INF554 - Team AJA report

For this problem of extractive summary generation, we chose to work with a Graph Neural Network (GNN) structure. After several attempts on standard datasets, we concluded that this structure best captures the behavior and dependencies within the graph. We added all relevant information as features.

1 Feature selection and extraction

1.1 Features creation and selection

Let's first revisit the choice of features. Regarding text features, the selected ones are as follows :

- We created a `sentence_length` feature that calculates the number of words in a sentence. See Figure 1 below, we observe that longer sentences tend to be more important than shorter ones.
- The TF-IDF score is a measure of word occurrence weighting in a document group. TF calculates the word frequency, and IDF calculates its rarity. As features, we chose to consider, for each sentence, the sum of word scores and the maximum score, see Figure 2 and Figure 3. For this feature, we used research done in the paper [1]
- We found that the presence of interjections (uh, um, ah, okay, oh) or markers such as '<vocalsound>' can be correlated with the label : we identified this looking at the most frequent words for sentences labeled 0 or 1 (except for stop words).
- After analyzing the correlation between the target variable and the presence of long words in a sentence, we added a feature counting the number of words exceeding 8 characters. See Figure 5.
- We also encoded the speaker type with one-hot encoding (OHE).
- Finally, since the model can only process numerical data, we performed text embedding using BERT.

As graph features, after several attempts, we decided that used a GNN would be best in order to have all the graph structure as a feature. The model's structure is described in a following subsection.

We also added OHE on the type of relationship between two sentences to incorporate this information as features.

We took care to normalize all our values to avoid influencing the model in case the data scales were very different among certain features.

1.2 Experiments to test feature performance

To test the performance of the features, we plotted various graphs (as mentioned in the previous section) and a correlation matrix see Figure 4.

To study the impact of features and their combination, we simply studied the model's performance with the addition of each feature, keeping track of the results. We also attempted to use SHAP values to enhance the model's interpretability and observe the impact of each feature. Unfortunately, our model is too customized to use this library.

2 Model choice, tuning, and comparison

2.1 Comparisons with other reference models

To compare with standard models, it is necessary to "flatten" the graph structure. We initially started working on such a structure. However, this flattening loses certain inherent graph information.

We tested different models with tuned parameters : KNN, RandomForest, SVM, whose results are presented in the following table :

Model	**Parameters**	**Accuracy**	**F1-score**
KNN	n_estimators = 9	0.835	0.439
RandomForest	n_estimators = 5, 'entropy'	0.814	0.424
SVM	C=1.0, kernel='rbf'	0.849	0.488
GNN (selected model)	$\alpha = 0.15$, lr = 0.005	0.83	0.604

We quickly notice that for the given problem, the use of a GNN is much more suitable and leads to a better F1-score. Flattening the graphs for conventional models results in a loss of essential contextual information. Therefore, we prefer a model based on the Graph Neural Network for our work.

2.2 Our Deep Learning Model

Given the graph structure of the data, we chose a model capable of handling such structures.

Our model is based on the PyTorch interface. It takes a graph of the "torch_geometric.data" class with a vector for each node corresponding to features extracted from a sentence. A graph corresponds to a conversation. The model's output is a vector corresponding to the binary classification of each graph node.

To consider the graph structure, the model also takes adjacency matrix equivalents as input. There is one for each type of relationship between sentences. We perform a kind of "channel decomposition" like with an image.

The model consists of graph convolution layers (`torch_geometric.nn.GCNConv`) applied in parallel for each channel (each type of relationship in the graph). A dense layer then extracts the information. Finally, the output is a sigmoid for binary classification.

We use a loss function assumed to be suitable for imbalanced class classifications : `torch.nn.BCEWithLogitsLoss`.

2.3 Parameters tuning and overfitting

For parameter tuning, after some manual tests, we selected relevant intervals for our important parameters. Several parameters need tuning :

- Learning rate : essential for learning speed and preventing over/underfitting.
- Alpha : a parameter in the loss function related to class imbalance (class 1 is the minority).
- The number of training epochs : too many epochs led to overfitting, and too few epochs prevented the model from learning properly from the data.

It was quickly noticed that our model tended to overfit. This is evident when comparing the F1 score difference between validation and test sets : a pronounced difference (high validation F1 score and significantly lower test F1 score, around 15 to 20% worse) indicates overfitting. To address this, we chose to implement a bagging technique and to add a dropout layer to our model. This rejects part of the data during training and limits overfitting.

After determining the intervals of interest, we chose to automate parameter tuning with Bayesian optimization. Unlike Grid Search, Bayesian optimization relies on probabilistic calculations and may not provide the most optimal results. However, given our data, Grid Search was very time-consuming and yielded similar results, so we preferred Bayesian optimization with fairly broad search intervals.

