Project Description

In this project, I was building a model to predict the difficulty level of French sentences, based on the labeled dataset (consisting of about 5,000 French sentences). The important limitation was the constraint on the data augmentation: *"you are NOT allowed to take prelabelled data, but it is allowed to create the data augmentation on your own (automatically or not)".* My personal limitation was that I worked completely alone-independently, unlike other teams, due to the desire to better master the subject and the odd number of students.

In this document I will talk in detail (step by step) about the work done, a description of the GitHub and the results obtained. I hope you will enjoy this journey with me 😊

<u>The result</u>: **0.604** accuracy, and 12/20 place.

Description of the Files

New GitHub: https://github.com/aleksandr-dubinin/french-language

Old GitHub: https://github.com/aleksandr7du/french_sentences/tree/main

Main notebook: https://github.com/aleksandr-dubinin/french-language/blob/main/DS%26ML_DubininAleksandr_Migros_main.ipynb

My best trained model: https://github.com/aleksandr7du/french_sentences/tree/main/models

Streamlit application (best work): https://github.com/aleksandr-dubinin/french-language/blob/main/LogReg_app.py

Based on my model (it didn't work, also that GitHub was blocked.):
https://github.com/aleksandr7du/french_sentences/blob/main/streamlit_app.py

Big Journey *(description of the main notebook)*

I divided the notebook into 4 main parts:

1. <u>Pre-processing</u>: the simplest one, processing the data.
2. <u>Basic models</u>: trying the simplest models.
3. <u>Advanced models</u>: the most intense section of the report, including different models and techniques. Also, this part includes some subsections, highlighting the most important models/embeddings such as BERT, CamemBERT and others.
4. <u>Appendix</u>: this section includes some unsuccessful attempts and runnings of the code.

Basic Models

In this part, I quickly analyzed the following models: Logistic Regression, KNN, Decision Tree, Random Forest, and simple Neural Network. The Table 1 shows the results.

	Logistic Regression	**KNN**	**Decision Tree**	**Random Forest**	**Neural Network**
Precision	0.3	0.3	0.3	0.3	0.45
Recall	0.31	0.31	0.31	0.31	0.45
F1-score	0.3	0.3	0.3	0.3	0.45
Accuracy	0.45	0.33	0.31	0.42	0.45

Table 1. Basic models: results. (*The accuracy can be not reliable as I didn't upload it to the Kaggle.)*

After a short analysis, it became clear that simple models are not enough for high results. Accordingly, more advanced models based on neural networks and with embeddings are needed.

Advanced Models

Realizing that my data volume was quite limited, I realized the importance of generating new data. Therefore, as step 0 of this chapter, I took the **NLTK** model.

*"**NLTK** is a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries,…"*

BERT *(google-bert/bert-base-multilingual-cased) -* *https://huggingface.co/almanach/camembert-base*

My first full-fledged step was the implementation of the Bert model. At this step I immediately crossed the 54-55% accuracy mark, and one of the best results brought **56.2%** accuracy.

*"**BERT multilingual base model (cased)** – pretrained model on the top 104 languages with the largest Wikipedia using a masked language modeling (MLM) objective."*

Using the model with the **NLTK**, I was able to reach **57%+** accuracy, however the combination of BERT and NLTK & lemmatization technique didn't succeed. One of the first result was around 51-52%, and I preferred not to spend time on this. Further, I moved to the **CamemBERT** model.

It should be noted that I spent most of my time on both models (BERT and CamemBERT), selecting various parameters and coming up with optimizations, since they showed themselves best in the initial (and subsequent stages) + are considered one of the most developed in the world. I tried to take more advanced ones, but usually they are private: they are not available or require a paid API.

CamemBERT (*almanach/camembert-base*) - https://huggingface.co/almanach/camembert-base

"CamemBERT is a state-of-the-art language model for French based on the RoBERTa model. (More than 3M+ downloads last month.)"

This model performed better, first achieving 57.5% and then 58.5%. After which there was a relatively long downtime with a further breakthrough to the **60.4%** accuracy. *(More on this later.)* I also tested this model with NLTK, initially without a big success.

As I mentioned before, due to paying a lot attention to these "bert"-based models, I ran them a lot testing different hyperparameters. Some of the tests are available in the **Appendix**.

