Report 22 - GxG: no genre distinction

Anonymous ACL submission

1 Introduction of the Dataset

The task I worked on aims to correctly classify the gender of individuals who have written specific tweets. The task is formulated as a binary classification task, where we have two genders, male and female. The purpose of my work is therefore to accurately predict the gender of the person simply by knowing the tweet they have written. As for the dataset, I can say that it is balanced, with a 50-50 distribution. In particular, I was provided with two sets of data, the training set, which I subsequently split into training and validation sets, and the test set. Both provided sets are roughly the same size. On the second page, you can find a chart showing the number of samples for each class.

2 Architecture of my model

Regarding the architecture of the model I implemented, first, there is an Embedding layer, which is responsible for converting the input tokens, represented as integers, into dense vectors of fixed size. Next, I implemented a BiLSTM layer, which processes the input sequence in both directions, thus obtaining more information. The third layer is a projection layer, which maps the hidden states obtained from the BiLSTM layer to the number of classes. I then implemented two fully connected layers. Basically, these two layers are used to allow my model to learn complex patterns in the data. I included a Dropout layer to prevent overfitting. Finally, I added a batch normalization layer for improving generalization

3 Design choices of my model

Regarding the design choices of my model, first of all, I chose to implement two fully connected layers as they help me learn complex patterns. In particular, during various implementations I tested, I noticed that the model performed better with the addition of a second fully connected layer, so I chose to keep it accordingly. As for the Dropout layer, I chose to include it to avoid overfitting issues in my model, as it could happen that neurons become too dependent on neighboring neurons. Additionally, it also makes the model more robust. Basically, I tested various models, some quite different from this one, but I chose to keep this one because it offered the best performance.

4 Baselines implemented

At the beginning of my notebook, I implemented 3 baselines: a Random baseline that selects a label uniformly at random among the possible labels, a Majority baseline always predicts the most frequent class in the training set, and finally a Stratified Baseline that predicts the classes following the distribution of labels in the training set. The results obtained from these 3 are in line with what I expected. In particular, as I mentioned at the beginning of the report, my dataset is balanced and has only 2 classes. Therefore, I expected an accuracy of the majority exactly equal to 0.5, and indeed I obtained what I expected. Also, regarding the random baseline, we are correctly around 0.5, and the same applies to Stratified.

5 Results section

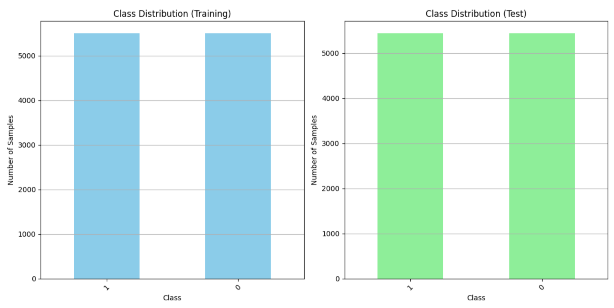
As for the results obtained from my model, I can say that they are slightly superior to the baseline regarding the test set. In particular, using my model, I achieved an accuracy of 0.5224 and an F1 score of 0.5142. I was not particularly satisfied with the results obtained, so I conducted some experiments to understand more about the datasets provided to me. It seemed to me that my results are also the result of the significant difference between the training and test data of the two datasets. I noticed that the learning on the validation set was very good, while it deteriorated significantly on the test set. To improve the performance of my model, I also chose to use "optuna" for hyperparameter tuning, aiming to achieve better performance from my model. On the third page of my report, you can find, in addition to the table showing the results of my three models, also the table related to the best hyperparameters I found, specifically referring to table number 2. On the fourth page, instead, in Figure 2, you can see the trend of the Loss, accuracy, and F1 score in the validation set.

