Retraining strategies for an economic activity classification model

Thomas Faria[1], Nathan Randriamanana[1], Tom Seimandi[1]

[1]Insee, Montrouge, France

Abstract

The French company registry, SIRENE, lists all companies in France and assigns them a unique identifier, the Siren number, for use by public institutions. As part of the registration process, companies must provide a description of their economic activity. Since the end of 2022, SIRENE leverages a simple text classification model to code each description into an industry from the French classification of activities (NAF).

Using a machine learning model in a production environment comes with challenges, in particular regarding model monitoring and maintenance. In this talk, we will first present the monitoring system developed to track model behavior and detect potential drifts for SIRENE. Then we will address the question of model retraining, including the following considerations:

- Evaluation data: how should evaluation data be collected (data quantity, frequency, sampling strategy)?
- Training data: what training data should be used for retraining? For example, should historical data be used systematically or does the model perform better when only trained on recent data? To what extent should data classified automatically by the model in production be part of the training set to keep it balanced?
- Retraining strategy: at what frequency? What are the differences between fine-tuning and retraining from scratch?
- Algorithmic adjustments: does a more complex text classification model allow performance gains on new real-world data?

This talk aims to equip practitioners with an improved understanding of the technical and practical considerations involved in retraining text classification models. As such models are becoming an essential component of official statistics, it is crucial to ensure the quality of their outputs in production environments.

Introduction

Machine learning (ML) systems are increasingly used within national statistical institutes (NSIs) for the production of official statistics. Such systems are already in place today in several NSIs (Gjaltema 2022) for coding and classification of text descriptions, data editing and imputation for example. In addition to this, experiments are conducted on other natural language processing tasks, and even on computer vision systems that are very promising.

The European Statistical System has developed of common quality framework that heavily relies on the European Statistics Code of Practice (CoP) (European Union 2018). This code of practice was first adopted in 2005 and later revised in 2017. It sets standards for developing, producing and disseminating statistics, in particular by defining quality principles regarding statistical processes and output. For example, the CoP affirms that "sound methodology [and] appropriate statistical procedures, implemented throughout the statistical processes, underpin quality statistics" (principles 7 and 8). Furthermore, it states that "European Statistics [must] accurately and reliably portray reality" (principle 12). To do so, "source data, integrated data, intermediate results and statistical outputs [must be] regularly assessed and validated" and "revisions [must be] regularly analysed in order to improve source data, statistical processes and outputs".

If the ESS common quality framework drove European NSIs to implement quality procedures for a large part of their statistical production, there is not a lot of shared experience on what such procedures should look like for statistical outputs leveraging ML systems. Developing these procedures is closely linked to MLOps, a set of good practices that aims to deploy and maintain ML systems in production reliably and efficiently.

MLOps is derived from the concept of DevOps, a general software development framework whose idea is to integrate the entire lifecycle of a project into an automated continuum. In the DevOps framework, continuity is achieved with the help of CI/CD pipelines. Continuous integration (CI) is the process of integrating code changes into a common source repository in a regular fashion. Continuous deployment (CD) is the process of automatically distributing the changes made to the source code. Consequently, the MLOps principles strive to automate the lifecycle of ML systems (McMahon 2023). Building a ML system begins with an experimental phase, where a data scientist prepares and analyses data before training a model to solve the task at hand. This model is then served to end-users through an application (typically an API). This application should be continuously monitored so as to detect potential performance losses over time. Major performance losses should trigger some form of model retraining, going back to the data preparation and analysis of the experimental phase.

This paper shows how MLOps principles are applied on a specific ML system at the French NSI (Insee) – namely a coding engine for the economic activity of companies – to ensure quality standards for downstream official statistics.

1 Coding system

1.1 Context

The French company registry, SIRENE 4, lists all companies in France and assigns them a unique identifier, the Siren number, for use by public institutions. As part of the registration process, companies must provide a description of their economic activity. Since the end of 2022, SIRENE 4 leverages a simple model to classify each text description into an industry from the French classification of activities (NAF) that contains 732 different codes[1]. NAF activity codes are constantly used by Insee and others to produce business statistics, whenever it comes to studying companies by business sector. It is therefore paramount to ensure that the

[1]The classification has a hierarchical structure, meaning that the 732 codes at the finest level are grouped in 615 *classes*, grouped into 272 *groups*, themselves grouped into 52 *divisions*, themselves finally grouped into 21 *sections*.

NAF codes assigned within SIRENE 4 reflect the truth. For this purpose, part of the activity descriptions are coded manually, when the model is not confident enough in its prediction. This will be detailed in Section 1.3.

The model currently used in a production setting (detailed in Section 1.2) has been almost entirely trained with historical data from the former SIRENE 3 registry. Approximately 10 million observations covering the period 2014-2022 were gathered for the training procedure, with part of the ground-truth labels coming from the former rule-based coding engine Sicore, and the rest from manual annotation (when Sicore could not perform coding). When evaluated on SIRENE 3 data, the model naturally exhibits a very high accuracy of 89% (along with other high performance metrics). However, SIRENE 4 data significantly differs from SIRENE 3 data (text activity descriptions are for instance longer on average), as the registration channel has been refactored in the new system. This distribution shift in the data comes with a reduced accuracy of 80% on a small evaluation set originating from the new registration channel that has been completely hand-coded. It is important to note that this first SIRENE 4 evaluation set is biased as only certain types of companies registered through the new channel at first.