<u>What I learnt from this?</u>

I figured out how these parameters interact and influence model performance. First, the learning rate and batch size had a synergistic relationship: a higher learning rate required a smaller batch size to maintain stability, whereas a lower learning rate could be paired with a larger batch size for smoother convergence. Then, dropout and weight decay worked together to regularize the model, preventing overfitting. Too much dropout, however, could counteract the benefits of a carefully chosen learning rate and batch size, while weight decay required fine-tuning to avoid diminishing the model's capacity to learn from the data. Warmup steps also played a crucial role in stabilizing the initial phase of training, gradually increasing the learning rate to avoid the pitfalls of a high starting rate. At the same time. It was crucial to follow a certain number of epochs, finding a balance between fitting and overfitting. Finally, I used the Adam optimizer, which combines the advantages of both Adaptive Gradient Algorithm (AdaGrad) and Root Mean Square Propagation (RMSProp). It computes adaptive learning rates for each parameter. The Adam optimizer was particularly effective for training BERT-based models due to its ability to handle sparse gradients on noisy data.

It is worth mentioning that periodically here and in the following stages I included early stopping in the code: this is when it begins to overfit and there is no point in further iterations (epochs).

Other Models

Having tested BERT and CamemBERT, I went further, trying to find new efficient models. Unfortunately, I was not able to use any of the models listed below, but I would definitely try to invest more time and money if the goal was to improve the results of any value.

My first glance fell on ChatGPT 4.0. Obviously, this is the most successful model at the moment. However, without paid APIs, even chagpt-3.5-turbo could not be loaded. Next attempt was to use the large version of CamemBERT, camembert-large (https://huggingface.co/almanach/camembert-large), 335M parameters instead of 110M in the base one. This model as well as **FlauBERT** (both large and base; https://huggingface.co/flaubert/flaubert_large_cased) didn't work out due the limited access. Finally, I tested **T5 Model** (a small version, https://huggingface.co/google-t5/t5-small) but the result was quite poor for my model.

*"**FlauBERT** is a French BERT trained on a very large and heterogeneous French corpus. Models of different sizes are trained using the new CNRS (French National Centre for Scientific Research) Jean Zay supercomputer."*

In this section, I also tested a cross validation technique but without success.

Using Optuna

Looking for an opportunity to improve the result, I came across the **optuna** model. Its peculiarity is that it goes through the given parameters to achieve the best result. This is twofold. On the one hand, this allows you to set up a model with the parameters of interest, shifting responsibility to the algorithm and doing other things. On the other hand, it can be very resource-intensive and ineffective. However, I have used it several times: first with less limited parameters and fewer (learning rate, drop outs, the size of batch) and then with more limited ones (less range in the variables), but added weight decay.

Based on this model, I received the best accuracy at ~57%+. After getting this result, I decided to test it, based on the best hyperparameters & NLTK, and got the best accuracy! **(60,4%)**

I realized that I could continue to burn power (at that time I had already spent 2-3 subscriptions to Colab Pro), trying to improve the result by tenths of a percent using Optuna and adding various parameters. However, this would not add value to the training, so I decided to try changing the methodology and starting from scratch. *(I saved the model and in the requirements section delivered some details of the model.)*

Starting from scratch

As a first step, I decided to calculate the number of the words in the total corpus, then spreading them between different levels of difficulty. To create a final list of the words, I put two main conditions:

1. At least 70% of a specific word belongs to a certain level;
2. Removing Top-150 most popular words (usually it is stop words).

I was also thinking about lemmatization but decided that it can partially spoil results (in the current case-scenario) as the verb can be both in present simple and subjunctive but it will have the same root. This led me to the **28%** accuracy, so I continued to search for the solution.

As a next step, I decided to check different quantitative characteristics:

1. The average length of the sentence for each difficulty level;
2. The average number of words in the sentence for each difficulty level;
3. The average number of apostrophes in the sentence for each difficulty level;
4. The average number of punctuation marks in the sentence for each difficulty level.

In the future this list can be continued and include different words and features of the words/sentences/levels! This model with in a consideration with a previous one resulted in **42%** accuracy. Then, I tried to implement these conditions with text embedding, however with a very modest result – **~52%** accuracy.

