Rapport de stage
Extraction d'entités nommées dans le cadre du projet EXO-POPP

V. BONSI

M1 Science et Ingénierie des Données 2022-2023

Table des matières

1 Introduction

Dans le cadre de ma première année de Master en Science et Ingénierie des Données, j'ai eu l'occasion d'effectuer un stage d'une durée minimale de 8 semaines. J'ai choisi de me tourner vers le laboratoire LITIS au sein de l'Université de Rouen, qui m'a accueilli au sein de son équipe APP (Apprentissage) aux cotés de M. Thierry Paquet et du doctorant Thomas Constum qui m'ont encadré tout au long du stage et avec qui j'ai pu participer à des réunions hebdomadaires. Tout au long de mon stage ma mission a été d'expérimenter et mettre en place différentes méthodes d'extraction d'entités nommées pour permettre leur application dans le cadre du projet EXO-POPP, qui consiste en une base de données de scans d'actes de mariages français manuscrits et tapuscrits riches en entités à relever. Le projet sera détaillé plus en détails dans les parties suivantes.

Le stage s'étale sur une période de 3 mois et demi, du 26 mars au 21 juillet 2023. Ce rapport de stage porte donc sur le travail qui a été effectué durant la période de 8 semaines le précédant, au cours desquelles j'ai d'abord appris les bases de l'extraction d'entités nommées avec la librairie FLAIR [8], faisant partie de l'état de l'art dans ce domaine, puis j'ai pu démontrer l'intérêt de l'utilisation d'un modèle de langage génératif (GPT3.5 [14]) pour cette tache en me basant sur les travaux effectués par d'autres étudiants plus tôt dans l'année.

2 Contexte et présentation

2.1 Le projet EXO-POPP

Le projet EXO-POPP (Extraction Optique des entités nommées manuscrites pour les actes de mariage de la population de Paris)[7], mené par Sandra Brée, porté par Thierry Paquet et financé par l'ANR (Agence Nationale de Recherche), vise à élaborer une vaste base de données a partir de 300.000 actes de mariage de Paris et sa banlieue (départements des Hauts-de-Seine, Seine-Saint-Denis et Val-de-Marne) datant de 1880 à 1940, récupérés auprès des archives départementales concernées.

Le projet a pour ambition de développer de nouveaux algorithmes basés sur les récentes avancées en reconnaissance d'écriture manuscrite et en traitement du langage naturel pour permettre l'extraction des entités nommes dans les actes et ainsi la construction de la base de données, qui permettra des avancées considérables pour la connaissance de la population.

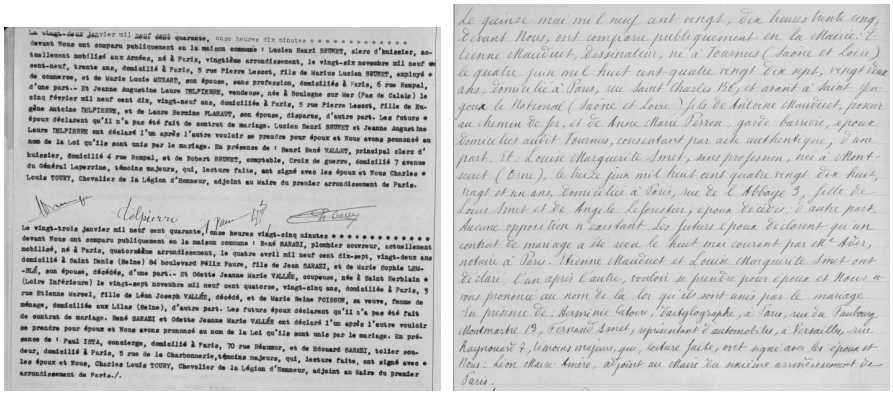
2.2 Le jeu de données disponible et les entités à extraire

2.2.1 Les données

Comme dit précédemment, les données disponibles pour le projet sont conséquentes, et partagent une structure similaire mais ils diffèrent en fonction des

années et du scripteur. Ils peuvent globalement être organises comme suit, par décennie :

- 1880 : Les actes sont très longs. Il y a beaucoup de texte autour des entités, il est fait mention des enfants légitimés et de 4 témoins.
- 1890 : La structure est similaire mais les actes sontbien moins longs, il y a moins de 'blabla'.
- 1900/1910 : La Structure similaires maisil y a des changements dans certaines formules administratives.
- 1920 : On assiste à une grosse simplification des actes, il n'y a plus que 2 témoins (et leurs liens ne sont plus mentionnés) et plus de légitimation des enfants.
- 1930/1940 : La structure est la même qu'en 1920 mais un changement majeur est présent : les actes sont désormais **tapuscrits**.



(a) Actes tapuscrits (b) Acte manuscrit

FIGURE 1 – Illustrations d'actes de mariage @PIVAN

Enfin il est important de mentionner la présence de nuisance pouvant compliquer la tache :

- Les registres d'état-civil peuvent comporter des actes de divorce, ainsi que la transcription de mariage ayant lieu a l'étranger. Ces données ne nous intéressent pas et sont donc à détecter pour les ignorer.
- Certains actes peuvent être scindes sur deux pages différentes. On relève donc 'deux' actes partiels (un contenant le début et un autre la fin)
- Pour les actes manuscrits, il peut y avoir des ratures, des corrections, des rajouts dans la marge ou des oublis.

Dans le cadre du stage, mon travail n'a, pour l'instant du moins, porté uniquement sur des actes tapuscrits de 1930 a 1940, et dont la reconnaissance manuscrite a déjà été effectuée par Thomas Constum via le modèle DAN [6]. Il s'agit donc ici uniquement d'un travail de traitement du langage naturel, et plus particulièrement d'extraction d'entités nommées.

2.2.2 Les entités

Les actes de mariages sont très riches en informations, et par conséquent les entités à relever sont nombreuses. On peut les séparer en deux catégories : Les personnes (incluant l'administratif) et les informations les concernant. On peut classer ces entités de manière hiérarchique comme dans le tableau ci-dessous :

Niveau	Tags					
1	Administratif	Mari	Epouse	Témoin		
1.1	Père	Mère	Ex-Epoux			
2	Naissance	Residence	Décès	Disparu	Age	Veuf
3			Profession	Prenom	Nom	
4	Pays	Département	Ville	Numéro voie	Type voie	Nom voie
	Année	Mois	Jour	Heure	Minute	

FIGURE 2 – Répartition hiérarchique des entités présentes dans les actes de mariage

On va donc pouvoir retrouver, par exemple, les noms et prénoms du mari et son épouse, la profession d'un témoin, le département de résidence de la mère du mari ou encore la date du mariage (administratif).

Il est important de noter que chaque tag de niveau 1 n'entraîne pas nécessairement la présence de toutes les informations qui en découlent. Certaines informations (comme la date de naissance des témoins) ne sont jamais mentionnées (du moins dans les actes tapuscrits) ou peuvent parfois ne simplement pas apparaître (Si un mariage se tient à midi pile, les minutes ne seront pas mentionnées). Il est aussi possible qu'une personne entière soit omise dans le texte (exemple : un des époux ne connaît pas un de ses parents).

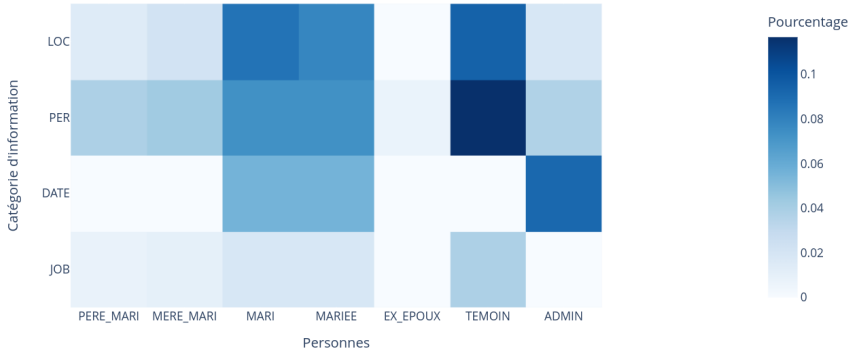


FIGURE 3 – Répartition des labels au sein du dataset

Enfin, il existe le cas des pluriels : Parfois si des personnes partagent des éléments en commun, certains scripteurs peuvent les regrouper (c'est souvent le cas pour les métiers et les adresses). Dans ce cas on a une même entité faisant références a plusieurs tags d'un même niveau, et il est important de la traiter comme tel et de ne pas l'associer à un seul des tags auxquels elle se rattache.