6 Instructions to run my code

The Notebook is divided into 4 sections, the first one concerns the baselines, the second one the LSTM model implemented by me, the third one the CNN network (an additional model that I chose to keep as a benchmark), and finally the Word2vec model. To run the code, simply run all the cells of the notebook in order. (The notebook should be run using the GPU)

7 Simpler model

As for the simpler model to implement, I chose to test two approaches. First, I implemented a CNN to see how it could perform with the provided datasets. Then, I implemented a simple model using Word2Vec to create embeddings (vector representations) of the words in the dataset sentences. These embeddings are used as features to train a logistic regression model. The performance of this latter model is almost similar to the previously described model, with an accuracy of 0.5454. What changes are probably the performance of the F1 score, which is equal to 0.4940. Regarding the CNN, I chose to keep the implementation, as I obtained results very similar to the main model, with the accuracy being nearly the same. However, the f1 score is much worse, demonstrating that a CNN is not the best choice for this type of dataset. As for other implementations of this kind, I tried various models, such as a fully connected one, without achieving excellent results. Consequently, only these two models seemed interesting to me. The implementation of the CNN, in particular, was more of a personal curiosity that arose after several attempts to test different models. Similarly, in this case, both the accuracy and F1 scores obtained are included within table number 3 on page 3, along with the best hyperparameters I found for the CNN. On page 4, you will find, just like for the LSTM model, the trend of Loss, accuracy, and F1 score in the validation set.



Hyperparameter	**Value**
Embedding Hidden Dimension	216
Num filters	116
Filter sizes	[5, 6, 3]
Dropout$_{prob}$	0.3920

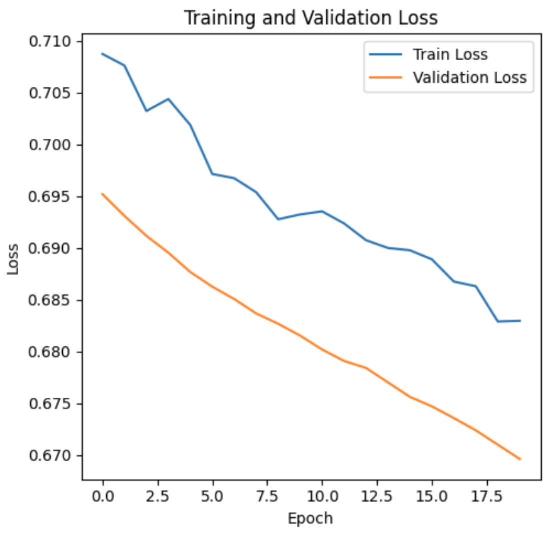
Table 1: Final hyperparameters for CNN model

Hyperparameter	**Value**
Learning rate	1.4599e-05
Dropout	0.2867

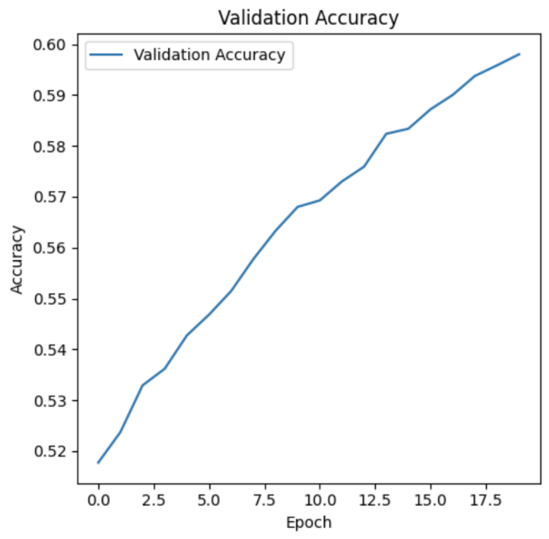
Table 2: Final hyperparameters for LSTM model

Model	**Loss**	**Accuracy**	**F1-score**
LSTM model	0.6945	0.5224	0.5142
CNN model	0.8216	0.5419	0.4892
Logistic Regression model with Word2Vec	NULL	0.5224	0.5142

