

# T-AIA-901

---

## *TRAVEL ORDER RESOLVER*

**Groupe N°5**

<b>1. Préambule</b>	<b>5</b>
1.1. Contexte du projet	5
1.2. Architecture et stack technique	6
<b>2. Pré et post NLP</b>	<b>8</b>
2.1. Speech-To-Text	8
2.2. Dijkstra	9
2.2.1. Données	9
2.2.2. Graphe	11
2.2.3. Algorithme	12
<b>3. NLP</b>	<b>14</b>
3.1. Définitions	14
3.1.1. Présentation du NLP	14
3.1.2. Classification de texte	15
3.1.3. Reconnaissance d'entités nommées (NER)	16
3.2. Pipeline	17
3.2.1. Prétraitement	17
3.2.2. Tokenization et normalisation	17
3.2.3. Représentation numérique	18
3.2.4. Modélisation et analyse	18
3.2.5. Optimisation et déploiement	18
3.2.6. Suivi continu	18
3.3. Modèles	18
3.4. Enchaînement	19
<b>4. Jeux de données</b>	<b>20</b>
4.1. Fondation	21
4.2. Etiquetage	23
4.3. Complexification	25
4.4. Problématiques rencontrées	26
4.4.1. Valeurs dupliquées	26
4.4.2. Problème du “point final”	26
4.4.3. De et d’	27
4.4.4. Erreurs de templates	27
4.4.5. Equilibre des jeux de données	28
4.4.6. Fine-tuning	29
4.5. Ouverture	30
4.5.1. Caractères inversés et supprimés	30
4.5.2. Départ et arrivée identiques	32
<b>5. Expérimentations</b>	<b>33</b>
5.1. Techniques de Machine Learning classiques	33
5.1.1. Algorithmes	33
5.1.1.1. Classification naïve bayésienne multinomiale	33
5.1.1.2. Régression Logistique	34
5.1.1.3. LinearSVC	35
5.1.1.4. Arbres de décision et Random Forest	36

5.1.1.5. Utilisation du One VS Rest	36
5.1.2. Vectorisation	37
5.1.2.1. Bag of Words	37
5.1.2.1.1. Présentation	37
5.1.2.1.2. Application et résultats	38
5.1.2.2. TF-IDF	43
5.1.2.2.1. Présentation	43
5.1.2.2.2. Application et résultats	44
5.1.2.3. N-grams	45
5.1.2.3.1. Présentation	45
5.1.2.3.2. Application et résultats	46
5.1.2.3.2.1. Bigrams	46
5.1.2.3.2.2. Unigram-Bigram	47
5.1.3. Normalisation	49
5.1.3.1. Stop Words	49
5.1.3.1.1. Présentation	49
5.1.3.1.2. Application et résultats	50
5.1.3.2. Lemmatization	50
5.1.3.2.1. Présentation	50
5.1.3.2.2. Application et résultats	51
5.1.3.3. Stemming	51
5.1.3.3.1. Présentation	51
5.1.3.3.2. Application et résultats	52
5.2. RNN	52
5.2.1. LSTM	52
5.2.1.1. Présentation	52
5.2.1.2. Application et résultats	53
5.2.2. GRU	56
5.2.2.1. Présentation	56
5.2.2.2. Application et résultats	57
5.3. Transformers	60
5.3.1. Concept	60
5.3.2. BERT	62
5.3.2.1. Présentation	62
5.3.2.2. Application et résultats	63
5.3.3. DistilBERT	65
5.3.3.1. Présentation	65
5.3.3.2. Application et résultats pour la classification de texte	66
5.3.3.3. Application et résultats pour la NER	68
5.3.4. CamemBERT	70
5.3.4.1. Présentation	70
5.3.4.2. Application et résultats pour la classification de texte	71
5.3.4.3. Application et résultats pour la NER	71
5.4. SpaCy	72

