

Année académique
2024-2025



Projet tutoré

Classification automatique de questions sur un
forum politique en anglais

Alexandre Fostier

Venera Gareeva

Theodora Pazakou

Lucas Prévot

Master Technologies de Langues

Enseignant : Pablo Ruiz Fabo

Introduction.....	2
Description des tâches.....	2
1. Téléchargement des données depuis l'API stack exchange.....	2
2. Prétraitement des données.....	3
3. Entraînement d'un modèle de type BERT pour la classification.....	5
4. Entraînement d'un modèle classique : Linear SVC pour la classification.....	7
Planification du travail.....	8
Analyse des résultats.....	9
Comparaison transformers et méthodes classiques.....	13
Le dépôt.....	14
Bibliographie.....	14

Introduction

Le projet présent a pour but de créer un classifieur permettant d'assigner automatiquement un tag aux questions, hébergées sur la plateforme collaborative anglophone Stack Exchange. Il s'agit d'un site contenant un grand nombre de forums thématiques ayant la structure question-réponse. Le contenu des questions (leur texte et éventuellement leur titre) sera pris en compte pour assurer la classification.

La classification automatique est une méthode d'apprentissage automatique largement utilisée dans divers domaines. Ce processus permet de traiter un grand volume de données en s'appuyant sur les données préalablement étiquetées (données d'entraînement). À la suite de l'entraînement sur ces données, il est possible de tester le modèle sur les nouvelles données (données de test), pas utilisées pendant le processus d'entraînement.

Dans ce travail nous allons nous concentrer sur les questions ayant les tags *economy*, *election*, *immigration*, *international relations*, *law* et *media* (les étiquettes). Le texte et le titre des questions seront utilisés comme les caractéristiques de notre jeu de données.

Pour atteindre notre objectif, nous avons divisé le travail en tâches suivantes: collecte de données, prétraitement des données, algorithmes d'apprentissage classiques et apprentissage automatique basé sur l'architecture Transformers.

Description des tâches

1. Téléchargement des données depuis l'API stack exchange

Par Lucas Prévot (Entre 6 et 7 heures de travail)

La première étape dans la création d'un outil de classification automatique est la collecte des données nécessaires à l'entraînement du modèle. Cette phase est essentielle, car la qualité du modèle repose directement sur celle des données utilisées pour son apprentissage. Dans ce projet, cette étape consiste à télécharger les données, composées des caractéristiques (texte et titre des questions) ainsi que des étiquettes (« tags »), indispensables à l'apprentissage supervisé. Théoriquement, ces données peuvent être obtenues de différentes manières, notamment par le biais du « web scraping », une pratique légale mais parfois restreinte par les conditions d'utilisation des sites web, pour des raisons de propriété intellectuelle, mais aussi de charge serveur. La méthode la plus efficace et recommandée par la plateforme Stack Exchange pour obtenir un corpus de grande taille est l'utilisation de son API. Bien que cette API impose une limite quotidienne de requêtes, celle-ci reste suffisante pour notre besoin. (Jusqu'à 10 000 requêtes par jour). La récupération des questions se fait en deux étapes. La première consiste à interroger l'API pour obtenir les questions associées aux étiquettes, mais seules leur titre et un identifiant unique sont renvoyés, sans le texte. Ainsi, la deuxième étape consiste à exploiter chaque identifiant afin d'obtenir les questions dans leur intégralité via l'API. Une fois l'intégralité du corpus téléchargée, celui-ci doit être sauvegardé dans un format approprié et mis à disposition sur GitLab pour l'apprentissage.

Pour cette étape, j'ai travaillé sur la conception du script nommé « `download_questions.py` » disponible sur GitLab. Afin de simplifier la tâche, je me suis largement appuyé sur l'annexe relative à l'utilisation de l'API, fournie dans la description du projet, ce qui m'a rapidement permis de comprendre que cette procédure devait s'effectuer en deux étapes, et était limitée à 100 résultats par requête.

Cette limite, indiquée dans l'annexe, n'a pas posé de problème particulier. J'ai mis en place une boucle « `while` » permettant d'incrémenter les pages jusqu'à épuisement des résultats. Après observation, le fichier JSON renvoyé par l'API précisait, grâce à la clé « `has_more` », la présence ou non de résultats supplémentaires. J'ai donc conditionné la boucle « `while` » à ce paramètre.

Après avoir récupéré l'ensemble des ID des questions, il a été nécessaire de réfléchir à un éventuel filtrage. J'ai travaillé sur ce projet début décembre, et comme je n'étais pas certain des intentions de mes camarades, j'ai décidé de télécharger et de conserver l'entièreté des questions.