Company activities evolve over time: completely new businesses appear, businesses traditionally associated with a certain activity may see this activity evolve, etc. As a result, the accuracy of a coding engine leveraging a static model trained on a fixed training set will decrease over time. Indeed, the model would in principle not be able to correctly classify text descriptions totally unlike any description in the training set, or would tend to classify businesses associated with a changing activity according to the code mostly found in the training set.

Therefore, to maintain a highly accurate coding engine, the ML model used must be periodically retrained with an updated training set. Monitoring the activity of the coding engine will help decide when to retrain the ML model, as discussed in Section 2. Different retraining methods are discussed in Section 3.

1.2 Modeling

The text classification model leveraged by the SIRENE coding engine is based on a simple bag-of-ngrams model (Joulin et al. 2016), along with a standard multinomial classification head. It is implemented by the fastText library. It is an extension of the bag-of-words approach that represents a text as the average of the vector representations of each of its constituent words. In the bag-of-ngrams extension, embeddings are not only computed on words but also on word n-grams and character n-grams, providing a form of context and reducing biases due to spelling mistakes.

In our context of supervised text classification, the embedding matrix and the classifier parameters are learned simultaneously during training by gradient descent, minimizing a cross-entropy loss function.

Additional categorical variables are in fact used to classify the text description into NAF codes. As the fastText implementation of the bag-of-ngrams model is currently used in production and does not allow using categorical variables out of the box, a hack is used that consists in concatenating specific strings for the relevant modalities of each of the categorical variables to the text description[2].

1.3 Model serving

Providing the trained model as an artifact is not a convenient way to make it available to end-users, as it assumes that they have the required knowledge to use the artifact to get predictions correctly. This often requires a specific computation environment, etc. Serving a model this way always leads to code duplication and very often to errors. In an effort to build an interoperable and easy-to-use coding engine, available for query from various programming languages, our text classification model is served through a REST

[2]This Pytorch implementation gives a cleaner way to use categorical variables for classification: https://github.com/InseeFrLab/codif-ape-train/tree/main/src/pytorch_classifier.

API. REST APIs have become a standard way to serve ML models. Among their advantages, they rely on standards technologies for queries (HTTP requests) and responses (generally, JSON-formatted strings), making them agnostic to the programming language used to query them, and they offer great scalability because of their stateless design.

The REST API is developed using FastAPI, a modern, high-performance, web framework for building APIs with Python. We encapsulate the API code and its dependencies into a Docker image, deployed as a container on a Kubernetes cluster. This grants the ability to easily scale the API according to demand thanks to automatic load-balancing. Upon startup, the API automatically retrieves the text classification model from storage. The model is packaged as an MLflow model (Chen et al. 2020), that integrates pre-processing steps in its inference method. This means that clients do not have to apply these steps themselves. This prevents many potential mistakes.

The target coding procedure is as follows: when a company registers, a query is made to the API containing the text description of the activity and the relevant categorical variables. The API returns the five most probable predictions, together with a confidence index equal to the difference between the output probabilities of the two most probable codes. If the confidence exceeds a specified threshold, then the top code predicted by the API is automatically assigned to the company. If it does not, the code is assigned manually in a user interface that displays the five most probable codes to speed up annotation.

2 Monitoring

2.1 Design

As explained in Section 1.1, the activity of the coding engine is monitored in almost real-time. This monitoring is essential to detect anomalies in real life data, or to be able to make decision on when to retrain the ML model behind the engine. The monitoring system is made up of two distinct blocks:

- A simple *Extract-Transform-Load* (ETL) process is run nightly from a Python script to fetch the logs recording the activity of the coding API. The logs are parsed and their content is saved as partitioned Parquet files;
- An interactive dashboard is built with Quarto[3] to offer insight on the data fed to the API and how it is coded.

The relevant computations are done with direct SQL queries on the Parquet files storing log content using the DuckDB engine[4]. The dashboard is also rebuilt nightly to integrate the latest data in its static visualizations. It is deployed as a container on a Kubernetes cluster.

2.2 Engine monitoring

The monitoring dashboard offers insight about the codification on several levels:

- The daily and weekly number of queries made to the API are displayed in two separate graphs on a tab. The daily and weekly automatic coding rates are also displayed, in reactive visualizations that are modified according to the confidence threshold for automatic coding displayed by the user. The aggregate automatic coding rate on all data is also displayed (see Figure 4);
- The distribution of confidence intervals is displayed for a time window specified by the user (Figure 5);
- For a time window, a level of the NAF classification system and a confidence threshold specified by the user, the number of predictions in each possible class is displayed in a barplot, colored with the corresponding automatic coding rates for these classes (Figure 6);

[3] https://quarto.org/docs/dashboards/
[4] https://duckdb.org/

- To detect potential data drifts, the user can specify two distinct time windows as well as a level of the activity classification to visualize the distributions of codes predicted at a desired level within the time windows (Figure 7);
- All data fed at some point to the coding API is available in a table (text descriptions and categorical variables) along with the predictions made by the ML classification model on each data point and the associated confidence. This table can be filtered on any of its fields to allow users to look for specific data points (Figure 8);
- Finally, a dashboard tab focuses on text statistics, and allows to user to mean number of words and of sentences of text descriptions fed to the API within a specified time window. The user can also visualize the most frequent identical text descriptions as well as the most frequent words within that time window (Figure 9).

While these tabs give insight on the data distribution and on the predictions of the classification model on that data, a proper continuous evaluation of the performance of the coding engine is not available at this point. An evaluation dataset with enough data collected in a continuous fashion is therefore necessary.