5.4.1. Présentation	72
5.4.1.1. C'est quoi Spacy ?	72
5.4.1.2. Principes de SpaCy pour le NER	73
5.4.2. Application et résultats pour le NER	73
5.4.2.1. Problématique	73
5.4.2.2. Hypothèses de travail	73
5.4.2.3. Choix du modèle	73
5.4.2.4. Modifications du modèle	74
5.4.2.5. Préparation des données	74
5.4.2.6. Entraînement	75
5.5. Refactorisation	77
<b>6. Choix finaux</b>	<b>79</b>
6.1. Benchmark sur la classification de texte	79
6.1.1. Méthode d'expérimentation	79
6.1.2. Classifieurs	80
6.1.3. Résultats	80
6.2. Choix de DistilBERT pour la NER	82
<b>7. Conclusion</b>	<b>84</b>

# 1. Préambule

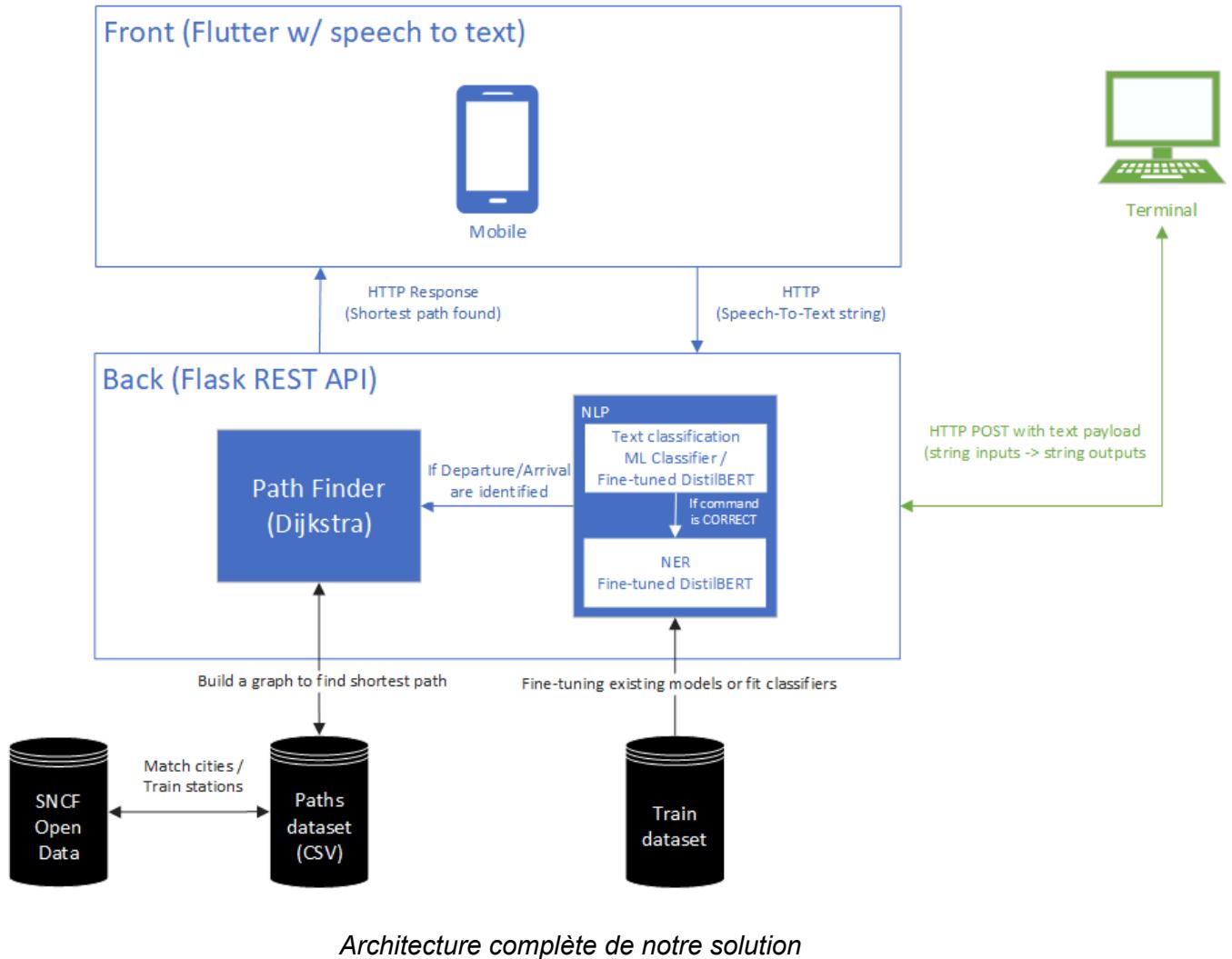
## 1.1. Contexte du projet

Au cœur de l'ère numérique, l'interaction entre l'homme et la machine s'avère être un domaine en constante évolution, poussé par les avancées fulgurantes de l'intelligence artificielle et du Traitement du Langage Naturel (NLP). Notre projet s'inscrit dans cette dynamique, cherchant à explorer et à repousser les limites de la compréhension de texte par les machines. Face à la complexité croissante des interactions humaines et à la diversité des langages naturels, le besoin de systèmes intelligents capables de comprendre et d'interpréter du langage de manière autonome n'a jamais été aussi crucial.

La demande initiale de ce projet émane d'une volonté de concevoir une solution capable de naviguer à travers un grand nombre des données textuelles, de les comprendre et de les traiter de manière efficace et pertinente pour pouvoir classer des **commandes de trajet passées par des utilisateurs** en **Speech-To-Text** sur une interface puis d'identifier le départ et l'arrivée de la commande de trajet si cette dernière est jugée valide. Lorsqu'un départ et une arrivée ont été extrait de la commande, **le chemin le plus court** en train doit être déterminé entre ces deux points.