Exemple 1 : "Michel et Bérénices, marchands de bestiaux, domiciliés 6 rue Jean Martin."

Exemple 2 : "Jean, domicilié 8 rue de l'Arche, avec ses père et mère."

Dans le contexte du projet EXO-POPP, labeliser les données est extrêmement long et fastidieux. En réalité les seules données corrigées disponibles sont composées de reconnaissances faites automatiquement a l'aide d'un script utilisant des regex, contenant évidemment de nombreuses erreurs. Cela rend un apprentissage classique particulièrement compliqué. L'entreprise NUMEN [**a**] été mandatée afin d'effectuer la correction d'un jeu d'environ 2200 actes manuellement, mais cela prend du temps et ces données ne nous sont arrivées qu'environ deux semaines avant la rédaction de ce rapport.

3 Travail préliminaire

Afin de m'initier à l'extraction d'entités nommées et au traitement du langage naturel, j'ai commencé mon stage en travaillant sur la librairie flairNLP [3] et sur une variante du dataset IAM, destiné à la reconnaissance d'écriture manuscrite, qui a été labellisée pour de l'extraction d'entités nommées.

3.1 Présentation de l'architecture FLAIR

Les embeddings sont une façon de représenter des caractères, des suites de caractères, de mots ou de phrases. Il en existe différentes méthodes, allant du one-hot encoding qui associe à chaque mot un vecteur de taille n valant 0 pour chaque indice et 1 pour l'indice i, avec n la longueur de la phrase et i l'indice du mot dans le phrase, à d'autres beaucoup plus évolués permettant de stocker des informations sur la sémantique, le sens ou le type de mot, dont le plus connu est GloVe [9]. Les méthodes de l'état de l'art actuel peuvent concaténer jusqu'à 3 méthodes parmi les suivantes :

- Word Embeddings 'classique' : pré-entrainé sur un large corpus pour extraire des informations syntaxiques et sémantiques.
- Character-Level features : non pré-entrainé, entrainé sur la tâche à réaliser pour apprendre des informations et sous-mots spécifiques à celle-ci.
- Contextualized Word embedding : Apprend la sémantique et le contexte d'utilisation des mots pour différencier des homonymes ou des sens dépendants du contexte.

La librairie flairNLP introduit une nouvelle approche baptisée Contextual String Embeddings [2] qui regroupe les avantages de ces 3 méthodes en étant entraîné sans notion de mot et dont les embeddings sont contextualisés par le reste de la phrase.

En 2020, une évolution majeure survient dans la librairie avec l'introduction de FLERT [13] dont le changement vient de l'utilisation des embeddings fournis par n'importe quel Encoder d'un Transformer pré-entraîné existant, qui sera fine-tuné sur la tâche à réaliser en ajoutant du contexte au niveau du document complet et non uniquement des phrases. Cela permet d'étendre les limites du contexte et de cerner le sens induit de certains mots qui n'aurait pas pu être bien saisi au niveau de la phrase elle-même.

Ces deux méthodes fournies par flairNLP représentent, dans leur époque et même aujourd'hui pour la plus récente, des modèles capable de dépasser ou égaler l'état-de-l'art sur de nombreuses tâches de NER.

3.2 Le dataset IAM

Le dataset IAM [10] est un dataset conçu pour évaluer des modèles de reconnaissance d'écriture manuscrite (OCR). Il a été adapté par l'univesité RWTH pour de la reconnaissance d'entité nommées. Il contient 6 tags différents et les phrases contenant des erreurs de retranscription ont été supprimées. Il est réparti tel que :

- 804 PERSON
- 447 LOCATION
- 367 TIME
- 334 CARDINAL
- 263 ORGANIZATION
- 160 NORP

3.3 Résultats obtenus

J'ai pu tester au cours de mes expériences différentes méthodes avec flairNLP, que ce soit concaténer différents embeddings classique, ou essayer différents embeddings de Transformer.

Ci-dessous se trouvent les 3 résultats les plus représentatifs réalisés, à savoir :

- FLAIRembeddings backward + forward : Donc une configuration LSTM bidirectionnelle.
- TransformersEmbeddings en utilisant BERT [5] l'un des premiers et plus connu Transformers.
- TransformersEmbeddings en utilisant XLM-RoBERTa-Large, une très grosse évolution multi-lingue de BERT, plus gros et plus récent. Cette association de Flair et RoBERTa représente actuellement l'état-de-l'art sur le site paperswithcode sur de nombreux benchmark de NER.

Dans un contexte de classification NER multi-classe comme ici, les métriques utilisées sont :

— Precision : Le rapport entre le nombre d'éléments d'une classe correctement reconnus, et le total d'éléments reconnus comme appartenant à cette classe.

$$\text{Precision(class = a)} = \frac{TP(class = a)}{TP(class = a) + FP(class = a)}$$

— Recall : Le rapport entre le nombre d'éléments d'une classe correctement reconnus, et le total d'éléments appartenant réellement à cette classe.

$$\text{Recall(class = a)} = \frac{TP(class = a)}{TP(class = a) + FN(class = a)}$$

— Score F1 : un rapport entre le produit et la somme de la précision et du recall.

$$\text{F1(class = a)} = \frac{2 * Precision(class = a) * Recall(class = a)}{Precision(class = a) + Recall(class = a)}$$

	precision	recall	f1-score	support
PERSON	0.8405	0.8705	0.8553	448
LOCATION	0.8000	0.7975	0.7988	326
CARDINAL	0.7965	0.7764	0.7863	237
TIME	0.6287	0.6649	0.6463	191
ORGANIZATION	0.5114	0.5422	0.5263	166
NORP	0.7561	0.8158	0.7848	114
	0.0000	0.0000	0.0000	7
micro avg	0.7521	0.7683	0.7601	1489
macro avg	0.6190	0.6382	0.6283	1489
weighted avg	0.7504	0.7683	0.7590	1489

(a) Flair F1avg=0.76

	precision	recall	f1-score	support
PERSON	0.9017	0.9219	0.9117	448
LOCATION	0.8656	0.8497	0.8576	326
CARDINAL	0.8417	0.8523	0.8470	237
TIME	0.6835	0.7801	0.7286	191
ORGANIZATION	0.5510	0.6506	0.5967	166
NORP	0.7891	0.8860	0.8347	114
	0.0000	0.0000	0.0000	7
micro avg	0.8013	0.8395	0.8199	1489
macro avg	0.6618	0.7058	0.6823	1489
weighted avg	0.8043	0.8395	0.8208	1489

(b) BERT F1avg=0.82

	precision	recall	f1-score	support
PERSON	0.9574	0.9531	0.9553	448
LOCATION	0.8889	0.9325	0.9102	326
CARDINAL	0.8846	0.8734	0.8790	237
TIME	0.7783	0.8639	0.8189	191
ORGANIZATION	0.7602	0.7831	0.7715	166
NORP	0.8760	0.9298	0.9021	114
	0.6667	0.2857	0.4000	7
micro avg	0.8770	0.9006	0.8887	1489
macro avg	0.8303	0.8031	0.8053	1489
weighted avg	0.8783	0.9006	0.8886	1489

(c) RoBERTa F1avg=0.89

FIGURE 4 – Résultats obtenus par classe et au total en precision,recall, et F1

Les 3 architectures proposent de bonnes performances, mais une chose saute directement aux yeux : les performances des modèles basés sur la dernière évolution de flairNLP, FLERT, proposent des performances bien supérieures à la précédente génération. RoBERTa améliore encore grandement les performances de BERT notamment sur les entités les moins représentées (NORP et ORGANIZATION).

