Department Génie Mathématiques et Modélisation
5GMM MMS

IA FRAMEWORKS

Défi Intelligence Artificielle

Camille Bason
Margaux Berthaud
Ana Cavillon

Janvier 2021

Table des matières

Table des figures

Liste des tableaux

1 Introduction

Le défi de cette année s'appuyait sur du Natural Language Processing. L'objectif était d'attribuer la bonne catégorie de métier à une description de métier, donc il s'agissait de classification multi-classes avec 28 classes. Les données sont issues de CommonCrawl et représentent donc ce que l'on peut trouver sur la partie "anglaise" d'internet, ce qui explique également qu'il y ait du biais. L'objectif de la compétition est d'implémenter une solution exacte et juste.
Le classement est réalisé selon la métrique Macro F1 utilisée pour construire le tableau des scores sur Kaggle. L'objectif est d'avoir le score le plus élevé sur les données de test privées sur Kaggle. Les classements secondaires s'appuient sur l'équité entre les genres, c'est à dire l'absence de biais à ce niveau. La reproductibilité, la lisibilité et la créativité sont les derniers critères de classement.
Dans ce rapport nous présentons les méthodes et algorithmes que nous avons utilisés afin de réaliser ce défi.

2 Exploration des données

Pour réaliser ce défi nous disposons de différentes données :

— train.json : contient les descritpions de métiers et les genres de la base d'apprentissage avec 217 197 échantillons.
— train_label.csv : contient les labels des métiers associés à la base d'apprentissage.
— categories_string.csv : contient la correspondance entre les labels des métiers et un nombre entier qui sert pour les soumissions sur Kaggle.
— test.json : contient les descriptions des métiers et les genres pour la base de test avec 54 300 échantillons.

Nous disposons également d'un modèle de format pour les soumissions.

Dans un premier temps nous avons effectué une analyse exploratoire des données afin de mieux visualiser le dataset. Pour commencer, nous avons observé la répartition des données d'entrainement dans les différentes classes de métier, représentée dans la figure 1. On observe ici, une grande disparité au niveau des classes, avec notamment les *professor* qui sont sur-représentés. Les métiers les moins bien représentés sont les métiers de *DJ* et de *rapper*. Nous avons ensuite représenté la quantité de vacabulaire pour chacun des métiers en figure 2. On retrouve également la classe *professor* qui se distingue grâce à la diversité des mots qui la décrivent. Il semblerait que la diversité de vocabulaire ait un lien avec le nombre d'occurence de chaque métier : plus un métier est présent dans le data set, plus son vocabulaire sera diversifié. Par exemple la classe *professor* a un vocabulaire plus diversifié que les autres classes, nous pouvons alors penser qu'il sera plus difficile de prédire ce métier. Cela dit, comme son occurence dans le data set est très importante on peut supposer que cette diversité de vocabulaire n'aura pas d'effet visible.