Pour répondre à cette demande, notre approche s'est reposée sur une exploration approfondie des technologies de pointe en matière de NLP. Cette exploration nous a conduits à examiner plus particulièrement les tâches de **classification de texte** et de **reconnaissance d'entités nommées** (NER) pour répondre à notre besoin. Ces tâches ont été appréhendées avec des algorithmes de Machine Learning ou des méthodes plus avancées telles que les **réseaux de neurones récurrents** (RNN), **spaCy** et les modèles de **transformers**.

## 1.2. Architecture et stack technique



L'architecture du système conçue intègre une interaction front-end/back-end pour le traitement de commandes vocales. Le front-end, développé avec Flutter, active la reconnaissance vocale sur des dispositifs mobiles, convertissant la parole des utilisateurs en requêtes textuelles (Speech-To-Text). Ces requêtes sont ensuite transmises via HTTP au back-end.

Le back-end repose sur une API REST Flask qui orchestre le traitement des données. Il comprend un moteur de traitement du langage naturel (NLP) qui effectue deux tâches principales : la classification de texte et l'extraction d'entités nommées (NER), toutes deux utilisant des modèles DistilBERT finement ajustés sur nos jeux de données pour interpréter correctement les commandes de trajet.

En parallèle, le "Path Finder" du back-end, s'appuyant sur l'algorithme de Dijkstra, se base sur un graphe pour trouver l'itinéraire le plus court entre des villes ou des gares, en utilisant des données ouvertes fournies par un dataset CSV des chemins possibles fusionnés avec des données des gares françaises disponibles sur SNCF Open Data. Une fois l'itinéraire optimal déterminé, le résultat est renvoyé au front-end sous forme de réponse HTTP, complétant ainsi le cycle de l'interaction utilisateur avec le système.