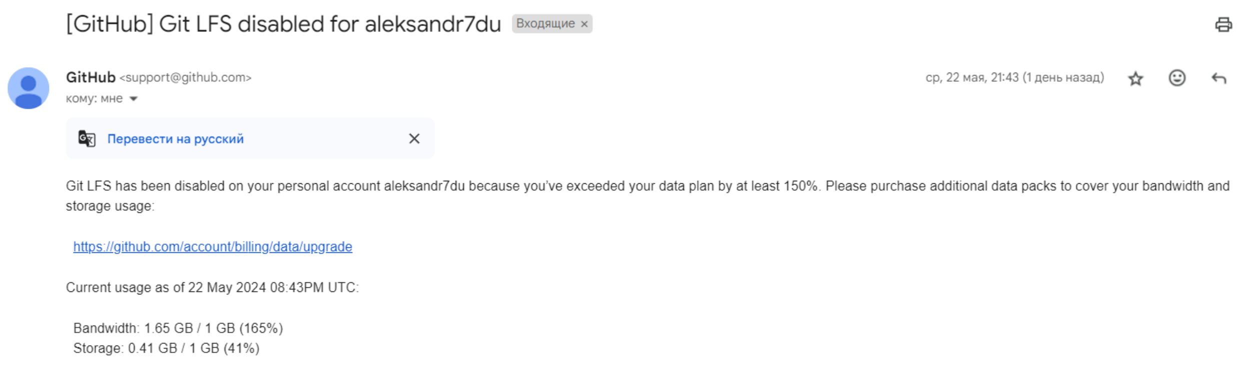
At this stage, I decided to stop, since we were told that 60% accuracy was enough to get a good result and it was better to concentrate on other aspects + there were a couple of days left before the deadline for the rest.

Appendix

My attempts with changing various coefficients in the target models, as well as attempts to generate data volume, are listed.

Streamlit

Streamlit turned out to be more stressful than we had discussed in class, since the implementation of the model, interaction with Github and the lack of practice were noticeable. In addition, after several hours of adjusting the code and playing the streamlit on my model, my free Github tariff was exceeded and I had to use a regular model for the streamlit sample. However, I got a good result and experience.


[GitHub] Git LFS disabled for aleksandr7du Входящие ×

GitHub <support@github.com>
кому: мне

ср, 22 мая, 21:43 (1 день назад)

Перевести на русский

Git LFS has been disabled on your personal account aleksandr7du because you've exceeded your data plan by at least 150%. Please purchase additional data packs to cover your bandwidth and storage usage:

https://github.com/account/billing/data/upgrade

Current usage as of 22 May 2024 08:43PM UTC:

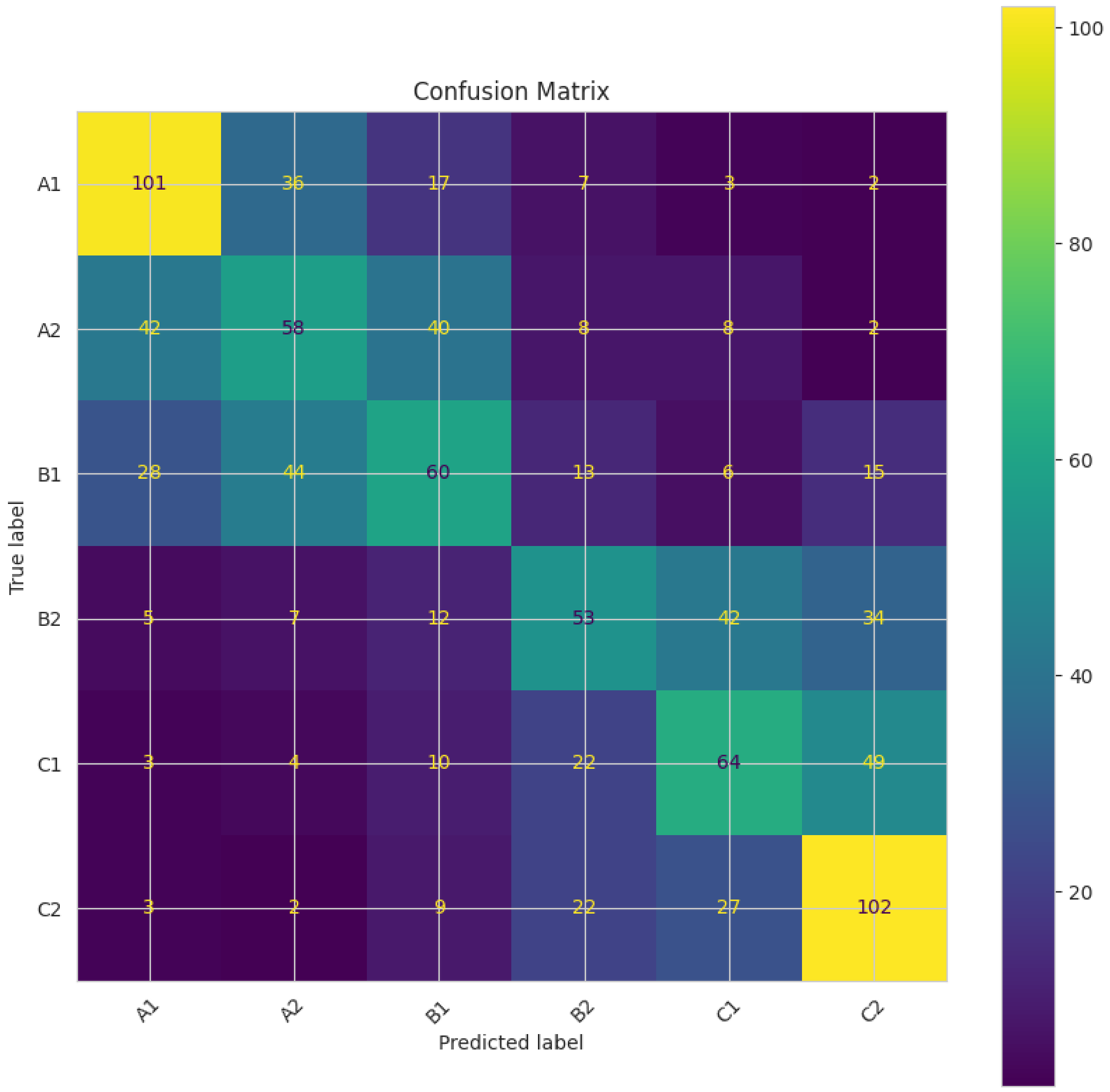
Bandwidth: 1.65 GB / 1 GB (165%)
Storage: 0.41 GB / 1 GB (41%)

