Journées de méthodologie statistique de l'Insee 2025

fastText, tout juste en production et déjà obsolète? Présentation du package torchTextClassifiers pour la modernisation de la codification automatique via l'exemple de l'APE

Meilame Tayebjee (*), Cédric Couralet (**), Nathan Randriamanana (***), Julien Pramil (****)

(*) Insee, DMCSI, Unité SSP Lab
(**) Insee, DMCSI, Unité SSP Lab
(***) Insee, DSE
(****) Insee, DMCSI, Unité SSP Lab

meilame.tayebjee@insee.fr *cedric.couralet@insee.fr* *nathan.randriamanana@insee.fr*
julien.pramil@insee.fr

Mots-clés : Classification de texte, codification automatique, fastText, torchTextClassifiers, deep learning, PyTorch

Domaines : Données administratives et qualité, Données et traitements statistiques

Résumé

La codification automatique de l'Activité Principale Exercée (APE) à partir de libellés textuels représente un enjeu opérationnel majeur pour l'Insee. Depuis 2020, un modèle fastText 1 est utilisé en production pour cette tâche. À la fois méthodologie et librairie logicielle développée par Facebook AI Research, fastText repose sur des classifieurs linéaires efficaces, combinés à des représentations vectorielles de mots et de sous-mots. Il offre une solution simple, rapide et précise pour la classification de texte à large échelle, particulièrement adaptée aux nomenclatures métiers.

Mais quatre ans après sa mise en oeuvre, ce modèle soulève une double préoccupation. D'une part, la pérennité technique est fragilisée par l'usage d'un binaire compilé, difficile à maintenir dans le temps, susceptible d'engendrer des conflits de version, et éloigné des standards actuels de développement collaboratif. D'autre part, le besoin de modernisation se fait sentir - les évolutions rapides des outils de deep learning ouvrent de nouvelles perspectives que l'approche fastText ne permet pas d'exploiter.

Pour répondre à ces défis, l'Insee a initié le développement de torchFastText, un nouveau package Python basé sur PyTorch, conçu pour reproduire l'approche de fastText dans un cadre plus moderne, transparent et modulaire. Ce package évolue désormais vers torchTextClassifiers, qui distribuera toutes les architectures classiques de classification de texte, à destination de tous usagers souhaitant entraîner rapidement et efficacement un modèle d'apprentissage profond pour la classification de texte.

Ce développement, réalisé en open source, répond à un besoin partagé entre plusieurs instituts statistiques européens, dans une logique - espérée - d'interopérabilité et de mutualisation. L'objectif - garantir la frugalité des modèles, tout en préservant les performances du modèle initial en termes de rapidité et de précision. torchFastText

a été validé à l'issue d'une série de tests rigoureux sur les jeux de données de production. Il est aujourd'hui en production.

Outre l'aspect technique, ce projet incarne une démarche d'appropriation collective des outils d'IA, en s'appuyant sur des langages et frameworks largement adoptés dans la communauté scientifique, favorisant leur diffusion et leur maintenance dans le temps.

En plus de la simple réimplémentation, le passage à PyTorch ouvre des perspectives de développement ambitieuses :

1. Amélioration du modèle - grâce à la flexibilité du framework, il devient possible d'explorer de nouvelles architectures, d'optimiser les fonctions de perte ou encore d'incorporer la hiérarchie propre aux nomenclatures dans l'apprentissage. C'est l'objectif de la nouvelle mouture, torchTextClassifiers.
2. Explicabilité - les outils d'interprétabilité compatibles avec PyTorch (comme Captum) permettent d'apporter des éclairages plus fins sur les prédictions du modèle, favorisant leur auditabilité.
3. Intégration à l'ecosystème deep learning - l'intégration de tokenizers entraînés sur des corpus spécifiques, ou encore l'utilisation de modèles préentraînés issus de bibliothèques comme Hugging Face (de type BERT ou CamemBERT3), autorise une personnalisation plus poussée, éloignant progressivement l'approche du modèle fastText originel.
4. Meilleure expérience pour les data scientists - l'environnement PyTorch facilite le suivi de l'apprentissage (logs, visualisation des courbes de perte, early stopping. . .), accélérant les cycles de développement et de validation.

Au-delà de ce cas d'usage spécifique, l'objectif est aussi de diffuser la culture PyTorch à l'Insee, pour en faire un socle commun pour les futurs projets de deep learning en statistique publique. Des formations internes sont en cours pour accompagner cette montée en compétence collective.