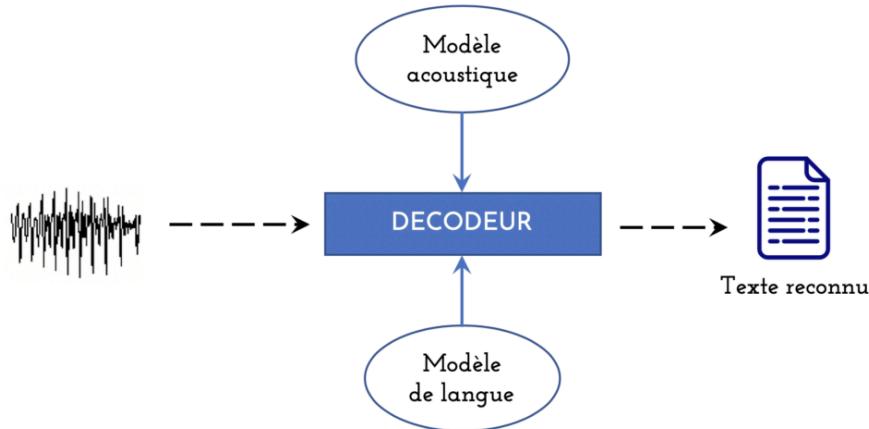
Cette structure reflète une fusion de technologies de pointe dans le domaine de l'IA et des interfaces utilisateur modernes, assurant une expérience utilisateur fluide et intuitive depuis la saisie vocale jusqu'à la réception de l'itinéraire calculé.

*A noter qu'il est possible de requêter le back-end, et individuellement le NLP et le path-finder depuis un terminal.*

## 2. Pré et post NLP

Le NLP est la partie au centre de notre projet. Elle fait le lien entre deux parties aux extrémités: la partie Speech-To-Text qui transforme un audio donné en texte, et la partie recherche du chemin le plus court entre deux endroits pour obtenir le résultat le plus optimal selon la commande de trajet passé par un utilisateur.

### 2.1. Speech-To-Text



*Schéma simplifié de la conversion de la parole en texte (Speech-To-Text)*

La conversion de la parole en texte (Speech-to-Text ou STT) est une technologie sophistiquée qui vise à transcrire les informations vocales en une forme écrite, ou texte. Du point de vue général, cette technologie est largement utilisée dans divers domaines tels que la transcription automatique, les interfaces vocales et l'accessibilité des applications.

Mathématiquement, le processus de STT peut être décrit comme une transformation complexe du signal audio en une séquence de mots ou de phrases. On peut le modéliser en utilisant des concepts de traitement du signal et d'apprentissage automatique. À un niveau élémentaire, cela implique la décomposition du signal vocal en segments plus petits appelés phonèmes, qui sont ensuite associés à des unités linguistiques telles que les mots. Les modèles acoustiques et linguistiques, souvent basés sur des réseaux neuronaux, sont utilisés pour apprendre les relations entre les phonèmes et les mots, améliorant ainsi la précision de la transcription.

En termes mathématiques, supposons que le signal vocal soit représenté par une fonction d'onde temporelle. On peut appliquer des techniques de traitement du signal telles que la transformée de Fourier pour convertir ce signal dans le domaine fréquentiel, permettant ainsi d'analyser les composantes sonores. En parallèle, des modèles de langage, basés sur des probabilités conditionnelles, sont utilisés pour estimer la séquence de mots la plus probable compte tenu des phonèmes détectés. Les algorithmes d'optimisation, tels que la descente de gradient, sont souvent utilisés pour ajuster les paramètres du modèle et améliorer la performance de la transcription.

Notre interface consistant en une application mobile développée avec Flutter, [nous avons utilisé ce package Flutter pour la partie Speech-To-Text](#). Cette librairie expose les capacités de reconnaissance

vocale spécifiques à l'appareil. Si l'on installe l'application sur un téléphone Android, la librairie fera alors appel à **Google Speech-to-Text**.