3 Email adresses

- julien.collard@polytechnique.edu
- alice.gorge@polytechnique.edu
- arnaud.teinturier@polytechnique.edu

4 Appendix

Références

[1] S. Verma, V. Nidhi, *Extractive Summarization using Deep Learning*, Jan. 2019.

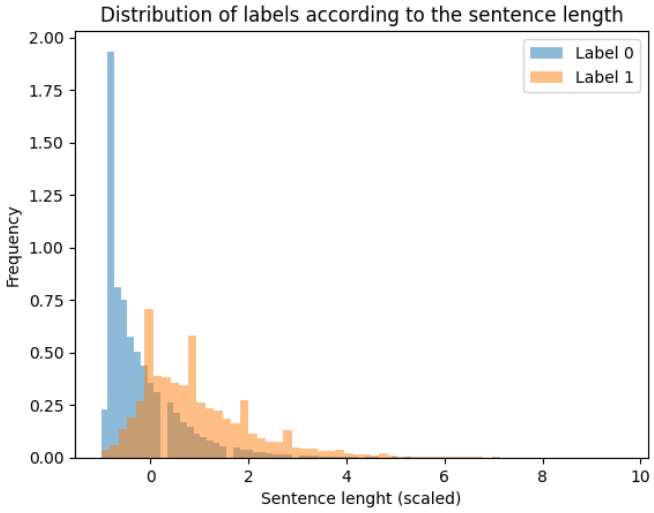

FIGURE 1 – Plot for `sentence_length` feature