Abstract

The automatic coding of the Activité Principale Exercée (APE) from textual labels is a major operational challenge for Insee. Since 2020, a fastText-based model has been used in production to perform this task efficiently. However, after four years, both technical sustainability and methodological modernization have become pressing issues. To address them, Insee has developed torchFastText, a new open-source Python package built on PyTorch that reproduces the fastText approach within a modern, modular, and maintainable framework. Now evolving into torchTextClassifiers, the package aims to provide a suite of lightweight, high-performance text classification models for broad institutional and public use. Beyond its technical benefits, this initiative reflects a collective effort to strengthen sovereignty over AI tools while fostering interoperability and shared development across European statistical institutes. It also paves the way for future advances in model interpretability, hierarchical learning, and deep learning adoption within public statistics.

1. Introduction et contexte

1.1. Historique de la codification automatique de l'APE : de Sicore à fastText

Tâche commune au sein des instituts de statistique publique, la codification automatique de libellés textuels dans une nomenclature donnée peut s'avérer complexe à grande échelle, complexité d'autant plus relevée que la nomenclature est de grande taille.

Historiquement, le système Sicore a été développé dans les années 90 (E. Meyer et P. Rivière [1]) pour les grands usages de codification à travers l'Institut. Basé sur un principe d'« apprentissage », l'algorithme permettait d'identifier de façon déterministe un code unique, ou bien de choisir de ne rien

proposer, menant à une reprise manuelle par un gestionnaire. C'était justement Sicore qui était utilisé dans le cadre de la codification automatique de l'Activité Principale de l'Entreprise dans le répertoire Sirene des entreprises.

Cependant, l'arrivée du guichet unique dans le cadre de la loi PACTE (n° 2019-486 du 22 mai 2019) a offert aux chefs d'entreprises plus de flexibilité dans la description de leurs activités principales mais les rendant ainsi mécaniquement plus verbeux que précédemment. Sicore s'est retrouvé hors de fonctionnement, avec seulement 30% de codage automatique et donc près de 70% de reprise manuelle, surchargeant les équipes Sirene.

L'arrivée des modèles de traitement automatique du langage (NLP) a ouvert la voie à une nouvelle génération de systèmes de codification automatique. Parmi eux, fastText, développé initialement par Facebook AI Research, a été retenu pour son excellent compromis entre rapidité et performance. Son architecture simple, fondée sur des représentations vectorielles de mots et de n-grammes, permet une adaptation rapide à la structure particulière des libellés d'activités.

Depuis le 1er janvier 2023, un modèle *fastText* a donc remplacé aves succès l'algorithme Sicore en production, permettant une automatisation à 80%/ de la codification, avec un temps de réponse unitaire de moins de 100ms et une précision satisfaisante.

1.2. fastText et MLOps

fastText se base sur une représentation vectorielle de *n-grams* (association de sous-éléments) de mots et de caractères. Cette représentation est apprise, comme dans tout modèle d'apprentissage profond, à partir d'un jeu de données d'entraînement. Nous invitions le lecteur à lire le papier original (A. Joulin, E. Grave, P. Bojanowski, et T. Mikolov [2]) pour comprendre les détails de cette méthodologie et l'architecture du modèle

Le déploiement d'un modèle de *machine learning* en production a nécessité de nombreuses adaptations, dans une philosophie *MLOps*. L'idée fondamentale est d'intégrer tout le cycle de vie d'un projet dans un continuum automatisé, de la gestion des données à la surveillance du modèle déployé en passant par l'entraînement et le versionnage des modèles.

Cette approche MLOps s'inscrit fondamentalement dans une perspective d'utilisation de technologies *cloud-native*, en particulier Onyxia. La description des travaux et la philosophie générale du déploiement d'un modèle *fastText* est décrite dans le document de travail *L'apport des technologies cloud pour industrialiser le processus d'innovation statistique* par par Romain AVOUAC, Thomas FARIA et Frédéric COMTE (N° M2025-05– Juillet 2025).

Elle a abouti à des principes clés qui servent de base à la mise en production d'un modèle de *machine learning*:

- séparation stricte des environnements de calcul et de stockage des donées : utilisation de l'espace S3 du SSPCloud pour la gestion des données
- automatisation des pipelines d'entraînement via ArgoWorkflows avec un minimum d'intervention humaine
- utilisation extensive de MLFlow pour la surveillance de l'entraînement, le stockage et le versionnage des modèles, ainsi que son encapsulation pour service (*wrapper* contenant les logiques de traitement du texte)
- service du modèle via une API REST en Python
- travail conjoint entre métiers et innovation (SSP Lab)
- surveillance du modèle via des dashboards basés sur Quarto