3.4 Limites de l'approche sur le projet EXO-POPP

Comme on a pu le voir, l'approche ci-dessus demande un entraînement. Celui-ci peut durer plusieurs heures (sur le cas de RoBERTa sur la machine à ma disposition) même en ne réalisant que du fine-tuning. Les modèles utilisés étant de surcroît de très grande taille, ils ont besoin d'une quantité importante de données pour apprendre. Ces données doivent aussi être labellisées, dans le cadre d'un entraînement qui est supervisé. Enfin, malgré la taille relativement conséquente du dataset IAM, nos tests ci-dessus on montré des faiblesses dans la détection d'entités faiblement représentées, indiquant un manque de données d'entraînement.

Le projet EXO-POPP, comme dit plus haut, comporte une base de données d'une taille considérable. Cependant ces données ne sont pas, peu ou mal étiquetées. De plus, la quantité de tags qu'il faut extraire de chaque texte est extrêmement conséquente (environ 40 tags même en effectuant des combinaisons pour réduire leur nombre) et certains sont très rarement représentés. De plus, en admettant un étiquetage des données parfaitement réalisé à la main, coûteux, l'apprentissage efficace d'un modèle sur celle-ci demanderait beaucoup de temps et de ressources.

Comment dès lors peut-on tenter de remédier à ces problèmes ?

4 Extraction d'entités nommées avec GPT3.5

4.1 L'architecture GPT

Les architectures a base de Transformers, qui ont suivi le papier "Attention is all you need" [4] ont permis de révolutionner de nombreuses taches dans les domaines du traitement du langage naturel en introduisant une mécanique appelée self-attention permettant de grandement améliorer la mise en contexte des éléments d'un texte.

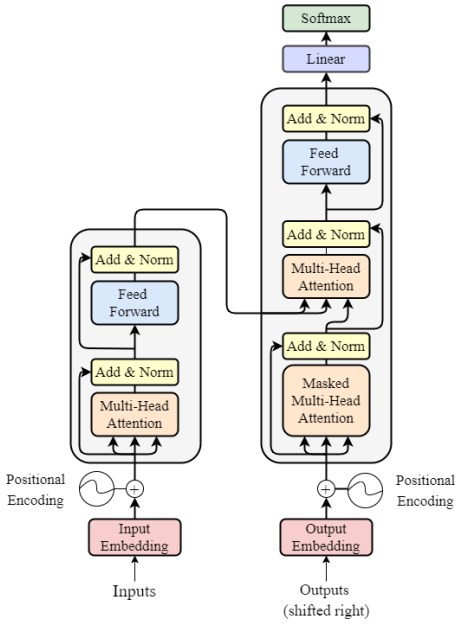


FIGURE 5 – Architecture d'un transformer @Attention is all you need

Les Transformers fonctionnent en utilisant un procédé de Tokenization en sous-mots qui sont ensuite passés dans un système Encoder-Decoder : L'input de l'utilisateur est d'abord tokenisé , puis la partie Encoder se charge de créer une représentation vectorielle des informations présentent appelée embedding. Enfin une réponse est générée par la partie Decoder en se basant sur ce qui a été généré jusqu'à présent et sur les embeddings fournis par l'Encoder.

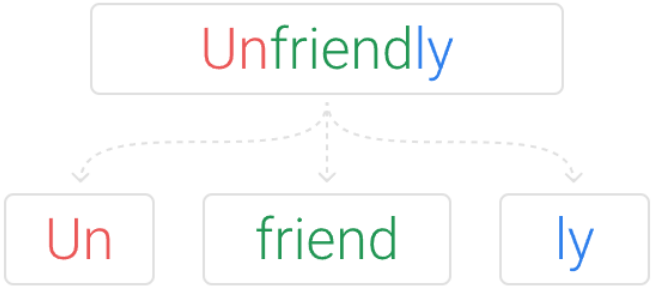


FIGURE 6 – Exemple de subword tokenization

Les modèles a base de Transformers permettent aussi d'obtenir d'excellentes performances sur des datasets ou labeliser les données manuellement serait extrêmement fastidieux voire impossible. Grâce au concept de Masked Multi-Head Attention, ces modeles vont pouvoir effectuer du self-supervised learning sur le jeu de donnees fournis en masquant automatiquement des mots ou des fins de phrases ou de texte.

Parmi ces modèles, certains peuvent être appelés 'Fondation'. Il s'agit de modèles entraînés sur d'énorme quantités de données dans des domaines hétéroclites, qui peuvent être réutilisés ou intégrés tels quels ou avec du fine-tuning dans de nouvelles applications. Ces modèles ont la capacité de pouvoir être adaptés a de nouvelles taches avec très peu de données ce qui peut être intéressant dans de nombreuses applications.

L'architecture GPT fait partie de ces modèles fondation. Elle est basée sur une architecture Decoder uniquement, c'est-a-dire que l'input de l'utilisateur est en quelque sorte 'constant' : l'embedding d'un sous-mot après la tokenization est réalisé en associant chacun d'eux a un vecteur grâce a un dictionnaire que le modèle a créé lors de son pré-entraînement.

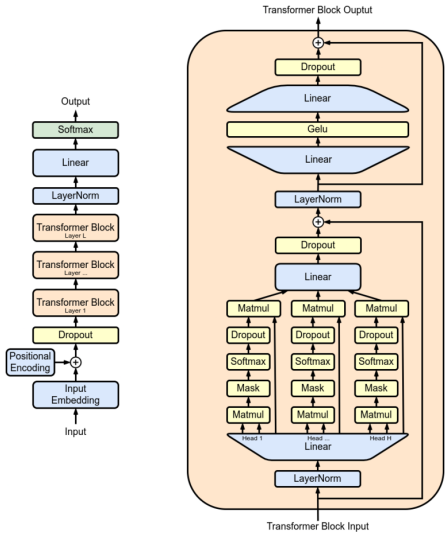


FIGURE 7 – Architecture GPT @Wikipedia

Ce modèle fonctionne ensuite de manière dite autoregressive : Chaque mot prédit par le modèle l'est en utilisant tout ce qui a été fourni ou généré jusqu'à présent. Le modèle prédit les mots un a un, en concaténant a chaque fois de manière récurrente son précèdent output a l'input fourni au départ avant de prédire le mot suivant.

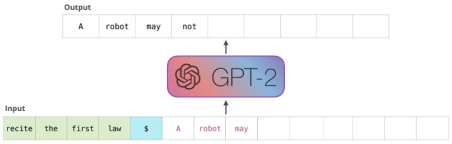


FIGURE 8 – Fonctionnement d'un modèle autoregressif @Wikipedia

L'architecture GPT a ensuite reçue plusieurs évolutions majeures, notamment avec l'apparition du Reinforcement Learning with Human Feedback [15] et instructGPT qui a démontré de grandes capacités a s'adapter aux instructions fournis, ou chatGPT, sa dernière évolution accessible au public, capable de répondre de manière humaine a des questions posées.

GPT est particulièrement adapté pour des taches de traduction ou de génération de texte, mais nous allons voir que grâce aux grandes performances du modèle, les capacités de génération peuvent être dérivées en d'autres utilisations.

4.2 Différences face aux méthodes conventionnelles de NER

Les méthodes conventionnelles de NER fonctionnent toutes plus ou moins de la même manière que la librairie FLAIR utilisée précédemment : on fournit des exemples labellisés, souvent au format BIO. Le modèle renvoie ensuite des associations entre des mots et des labels (exemple : personne, lieu, date...), certains labels pouvant s'étendre sur plusieurs mots, que l'on peut ensuite remettre facilement au format BIO.