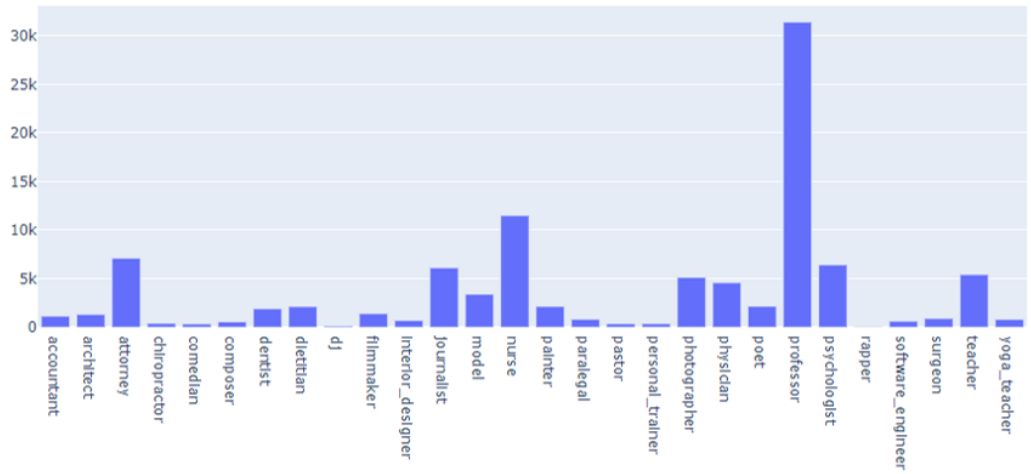

FIGURE 1: Distribution des classes

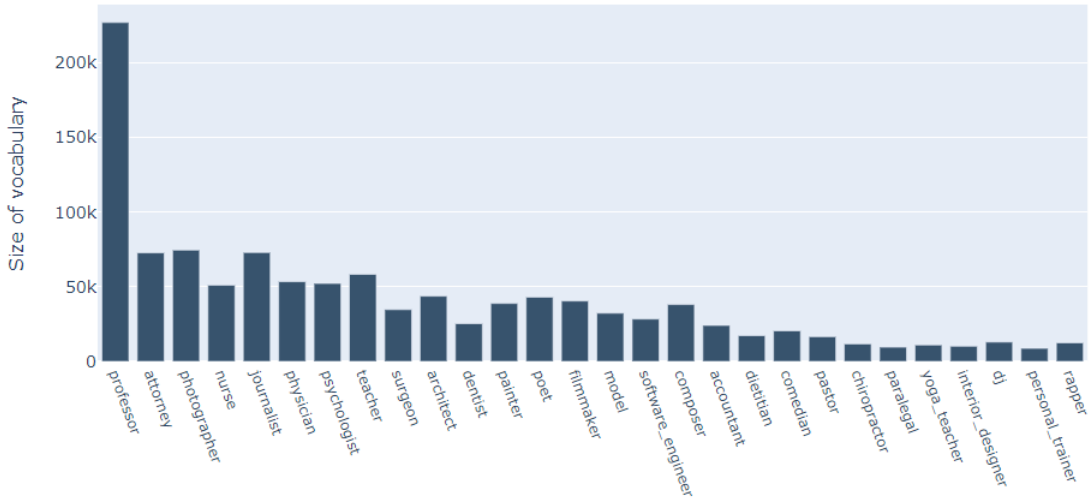

FIGURE 2: Vocabulaire selon les classes

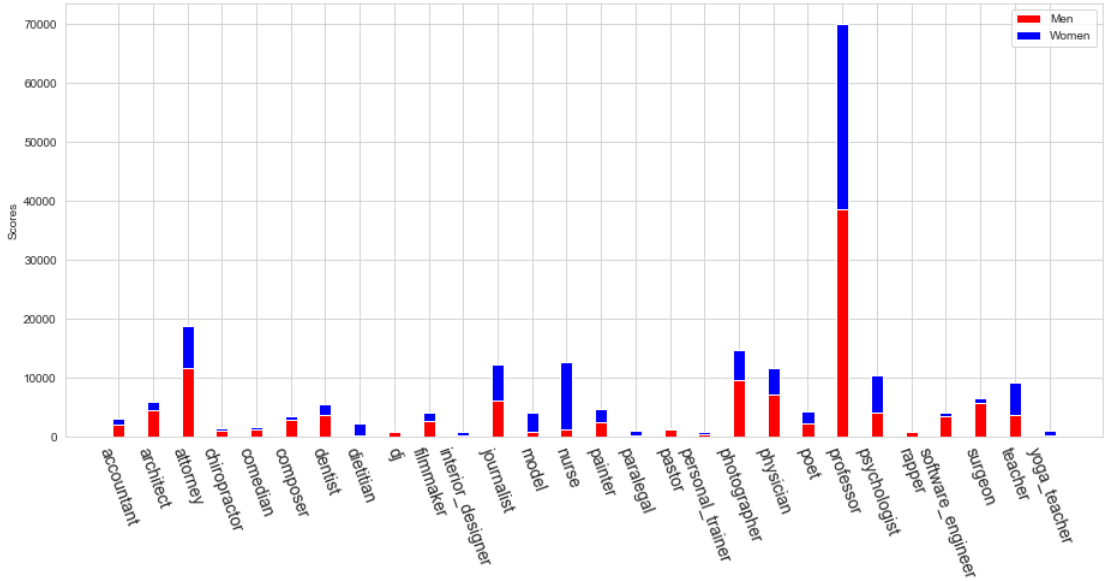

FIGURE 3: Distribution des classes pour les femmes et les hommes

Enfin, dans la figure 3, nous représentons la distribution des métiers selon le genre. Dans ces barplots nous remarquons que tous les métiers ne sont pas représentés avec les mêmes quantités et que la fréquence de chaque métier dépend aussi du genre. Par exemple, nous retrouvons plus de *nurse* et de *yoga teacher* chez les femmes et plus de *surgeon* et de *rapper* chez les hommes. Cet aspect fait d'ailleurs partie des objectifs du défi, avec une volonté de limiter le biais entre les genres dans les classifications. Malheureusement, nous n'avons pas traité cet aspect dans notre code.

L'objectif est donc d'entrainer nos différents algorithmes sur la base d'apprentissage et de les appliquer sur la base de test pour ensuite soumettre cette classification et obtenir le meilleur score sur Kaggle.

3 Méthode

3.1 Nettoyage des données

Nous avons commencé par nettoyer les données d'apprentissage, cela consiste en plusieurs méthodes. Tout d'abord, les caractères de code html sont retirés et tous les mots sont mis en minuscule. Les accents et les caractères qui ne sont pas des lettres sont retirés également. Ensuite, l'exclusion des "stopwords" permet de retirer tous les mots de langage qui n'ont pas d'importance dans la classification. Et enfin tous les mots sont transformés en leur racine afin d'éviter les différentes conjugaisons, par exemple, qui compliqueraientt la classification.