That is why I decided to move to a new GitHub and build the app, based on the simple model. As an App, I built the prediction and recommendation system.

My best model

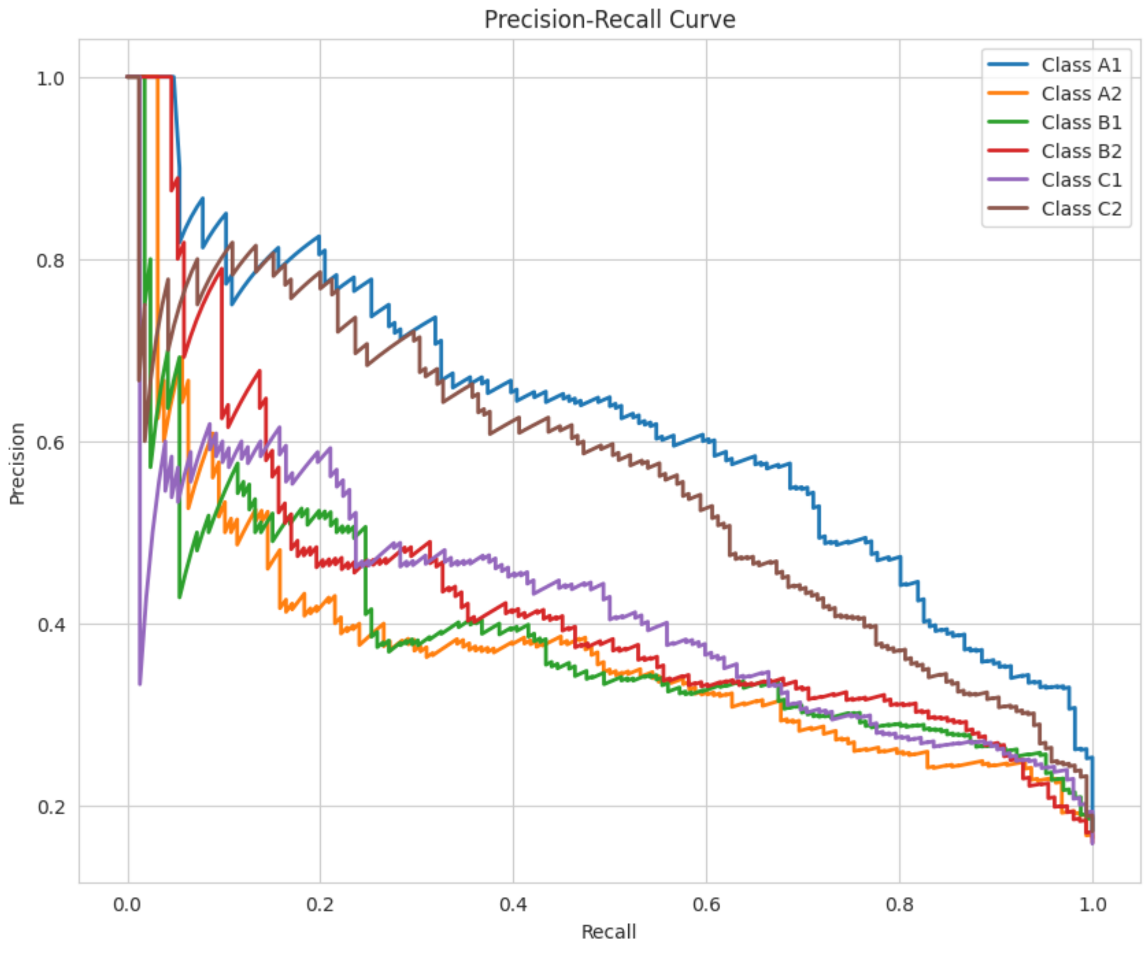
Based on the reason above and that my Colab didn't download-upload the most advanced model, I decided to build a confusion matrix and make some analysis on almost the same model (logistic regression) as I used for the design of the application (to show my understanding).

