introduction

The goal of this project is to build a neural network model able to classify time series into 12 different classes. In this report, we illustrate the process that brought us to build a model that reached 73% accuracy on the test set. All our work has been done on Google Colab as it offers free computational units for code execution. Throughout the challenge, we used a bottom-up approach starting from simple models and progressively increasing their complexity.

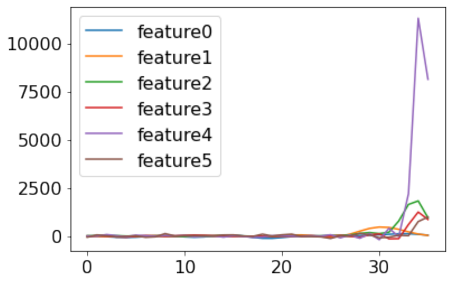
2 Exploration Data Analysis

2.1 Dataset split

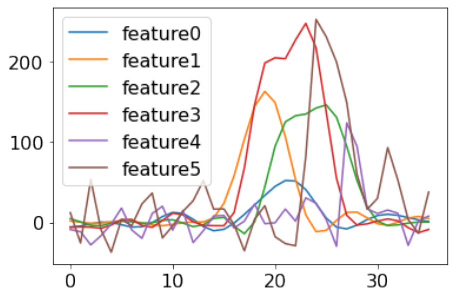
We divided the entire dataset into training, validation and test sets with a ratio of 80/10/10. In particular, we selected for the validation and test sets the same percentage of images from each class.

2.2 Preprocessing

During this phase we studied the shape of our data and tried to understand their common properties. For example this is how samples appeared to us:



(a) Sample belonging to class 0



(b) Sample belonging to class 1

Figure 1: plots of samples belonging to different classes

2.2.1 Standardization

Standardization of a dataset is a common requirement for many machine learning estimators. Typically this is done by removing the mean and scaling to unit variance. However, outliers can often influence the sample mean and variance in a negative way. In such cases, the median and the interquartile range often give better results. So we applied as preprocessing `RobustScaler()` that removes the median and scales the data according to the quantile range.

2.2.2 Feature Reduction

In order to prevent overfitting, we tried to select a subset of features considering the correlation coefficient between each feature and the target variable, then we extracted the correlation coefficients for each feature at each time step and finally selected the indices of the features with the highest absolute correlations at each time step. However, this approach did not improve our performance, since it decreased the training accuracy without increasing the validation one.

2.2.3 Window Sliding

This method refers to the process of dividing a time series into overlapping segments, or windows, and using each window as an input to a classifier. Each of these windows can be used as an input to a classifier, which would make a prediction based on the values within the window. The predictions made by the classifier can then be combined in some way to make a final prediction for the entire time series. We noticed that window sliding is a useful technique for time series classification because it allows the classifier to take into account the dependencies between time points in the time series. By using overlapping windows, the classifier can consider both the the values at current and previous time points when making a prediction. If on the one hand this can improve the accuracy of the prediction, in our case it led to overfitting and did not improve performances on the Codalab platform test set.

2.2.4 OverSampling

We noticed that data weren't equally distributed among the 12 classes, so, to reduce the scarcity of data of the poorest classes, we used oversampling, which increases the number of samples by generating new ones from existing data.

2.2.5 Data Augmentation

We performed data augmentation through 2 different approaches. Firstly, we implemented by hand a function that modifies training data by adding some noise, scaling, shifting and cropping, but this did not improve our accuracy. Then we used library `tsaug` in order to augment in a more efficient way, cropping subsequences with length 36, random quantizing to 10, 20, or 30 level sets, with 80% probability, random drifting the signal up to 10% - 50%, with 50% probability, reversing the sequence, and, in the end, concatenating our original data with augmentated ones.

2.3 Trying some networks

Starting from exercise sessions, we decided to adapt the networks presented during the lectures (LSTM, BiLSTM, Conv1D) editing both the input shape and the class parameter. Then, we increased the number of hidden layers adjusting their number of units and setting dropout layers to reduce overfitting. Finally, we obtained the best model combining Conv1D with BiLSTM. In order to compare performance from different types of architecture we made some prediction on the test set, and to improve them, through confusion matrix, we saw in which classes the model performed worse.

2.3.1 Adding the attention layer

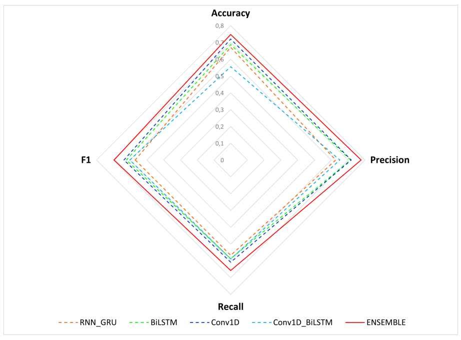
Following the advices given in the logbook file, we implemented a type of Attention mechanism between LSTM layers. This can be helpful because it allows the model to focus more on the most relevant time steps and less on the less relevant ones, potentially improving its performance. However, they are computationally intensive to train and implement, since they require the implementation of a custom layer that computes attention on the positional/temporal dimension. Thus we decide to not continue.

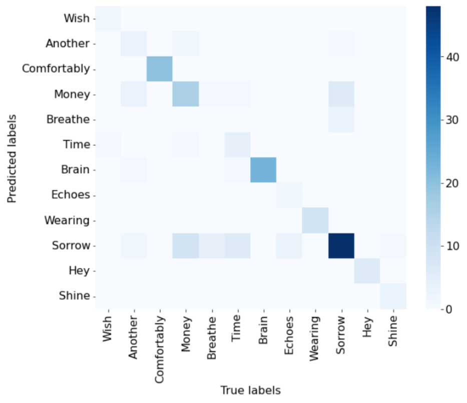
3 Ensemble

As explained above, our first approach to the challenge was to focus on a single model. However, after several attempts, we noticed that each architecture, when optimized, struggled to obtain an accuracy on the test set higher than 70%. If a single network could not break such a barrier, maybe more networks might succeed instead.

3.1 Improving the accuracy by combining predictions

Ensemble methods are numerous and, according to their implementation, they are suitable for different purposes. What we wanted to achieve was to take advantage of the whole information learnt by our architectures by combining predictions from different feature extraction processes. Thus, we computed for each class the average of the scores assigned to it. For this method to be effective, the involved predictions must rely on intrinsically different architectures. Firstly, we included the best versions of LSTM, BiLSTM, and Conv1d. With this arrangement the method already showed notable improvements on our test set. Despite this, we understood that the ensemble might benefit from a more differentiated set of architectures, as LSTM and BiLSTM differ uniquely on the direction of the timeseries' inspection. Therefore, we implemented a new architecture based on RNN and GRU layers being keen to optimize it, as the advantage brought by a new model might be lost if its performances are sensibly poorer than the others.
The inclusion of a fourth model made the ensemble obtain a 4% higher accuracy than single models and clear improvements on precision and recall, with a gain of about 8-10%.





These local scores were plainly confirmed when we tested the new model on the platform test-set, reaching an overall accuracy of 73,38%.

3.2 Further developments

Having understood the potential of this method we further investigated possible variants. Our goal was to find the model combination which generated a slightly higher class probability for the correct one the most of the times. Thus, we adopted the weighted sum combination of predictions. In order to find the best weight vector, we tested all the weight combinations against our local test set. In this phase, we simply included in the group every architecture we had implemented, as counterproductive networks would have been excluded by null weights. However, we were not able to properly exploit this method as, due to the scarcity of test samples, the results were strictly split-dependent, and the submitted weighted versions obtained poor results.