Pour retirer les stopwords, nous utilisons les stopwords anglais contenus dans le package nltk.corpus. Cependant, en affichant les wordcloud des mots les plus présents dans les descriptions de métier après nettoyage, nous nous sommes rendues compte que certains mots étaient très fréquents alors qu'ils n'apportaient rien à la classification selon nous (voir figure 4b). Ainsi, nous avons ajouté des stopwords à supprimer de la description : 'interest', 'interests', 'include','includes', 'including', 'received', 'new', 'work', 'works', 'working', 'worked', 'current', 'currently' (voir figure 4c).

(a) Wordcloud avant nettoyage

(b) Wordcloud après nettoyage

(c) Wordcloud après nettoyage et suppression de davantage de stopwords

FIGURE 4: Wordclouds

3.2 Vectorisation

Ensuite nous avons mis en place une vectorisation, c'est-à-dire que nous avons transformé tous les objets string en objets numériques. Il existe pour cela plusieurs méthodes comme TF-IDF, OHE, WORD2VEC Skip-Gram, WORD2VEC CBOW, FastText, Glove...
TFIDF et One-Hot-Encoder sont basées sur la fréquence des mots dans les descriptions. Ces deux méthodes se différencient par la numérisation des mots qui est faite sans tenir compte de l'importance du mot dans OHE mais en utilisant des poids pour TF-IDF. Cette étape est assez rapide et est facilement réalisable sur nos ordinateurs.

Nous avons également testé les méthodes de vectorisation qui s'appuient sur l'apprentissage avec le Word Embedding, afin d'exploiter les relations entre les mots. Par exemple avec Word2Vec qui contient elle-même deux versions : Continuous Bag Of Word (CBOW, le contexte est pris en entrée alors que la cible est en sortie) et Skip-Gram (au contraire, la cible est en input et le contexte en output). Enfin nous pouvions utiliser Glove qui effectue une vectorisation avec des propriétés globales, et non locales comme Word2Vec, mais l'algorithme ne fonctionnait pas sur nos ordinateurs qui n'étaient pas des MAC...