Il est ensuite facile de comparer les résultats de références et ceux fournis par la modèle en comparant la suite de labels de l'un et de l'autre, de la façon suivante, par exemple en utilisant le format MUC (Message Understanding Conference) :

- Correct (COR) : Les labels sont similaires
- Incorrect (INC) : Les labels sont différents
- Partial (PAR) : Une partie d'un label (en plusieurs mots) est similaire
- Missing (MIS) : Un label n'a pas été détecté
- Spurius (SPU) : Le modèle a labellisés un mot qui n'aurait pas du l'être

Ensuite on peut évaluer les performances du modèles en pondérant les différentes réponses plus ou moins sévèrement :

Scenario	Golden Standard		System Prediction		Evaluation Schema			
	Entity Type	Surface String	Entity Type	Surface String	Type	Partial	Exact	Strict
III	brand	TIKOSYN			MIS	MIS	MIS	MIS
II			brand	healthy	SPU	SPU	SPU	SPU
V	drug	warfarin	drug	of warfarin	COR	PAR	INC	INC
IV	drug	propranolol	brand	propranolol	INC	COR	COR	INC
I	drug	phenytoin	drug	phenytoin	COR	COR	COR	COR
I	Drug	theophylline	drug	theophylline	COR	COR	COR	COR
VI	group	contraceptives	drug	oral contraceptives	INC	PAR	INC	INC

FIGURE 9 – Schéma d'évaluation d'un label en fonction de la rigueur choisie

Cependant le fonctionnement des modèles conventionnels présente des limitations :

- Ils ont besoin d'un apprentissage long sur beaucoup de données, qui doivent avoir été labellisées manuellement et rigoureusement. Cela peut être coûteux autant en temps que financièrement.
- Il est difficile de disposer d'un grand nombre de tags différents, et des tags spécifiques doivent être reconstruit a partir de tags plus généraux afin de limiter leur nombre.
- Des tags apparaissant rarement seront très difficilement appris.

Dans le contexte du projet EXO-POPP, labeliser les données est extrêmement long et fastidieux. En réalité les seules données corrigées disponibles sont composées de reconnaissances faites automatiquement a l'aide d'un script utilisant des regex, contenant évidemment de nombreuses erreurs. Cela rend un apprentissage classique particulièrement compliqué. L'entreprise NUMEN [numen]a été mandatée afin d'effectuer la correction d'un jeu d'environ 2200 actes manuellement, mais cela prend du temps et ces données ne nous sont arrivées qu'environ deux semaines avant la rédaction de ce rapport.

A l'inverse des méthodes conventionnelles de NER, GPT3.5 n'est pas conçu pour de l'extraction d'entités nommées. Il possède cependant une formidable capacité d'adaptation a différentes taches [15] et surtout, comme les autres modèles de langage pré-entraînés , il est théoriquement capable de devenir tres performant sur une tache avec seulement un ou peu d'exemple, ce que l'on appelle **One-shot learning** et **Few-Shot learning** [11]

En se basant sur les capacités d'adaptation rapide de GPT3.5 et du travail très encourageant mené par Diego Rio et Owen Ruffin, deux étudiants de l'Université de Rouen, quelques mois plus tôt, nous allons nous poser la question suivante : est-il possible d'utiliser GPT pour générer du texte de manière a mimer le fonctionnement d'un modèle de NER ?

4.3 Méthodologie et métriques utilisées

Pour réaliser mes expériences, je disposais d'une base de données composée de la transcription d'actes de mariages tapuscrits de 1930 a 1940 ainsi que leurs labels au format JSON générés automatiquement avec un script utilisant des regex, comme dit plus haut. Ces labels contiennent chaque labels indépendamment, c'est-a-dire que l'on a pas nn mot associé aux labels 'mari', 'mere', 'residence' et 'rue' par exemple, mais une clé de dictionnaire 'rue-de-residence-de-la-mere-du-mari' dont la valeur sera le mot (ou la suite de mots) du texte qui y correspond.

Le travail réalisé par Owen et Diego consistait en le cheminement suivant :

- Démarrer un "chat" avec l'API GPT3.5.
- Donner une phrase d'initialisation. En l'occurence *"Nous allons te fournir un certificat de mariage, un document ayant toujours la même mise en forme.Tu vas devoir procéder à l'extraction de certaines données sur plusieurs certificats ensuite. Voici le premier certificat, je précise qu'il est extrait d'un document au format Json et que tu auras toutes les réponses fournies à la fin, cela te permettra de mieux reconnaître ce qu'il te faut obtenir dans les contrats suivants."*.
- Donner la retranscription d'un acte de mariage 'témoin' du dataset, servant d'exemple. Il s'agit toujours du même acte.

— Donner la liste des labels qui y est associée, au format json converti en chaîne de caractères.
— Ajouter *"Maintenant, voici un autre certificat de mariage : je veux que tu m'extrais des données sous la même forme que les réponses que je t'ai fourni."*.
— Donner la transcription d'un acte de mariage du dataset différent de l'exemple.
— Récupérer la réponse renvoyée par l'API. Celle-ci devrait normalement prendre la forme d'une chaîne de caractère contenant un json, sur le même modèle que l'exemple, c'est a-dire avec chaque clé du dictionnaire contenant un label complet, suivi de sa valeur correspondant a un mot ou une suite de mot du texte.

Cette méthode propose des résultats encourageants, mais elle présente de nombreuses limites qui seront détaillées plus bas, ainsi qu'un fonctionnement très aléatoire en fonction de l'acte a tester.

Pour évaluer les performances des expériences plus bas, il faudra comparer le JSON que l'on obtient en réponse de l'API a celui que l'on possède comme référence. Cette méthode ne permet pas d'utiliser les métriques classiques utilisées en NER. En effet, la plupart des cas n'ont pas de sens. Surtout, on associe pas un label a un mot (*[PARIS] ->(LOC, MARI)*), mais un mot a un label (une clé de dictionnaire). Ainsi on ne peut pas par exemple comparer la présence et la valeur d'un label sur le mot 'Paris' dans le texte. On ne peut que regarder le dictionnaire et vérifier si la clé 'Ville-de-residence-du-mari' contient le mot recherché. Il est aussi nécessaire de considérer les cas ou des réponses sont similaires mais différentes a quelques caractères près (surtout dans le cas d'une suite de mot) ainsi que le cas ou une clé dans le dictionnaire de différence est vide. Enfin dans le cas d'une NER classique, associer le tag 'Departement' a Paris au lieu de 'Ville' constituerait une erreur. Ici, la clé 'Ville' sera vide dans la réponse (elle contient Paris dans la référence) et celle 'Departement' contiendra Paris au lieu d'être vide.

Nous allons donc utiliser 3 métriques dans la suite de ce rapport :
— La distance de Levenshtein, ou distance d'édition. Elle correspond au nombre minimal de caractère qu'il est nécessaire de supprimer, insérer ou modifier pour passer d'une chaîne a une autre.

$$lev(a,b) = \begin{cases} max(|a|,|b|) & \text{si min(|a|,|b|) = 0} \\ lev(a-1,b-1) & \text{si a[0] = b[0]} \\ 1 + min(lev(a-1,b), lev(a,b-1), lev(a-1,b-1)) & \text{sinon} \end{cases}$$

— Une 'erreur' de label :

$$err(a,b) = \begin{cases} 1 & \text{si len(a) = 0 et len(b) > 0 ou len(a) > 0 et len(b) = 0} \\ 1 & \text{si lev(a,b) >= len(b // 2)} \\ 0 & \text{sinon} \end{cases}$$

— Une metrique que l'on va appeler F1 qui se rapproche du score F1 : **INCLURE L'EQUATION ET LE CALCUL DU SCORE F1 ICI**

4.3.1 Création d'un dataset de test

Comme mentionné plus haut, les ressources disponibles sont constituées de données n'ayant subies aucune vérification humaine et dont le taux d'erreur est haut. Les travaux de Diego et Owen sont basés sur ce dataset et cela nuit fortement aux performances des modèles. J'ai donc créé un nouveau dataset composé de 22 actes que j'ai pris soin de corriger a la main. Toutes les mesures et expériences qui suivent ont été réalisées avec celui-ci. Le dataset est composé comme suit :

— 13 actes de 1940.
— 4 actes de 1935.
— 5 actes de 1930.