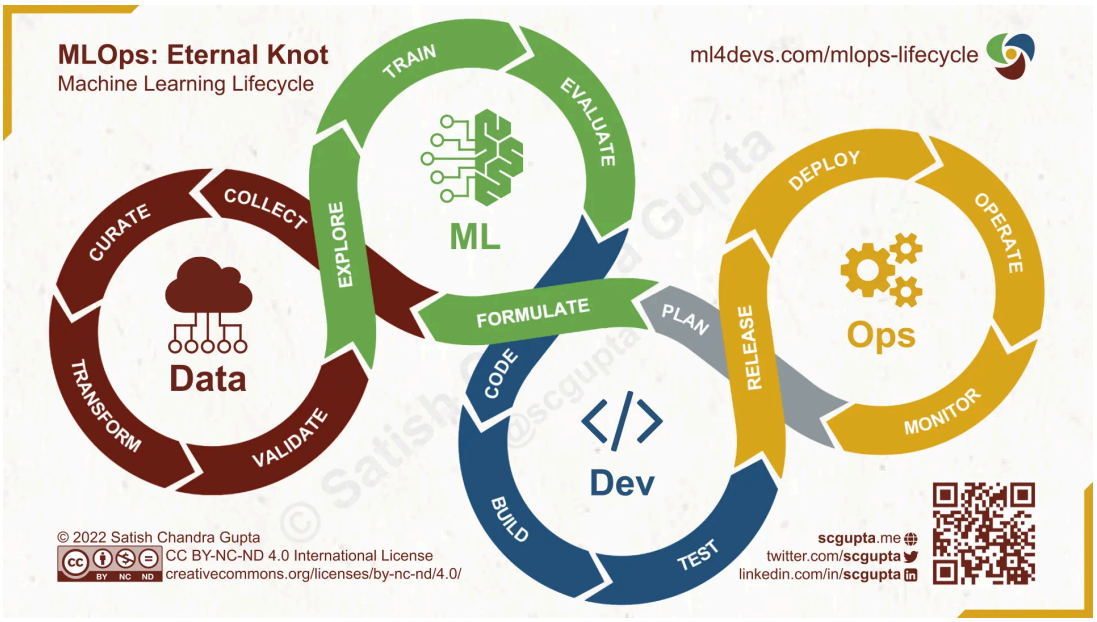


Figure 1. – La pipeline MLOps idéale

2. PyTorch : enjeux et défis de la mise en production

La librairie *fastText* développée par Meta a été archivée en juin 2024, posant différentes question quant à la maintenabilité et la viabilité d'un modèle *fastText* en production.

Le passage à l'écosystème **PyTorch**, standard actuel pour le développement de réseaux de neurones, a donc été envisagé. Il entrouvre de nombreuses possibilités:

- maintenance et développement en interne du modèle de codification automatique (par exemple, avec une prise en compte mieux gérée des variables additionnelles et la possibilité d'améliorer les performances avec des architectures plus sophistiquées)
- connexion à un écosystème dynamique et donc à de nombreuses librairies permettant des ajouts utiles : utilisation de modèles et/ou de tokenizers pré-entraînés depuis *Hugging Face*, explicabilité avec la librairies Captum, calibration des modèles avec torch-uncertainty etc.

La question principale qui restait en suspens était celle de la performance et du temps d'inférence sur CPU. PyTorch est une librairie généraliste, destinée à pouvoir implémenter n'importe quelle architecture de réseaux de neurones, contrairement à la librairie fastText (qui par définition n'implémente que la méthodologie fastText). PyTorch est par ailleurs particulièrement destinée à une utilisation sur GPU : si les GPUs du SSP Cloud permettent un entraînement rapide, la question de l'inférence en production (sans GPU) était cruciale.

Après des tests conclusifs, un modèle PyTorch d'environ 1 million de paramètres a été mis en production, avec des temps d'inférence satisfaisants de moins de 300 ms - bien que supérieurs à ceux de fastText. Mais cette légère dégradation s'accompagne d'une meilleure précision, d'une meilleure calibration, d'un meilleur contrôle sur l'entraînement et des fonctionnalités supplémentaires (voir ci-après).