3.3 Classification

Après avoir sélectionné une méthode de vectorisation, nous nous sommes penchées sur les méthodes de classification, afin de voir si cela pouvait influer sur le résultat de la métrique Macro F1.
Nous avons utilisé principalement trois méthodes : Regression logistique, Random Forest et SVC. Random Forest est un algorithme incontournable en Machine Learning qui se compose de plusieurs arbres de décision indépendants qui utilisent le tree bagging et le features sampling. L'algorithme utilise ensuite le vote de chaque arbre pour prendre une décision de classification. Nous avons implémenté Random Forest grâce à la librairie *sklearn.ensemble.RandomForestClassifier*. La régression logistique est un modèle statistique et linéaire. Grâce à l'optimisation des coefficients de régression on obtient la probabilité d'appartenance aux classes. Nous avons utilisé la librairie *sklearn.linear_mode.LogisticRegression* pour mettre en place ce modèle. Nous détaillerons le fonctionnement de Support Vector Classification dans la partie suivante.

Nous avons également décidé d'utiliser les modèles MLP (Perceptron multi-couches) et Naive Bayes classifieur. Cependant, nous n'avons pas effectué de soumissions pour ces modèles car nous obtenions déjà une accuracy faible lors de la cross validation pour l'optimisation des paramètres. En effet, les meilleurs paramètres pour MLP permettaient d'obtenir une accuracy de 0.551 et ceux de Naives Bayes permettaient d'obtenir une accuracy de 0.601.

Nous avons ensuite tenté d'ajouter des méthodes à TF-IDF et SVC optimisé dans le but d'augmenter le score obtenu. Tout d'abord nous avons voulu pallier au problème des classes déséquilibrées, il y avait donc deux possibilités : sous-échantillonner ou sur-échantillonner les données. Dupliquer les données peut entraîner du sur-apprentissage donc nous avons utilisé SMOTE (synthetic minority oversampling technique) afin de sur-échantillonner les classes qui étaient trop peu représentées dans notre base d'apprentissage comme *rapper*, *yoga teacher*... Cette méthode vise à équilibrer les classes en augmentant aléatoirement les exemples des classes les moins représentées, comme nous pouvons le voir dans l'article [1]. SMOTE parcourt les éléments de ces classes et cherche les k plus proches voisins des points afin de créer de nouvelles observations. Ainsi, au lieu de 217 197 descriptions nous en obtenons 286 430.

Nous avons également tenté d'associer des poids aux classes afin de réduire les déséquilibres. Pour cela nous avons utilisé le paramètre *class_weight* en mode *balanced*, cet argument permet d'associer à chaque classe un poids inversement proportionnel à sa fréquence, c'est à dire que l'algorithme va pénaliser les erreurs de classification des classes minoritaires. Cette méthode insère donc un biais afin d'améliorer la prédiction des classes sous représentées (voir [4]).

Nous avons enfin ajouté l'algorithme de classification OneVsRest comme suggéré dans les articles [3] et [2]. Le principe est de considérer linearSVC en classifieur binaire et d'avoir une classification pour chaque classe, c'est-à-dire que l'on compare une classe à toutes les autres. Ainsi au lieu d'avoir une classification multi-classe nous avons plusieurs classifications binaires, et l'algorithme est entraîné sur chacun d'entre eux. Cet algorithme s'appuie sur la méthode one-vs-rest et prédit donc une probabilité d'appartenance à chaque classe, et l'indice de la classe ayant la plus grande probabilité est alors utilisée pour prédire la classe. OneVs-

RestClassifier peut s'appliquer directement sur un autre algorithme de classification, comme nous le faisons ici avec linearSVC alors que seul, linearSVC était utilisé comme classifieur multi-classe. Cette méthode permet d'avoir une interprétation plus simple car la classification est individuelle pour chaque classe, et ce plus rapidement car il y a moins de classifieurs nécessaires (autant que le nombre de classes).