J'ai commis une erreur à ce stade : en conservant toutes les questions, y compris celles associées à plusieurs « tags », des doublons se sont glissés dans les données sans que je m'en aperçoive immédiatement, car certains ID sont ressortis sur plusieurs « tags » à la fois. C'est lors de l'exploration manuelle des données, par précaution, que j'ai pris conscience du problème.

J'ai donc édité le code pour ajouter une compréhension de dictionnaire, dont les ID sont les clés, ce qui supprimerait les doublons.

J'ai choisi de sauvegarder les données au format JSON pour sa flexibilité, permettant d'organiser facilement les données sous forme de paires clé-valeur. Afin de conserver un maximum d'informations, j'ai inclus l'ID, le titre, les tags, ainsi que le texte des questions en HTML sous la clé « body_html », ma théorie est que les balises pourraient être utiles pour l'apprentissage, même si cela est peu probable. J'ai ensuite traité ce texte avec *Beautifulsoup* pour obtenir une version nettoyée sans balises HTML, sauvegardée sous la clé « body_text », qui devrait être la version de travail que mes camarades vont utiliser. De plus, j'ai entrepris un prétraitement en supprimant les URLs, sauts de ligne et espaces consécutifs via expressions régulières, et sauvegardé cette version dans « body_text_clean ». Incertain des futurs besoins du projet, conserver ces trois versions du texte permettrait à mes camarades de travailler selon leurs besoins spécifiques.

Une fois le script et les données prêtes à être envoyées sur Gitlab, j'ai créé une « branche » et fait une « merge request », pour que mon travail soit examiné au préalable. Pour réaliser la première version de ce script, le processus de réflexion et l'écriture du code a pris entre 4 et 5 heures le 13 décembre. Cependant, non satisfait de la qualité du code qui me semblait difficile à lire, j'ai procédé à une révision le 28 décembre, améliorant la structure, le nom des fonctions et des variables, et en multipliant les commentaires, ce qui m'a pris environ deux heures.

2. Prétraitement des données

Par Venera Gareeva

La deuxième étape, qui suit la collecte des données, est le prétraitement des données. Cette étape est cruciale pour le processus d'apprentissage automatique, surtout pour les algorithmes d'apprentissages dites « classiques », comme la régression logistique, par exemple. Il est important de noter que pour les modèles plus complexes, comme Transformers, il ne s'agit pas de même type de traitement. Ces modèles ont souvent leur propre tokeniseurs intégrés, qui transforme le texte brut en tokens, adapté à leurs vocabulaires pré-entraînés.

Ainsi, dans ce travail, après avoir chargé les données collectées en format json, pour faciliter leur manipulation je les ai convertis en dataframe à l'aide d'une bibliothèque pandas, qui est largement utilisé pour l'analyse des données en TAL.

En explorant notre dataframe, je me suis rendu compte de quelques coquilles dans les données, par exemple, lors de la lecture du fichier isn't apparaît comme isn't, ce qui correspond à un code HTML. Pour pallier ce problème, j'ai utilisé la bibliothèque Python html pour décoder les entités HTML dans les colonnes contenant le texte et titre des questions. Ensuite, j'ai filtré les données pour avoir seulement les questions ayant les tags qui nous intéressent, à savoir economy, immigration, election, international relations, law et media, en ajoutant la condition que ces tags ne se superposent pas, c'est-à-dire une question ne devrait pas avoir plus qu'un tag parmi ceux mentionnés ci-dessus. Etant donné que les tags seront les étiquettes à deviner par le modèle je les ai transformés en type category.

Lors du comptage de valeurs de ces étiquettes, on observe que la distribution des questions dans notre jeu de données n'est pas homogène. Le tag « election » est presque 4 fois plus représenté que le tag « immigration » (322 valeurs contre 83). Comme notre but c'est de réaliser un apprentissage automatique supervisé à partir des données textuelles, il est nécessaire de les prétraiter. La transformation de données textuelles en données numériques suppose la conservation de leurs propriétés sémantiques.

Dans ce projet nous travaillons avec les données en anglais. Comme c'est une langue dite « bien dotée », nous avons un choix assez large des librairies qui proposent le traitement correspondant (NLTK, Spacy, Keras). Je choisis à cette fin Spacy, une bibliothèque puissante dans le domaine TAL, qui permet d'effectuer la tokenisation (découpage en tokens), lemmatisation (qui consiste à trouver la forme canonique d'un mot), la reconnaissance des entités nommées et encore beaucoup d'autres traitements. Je téléchargé donc un modèle pour la langue anglaise (en_core_web_sm) et je crée un pipeline (une séquence de traitements) qui retourne les données traitées. Ainsi, le prétraitement que j'ai prévu pour nos données sont:

- Tokenisation
- Suppression des mots outils
- Suppression de la ponctuation
- Lemmatisation

J'ai également utilisé la bibliothèque NLTK pour pouvoir appliquer la désuffixation.