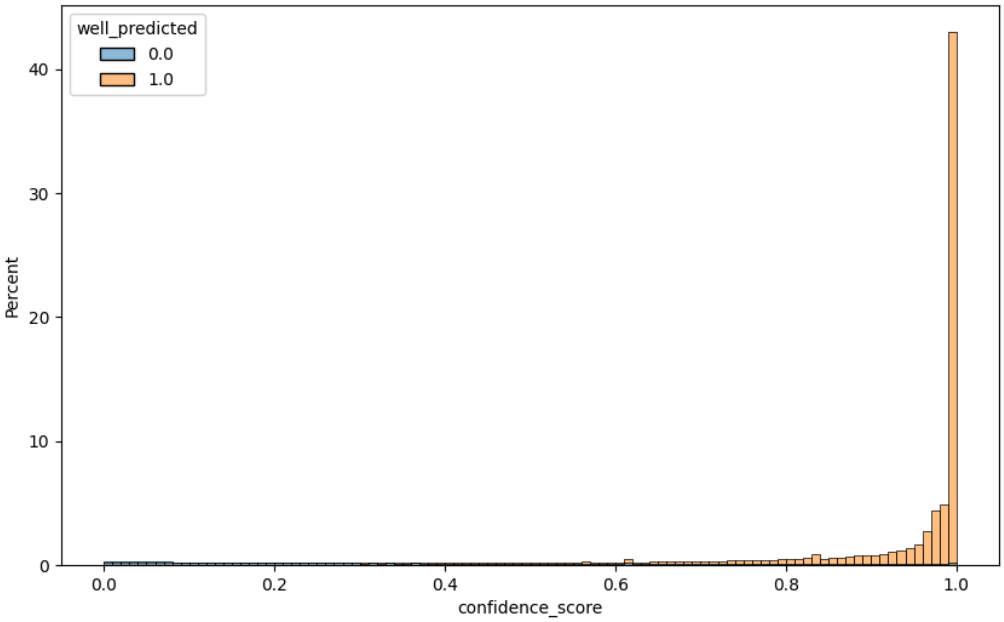


Figure 2. – Sur un échantillon de test de 300 000 libellés, histogramme des niveaux de confiance prédits par le modèle, sur les libellés correctement ou mal classés. Le modèle arrive à bien discriminer, ne se trompant que lorsqu'il a une faible confiance.