D'autre part, nous avons essayé d'utiliser des transformers Bert (Bidirectional Encoder Representations from Transformers) et Roberta (Robustly Optimized BERT Pretraining Approach) de la librairie Simple Transformers, qui sont des modèles pré-entraînés. Simple Transformers est une librairie qui simplifie l'utilisation des transformers comme le détaille l'article [5]. Les transformers sont des réseaux de neurones qui utilisent un encodeur et un decodeur. Les Simple Transformers ont besoin en entrée de seulement une colonne de texte et une colonne de labels sous forme d'integer, soit pour nous les descriptions et les numéros des classes. Il faut ensuite renseigner un type de modèle (Bert, Roberta etc), un nom de modèle pré-entraîné et le nombre de classes différentes. En sortie, l'algorithme (avec la fonction *eval_model*) nous renvoie les résultats de la prédiction au format *dict* ou alors il faut utiliser la fonction *predict* comme avec les algorithmes usuels pour obtenir les prédictions. Même si BERT est plus rapide que ROBERTA (4 à 5 fois plus rapide) il s'agit d'algorithmes dont l'exécution est assez longue, et ce même sur les GPU des ordinateurs qui sont à notre disposition en GMM. De plus, la librairie Simple Transformers entraînait des conflits de versions sur la plateforme de Kaggle, ce qui nous a empêché d'utiliser cette alternative également. Ayant des contraintes de temps et techniques, nous avons fait tourner ces algorithmes sur des sous échantillons d'entraînement (avec 100 occurrences des 28 classes soit 2800 données d'entraînement pour Roberta, et avec 783 occurrences des 28 classes soit 21840 données d'entraînement pour Bert).
Bert s'appuie sur de l'apprentissage avec Masked Language Model (MLM) et Next Sentence Prediction (NSP), tandis que Roberta n'utilise pas le NSP, ce sont des algorithmes qui prennent en compte le sens des mots et le contexte des éléments. L'apprentissage avec MLM consiste à masquer 15% de chaque séquence en entrée et il doit ensuite prédire les bons éléments manquants, ce qui en fait un algorithme bidirectionnel. Simultanément, NSP est une fonction de classification binaire qui prédit, avec deux phrases en entrée, si la deuxième suit bien la première dans le contexte. Il est entraîné ainsi sur 50% de paires correctes de phrases et 50% de paires aléatoires.

Roberta est connu pour avoir une meilleure performance que Bert mais ici sur des sous-échantillons leur efficacité n'était pas optimale.

4 Résultats

4.1 Vectorisation

Pour ce qui est de l'étape de vectorisation, nous avons implémenté certaines des méthodes évoquées plus tôt afin de comparer les résultats (voir tableau 1). Pour tester et comparer ces méthodes, nous avons gardé le même algorithme de classification (régression logistique). L'objectif était de déterminer quel algorithme de vectorisation permettait d'obtenir une meilleure métrique.

Vectorisation	Classification	Macro f1
TFIDF	Logistic regression	0.73308
WORD2VEC SKIP GRAM	Logistic regression	0.71823
WORD2VEC CBOW	Logistic regression	0.71175
One hot encoder	Logistic regression	0.70691

Table 1: Vectorisation

Nous avons obtenu un meilleur résultat avec TF-IDF (0.73308) qu'avec One-Hot-Encoder (0.70691), ce qui semble logique car elle est mieux adaptée à l'importance des mots. Aucune des deux versions de Word2Vec ne nous a permis de dépasser TF-IDF (CBOW : 0.71175 et Skip-Gram : 0.71823). Le meilleurs résultats étaient finalement obtenus avec la vectorisation TF-IDF.

4.2 Classification

Nous avons ensuite gardé la méthode de vectorisation TF-IDF pour tous les tests de méthodes de classification. Les modèles du tableau 2 ont dans un premier temps été utilisés sans optimisation des paramètres. Nous n'effectuons au final pas d'optimisation des paramètres pour Random Forest au vu du temps d'exécution de l'algorithme.

Vectorisation	Classification	Macro f1	Temps d'execution
TFIDF	SVC	0.73365	quelques minutes
TFIDF	Logistic regression	0.73308	quelques minutes
TFIDF	Random Forest	0.62091	quelques heures

Table 2: Modèle de classification

Random Forest a donné 0.62091 ce qui est plutôt bas au vu des scores des autres algorithmes, même sans optimisation des paramètres.
Comme le montre le tableau 3, l'optimisation des paramètres de régression logistique donne les mêmes paramètres que les paramètres par défaut. Cela ne change donc pas la métrique. En revanche, nous obtenons une légère augmentation pour le modèle SVC.