- Désuffixation (stemming)

Suppression des mots outils consiste à éliminer les mots très fréquents qui ne porte pas de valeur sémantique importante, ce qui permet de réduire la taille des vecteurs. Ce sont les mots qui apparaissent dans toutes sortes de documents, quelle que soit leur thématique. Comme il s'agit d'une tâche de classification, nous considérons que la conservation de ces mots n'est pas nécessaire, contrairement à d'autres tâches (par exemple, analyse des sentiments). Spacy à cet égard contient une liste intégrée de mots-outils (stopwords) pour chaque

langue. Spacy permet également de supprimer la ponctuation dans notre jeu de données en filtrant les tokens qui ne sont pas marquées comme `is_punct`. D'une manière similaire je procède à la lemmatisation en l'intégrant dans un pipeline et en l'appliquant aux colonnes contenant le texte et le titre des questions.

Un autre traitement que je décide à appliquer à nos données est la désuffixation (stemming), appelé également racinisation. Cette étape consiste à trouver le radical d'un mot en supprimant certains suffixes. Pour chaque langue la désuffixation se fait suivant ses propres règles. Il existe plusieurs algorithmes proposant ce traitement, par exemple Snowball, Porter, Krovetz ou Lancaster. Pour ce travail, j'ai choisi l'algorithme de Porter, qui est très populaires et efficace pour l'anglais (snowball pour le français). Les données prétraitées après chaque étape je conserve dans des colonnes correspondantes. À l'issue de toutes ces manipulations, je sauvegarde mon dataframe en format csv.

3. Entraînement d'un modèle de type BERT pour la classification

par Theodora Pazakou

La troisième étape consiste à utiliser le jeu de données produit par les étapes précédentes pour entraîner un modèle de type BERT. Étant donné que le modèle utilisé repose sur l'architecture Transformers, les prétraitements n'ont pas été utilisés pendant les expériences menées avec ce modèle. Seules les colonnes `body_text_clean`, `title` et la colonne catégorielle `tags` ont été exploitées dans le cadre de cette étape.

Le modèle que j'ai choisi est une version plus légère de BERT, appelée DistilBERT, qui est disponible sur Hugging Face¹. Comme décrit dans l'article qui a proposé ce modèle (Sanh et al., 2020), DistilBERT fonctionne en conservant 95 % des performances de BERT tout en étant 60 % plus rapide. Il est obtenu grâce à une technique appelée *knowledge distillation*, où un modèle plus petit (l'élève) apprend à reproduire le comportement d'un modèle plus grand (l'enseignant) en s'entraînant sur les mêmes données. Par rapport à BERT base, DistilBERT réduit le nombre de paramètres de 40% tout en conservant la structure des *transformers*, ce qui le rend plus efficace en termes de calcul et plus rapide à entraîner, tout en restant performant sur diverses tâches. J'ai choisi alors ce modèle pour des raisons pratiques, notamment la limitation des ressources du notebook sur Google Colab, avec une utilisation restreinte de la GPU.

Dans un premier temps, après la récupération du jeu de données depuis Git, j'ai réalisé une analyse exploratoire des données, similaire à celle effectuée par Venera lors de l'étape précédente. La distribution des données (Figure 1) n'est pas homogène : les catégories *international-relations* et *election* sont les mieux représentées, tandis que les catégories *media* et *immigration* sont les moins représentées, avec un écart significatif. Étant donné ce déséquilibre, je m'attendais

¹ <https://huggingface.co/distilbert/distilbert-base-uncased>

à ce que les catégories les moins représentées soient moins bien prédites dans les expériences.