Table 3: Results of my models on the test set

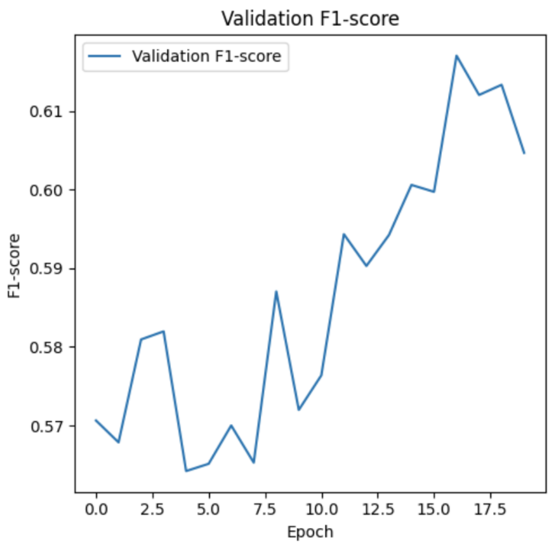


(a) Loss of LSTM



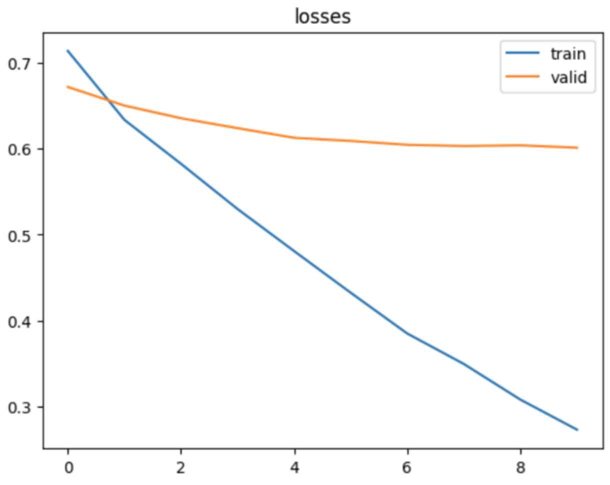
(b) Accuracy of LSTM

Figure 1: LSTM

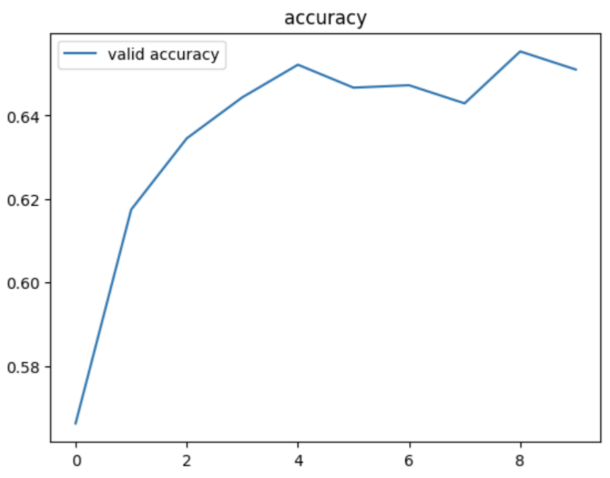


(a) F1 of LSTM

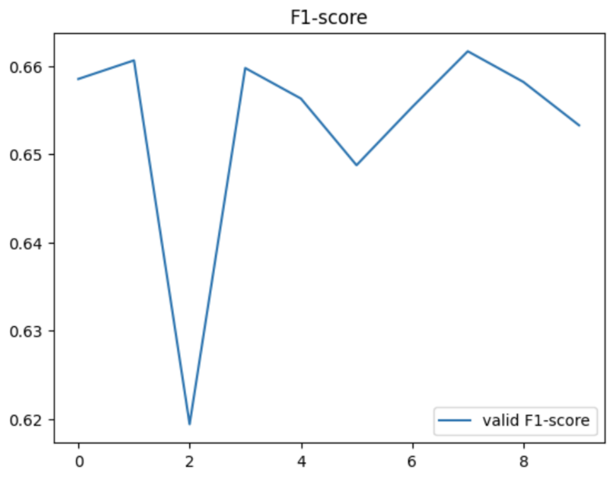
Figure 2: LSTM model



(a) Loss of CNN



(b) Accuracy of CNN



(c) F1-score of CNN

Figure 3: Caption for all images