2.3 Continuous performance evaluation

To perform continuous performance evaluation of the ML model, we regularly annotate data points sampled from the SIRENE 4 information system. We sample batches of data that are manually annotated using a Label Studio interface. The first batches were sampled uniformly at random in the subset covering the period going from January 2022 to February 2023. In the following batches, we only sample data points from a subset covering the latest dates not yet present in the evaluation set. Eventually we will only sample unique data points with inclusion probabilities proportional to their prevalence in the registry data.

Data batches are automatically uploaded on a Label Studio interface, to which several annotators have access. A single annotation is demanded for each point in the batch: the annotator is presented with the text description of the activity and the additional categorical variables useful for the classification task. Either a code from the NAF classification system is chosen or the text description is marked *uncodable*. The annotator also has the possibility to skip the annotation task, which will leave it available for an other annotator. Figure 1 shows the Label Studio user interface, that includes a tool to browse the classification system efficiently with keywords. With the current system, a single annotation takes about a minute on average. For now, 5 batches have been added to our evaluation set for a total number of around 8400 data points.

An issue with the annotation procedure for now is that one single code is collected for each text description, regardless of how simple it is to assign it a code. We would eventually like to modify the procedure by allowing annotators to assign a code but warn that the description is difficult to classify. Such descriptions would then be annotated by a different person not aware of the first code that was assigned (with the intervention of an arbiter in the case of a disagreement).

Two tabs are added to the monitoring dashboard for continuous performance evaluation:

- A first tab gives elements on the entire evaluation set, given a specified threshold on the confidence for automatic codification. Plots here assume manually-coded descriptions are perfectly coded (Figure 10):
  - The total number of observations in the evaluation set;
  - The automatic coding rate;
  - The accuracy of the coding engine;
  - The top-2 to top-5 accuracies of the coding engine;
  - The accuracies of the coding engine at all levels of the NAF classification system;
  - The distributions of confidence values on the entire evaluation set for both correctly and incorrectly classified descriptions, with a display of the chosen threshold;

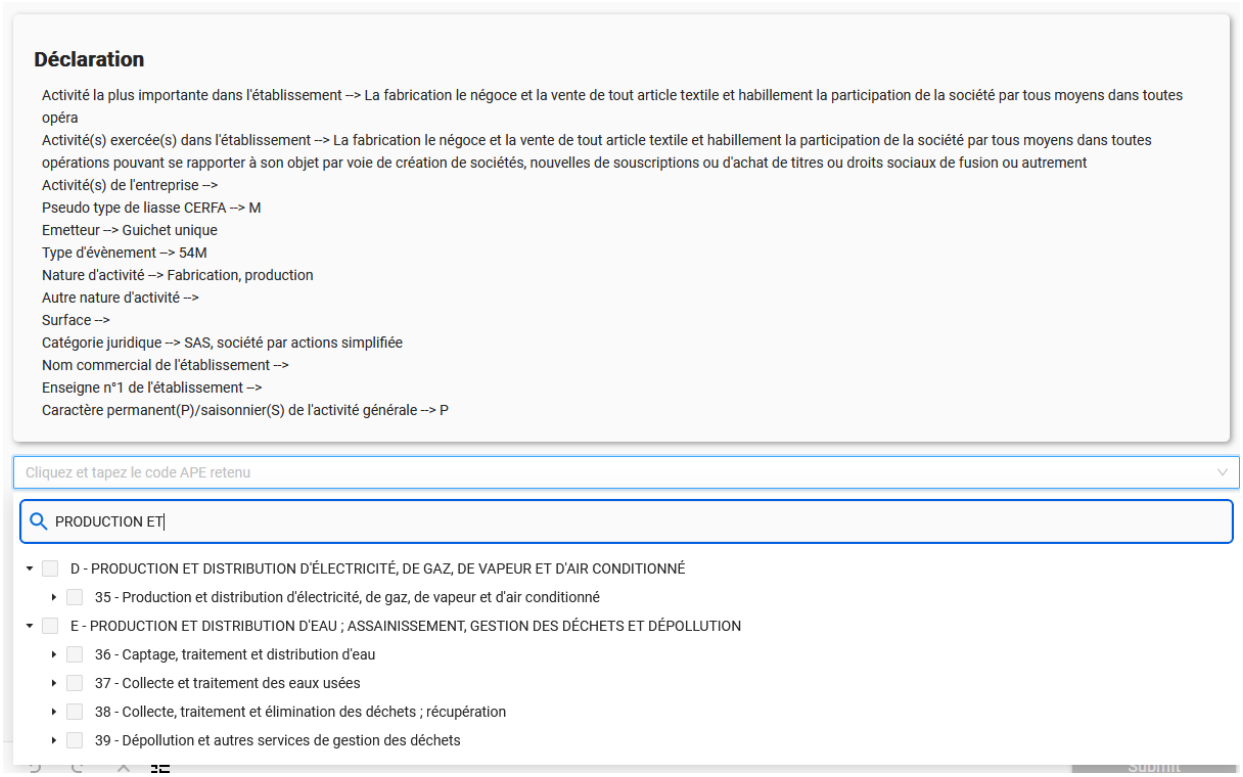

Figure 1: Screenshot of the Label Studio UI for a single annotation task.

- A second tab gives monthly estimates for the accuracy of the coding engine, with bootstrap confidence intervals (Figure 11).