Les actes ont été choisis de manière a couvrir a couvrir le plus de cas possibles afin d'évaluer au mieux les performances des modèles. Ils ont donc été écrits par différents scripteurs, contiennent des personnes décédées, disparues ou non mentionnées, des résidants étrangers ainsi que plusieurs cas de pluriels.

4.4 Base de travail : One-shot Learning

4.4.1 Approche

Le one-shot learning constitue les prémices de mon travail. Comme son nom l'indique, son but est de réutiliser un modèle pré-entrainé et de ne lui donner qu'un unique exemple pour s'adapter à une nouvelle tâche. C'est l'approche menée par Owen et Diégo dans leur travail.

J'y ai ajouté plusieurs axes d'améliorations :

— Premièrement, l'exemple fournit a été corrigé par mes soins. Les informations qu'ils comportent sont donc justes et vérifiées. Ce n'était pas le cas de celui utilisé par Owen et Diégo, qui comportait des éléments manquant ou bien faux.
— Deuxièmement, les clés du dictionnaire ont été largement raccourcies. Elles étaient précédemment en langage naturel contenant des caractères spéciaux ainsi que des déterminants. Certains éléments identiques étaient également référencés par des synonymes, et nom le même mot. Exemple :
  — "Numéro-de-rue-de-résidence-de-la-mère-de-la-mariée" -> "Numero-rue-residence-mere-mariee"
  — "Prénom-de-l'adjoint-au-maire" -> "Prenom-maire"
  — "Profession-... / Métier-..." -> "Profession-..."

  Cela a deux avantages : Non seulement l'aspect autoregressif de GPT a plus de chance de générer des clés inexistantes ou incorrects dans sa

réponse si celles-ci sont plus long et complexe, mais le mécanisme d'attention fonctionne bien mieux avec des mots-clés.

— Troisièmement, la conversation n'est plus initiée comme un message de l'utilisateur, mais comme un chat déjà existant, en utilisant les fonctionnalités de l'API d'openAI [12]. Cette api permet de créer un 'pré-chat' en utilisant 3 types de rôles :
  — System : Permet d'envoyer un message de conditionnement du comportement du modèle. Dans notre cas il s'agit de : *Tu extrais des entités nommées. Tu analyses les textes que je te donne. Tu réponds sous forme de json.*
  — User : L'utilisateur 'humain' du chat. On s'en sert désormais **uniquement** pour envoyer les actes de mariage au format texte, que ce soit les exemples ou les requêtes.
  — Assistant : il s'agit des réponses de GPT aux requêtes de 'User'. On peut donc mimer une réponses synthétiques précédemment envoyée par l'API. On l'utilise pour envoyer les json correspondant aux actes envoyés en exemple par 'User'.

  La requête prend donc la forme de l'image ci-dessous, avec la valeur retournée correspondant aux labels de *acte-requete* :

```
openai.ChatCompletion.create(
  model="gpt-3.5-turbo",
  messages=[
        {"role": "system", "content": "Tu extrais des entités..."},
        {"role": "user", "content": acte-exemple},
        {"role": "assistant", "content": label-de-lexemple},
        {"role": "user", "content": acte-requete}
    ]
)
```

FIGURE 10 – Structure d'une requête 'one-shot' à l'API GPT3.5

4.4.2 Résultats

Dans les résultats ci-dessous, F1 strict correspond à un score F1 calculé avec avec un match PARTIEL considéré comme faux. La distance de Levenshtein entre la référence et la réponse de GPT3.5 doit être de 0 pour qu'un label soit compté comme un True Positive. F1 relâché correspond à un score F1 calculé en autorisant un match partiel et en le comptant comme un true positive selon les conditions suivantes :

— La valeur pour une même clé dans le dictionnaire de référence.
— OK si lev(a,b) $<$ len(b // 2), ce qui correspond donc à une disanc d'édition autorisée de 50%

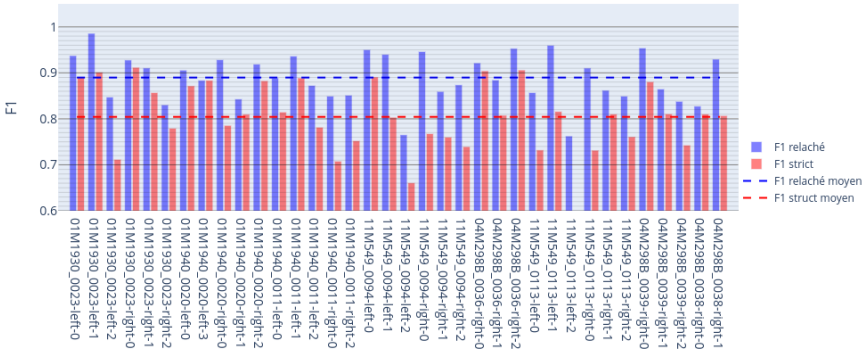


FIGURE 11 – Score F1 sctrict et relaché moyen en one-shot sur les différents actes

On obtient un score F1 strict de 0.805 et un score F1 relaché de 0.890 en moyenne. Certains actes ont des scores bien plus bas que les autres : il s'agit d'actes qui étaient incomplets et dont le texte est tronqué. Le modèle ayant reçu un unique exemple d'acte complet il n'est pas parfaitement capable de gérer un cas inconnu.

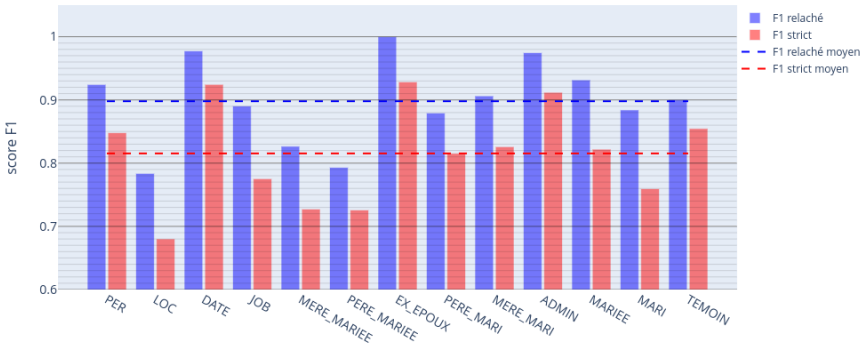


FIGURE 12 – Score F1 sctrict et relaché moyen par catégories en one-shot

Si l'on regarde plus en détail les résultats obtenus, on remarque une disparité dans les scores, notamment au niveau des pères et mères de la mariée mais aussi des localisation. C'est sans doute dû aux différentes formulation possible qui mettent à mal le seul exemple connu par le modèle.

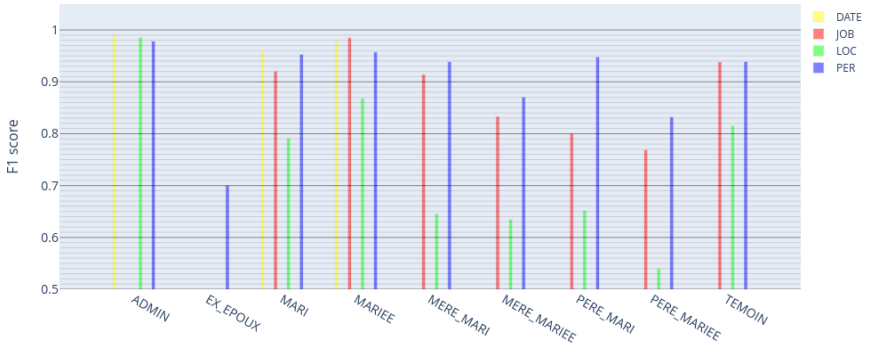


FIGURE 13 – Score F1 sctrict et relaché moyen par catégories par entitée

Enfin la photo ci-dessus semble confirmer l'hypothèse précédentes : les pères et mères vivent souvent ensemble, leurs adresses et donc métiers sont donc souvent mentionnés une seule fois, ce qui semble poser problème au modèles. Les métiers des pères posent plus de problème que ceux des mères supposément car celles-ci sont très fréquement dans le cas *'sans profession'*.