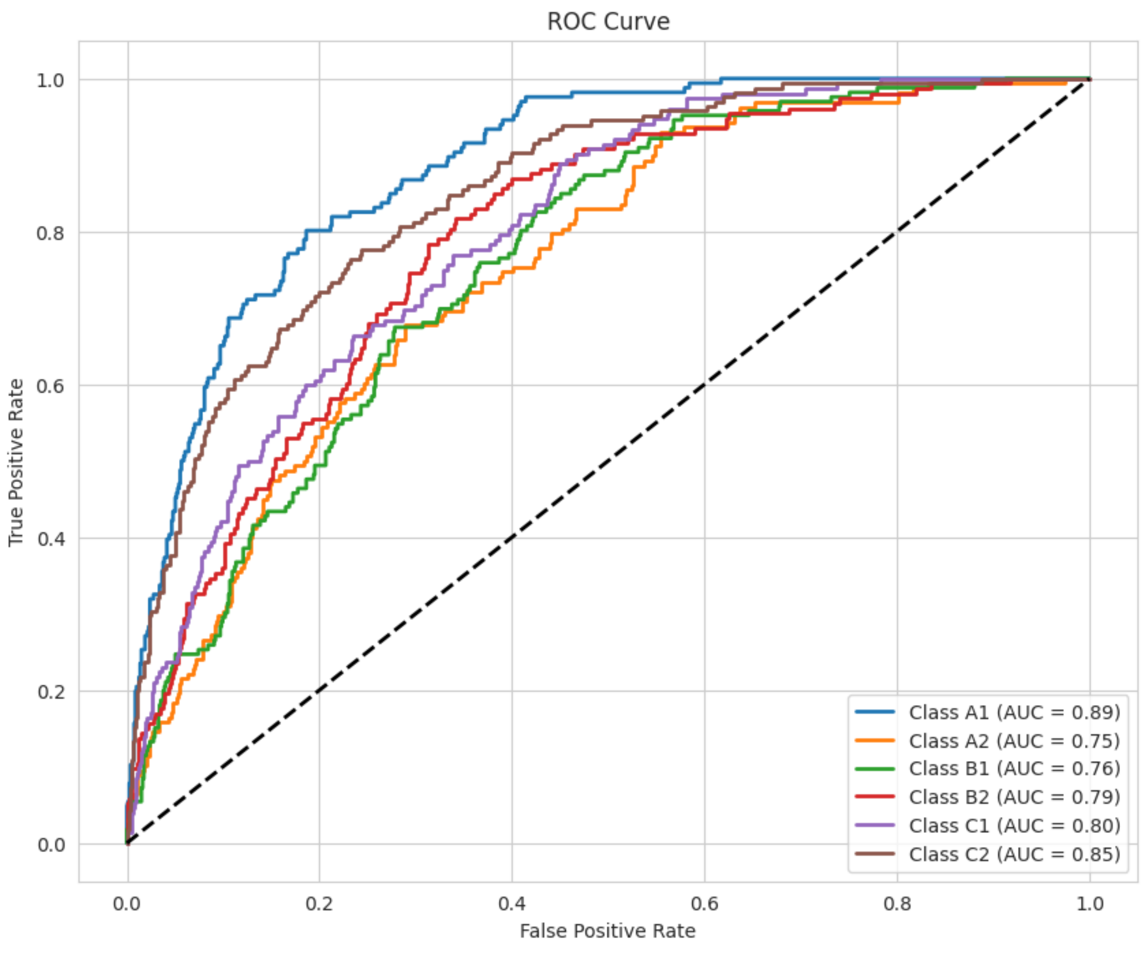
Here, the confusion matrix and some analysis.



```
                                              sentence true_label  \
2908  Les Français ne cèdent pas au chacun pour soi,...         B2
8     J'ai retrouvé le plaisir de manger un oeuf à l...         A2
2867  Je vois trois pommes de terre, mais elles ne s...         A1
3509                 Ce sera inoubliable, j'en suis sûr!         A1
3121  Et soudain elle est devant moi, mes lunettes à...         A2

     predicted_label
2908              B1
8                 B1
2867              B2
3509              A2
3121              A1
```