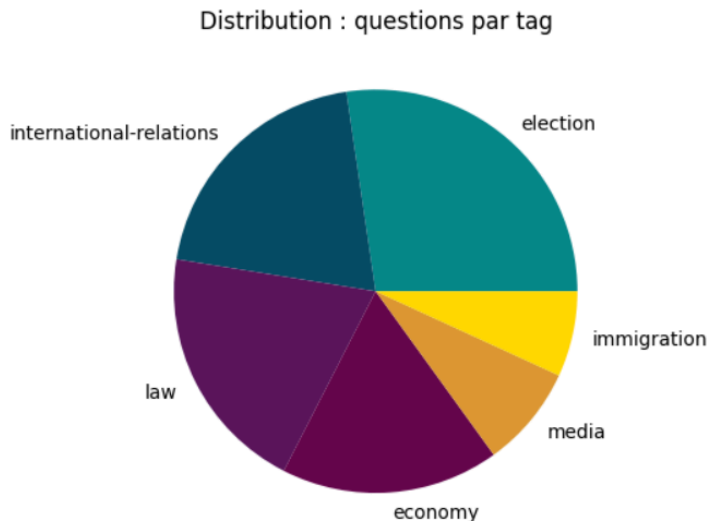


Figure 1 : Distribution des tags dans le jeu de données

Après avoir divisé le jeu de données en ensembles d'entraînement, de validation et de test en utilisant la fonctionnalité *train_test_split* de la bibliothèque *scikit-learn*, je l'ai préparé pour l'entraînement. Cette préparation inclut l'ajout d'une colonne supplémentaire dans le *DataFrame* afin d'attribuer un identifiant numérique à chaque ligne de la colonne catégorielle *tags*, de sorte que chaque tag corresponde à un nombre (par exemple, *economy*: 0). Ensuite, j'ai appliqué la tokénisation des données à l'aide du tokéniseur dédié à ce modèle, tel que défini sur Hugging Face. Étant donné que la tokénisation avec les Transformers consiste à découper les mots les moins fréquents en sous-mots, cette étape était essentielle pour garantir une bonne compatibilité avec le modèle.

J'ai poursuivi en testant deux méthodologies différentes : l'utilisation du modèle pour l'extraction des plongements contextuels, qui sont ensuite exploités par des modèles d'apprentissage classiques, et l'affinage du modèle directement sur le jeu de données. Ces deux approches ont été testées séparément sur les questions et sur leurs titres. J'ai commenté les résultats obtenus dans le notebook, et une discussion plus détaillée de ces résultats est présentée dans la section *Analyse des résultats*.

Pour l'extraction de plongements, j'ai récupéré le hidden state et testé ces plongements avec les algorithmes Linear SVC, régression logistique et Random Forest. Pour l'affinage du modèle, grâce à sa légèreté, j'ai pu tester 10 époques afin d'analyser l'évolution de la perte et de la mesure F1, et ainsi déterminer le

nombre d'époques le plus adapté à cette tâche. Au total, les expériences, l'analyse et la visualisation des résultats obtenus m'ont pris environ 10 heures.

4. Entraînement d'un modèle classique : Linear SVC pour la classification

par Alexandre Fostier

Nous avons également souhaité tester les performances de modèles classiques sur cette tâche de classification.

Après l'étape de pré-traitement, j'avais accès à trois versions de tokenisation des articles et de leur titre : une version 'normale', une version toknisée et une version avec stemming.

Pour exploiter le texte dans les modèles, les données doivent être vectorisés. A la différence des modèles testés par Théodora, ici, le vocabulaire est construit uniquement à partir des données d'entraînement.

J'ai donc construit trois vocabulaires avec un Vectorizer Tf-idf dont j'ai limité la longueur à 5000 tokens.

En me reposant sur des expériences similaires, j'ai pensé que la version lemmatisée serait celle qui donnerait les meilleurs résultats, c'est donc cette version que j'ai utilisée pour tester une sélection de modèles classiques avant d'effectuer un entraînement plus complet et de comparer les performances des différentes tokenisations.

Les modèles testés, issus de la bibliothèque scikit learn, sont les suivants : MultinomialNB, DecisionTreeClassifier, LogisticRegression, KNeighborsClassifier, RandomForestClassifier et LinearSVC. La baseline est simplement une prédiction de la classe la plus fréquente (27% d'exactitude).

Après une validation croisée à 5 plis, le modèle le plus performant s'est avéré être le Linear SVC (avec 81% d'exactitude).

J'ai donc ensuite entraîné ce modèle sur la totalité des données d'entraînement avant de l'évaluer sur les données de test. Les résultats étaient similaires à la validation croisée.

Ces performances sont assez bonnes étant donné la simplicité du modèle et la proximité et l'ambiguïté entre certaines classes. Néanmoins, les prédictions se basent sur une représentation 'sparse' des documents. J'ai donc tenté d'améliorer le système en ayant recours à la réduction de dimension par SVD (TruncatedSVD de scikit learn) pour obtenir des représentations denses. Après quelques essais, j'ai choisi une dimension de vecteur de 250 (contre 5000 précédemment). Ce

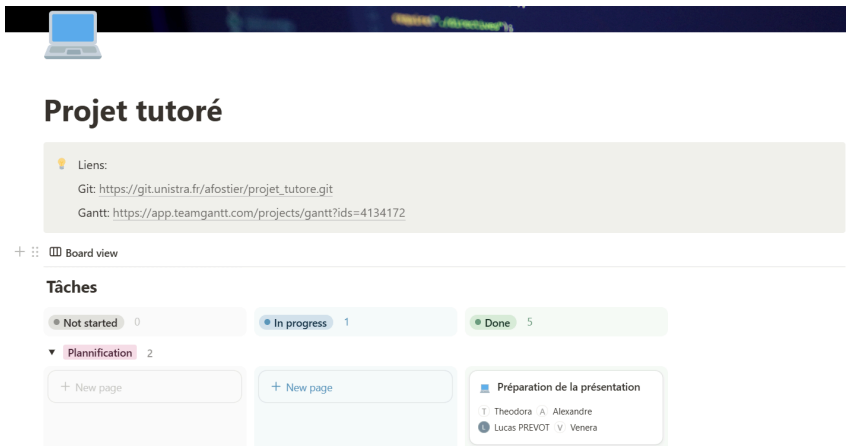
changement a permis une amélioration de 5 points de l'exactitude globale du modèle.