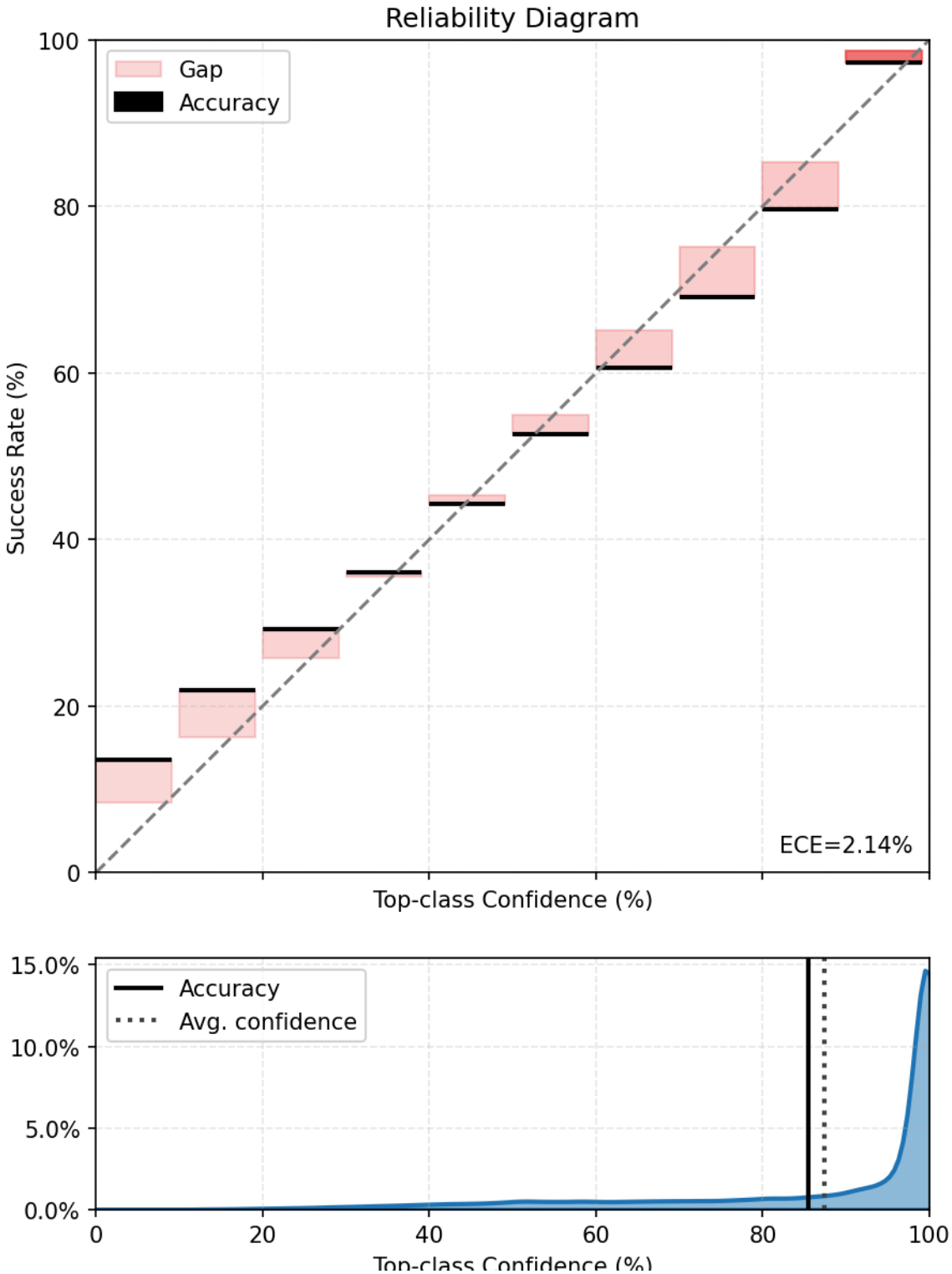


Figure 3. – Sur un échantillon de test de 300 000 libellés, courbe de calibration du modèle. Le modèle est très bien calibré : quand il indique un niveau de confiance de xx%, il est effectivement correct dans xx% des cas.

3. torchTextClassifiers: perspectives

Le SSP Lab a centralisé sous la forme d'un package Python ce modèle PyTorch (disponible sur ce lien), et ce pour plusieurs raisons :

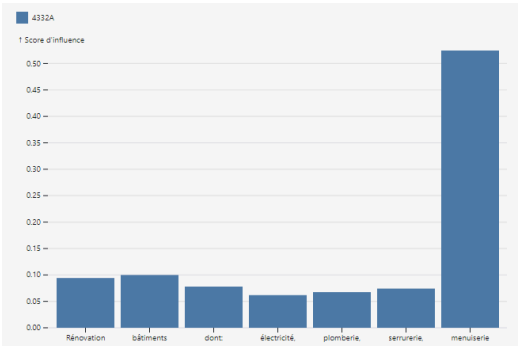
- le modèle a une existence propre et est indépendant de ce pour quoi il est entraîné
- portabilité et versionnage optimisés (l'architecture sous forme de package est accessible depuis PyPI qui agit comme un stockage distant)
- la méthodologie n'est pas propre à la codification de l'APE mais a pour but d'être diffusée et de servir de standard au sein de l'Institut : diffusion rapide et installation efficace
- opportunités de collaboration avec d'autres acteurs

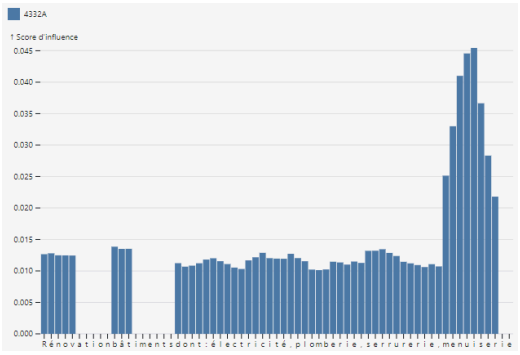
L'idée à terme est d'avoir une librairie permettant d'initialiser et entraîner très facilement un modèle de classification de texte parmi les architectures plus modernes. Les utilisateurs visés sont ceux qui ne peuvent pas utiliser de modèles lourds pré-entraînés depuis HuggingFace mais veulent entraîner efficacement leurs propres modèles de taille réduite - pour une inférence CPU par exemple. Les instituts de statistique publique européens font donc naturellement partie de la cible.

Des fonctionnalités supplémentaires sont également prévues, comme:

- utilisation et entraînement de tokenizers HuggingFace
- explicabilité
- calibration
- quantization

La libraries *torchTextClasifiers* repose sur du PyTorch natif ainsi que le framework PyTorch Lightning, standard actuel pour l'entraînement de réseaux de neurones. Des dépendances optionnelles en fonction des fonctionnalités utilisées peuvent être ajoutées - *Captum* pour l'explicabilité par exemple.





Bibliographie

[1] E. Meyer et P. Rivière, « SICORE, un outil et une méthode pour le chi*H*rement automatique à l'INSEE ». 1997.

[2] A. Joulin, E. Grave, P. Bojanowski, et T. Mikolov, « Bag of Tricks for Efficient Text Classification », *arXiv preprint arXiv:1607.01759*, 2016.