Vectorisation	Classification avec optimisation des paramètres	Macro f1
TFIDF	SVC	0.73668
TFIDF	Logistic regression	0.73308

Table 3: Modèle de classification après optimisation des paramètres

Les résultats obtenus avec la régression logistique sont ceux présentés précédemment avec le meilleur score après vectorisation TF-IDF. Enfin, SVC appliqué après TF-IDF et avec optimisation des paramètres donne un score de 0.73668, ce qui fait de cette association de méthodes la meilleure jusqu'à présent.

Une fois ce score atteint nous avons tenté d'ajouter des méthodes à TF-IDF et SVC optimisé dans le but d'augmenter le score obtenu. Cependant, le score obtenu avec l'utilisation du sur-échantillonnage SMOTE

est seulement de 0.73339 ce qui ne nous permet pas de dépasser le score précédent. Cela peut être dû à la taille de la base d'apprentissage, une solution aurait pu être d'utiliser le sous-échantillonage, qui diminue les exemples des classes trop représentées au lieu d'augmenter celles qui sont sous-représentées, mais l'algorithme ADASYN nécessitant beaucoup de temps d'exécution, nous n'avons pas pu évaluer son effet.

Enfin, l'ajout des poids aux classes afin de réduire les déséquilibres a donné un score de 0.71450, à nouveau plus bas que le score obtenu sans.
L'algorithme de classification OneVsRest appliqué sur linearSVC en classification binaire donne 0.73650, un résultat très proche du meilleur score obtenu jusqu'à présent.

Transformer	Taille des données d'entrainement	Macro f1	Temps d'exécution
BERT	21840	0.65400	13h
ROBERTA	2800	0.53973	14h

Table 4: Simple transformers

En ce qui concerne les Simple Transformers, nous voyons très clairement dans le tableau 4 que l'échantillon d'entraînement était trop petit dans le cas de Roberta. En revanche, la taille de l'échantillon de Bert nous permettait d'observer quelques résultats, bien qu'elle soit encore trop restreinte pour un apprentissage optimisé. Comme nous nous en doutions, l'implémentation de ces algorithmes après apprentissage sur une quantité réduite de données ne nous permet pas de concurrencer les autres algorithmes de classification.

Finalement, notre meilleur résultat est obtenu avec une vectorisation TF-IDF et la classification linearSVC avec des paramètres optimisés.

5 L'algorithme choisi : TF-IDF et SVC linear

La vectorisation des données la plus efficace est la vectorisation TF-IDF. Cette méthode permet d'évaluer l'importance d'un terme dans un texte. Plus l'occurrence du mot est importante dans le texte plus son poids est important. Il s'agit donc d'une méthode de vectorisation qui repose sur la fréquence d'apparition des mots dans un texte. Toutefois, si un mot est fréquent dans plusieurs documents sa pertinence est jugée moins importante par le modèle. Le score dépend donc de la fréquence des mots "rares"(dans le corpus) présents dans le document analysé.

L'algorithme de classification nous ayant donné les meilleurs résultats est celui de la classification Linear Support Vector Classification (linear SVC). Cette version de la classification par vecteurs supports est plus flexible que SVC pour le choix des "penalties" et fonctions de perte et devrait donc être plus adaptée aux grands échantillons. Elle s'applique seulement dans le cas de noyaux linéaires. Cette méthode établit l'hyperplan qui divise le mieux les différentes classes à partir des données. Ensuite, une fonction de décision est utilisée pour savoir si chaque échantillon est situé à droite ou à gauche de l'hyperplan ainsi que sa distance à ce dernier. La classification multi-classe est réalisée selon le principe "one-vs-the-rest", en s'appuyant donc sur des classifications binaires plutôt qu'une seule multi-classe. Comme évoqué plus tôt, la méthode one-vs-the-rest permet de rendre l'algorithme SVC adapté à la classification multi-classe sans utiliser le classifieur OneVsRest nous même.
Nous l'avons implémenté grâce à la librairie *sklearn.svm.LinearSVC*.