Enfin, j'ai tout de même vérifié les performances des classifieurs qui utiliseraient une tokenisation classique, et une tokenisation avec stemming. Comme prédit, c'est bien la lemmatisation qui offre les meilleurs résultats. Le stemming atteint une exactitude de 83% et la tokenisation classique, 81%.

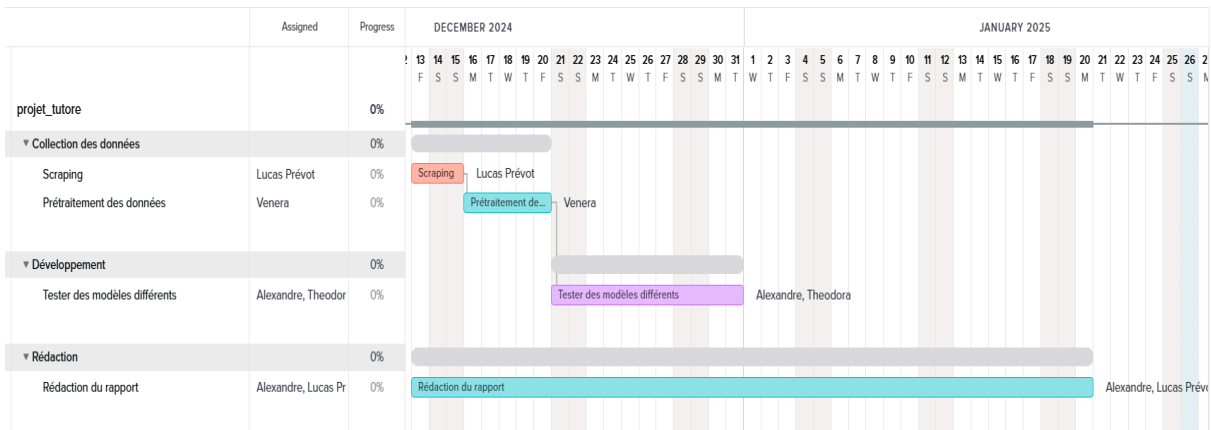
Planification du travail

Pour les besoins de ce projet, nous avons utilisé différents outils pour planifier notre travail. Le suivi du projet s’est fait sur GitLab, les communications ont eu lieu sur WhatsApp, et pour l’organisation du travail ainsi que la répartition des tâches, nous avons utilisé deux plateformes : Notion et TeamGantt.

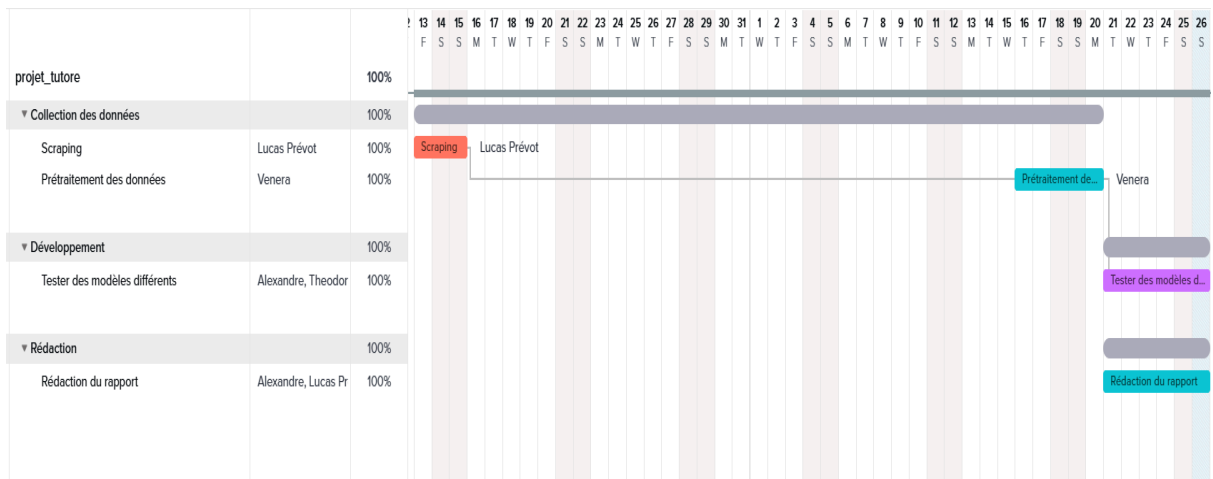
Notion, une plateforme collaborative créée pour la planification des tâches et le suivi des projets, nous a permis de diviser le travail et suivre le progrès de chacune de ces tâches.