3 Retraining

3.1 Triggers

"When" is one of the first questions arising when thinking about retraining the ML model behind the coding engine. As explained earlier, business activities evolve over time, so the coding engine should not be static. At a given time, the model in use should be trained with data that covers as much as possible the entire distribution of inference data. This means that if a new business appears at some point t in time, the ML model used for coding should be retrained or fine-tuned with training data including data points corresponding to this new business. In theory, this will be the case for an extraction of the registry following t. Indeed, the data points in question should be given a low confidence score, and thus be manually assigned the correct code in the registry.

It seems natural to advocate frequent retraining or fine-tuning procedures to avoid long periods of time where the data points mentioned above are all coded manually. However, it is good practice to have a thorough validation procedure when switching production models. This validation can be costly, which on the contrary encourages to retrain sparingly over time. This is all the more true that we are not dealing with extremely high-frequency data and that the emergence of new businesses is a relatively one-off phenomenon. All in all, it seems reasonable considering our validation procedure to periodically retrain our classification model every 3-6 months.

In addition to periodic updates, our monitoring allows us to trigger specific retraining procedures when certain events are observed – for instance, as soon as the monthly accuracy computed on our evaluation set (see Figure 9) falls under a certain threshold.

A third situation should trigger an adjustment of the coding engine. Companies that consider that they have been assigned a wrong activity code can contact Insee to make a claim. Claims are processed manually by Insee agents who modify the activity codes in the registry when it is justified. When many claims are accepted about similar businesses, it means that systematic coding errors are made. In this case it is relevant to retrain the ML model with a training set modified to avoid the errors in question. Training sets are based on extractions of the SIRENE registry. The updated training set would be an extraction where relevant observations have been identified and their code modified.

A similar situation occurs when businesses traditionally associated with a certain activity see this activity evolve. From a certain point on, the code that should be assigned to such a business changes. This should also trigger a retraining procedure of the model with an updated training set, where observations corresponding to the business in question have been assigned the new updated code. In practice this requires versioning training sets (see Figure 2).

3.2 Retraining strategies

There are multiple ways to conduct the retraining procedure. There are two following alternatives:

- Training a model from scratch, with an updated training set. The main question in this scenario is how to weight observations from the training set with respect to their date. In particular, should we stop including observations that are too old by setting a limit on the earliest date in the training set? This question should be dealt with empirically using evaluation data. Indeed, it is very dependent on the use-case and the answer is not obvious as two effects counterbalance each other:
  - First, model capabilities scale with training data. In general, including additional data points in the training set leads to a better accuracy;

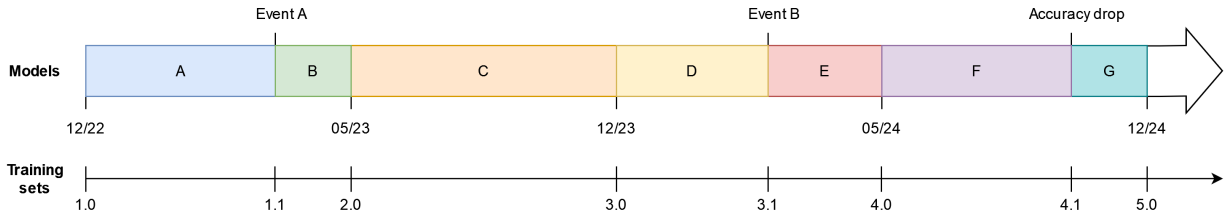

Figure 2: Illustration of model and training set versioning. Here the production model is retrained periodically every six months. It is retrained on three additional occasions, at events A and B when systematic coding errors have been identified and once because the monitoring system has identified a large accuracy drop. Training sets are versioned. They are incremented with an extraction of the latest additions to the registry at each retraining procedure. Additionally, at events A and B, historical data is modified to an extent to modify model behavior on certain text descriptions.

  - Old data from the registry has lower quality labels. This results from the evolution of business activities mentioned earlier. Let us consider businesses for which the activity code changed at some point. In the registry data, data points corresponding to the same business will partly be labeled as A up to some point and as B later on. Ideally, we would therefore like to retrain the ML model with data including only the latest B labels;
- Fine-tuning the current model with only new observations from the registry. In this scenario, we use pre-trained weights from the current model used in production that we adjust with the latest information from the registry. A key concern in this scenario is that the global performance of the simple classification model we are using is very sensitive to distribution shifts in inference data with respect to training data, which can easily happen in the case of a small training set. In any case, fine-tuning must be done to maximize performance metrics on a validation set.

Only the first alternative is tested here, as using the fastText library prevents from easily fine-tuning using pre-trained weights[5]. Figure 3 displays the accuracies on the Label Studio evaluation set of models trained on SIRENE 4 data and subsets of SIRENE 3 data as a function of the earliest year included in the SIRENE 3 training data. Training set sizes are also displayed. All accuracies measured range from 0.77 to 0.78. On a macroscopic level and for our use-case, there is no clear indication on how to set a date threshold for the training data. It seems reasonable to keep a few years (3-5 years for example) of data in the training set. A more in-depth analysis could help refine this policy.

4 Discussion

By monitoring the activity of our coding engine and its performance over time, we are able to minimize the risk of degradation by developing appropriate retraining strategies. Another area for improving the current system is to use a more complex language model.

As explained in Section 1.2, the classification model used by the current coding engine is a very simple bag-of-ngrams model. It offers the advantages of simplicity, speed and good handling of spelling mistakes, an important feature of SIRENE data to account for. It does come with obvious drawbacks however. Embeddings in our classification model are contextual only in a very limited way, through the use of word-ngrams. Activity descriptions similar to "Our company's main activity is <activity 1> but we are also involved in <activity 2>" will never be correctly embedded for the classification task at hand as the model will have no way of differentiating the two described activities.