## 2.2. Dijkstra

### 2.2.1. Données

Avant de pouvoir aborder l'algorithme permettant de trouver le chemin le plus court en temps entre deux gares, il faut d'abord présenter les données sur lesquelles nous allons nous baser pour réaliser cette tâche.

	trip_id	trajet	duree
1	OCESN003100F140147152	Gare de Le Havre - Gare de Paris-St-Lazare	138
2	OCESN003190F040047309	Gare de Dieppe - Gare de Paris-St-Lazare	145
3	OCESN003198F030037315	Gare de Paris-St-Lazare - Gare de Rouen-Rive-Droite	97
4	OCESN003300F030037323	Gare de Cherbourg - Gare de Paris-St-Lazare	194
5	OCESN003313F380387526	Gare de Caen - Gare de Paris-St-Lazare	149
6	OCESN003371F080087908	Gare de Paris-St-Lazare - Gare de Trouville-Deauville	147
7	OCESN003410F120128075	Gare de Granville - Gare de Paris-Montp.3-Vaug.	201
8	OCESN003460F010018083	Gare de Granville - Gare de Paris-Montparnasse 1-2	195

*Etat initial du fichier timetables.csv*

De base, nous avons à notre disposition un jeu de données listant l'ensemble des lignes de la SNCF pouvant être empruntés lors d'un voyage en train. Pour chaque ligne, nous avons le numéro, les deux gares qu'elle relie ainsi que la durée en minutes entre les deux gares. Les noms des gares sont présents dans la même colonne "trajet", on peut alors retravailler le fichier pour plus de commodité.

gare_a	gare_b	duree
Le Havre	Paris-St-Lazare	138
Dieppe	Paris-St-Lazare	145
Paris-St-Lazare	Rouen-Rive-Droite	97
Cherbourg	Paris-St-Lazare	194
Caen	Paris-St-Lazare	149
Paris-St-Lazare	Trouville-Deauville	147
Granville	Paris-Montp.3-Vaug.	201
Granville	Paris-Montparnasse 1-2	195

*Les noms de gares présents dans la colonnes "trajet" du timetables.csv séparés en deux colonnes*

Ici, nous avons séparé les noms des deux gares pour chaque ligne en deux colonnes (Gare A et Gare B). Selon cet état, on pourrait déjà construire le graphe qui sera alors parcouru par un algorithme pour trouver le chemin le plus court. **Cependant, il faut partir du principe que les commandes de trajet qui seront passées par Speech-To-Text (ou par écrit) ne contiendront pas explicitement les noms de gares entre lesquelles l'utilisateur veut effectuer son trajet.** On pourrait plutôt s'attendre à ce que l'utilisateur dise "Je veux aller à Nantes en partant de Paris" plutôt que "Je veux aller à la gare de Nantes en partant de la gare de Paris-Montparnasse".

Gares de voyageurs

Twitter Facebook LinkedIn Email

Informations									Tableau	Carte	Analyse	Export	API	
	Code UIC	Code gare	Intitulé gare	DTG	Région SNCF	Unité gare	UT	Code plate-forme						
1	0087313759	02266	Abancourt	DRG HDF-Normandie	REGION PICARDIE	UG Picardie	ABANCOURT P GARE	02266-1						
2	0087481614	01604	Abbaretz	DRG B-CVL-PDL	REGION PAYS DE LA LOIRE	UG Pays de la Loire	ABBARETZ GARE	01604-1						
3	0087317362	02219	Abbeville	DRG HDF-Normandie	REGION PICARDIE	UG Picardie	ABBEVILLE GARE	02219-1						
4	0087545269	01460	Ablon-sur-Seine	DGIF	REGION DE PARIS RIVE GAUCHE		ABLON GARE	01460-1						
5	0087683656	00773	Accolay	Agence Centre Est Rhône Alpin				00773-1						
6	0087386052	02020	Achères Grand Cormier	DGIF	REGION DE PARIS SAINT-LAZARE		ACHERES GRAND CORMIER GARE	02020-1						
7	0087381657	02062	Achères Ville	DGIF	REGION DE PARIS SAINT-LAZARE		ACHERES VILLE GARE	02062-1						
8	0087316745	02242	Acheux - Franleu	DRG HDF-Normandie			ACHEUX FRANLEU HALTE	02242-1						
9	0087342154	03098	Achicourt					03098-1						