Le plan construit au début de ce projet est résumé dans le diagramme de Gantt suivant :



Le charge de travail du master nous a amené toutefois à modifier ce plan. La version finale est la suivante :



Analyse des résultats

Transformers

Concernant les résultats obtenus avec des algorithmes d'apprentissage classiques utilisant les plongements contextuels générés par DistilBERT, l'algorithme qui a obtenu les meilleures performances est la régression logistique, avec un score F1 de 0.75 lorsque l'entraînement est réalisé sur les questions et de 0.71 sur les titres. Toutefois, ces résultats restent relativement insatisfaisants, ce qui peut s'expliquer par plusieurs facteurs.

Tout d'abord, le modèle utilisé pour générer les plongements est probablement inutilement complexe par rapport à la tâche, à la taille du jeu de données et à la longueur des questions. Les modèles encodeurs de type BERT, en raison de leur architecture complexe, ne sont pas toujours adaptés aux petits jeux de données. De plus, puisque l'entraînement avec des plongements contextuels n'a pas donné de meilleurs résultats qu'une méthode plus simple comme TF-IDF, la question de la nécessité d'une approche aussi complexe est soulevée, particulièrement si des méthodes plus simples s'avèrent plus efficaces.

Les résultats s'améliorent avec l'affinage du modèle sur notre jeu de données. L'affinage permet au modèle de s'adapter spécifiquement aux caractéristiques de la tâche et à la distribution des données. En ajustant les poids du modèle sur notre propre jeu de données, le modèle apprend à mieux comprendre les relations entre les données et leurs catégories, ce qui améliore les prédictions (F1-score (questions) : 0.84 et F1-score(titres) : 0.81). De plus, contrairement à l'utilisation des plongements contextuels extraits, l'affinage permet de tirer pleinement parti de la capacité du modèle à capturer des informations contextuelles et des nuances spécifiques au domaine de notre projet. Il est toutefois à noter que le modèle a été entraîné pendant 10 époques, avec les meilleurs résultats obtenus à la 9e époque. Cependant, les résultats se sont stabilisés à partir de la 5e époque, sans amélioration significative après ce point, ce qui signifie que l'entraînement devrait plutôt être arrêté à la 5e époque pour éviter que le modèle ne devienne sur-ajusté aux données d'entraînement.

Le déséquilibre des classes a effectivement influencé les performances du modèle, confirmant notre hypothèse, comme le montrent les matrices de confusion (Figure 2). La classe immigration est mal prédite dans les deux cas (affinage sur les questions et sur les titres), ce qui est logique, car le nombre d'exemples était probablement insuffisant pour que le modèle puisse bien généraliser.

La catégorie *media* est relativement bien prédite dans le cas de l'entraînement sur les questions, ce qui suggère que, malgré un nombre restreint d'exemples, le contenu des questions est suffisamment informatif pour la catégorie. En revanche, cela n'est pas le cas avec l'affinage sur les titres, probablement parce que le

contenu des titres ne fournit pas assez d'informations contextuelles pour une bonne prédiction.



Figure 2 : Matrices de confusion du modèle affiné sur les questions (à gauche) et sur les titres (à droite)

Concernant la catégorie law, le nombre de confusions n'est pas uniquement lié à sa distribution dans le jeu d'entraînement. L'analyse des exemples par perte décroissante suggère l'existence de chevauchements thématiques et d'un vocabulaire partagé entre plusieurs catégories, ce qui explique la difficulté du modèle à faire des prédictions précises.

La comparaison des modèles testés est résumée dans la Figure 3.