[5]A possibility for fine-tuning would be to use the alternative Pytorch implementation https://github.com/InseeFrLab/codif-ape-train/tree/main/src/pytorch_classifier.

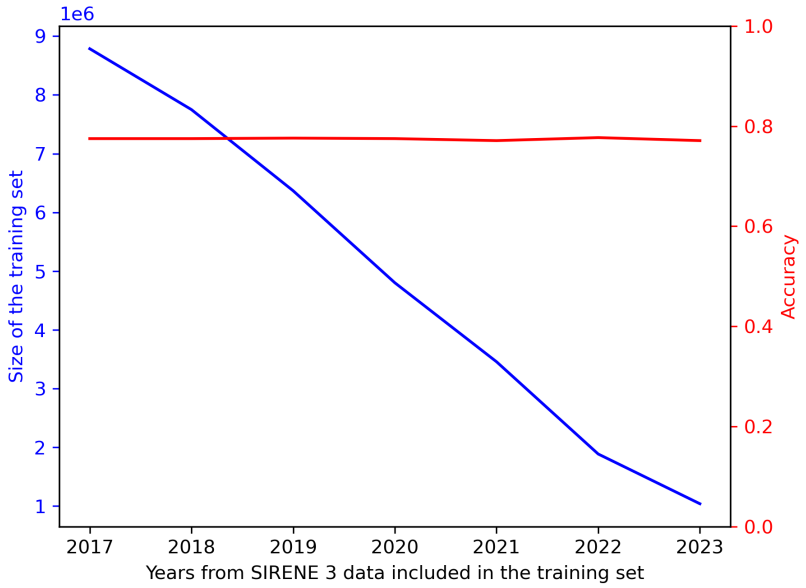

Figure 3: Accuracy on the Label Studio evaluation set and training set size as functions of the earliest year included in SIRENE 3 training data

However, the accuracy gains observed with a more complex model, a version of CamemBERT[6] fine-tuned on SIRENE data for our text classification task, are currently too low to completely justify switching models. When evaluated on part of our evaluation set, a fastText and a CamemBERT model trained on SIRENE 4 data and SIRENE 3 data going back to January 2020 have respective accuracies of 0.75 and 0.76. More precisely:

- The two models predict the code given by the annotator with a rate of 0.71 (case A);
- The fastText model predict the code given by the annotator that is different from the code predicted by the CamemBERT model with a rate of 0.04 (case B);
- The CamemBERT model predict the code given by the annotator that is different from the code predicted by the fastText model with a rate of 0.05 (case C);
- The two models predict the same code, but it is different from the annotation with a rate of 0.17 (case D).

Looking at the predictions more closely shows that when the two models predict the same code that differs from the annotation, it is often not clear that the annotation is correct. This pushes towards a more thorough annotation procedure with double coding and intervention of an arbiter in the case of a disagreement, either systematically or on a selection of observations identified as difficult to code. As mentioned earlier, it is possible to allow annotators to mark observations as difficult to code on the Label Studio interface. Another possibility to select these observations is to check whether the manual annotation matches the prediction(s) of one or more models. While this could help reduce the annotation cost, it could potentially introduce bias in the evaluation set.

An in-depth analysis of cases B and C would be interesting to get insight into the limitations of the

[6]CamemBERT is a language model for French based on the RoBERTa architecture pre-trained on the French subcorpus of the multilingual corpus OSCAR (Martin et al. 2020). The custom implementation of our CamemBERT classifier that allows to use additional categorical variables is available at https://github.com/InseeFrLab/codif-ape-train/tree/main/src/camembert.

bag-of-ngrams model or on the text descriptions currently not correctly dealt with by the more complex CamemBERT model. For instance, a first simple observation is that text descriptions corresponding to case B are on average slightly shorter than those corresponding to case C (84 versus 76 characters). This is consistent with the idea that CamemBERT models language better and should therefore have a higher classification accuracy on longer, more complex descriptions. The in-depth analysis could help guide improvements of the training procedure for the CamemBERT model (including tokenizer adjustments, etc.). We leave this for future work.

In this paper, we present the application of MLOps principles to a specific ML system at Insee for coding the economic activities of companies. In particular, we present our monitoring system. An ETL process fetches and transforms API logs, and an interactive dashboard built with Quarto provides insight into the data fed to the API and how it is coded. We also create an evaluation set that is continuously updated by sampling and annotating points from the SIRENE 4 information system. The evaluation set is used to include a proper performance evaluation in the monitoring dashboard. We discuss retraining strategies envisioned for the ML model. The exact retraining strategy adopted should be based on a thorough analysis of the metrics presented in the monitoring dashboard.