4.4.3 Limites

Les expérimentations au-dessus mettent bien en lumière un problème difficilement solvable avec la méthode actuelle. En effet malgré la structure similaires dans les données, celles-ci sont très différentes. Par exemple :

- Lorsqu'une personne est décédée, cela peut-être mentionné directement (*"...fils de Jean Valjean, décédé"*) ou indirectement (*"...fils de Jean Valjean et Emma Bovary, sa veuve"*).
- Les couples partageant le même travail peuvent le voir être cité séparément (*"Jean Valjean, libraire, et Emma Bovary, libraire"*) ou bien au pluriel (*"Jean Valjean et Emma Bovary, libraires"*).
- Il en est de même pour les adresses, avec en plus le cas où l'adresse des (ou du) parent(s) peut être directement citée avec celle de leur enfant (*"Domicilié 8 rue de la Paix, avec ses pères et mères"*).
- Les actes peuvent être complets, tronqués au début ou tronqués à la fin.

Dès lors, comment donner au modèles toutes ces informations différentes ? La réponse peut paraître simple : donner plus d'exemples pour couvrir de plus nombreux cas d'usages.

Mais elle ne l'est pas : GPT3.5 impose une limite de contexte de 4000 tokens, un token représentant un sous-mot ou un signe de ponctuation. Pire, le fonctionnement auto-régressif lui impose que cette limite est composée de la somme de la réponse mais aussi de l'input qui lui est fourni. Ainsi, même avec un seul exemple, il serait possible de dépasser cette limite dans le cas ou celui-ci est très

long, ce qui entraînerait des éléments manquant voire une erreur du programme (dictionnaire json non fermé ou mal formatté)

La seule solution pour fournir plus d'exemple serait de réussir à réduire la taille de ceux-ci.

4.5 Few-shot learning en paragraphes

Le concept de few-shot learning est assez simple : plutôt que de donner un unique exemple à un modèle pré-entraîné, on lui en donne un petit nombre. Ces exemples doivent cerner un maximum de cas possible et ne pas être trop similaires au risque de vite faire sur-apprendre le modèle. De plus il est important de noter que ces exemples peuvent être synthétiques : rien n'empêche d'ajouter à la main des informations ou des cas au sein d'un acte existant afin de le rendre plus riche.

Sur le même principe que le jeu de données mis en place pour les tests, j'ai donc réalisé un jeu de 5 exemples corrigés et modifiés pour différents scripteurs, différentes formulations de scripteurs, des résidants étrangers et des cas de personnes décédées, disparues ou non mentionnées.

Ces exemples sont utilisés à chaque requête, et toujours dans le même ordre, en étendant le notion de **chat préliminaire** vue plus haut, de la manière suivante :

```
6  messages = [{"role": "system", "content" : consigne_systeme}]
7  for i in range(nombre_exemple):
8      messages.append({"role": "user", "content" : acte_exemple[i]})
9      messages.append({"role": "assistant", "content" : json.dumps(labels_exemple[i], indent=4)})
10 messages.append({"role": "user", "content" : acte_requete})
11
12 openai.ChatCompletion.create(
13     model="gpt-3.5-turbo",
14     temperature=0.4,
15     messages=messages
16 )
```

FIGURE 14 – Boucle de création du pré-chat de conditionnement de l'api GPT.

4.5.1 Découpage en paragraphes

Le fonctionnement plus haut ne résout pour l'instant pas le problème du nombre de tokens maximum. Pour cela nous allons introduire un découpage en paragraphe.

Même si tous les actes sont différents, ils partagent tous une structure commune (du moins sur les actes tapuscrits), à savoir :

— Un premier paragraphe traitant de la date du mariage, commençant dès le début et s'arrêtant toujours à *"ont comparu devant Nous"* ou *"ont comparu devant Nous en la maison commune"*.

— Un second paragraphe traitant du marié contenant toutes les informations sur lui (nom, prénom, date de naissance, lieu de naissance, lieu de résidence, profession, nom/prénom d'une potentielle ex-épouse) ainsi que ses parents (noms et prénoms, lieux de résidence, professions). Il s'arrête toujours après *"d'une part"*.
— Un troisième paragraphe identique mais concernant la mariée et sa famille. Il commence le plus souvent par *"ET"* et s'arrête après *"d'autre part"*.
— Un quatrième paragraphe contenant à nouveaux les noms et prénoms des mariés ainsi que la présence ou non d'un contrat de mariage. Il s'arrête après *"et Nous avons déclaré qu'ils sont unis par le mariage"*.
— Un cinquième et dernier paragraphe contenant les information sur les témoins (Noms, prénoms, adresses, profession), la ville où se tient le mariage ainsi que les coordonnées du maire ou de l'adjoint au maire. Il commence par *"en présence de"* et va jusqu'à la fin.

On peut donc scinder les exemples en paragraphes, de même que les labels qui y sont associés, répartis comme tels :
— §1 : 5 labels
— §2 : 37 labels
— §3 : 37 labels
— §4 : 5 labels optionnels
— §5 : 21 labels

Ce qui nous donne un total de 101 labels + 4 optionnels servant à remplacer les noms et prénoms des mariés dans le cas où ils sont manquants dans les paragraphes 2 et 3.

Ensuite il suffit d'appliquer strictement la même méthodologie que celle donnée plus haut pour le few-shot learning, excepté que chaque paragraphe est traité indépendamment, puis que l'on concatène les résultats (voir photo ci-dessous).

Ce découpage se montre efficace. Mais un problème subsiste : le découpage en paragraphe n'est pas parfait. De légères différences de formulation, de ponctuations ou des erreurs dans la reconnaissance de texte peuvent induire en erreur un découpage 'force brute' à l'aide de regex. De plus on rappelle que les actes sont parfois tronqués.

Pour palier à cela on va introduire une nouvelle instance de GPT. Celle-ci se verra donner 3 exemples, ainsi qu'un json contenant le découpage à effectuer (*Noter que ci-dessous "texte" est à remplacer par le texte de l'acte associé*)
— Un acte relativement court mais complet, ainsi qu'un json sous la forme suivante : *p1 : texte, p2 : texte, p3 : texte, p4 : texte, p5 : texte*
— Un acte dont le début est manquant, ainsi qu'un json sous la forme suivante : *p1 : , p2 : , p3 : ...texte, p4 : texte, p5 : texte*

FIGURE 15 – Comparaison d'une requête one-shot vs few-shot en paragraphes

— Un acte dont la fin est manquante, ainsi qu'un json sous la forme suivante : *p1 : texte, p2 : texte, p3 : texte..., p4 : , p5 :*

On fournit également le message *"System"* suivant : *"You split the given texts into paragraphs. You are aware the text can be incomplete or truncated. The parapgrahs are : p1 = Date et heure maison commune, p2 = Le mari, p3 = La mariée, p4 = acte de mariage + uni, p5 = Témoin et maire"*

Chacun des paragraphes récupéré est ensuite passé s'il est vide, ou envoyé dans la seconde instance de GPT avec les exemples adéquats, selon la structure suivante :