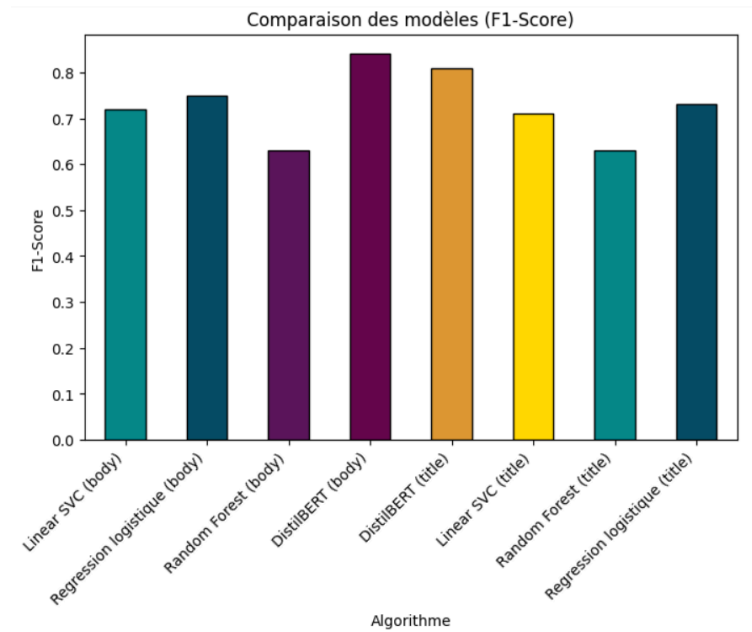


Figure 3 : Comparaison des résultats

Linear SVC avec données lemmatisées et réduction de dimension par SVD

On commentera ici uniquement le meilleur modèle classique obtenu.

Ce modèle atteint une exactitude de 0.86, soit semblable aux performances du modèle avec les plongements DistilBERT entraîné uniquement sur le corps des articles (0.85). Une comparaison entre ces modèles sera effectuée dans la partie suivante.

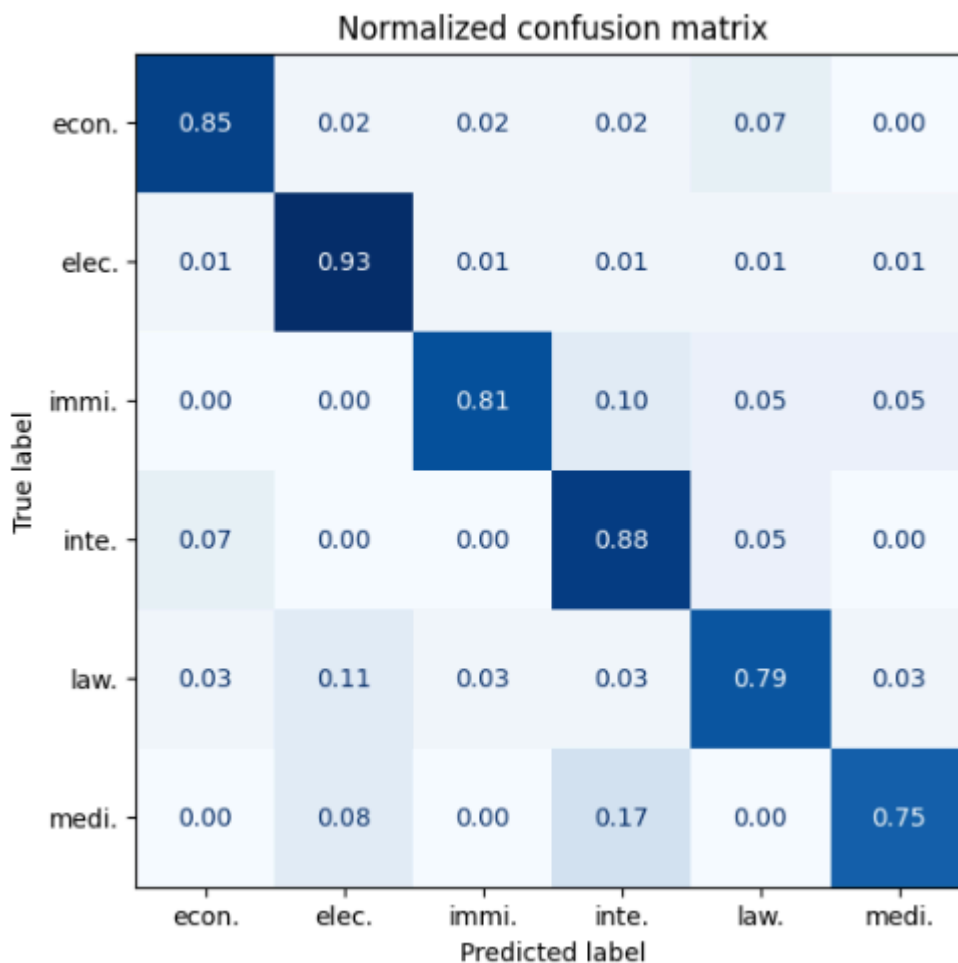
On remarque que les catégories 'law' et 'media' sont les moins bien classées. Il y a bien un déséquilibre entre les classes dans les données d'entraînement, mais cela n'explique pas tout. En effet, la catégorie 'law' était pourtant parmi les plus présentes dans les données d'entraînement. De plus, même s'il n'y a que 74 articles 'media', il y a encore moins d'articles 'immigration' (62) dont la catégorie est pourtant mieux classifiée (f1-score immigration : 0.83; f1-score media : 0.75). Cette performance n'est pas étonnante puisqu'on s'attend à ce que les articles traitant des médias mentionnent d'autres sujets.