Supplementary material

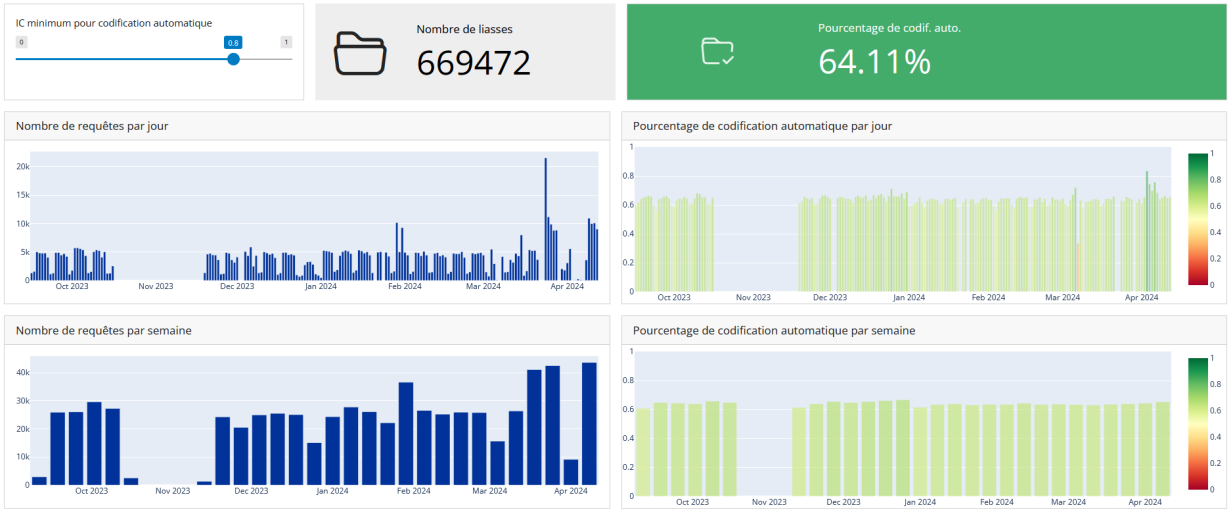

Figure 4: Dashboard tab offering daily and weekly insight on the number of queries to the API and its automatic coding rate.

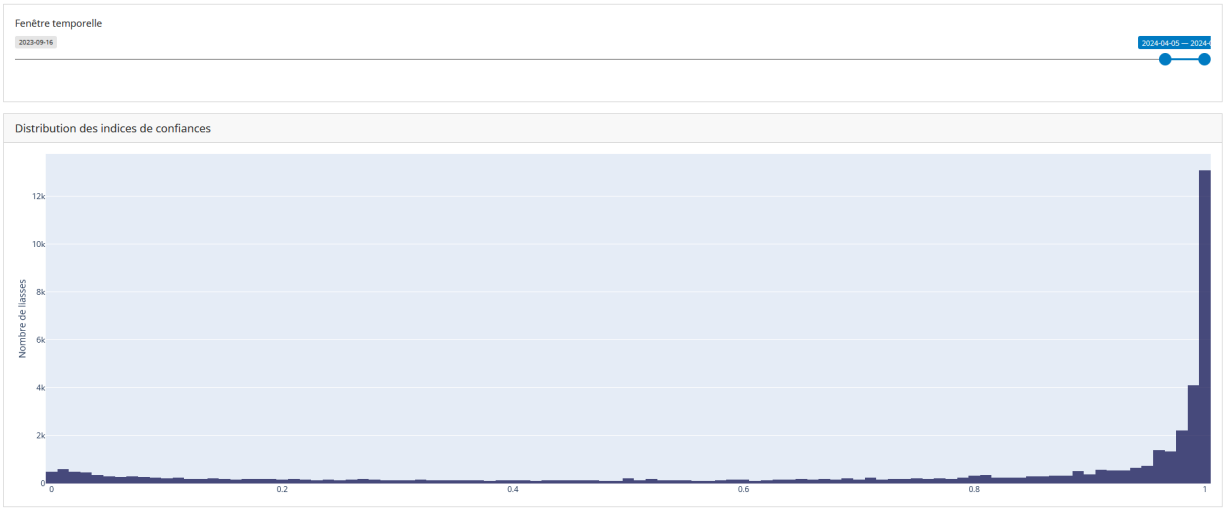

Figure 5: Dashboard tab displaying the distribution of confidence values for a specified time window.

Figure 6: Dashboard tab displaying numbers of queries and automatic coding rates for classes at a specified level of the classification system and for a specified time window.

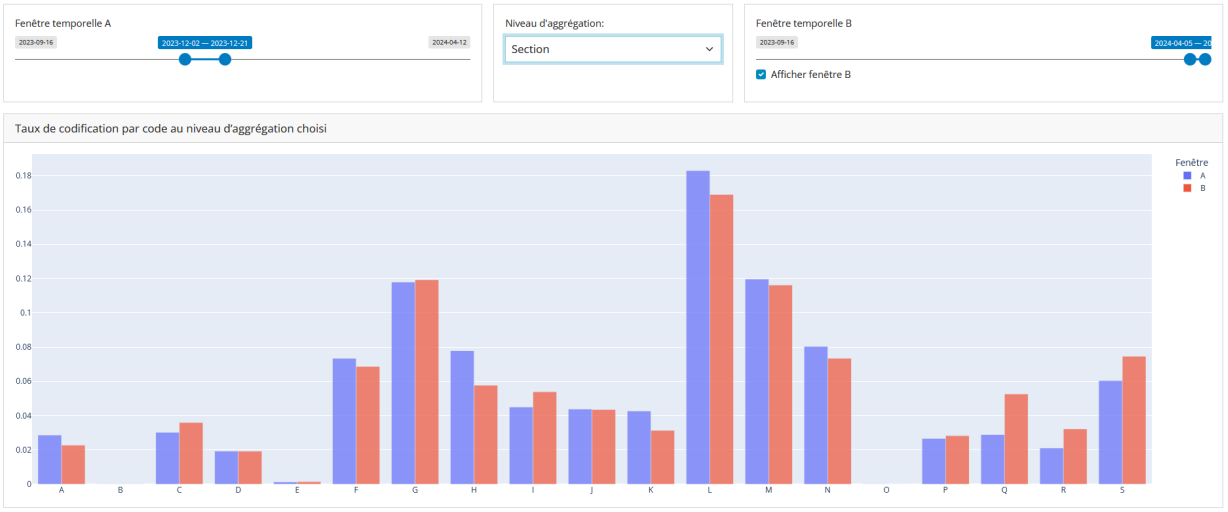

Figure 7: Dashboard tab displaying the two distributions of predicted classes at a specified level of the classification system for two specified time windows.

