Gatti Carlotta (ID: 10623228) – Montenegro Martina (ID: 10632933) – Zanella Francesco (ID:10835648)

HOMEWORK 2 – TIME SERIES CLASSIFICATION

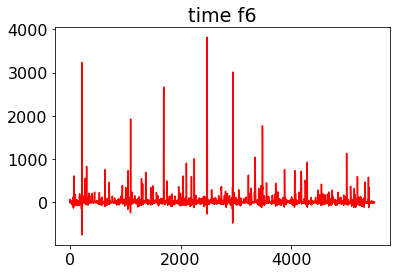
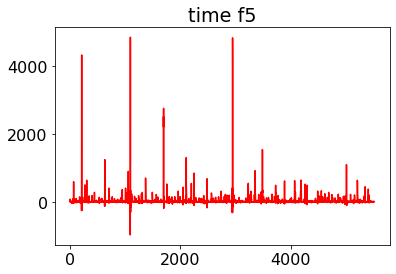
Time series is a sequence of data points that occur in order in a defined period. In our case, there were 2429 windows of length 36, and for each sample there were 6 features. Using supervised learning, it is possible to build a classifier able to predict the class to which a new dataset belongs.

DATA ANALYSIS

For building a robust classifier the best start relies on understanding the data. Therefore, samples had been split according to the label of belonging: it was clear that there was an outstanding issue of class imbalance; to report an example, class labeled as ‘*Wish*’ had 34 samples, while ‘*Sorrow*’ 777.

As a second step, a plot was built:

Immagine che contiene testo, antenna

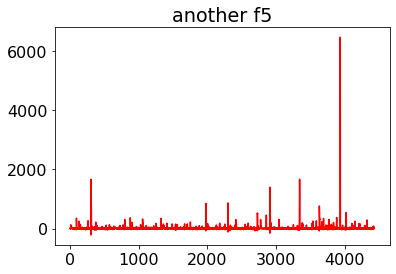
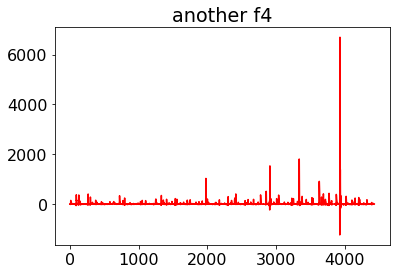
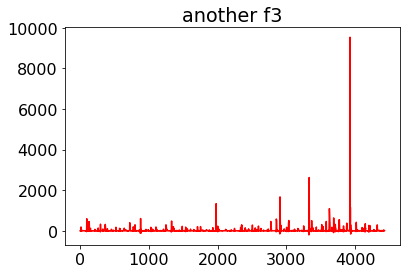
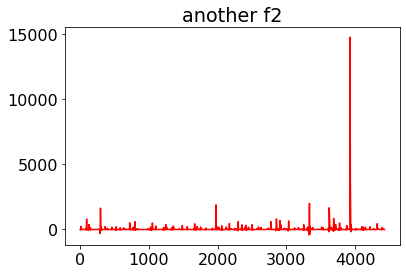
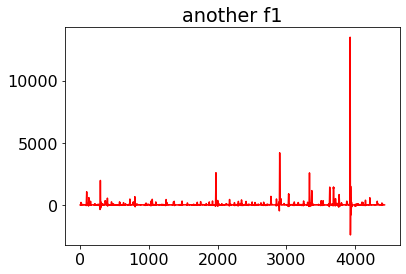
Descrizione generata automaticamenteImmagine che contiene testo, antenna

Descrizione generata automaticamenteImmagine che contiene testo, antenna

Descrizione generata automaticamenteImmagine che contiene testo, antenna

Descrizione generata automaticamente

It is shown the plot in all the time stamps of the class ‘*Time*’, divided by features (on the x-axis the timestamps, on y-axis value of data point). This was the class that has always reached 0.0 as F1-score on test accuracy – it shows many peaks and not very different in height. Thereby, the hypothesis is that by normalization or standardization the characterization of the signal is lost. This is better justified when looking at other labels. For example, ‘*Another*’:

Immagine che contiene testo, antenna

Descrizione generata automaticamente

In this case, it could be possible to generalize saying one high peak is particularly characterizing the class, which is far apart in height from all the others.

Looking at all classes, the decision was that all features were as important as the other ones, after being confirmed by feature importance measurements. To be even more sure, models were trained without some features and no improvement has been seen.