	precision	recall	f1-score	support
economy	0.88	0.85	0.86	41
election	0.92	0.93	0.93	75
immigration	0.85	0.81	0.83	21
international-relations	0.83	0.88	0.85	40
law	0.81	0.79	0.80	38
media	0.75	0.75	0.75	12
accuracy			0.86	227
macro avg	0.84	0.84	0.84	227
weighted avg	0.86	0.86	0.86	227

En s'intéressant à la matrice de confusion, on devine qu'il s'agit pour les données de test de questions relatives aux relations internationales et aux élections, ces classes étant souvent confondues.

Néanmoins, la catégorie 'elections' qui était largement sur-représentée est aussi celle étant la mieux classifiée, et ce largement au-dessus des autres. On ne peut donc pas complètement écarter l'influence de la quantité de données d'entraînement.

On décèle également des confusions entre les classes 'international relations' et 'immigration', mais aussi entre 'election' et 'law'. Cela est tout à fait cohérent puisque ces sujets se rejoignent et que les articles partagent donc une partie du vocabulaire.



Comparaison transformers et méthodes classiques

Tout d’abord, notons que le modèle classique (LinearSVC avec TF-IDF) est entraîné à la fois sur le titre et le corps de l’article alors qu’il y a deux modèles transformers (DistilBERT affinés) entraînés séparément sur ces différentes parties des données. On effectuera donc cette comparaison avec le modèle transformers entraîné sur le corps de l’article.

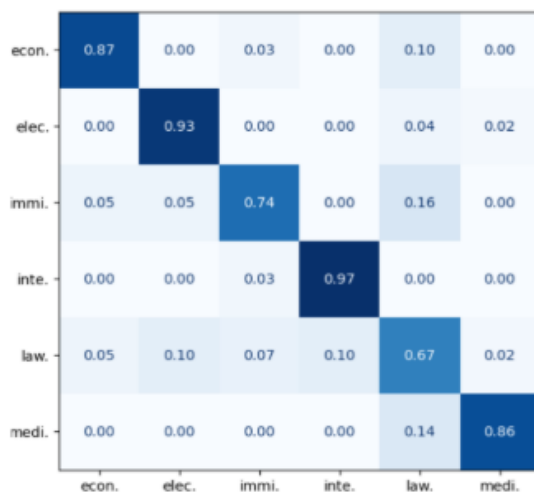
Ensuite, il faut considérer que la différence entre ces approches tient davantage de la représentation des documents que de la nature des modèles. L’approche à base de transformers repose sur des plongements affinés sur le corpus d’entraînement. On a donc ici des représentations plutôt sémantiques. Dans l’approche classique, par contre, les représentations sont plutôt d’ordre lexical.

De manière globale, les performances de ces deux modèles sont similaires : 0.86 d’exactitude pour le LinearSVC contre 0.85 pour le modèle DistilBERT affiné. Les performances de classification des différentes catégories suivent également une

tendance relativement similaire, comme on peut le remarquer facilement avec les matrices de confusion. On retrouve néanmoins des confusions assez différentes, et deux catégories avec des performances qui ne concordent pas entre les deux modèles : 'media'(+0.11, mieux catégorisée par transformer) et 'law' (+0.12, mieux catégorisée par méthode classique).

Dans le modèle transformers, la catégorie 'law' est plus souvent prédite à tort, notamment à la place de 'immigration' ou 'economy'. On peut imaginer que les plongements DistilBERT ont repéré et donné plus d'attention aux thèmes légaux dans des questions traitant de ces catégories alors que cela n'a pas été le cas avec le TF-IDF.

Pour la catégorie 'media', c'est une confusion avec la catégorie 'international relations', pourtant complètement absente dans l'évaluation du modèle transformer, qui fait baisser les performances de classification dans le modèle classique. Ici, on peut supposer qu'il s'est passé l'inverse et que le chevauchement du vocabulaire entre les catégories a induit le modèle classique en erreur.



DistilBERT



LinearSVC

Le dépôt

Le lien vers le dépôt Git qui comprend le code, les données, et le rapport :

https://git.unistra.fr/afostier/projet_tutore

Bibliographie

Sanh, V., Debut, L., Chaumond, J., & Wolf, T. (2020). *DistilBERT, a distilled version of BERT: Smaller, faster, cheaper and lighter* (arXiv:1910.01108). arXiv.
<https://doi.org/10.48550/arXiv.1910.01108>