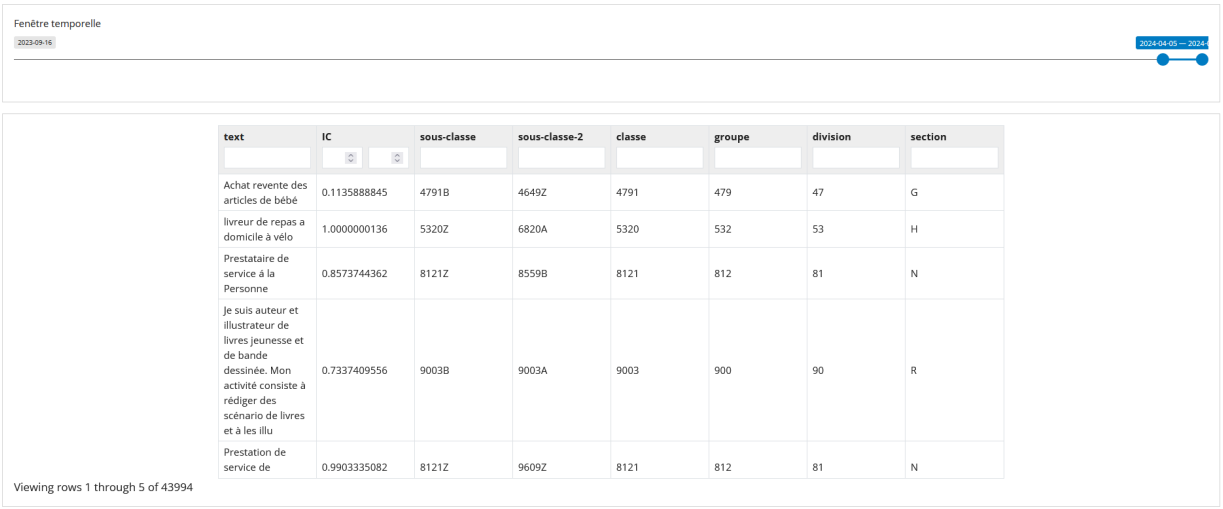

Figure 8: Dashboard tab allowing users to look through the data sent to the coding engine within a specified time window.

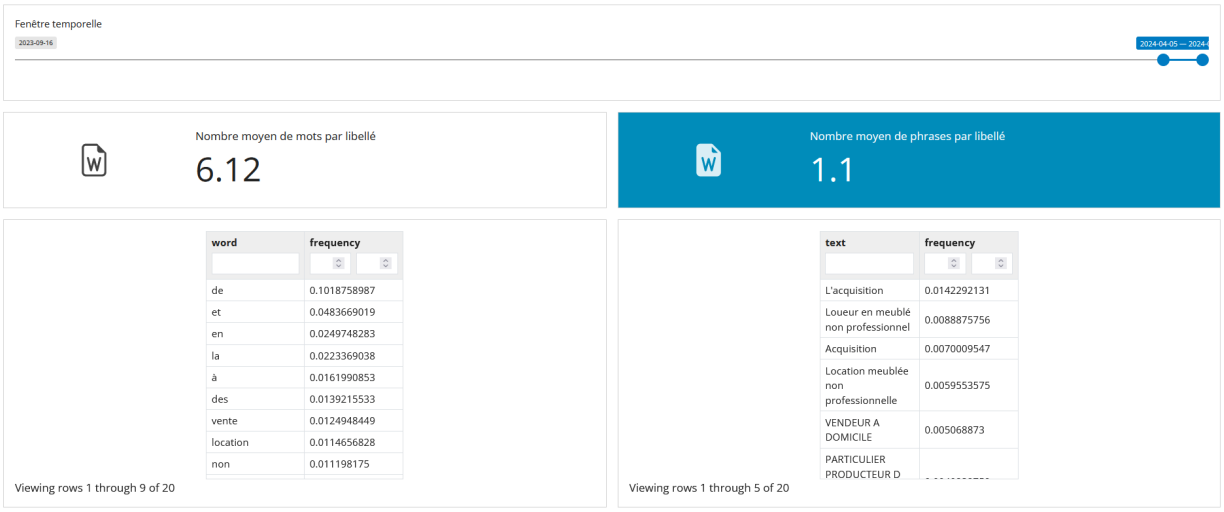

Figure 9: Dashboard tab allowing insight on the most frequent text descriptions, most frequent words and on the average length of the descriptions within a specified time window.