DATA PREPROCESSING

After having split the dataset using ‘*train\_test\_split’* function, with validation set accounting for 10%, many ways of scaling have been tried. The first attempt was ‘*MiniMaxScaler*’ applied per feature; in this case, validation accuracy was stuck at a fixed value from the beginning to the end of training. The same result was obtained through Peak Normalization per feature (*i.e.,* dividing each element in a column by the maximum absolute value of the column itself). Subsequently, ‘*RobustScaler*’ has been reharsed, but still the best result has been achieved by subtracting the mean and dividing by the standard deviation for each matrix (whereby matrix it is meant a window with all the features).

Regarding the window size, it is usually chosen by the user, while, in our case, data were given already shaped as presented before. Consequently, given that class imbalance matter has been pointed out before, we tried to change the window size, so that samples belonging to different labels looked less badly distributed. We tried both to enlarge and to shrink the window size. We reduced the window size from 36 to 18 or even 6 timestamps. Doing this we obtained in every model a worst accuracy. So, we tried to double the window size. We took the elements belonging to the same class and we concatenated them in pairs. In this way, we ended up with half the samples but, for each sample, we had 72 timestamps. Therefore, the validation accuracy increased in all the models we tried; we obtained good results in model with Bidirectional Long Short-Term Memory (*BiLSTM*) layers. We couldn’t apply the same preprocessing to the test set since, obviously, we didn’t know to which class the elements belong. To overcome this, we enlarged the window size of the test set concatenating each sample with itself or with the mirrored itself or simply adding 0 in all the other values. The results with the first technique were not bad but many degrees of accuracy were lost anyway.

Downsampling has also been applied without luck – samples from the most represented class (‘*Sorrow*’) have been randomly dropped off with no relevant improvements.

DATA AUGMENTATION

Data augmentation can be a useful technique for increasing the size and diversity of a dataset for time series classification tasks. The most appropriate method to augment data depends on the characteristics of the data and the requirements of the task. Since, it is important to ensure that the augmented data is representative of the types of patterns and variations that the model should learn.

Since the total amount of samples is 2429, which is low for training a deep learning algorithm, data augmentation has been applied. Above all, ‘*RandomHeight*’, ‘*RandomWidth*’, ’*RandomZoom*’ and ‘*RandomFlip*’ from *Keras* has been introduced in the model – in each model we built, they brought about improvements. Parameters have been adjusted by trial and error.

To further increase data, ‘*np.random.normal’* was used to generate random normal noise (mean equal to zero and standard deviation equal to the one of the matrices), thus it was added to each column of each matrix. Noise was scaled by a factor (namely, 0.25) tuned by hand.

Finally, data were also shifted by a random amount through ‘*np.random.randint*’ and scaled by a random factor by ‘*np.random.uniform’*.

BUILD THE MODELS

Long Short-Term Memory networks, already cited, are an extension of Recurrent Neural Networks that extend the memory. Generally, the advantage of *RNN*s is that they share weights for each position of the input vector, so that sequences with different lengths can be processed. *LSTM*s, introduced to solve the problem of vanishing and exploding gradients, recognize patterns in data sequences, since through a special architecture made of gates, the model decides whether to retain previous information in short term memory or discard it:

* *Forget gate*: how much information from previous step should be discarder
* *Input gate*: how much information the current memory will receive (what and whether to write)
* *Output gate:* controls the value of the next hidden state

When referring to Bidirectional *LSTM*s it means having two *LSTM*s, one taking the input in a forward direction, and the other in a backward one. It is useful and important since every component has information from both past and present.

Many models have been developed, trying different combinations: for example, merging Bilateral Long Short-Term Memory with Skip connections (identity shortcut connections, not adding parameters). Main role of Skip Connections is improving gradient flow through the network; even though our models were not deep, improvements were noticed when Skip Connections and *BiLSTM*s were used together.

Nonetheless, after tuning properly the parameters of the noise added to augment data, these models started showing overfitting, even when reducing layers, number of neurons or when using regularization terms. They became too complex, for a dataset that was finally brought to its optimal status, as shown below (*BiLSTM+Skip*), and highlighted by the comparison to the found optimal model (*1Dconv*).

In fact, in the end, the model with the best performance is the one with only 1D Convolution (*Conv1D*)layer: which is a way to compress the volume by reducing the depth without reducing the spatial extent. It substantially and impressively reduces the number of parameters, speeding up the training and helping in avoiding overfitting. Parameters have still been tuned by hand, to obtain 0.7046 accuracy on an external test set.