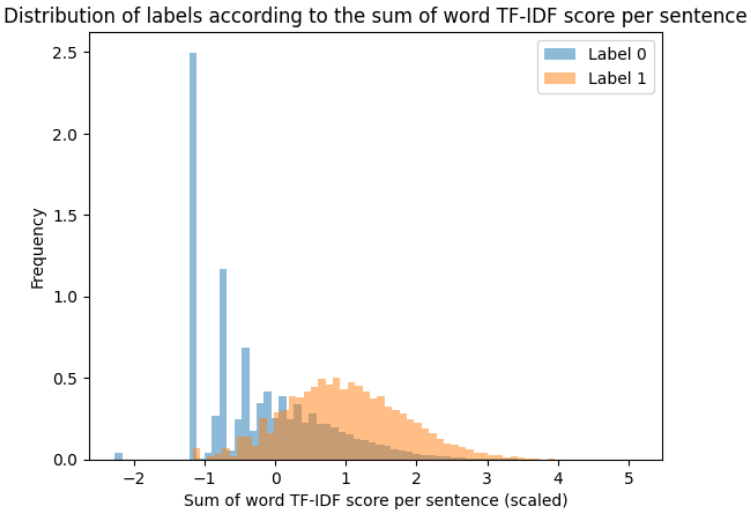

FIGURE 2 – Plot for `tfidf_sum` feature

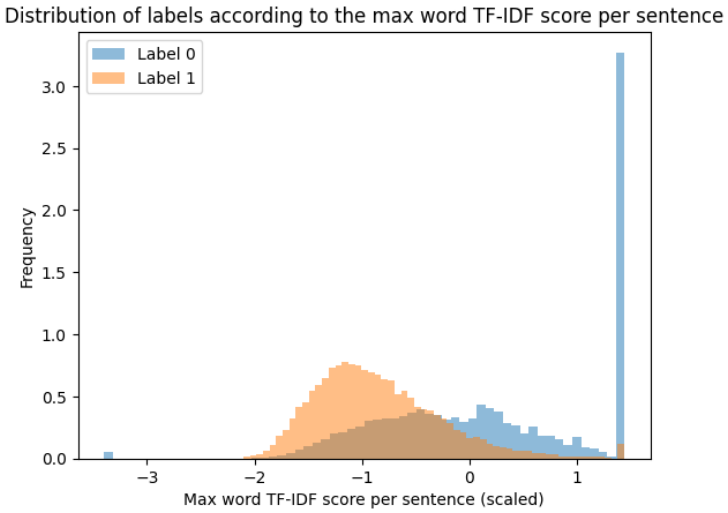

FIGURE 3 – Plot for `tfidf_max` feature

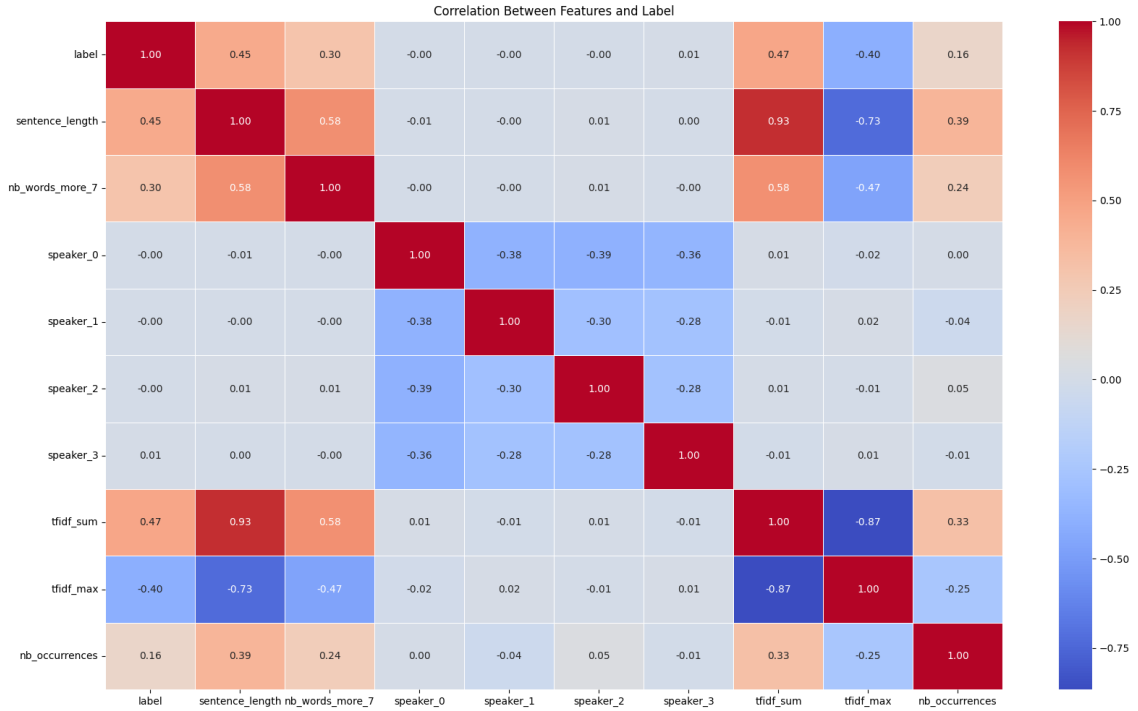

FIGURE 4 – Correlation matrix

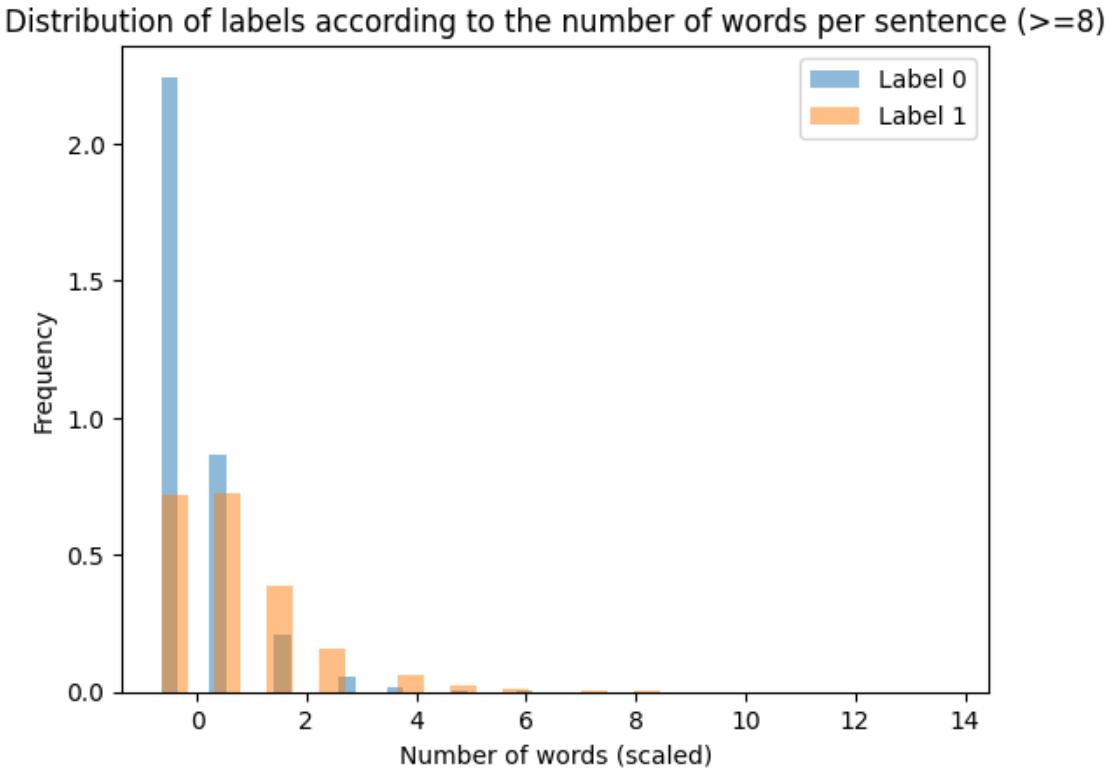

FIGURE 5 – Plot for `nb_words_8` feature