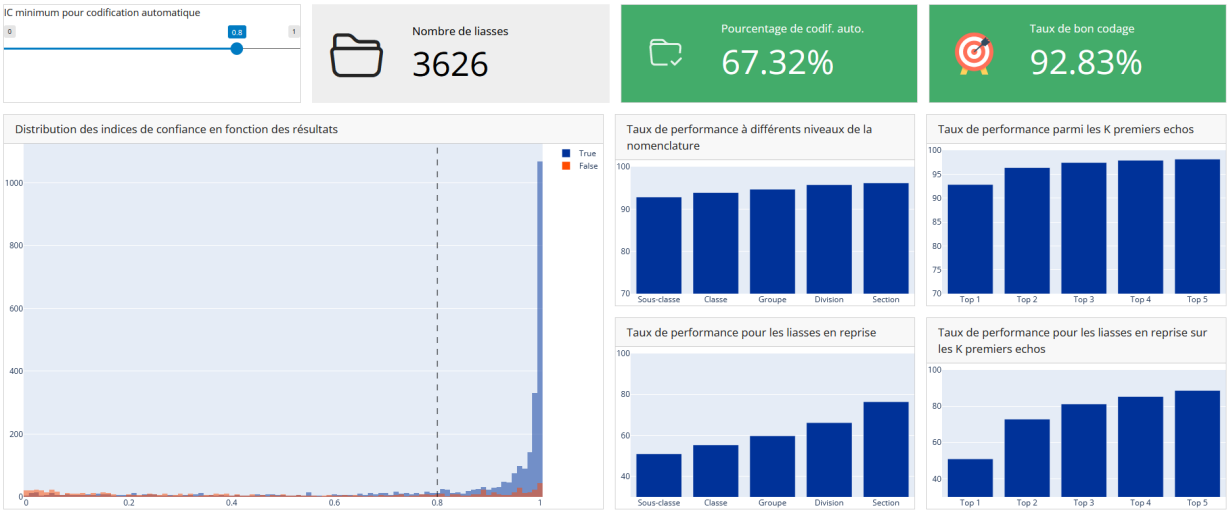

Figure 10: Dashboard tab giving insight on the aggregate performance of the coding engine on the evaluation set.

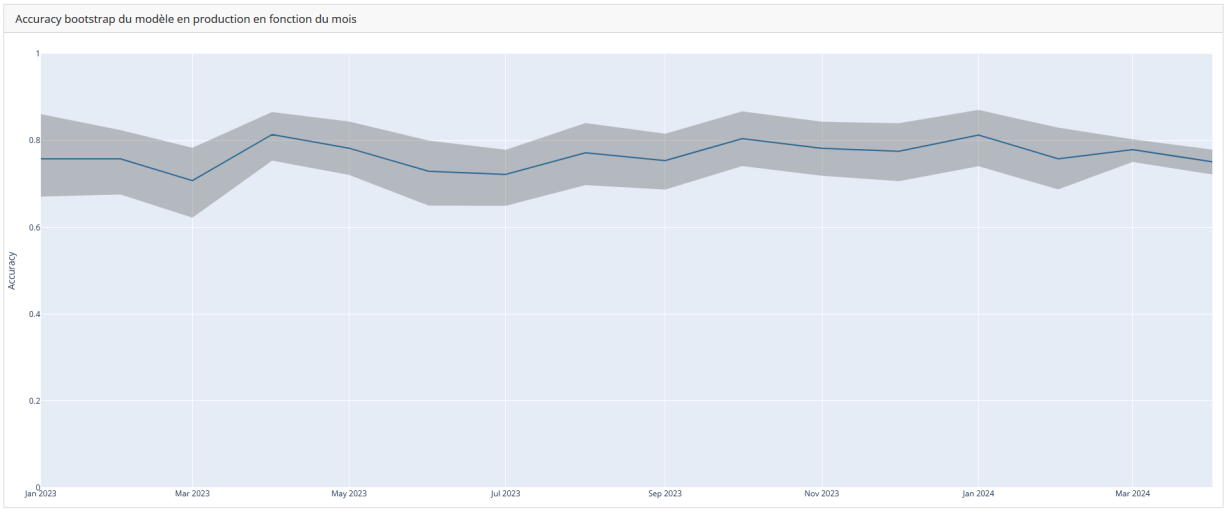

Figure 11: Dashboard tab giving insight on the monthly accuracy of the evaluation set.

References

Chen, Andrew, Andy Chow, Aaron Davidson, Arjun DCunha, Ali Ghodsi, Sue Ann Hong, Andy Konwinski, et al. 2020. "Developments in MLflow." In *Proceedings of the Fourth International Workshop on Data Management for End-to-End Machine Learning*. New York, NY, USA: ACM.

European Union, Statistical Office of the. 2018. *European Statistics Code of Practice: For the National Statistical Authorities and Eurostat (EU Statistical Authority)*. Publications Office. https://doi.org/10.2785/798269.

Gjaltema, Taeke. 2022. "High-Level Group for the Modernisation of Official Statistics (HLG-MOS) of the United Nations Economic Commission for Europe." *Statistical Journal of the IAOS* 38 (3): 917–22.

Joulin, Armand, Edouard Grave, Piotr Bojanowski, and Tomás Mikolov. 2016. "Bag of Tricks for Efficient Text Classification." *CoRR* abs/1607.01759. http://arxiv.org/abs/1607.01759.

Martin, Louis, Benjamin Muller, Pedro Javier Ortiz Suárez, Yoann Dupont, Laurent Romary, Éric de la Clergerie, Djamé Seddah, and Benoît Sagot. 2020. "CamemBERT: A Tasty French Language Model." In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 7203–19. Online: Association for Computational Linguistics. https://www.aclweb.org/anthology/2020.acl-main.645.

McMahon, Andrew P. 2023. *Machine Learning Engineering with Python*. 2nd ed. Birmingham, England: Packt Publishing.