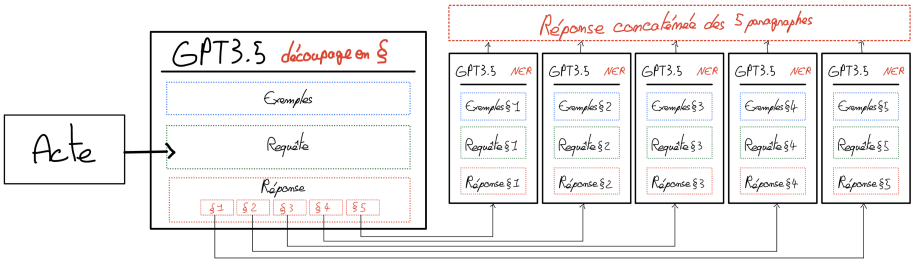


FIGURE 16 – Cheminement d'un acte à la récupération de ses labels

4.5.2 résultats

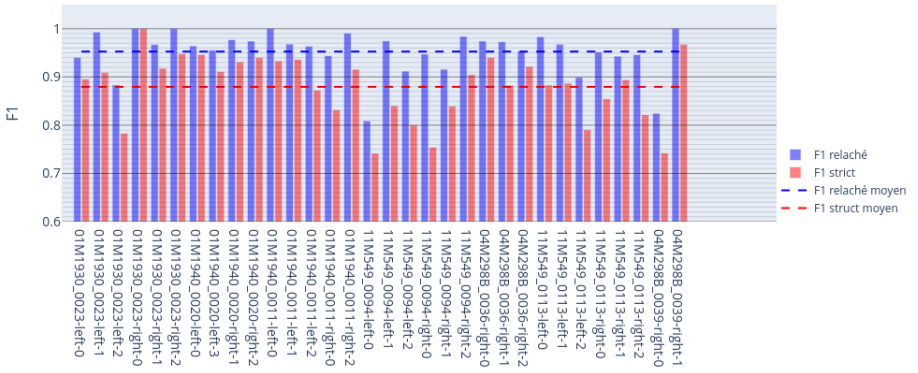


FIGURE 17 – Score F1 moyen strict et relâché few-shot

On remarque directement une belle augmentation du score, passant à un F1 relâché de 0.89 à 0.95. Tous les scores sont plus haut, avec plusieurs actes atteignant 100% de réussite.

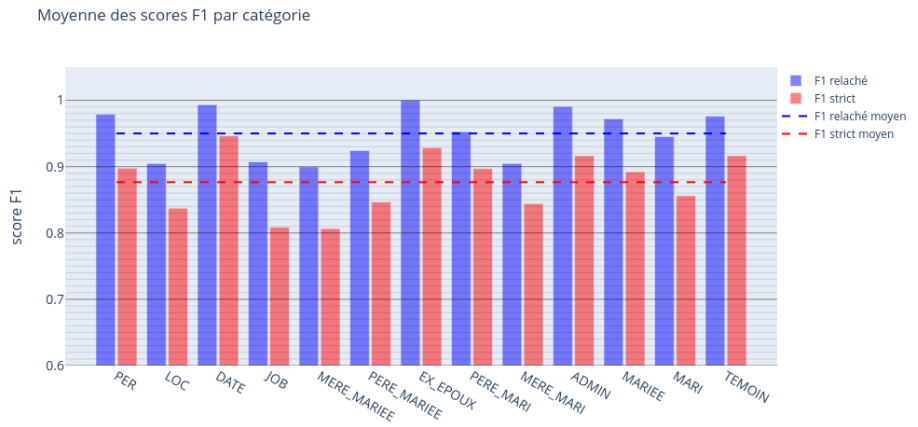


FIGURE 18 – Score F1 moyen par entité strict et relâché few-shot

Le surplus d'exemple fourni permet au modèle de mieux cerner les entités qui posaient problème précédemment, notamment au niveaux des adresses dont le score a beaucoup augmenté. Toutes les métriques proposent de meilleurs résultats.

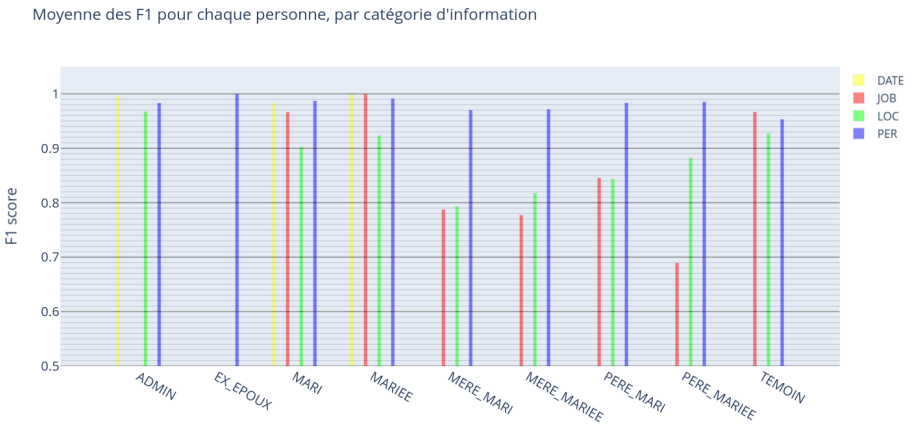


FIGURE 19 – Score F1 moyen par entité par information strict et relâché few-shot

En regardant plus en détails, certains phénomènes sont à noter. Il y a une baisse de performances au niveau de la reconnaissance des professions des pères et mères.

4.5.3 Problèmes rencontrés

En regardant le compte rendu des résultats, on constate en effet une possible forme de surapprentissage du modèle. Certains éléments sont relevés à tort car situés à la même position que dans les exemples :

— Certaines entités possèdent le métier *"disparu"* ou *"décédé"*.
— Pour les femmes veuves, *"sa veuve"* est souvent présent avant leur métier.

On remarque aussi d'autres incohérence qui semblent encore tenir d'un surapprentissage : le modèle différencie mal les département et les pays. En effet, dans les actes ils sont mentionnés de la même manière : *"*pays/département**"*, *"(pays/département)"*, ou encore *"-pays/département-"*. Ces erreurs n'étaient pas présentes en one-shot. Le modèle semble avoir mis de côté ses propres connaissances pour coller au mieux à la demande de l'utilisateur.

Enfin, si un acte tombe à une heure pile, les minutes ne sont pas mentionnées et ne doivent pas être relevées. Dans les résultats de tests le champ *"minute-mariage"* contient parfois "None", "non mentionné" ou encore "aucune", ce qui est incorrect.

4.5.4 Améliorations

Les erreurs citées au-dessus sont facilement remédiables. En effet, il s'agit d'erreurs que l'on pourrait appeler 'fixes'. On peut y remédier en analysant

après-coup le dictionnaire renvoyé par l'api.

- Si une profession est "décédé/é/s" ou "disparu/e/s" on la remplace par une chaîne vide.
- On fait de même pour les minutes du mariage si elles ne sont pas mentionnées.
- Si une profession contient "sa veuve, " alors on supprime cette partie
- Pour les pays et départements, on connaît la liste des pays. On peut donc y comparer les valeurs des champs *'departement-x'* et *'pays-x'*, et effectuer les permutations adéquates le cas échéant.

4.5.5 nouveaux résultats

Avec ces corrections faites, et en relançant un test, on obtient les résultats suivants :

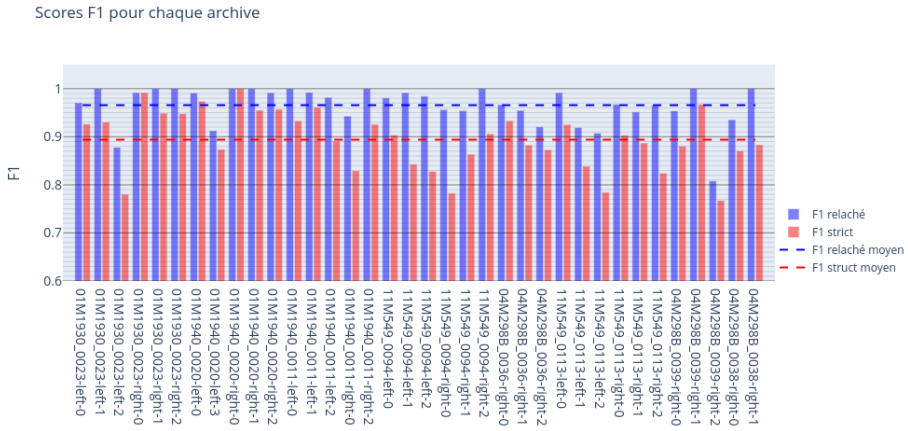


FIGURE 20 – Score F1 moyen strict et relâché few-shot corrigé

On obtient un score F1 moyen relâché de 0.965, tandis que le score sctrict, qui ne tolère que des réponses exactes au caractère près à celles du dictionnaire de référence, s'établit à 0.895, c'est-à-dire le même score que le modèle en fonctionnement one-shot en autorisant une distance de Levenshtein maximale de 50% de la longueur du label.