Precision-Recall Curve
Precision
Recall
Class A1
Class A2
Class B1
Class B2
Class C1
Class C2


ROC Curve
True Positive Rate
False Positive Rate
Class A1 (AUC = 0.89)
Class A2 (AUC = 0.75)
Class B1 (AUC = 0.76)
Class B2 (AUC = 0.79)
Class C1 (AUC = 0.80)
Class C2 (AUC = 0.85)

Here's a description and analysis of the steps taken:

Confusion Matrix

The confusion matrix visualizes the performance of the classification model. Each cell represents the number of instances of true class i that were predicted as class j.

From the confusion matrix provided, we can see that the model performs reasonably well with some noticeable misclassifications between certain classes. For example, there are significant misclassifications between A1 and A2, C1 and C2, indicating that the model sometimes confuses adjacent levels.

Erroneous Predictions:

The table with some erroneous predictions highlights sentences where the predicted labels do not match the true labels. For instance:

"Les Français ne cèdent pas au chacun pour soi,..." (True Label: B2, Predicted Label: B1)

"J'ai retrouvé le plaisir de manger un oeuf à l..." (True Label: A2, Predicted Label: B1)

These errors suggest that the model might struggle with subtle nuances that differentiate adjacent levels, possibly due to the overlap in vocabulary and sentence structure.

Precision-Recall Curve:

The precision-recall curve shows the trade-off between precision and recall for each class. The curves for each class indicate the model's performance across different thresholds.

Classes like A1 and C2 have relatively better precision and recall compared to the middle levels like B1 and B2, which might indicate that the model is more confident in predicting the extremes but less so for intermediate levels.

ROC Curve:

The ROC curve plots the true positive rate against the false positive rate. The Area Under the Curve (AUC) for each class provides a single metric summarizing the model's performance.

The AUC values range from 0.75 (A2) to 0.89 (A1), indicating good performance, with the model performing better for classes A1 and C2.

Analysis and Understanding Errors:

Vocabulary Overlap: many French words are common across multiple levels, making it challenging for the model to distinguish between adjacent levels. For example, A2 and B1 might share significant vocabulary, leading to confusion.

Sentence Structure: complexity in sentence structure might not be adequately captured by the model. Sentences with similar structures but different difficulty levels can lead to misclassifications.

Contextual Understanding: the model might need more sophisticated context understanding to differentiate between levels. For example, a sentence with complex grammatical constructs might be indicative of a higher level, but if such nuances are missed, it can result in errors.

For the further analysis, I would consider: feature importance, cross-validation, error distribution, text length impact.

Conclusion

To summarize and discuss how I would further improve the model, I see 4 ways:

1. Using Optuna or yourself, slightly selecting hyperparameters to increase accuracy.
2. Use other models and implementation techniques.
3. Explore new methodologies: think more deeply about which aspects and features of the (French) language can be paid attention to and thus classify the level of sentences.
4. Date augmentation: I made several attempts to expand the data, in particular, by reverse translation, but this required very significant power (for 3-4 high-quality language models for sure).

Overall, I can say that the goal was achieved: I used and tested different techniques and models, eventually using text inserts and reaching the 60% accuracy. It was a long but very valuable journey. Back in September, I didn’t know what Pandas was. However, now I used advanced versions and machine learning models, interacting with GitHub and doing basic design on Streamlite. Thank you for this subject and for your attention!