Nous avons également optimisé les paramètres par validation croisée. Ainsi la norme de pénalisation l2 est la norme optimale. Ensuite pour ce qui est de la fonction perte, la fonction "hinge" est la meilleure. Enfin, le paramètre de régularisation C est optimal pour une valeur de 1.0. Ces paramètres optimisés sont proches des paramètres par défaut de la fonction (penalty=l2, loss=squared_hinge, C=1.0), ce qui explique qu'il n'y ait pas une très grande différence entre le score avec et sans optimisation pour LinearSVC. Ils jouent sur la régularisation du modèle qui permet d'éviter le sur-apprentissage avec les paramètres de régularisation et la norme de pénalisation.

6 Discussion et limites

Nous avons principalement exécuté nos codes en local sur les CPU de nos ordinateurs. Il est arrivé également que nous utilisions google colab afin d'utiliser les GPU en ligne mais la vitesse d'exécution n'était pas toujours meilleure qu'en local. Enfin, nous avons tenté d'utiliser un GPU sur la plateforme en ligne de Kaggle pour l'exécution des Simples Transformers et avons fait face à un conflit de versions qui nous a finalement empêché de l'exploiter. Les temps d'exécution de nos algorithmes étaient variables, de quelques minutes à plusieurs heures pour certains apprentissages, une meilleure utilisation des ressources à notre disposition aurait sûrement diminué un peu ces délais.

Pour améliorer nos résultats, nous aurions aimé tester le simple transformer Roberta avec le jeu de données d'entraînement complet. Cependant, comme nous l'avons expliqué, nous n'avons pas été en mesure de le faire et avons donc obtenu des résultats faibles avec les Simple Transformers.
Nous pensions également que l'ajout de stopwords à retirer, spécialement liés à nos données, aurait augmenté davantage le score de la métrique mais l'effet a été très faible.

Enfin, nous avons utilisé l'algorithme linearSVC sous deux formes différentes : en tant que classifieur multi-classe qui utilise par défaut la méthode "one-vs-the-rest" et en tant que classifieur binaire sur lequel est appliqué ensuite le classifieur OneVsRest. Le meilleur résultat a été obtenu avec la classification multi-classe directe mais les résultats sont au final très proches, ce qui nous a rassuré sur l'utilisation de ces différentes méthodes ayant le même objectif.

Dans un second temps nous aurions voulu travailler sur l'équité homme/femme des prédictions. Mais n'étant pas satisfaites par nos scores de prédiction nous n'avons pas eu le temps de nous concentrer sur cet aspect du défi.

7 Conclusion

Pour conclure, ce projet nous a permis de mettre en application plusieurs méthodes de travail que nous avons découvertes en cours. Nous avons pu également utiliser de nouveaux algorithmes spécifiques au NLP.
Finalement, notre plus grande difficulté a été la gestion du temps, puisque les algorithmes que nous voulions utiliser avaient un temps d'exécution souvent long même en utilisant un GPU. Cette mauvaise gestion du temps ne nous a pas permis de nous concentrer sur un des objectifs principaux du projet : la diminution du biais Homme/Femme dans la classification. Pour cela, une des solutions aurait été d'ajouter des stopwords de pronoms féminins et masculin ("he", "she", "him", "her").
Enfin, ce projet a été l'occasion de participer à un premier concours Kaggle qui apporte une motivation supplémentaire pour trouver la meilleure solution.

Références

[1] Blue DME — Tech, *Comment traiter les problèmes de classification déséquilibrée en machine learning ?* 18 octobre 2018.

[2] Jason Brownlee, *One-vs-rest and one-vs-one for multi-class classification.* 13 avril 2020.

[3] Mayur-btc, *Understanding multilabel text classification and the related process.* 15 novembre 2019.

[4] Raphael K, *Comment gerer les problemes de classification desequilibree.* 25 mars 2020.

[5] Thilina Rajapakse, *Simple transformers — multi-class text classification with bert, roberta, xlnet, xlm, and distilbert.* 13 octobre 2019.