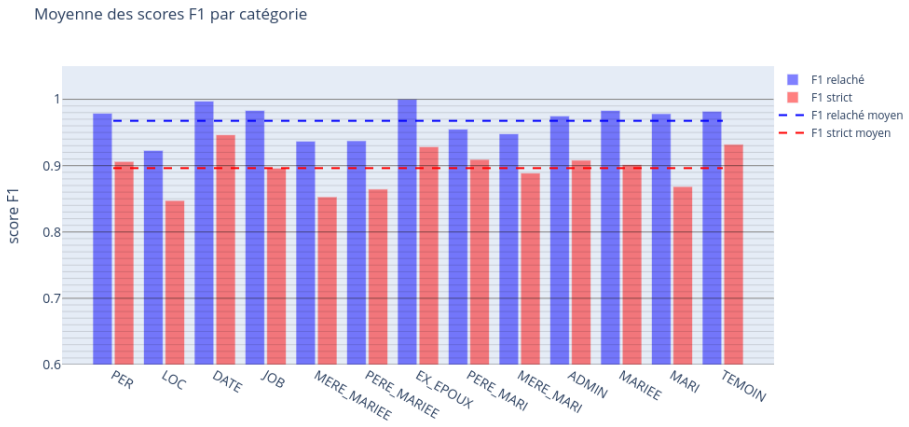


FIGURE 21 – Score F1 moyen strict et relâché par entité few-shot corrigé

Grâce aux corrections précédentes les groupes qui posaient problème proposent de bonnes performances. Cela est confirmé par l'affichage des métriques détaillées.

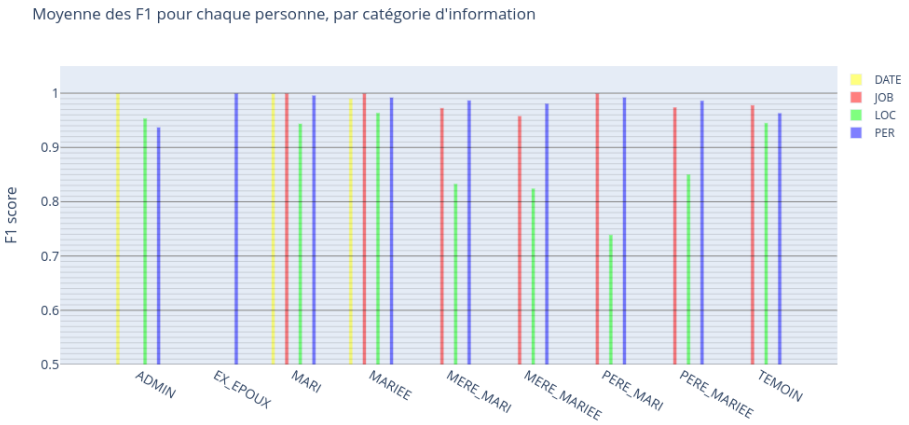


FIGURE 22 – Score F1 moyen strict et relâché par entité par information few-shot corrigé

Il n'y a en effet une belle amélioration sur la détection des métiers et les scores de localisation ont également augmenté. Bizarrerie cependant, celui du père de la mariée est légèrement plus bas, sans réelle explication de ma part.

5 Conclusion

Les méthodes de NER traditionnelles nécessitent un temps d'apprentissage conséquent ainsi qu'un grand nombre de ressources à portée de main. L'apprentissage est de plus supervisé et nécessite donc, en surcroît des ressources informatiques, d'importantes ressources humaines pour effectuer une annotation des données pouvant se montrer très fastidieuse. De plus, le temps et les ressources nécessaires à de bonnes performances augmente de manière croissante avec le nombre de types d'entités à relever, et obtenir de bonnes performances sur des entités apparaissant rarement est difficile.

Dans ce rapport, j'ai tenté de démontrer l'efficacité d'utiliser les Large Language Models et leur rapide capacité d'adaptation pour de l'extraction d'entité nommées en détournant leur comportement auto-régressif pour générer directement les entités à relever, et ce avec un nombre extrêmement réduit d'exemples.

L'application au projet EXO-POPP, particulièrement adapté à cette démonstration du fait de son très grand nombre d'entités présentes mais aussi de la quasi absence de données annotées, a montré des résultats très encourageants avec un score F1 capable d'atteindre 0.96 sur le jeu de test avec seulement 5 exemples d'apprentissage, et avec une vitesse d'exécution dé-corrélée du nombre d'entités à reconnaître.

6 Extension

Pour pousser plus loin, on pourrait envisager l'utilisation de GPT3.5 ou d'autres modèles disponibles pour effectuer une première annotation des données dans des projets de grande envergure afin de permettre l'apprentissage d'autres modèles dédiés spécifiquement au projet.

Cela permettrait de réduire, du moins dans un premier temps, grandement le coût nécessaire à l'annotation des données, qui nécessite souvent de mandater des entreprises spécialisées, procédure coûteuse mais aussi chronophage.

Avec un coût de 0.2ct par 1000 tokens, le coût d'utilisation de GPT3.5 avec la méthodologie utilisée ici serait d'environ 7€ pour annoter 1000 actes au rythme de 1 acte par minute, avec d'excellentes performances. Il faut aussi ajouter que l'humain n'est pas infaillible et que même une correction humaine des données peut laisser passer de rares erreurs.

Références

[1] *Accompagner votre métier vers la transformation digitale.* URL : https://www.numen.fr/.

[2] Alan Akbik, Duncan Blythe et Roland Vollgraf. "Contextual String Embeddings for Sequence Labeling". In : *COLING 2018, 27th International Conference on Computational Linguistics*. 2018, p. 1638-1649.

[3] Alan Akbik et al. "FLAIR : An easy-to-use framework for state-of-the-art NLP". In : *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*. 2019, p. 54-59.

[4] *Attention Is All You Need.* URL : https://arxiv.org/pdf/1706.03762.pdf.

[5] *BERT : Pre-training of Deep Bidirectional Transformers for Language Understanding.* URL : https://arxiv.org/pdf/1810.04805.pdf.

[6] *DAN : a Segmentation-free Document Attention Network for Handwritten Document Recognition.* URL : https://github.com/FactoDeepLearning/DAN.

[7] *Extraction Optique des entités nommées manuscrites pour les actes de mariage de la population de Paris (1880-1940).* URL : https://exopopp.hypotheses.org/.

[8] *FLAIR : A very simple framework for state-of-the-art NLP.* URL : https://github.com/flairNLP/flair.

[9] *GloVe : Global Vectors for Word Representation.* URL : https://nlp.stanford.edu/projects/glove/.

[10] *IAM Handwriting Database.* URL : https://fki.tic.heia-fr.ch/databases/iam-handwriting-database.

[11] *Language Models are Few-Shot Learners.* URL : https://arxiv.org/pdf/2005.14165.pdf.

[12] *OpenAI : Chat Completion.* URL : https://platform.openai.com/docs/guides/chat.

[13] Stefan Schweter et Alan Akbik. *FLERT : Document-Level Features for Named Entity Recognition*. 2020. arXiv : 2011.06993 [cs.CL].

[14] *Training language models to follow instructions with human feedback.* URL : https://arxiv.org/pdf/2203.02155.pdf.

[15] *Training language models to follow instructions with human feedback.* URL : https://arxiv.org/pdf/2203.02155.pdf.