### Jeu de données "Gares de voyageurs" issu du site SNCF Open Data

Il est possible de retrouver le nom des villes dans lesquelles se trouvent les gares de notre jeu de données grâce au site [SNCF Open Data](#) et plus précisément le jeu de données [Gares de voyageurs](#). Les colonnes "Intitulé gare" et "Commune" présentent dans l'export de données permettent de faire le lien entre le nom de la gare et le nom de la ville dans la plupart des cas.

```
"Issoudun-Place-de-la-G.": "Issoudun",
"Pont-Audemer-Gare-SNCF": "Pont-Audemer",
"Bois-d'Oingt-Centre": "Bois-d'Oingt-Légy",
"Bitche-Quartier-Driant": "Bitche",
"Biganos-Facture": "Facture-Biganos",
"Saint-Claude": "St-Claude",
```

### Renommage forcé de certaines gares pour correspondre au nom de SNCF Open Data

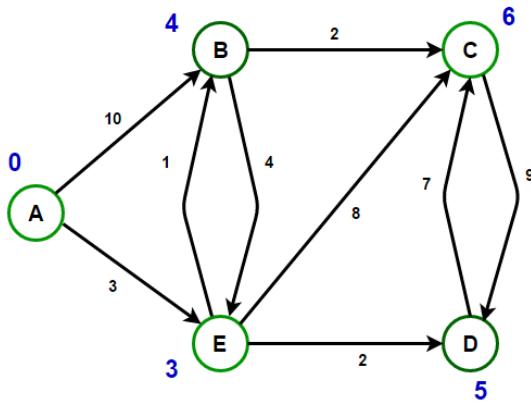
Certains noms de gare présents dans le fichier timetables.csv ne sont cependant pas présents dans les gares de l'export de la SNCF, ce qui empêche de faire la correspondance **Gare timetables.csv** →

**Gare SNCF Open Data** → **Commune**. Dans ce cas, on renomme au cas par cas les noms de gares de notre fichier de base non retrouvés en leurs potentiels équivalents dans le fichier de la SNCF.

gare_a	gare_b	gare_a_city	gare_b_city
Le Havre	Paris-St-Lazare	HAVRE (LE )	PARIS
Dieppe	Paris-St-Lazare	DIEPPE	PARIS
Paris-St-Lazare	Rouen-Rive-Droite	PARIS	ROUEN
Cherbourg	Paris-St-Lazare	CHERBOURG	PARIS
Caen	Paris-St-Lazare	CAEN	PARIS
Paris-St-Lazare	Trouville-Deauville	PARIS	DEAUVILLE
Granville	Paris-Montp.3-Vaug.	GRANVILLE	PARIS
Granville	Paris-Montparnasse 1-2	GRANVILLE	PARIS

Une fois la fusion effectuée, on arrive alors à associer pour la grande majorité des gares leur ville dans des colonnes "Ville Gare A" et "Ville Gare B".

## 2.2.2. Graphe



Les graphes sont des structures de données puissantes utilisées pour modéliser des relations entre des entités. Ils sont composés de nœuds (ou sommets; en vert sur le schéma) et d'arêtes (ou liens; en noir sur le schéma) qui définissent les connexions entre ces nœuds. Dans notre contexte, chaque ville serait représentée par un nœud, et les connexions par rail entre les villes seraient représentées par des arêtes.

Chaque arête du graphe a un poids. Le concept de poids associé aux arêtes d'un graphe est crucial pour modéliser des propriétés quantitatives telles que la distance, le coût, ou le temps de trajet entre deux nœuds. Les graphes avec des poids sont appelés des graphes pondérés. Dans notre cas, le poids représente le temps de trajet entre deux gares.

Les données formatées depuis le fichier timetables.csv avec les données de SNCF Open Data permettent de construire un graphe où l'on associe à chacune des villes, les différents nœuds liés (villes) et le poids de l'arête qui les relie (durée du trajet en minutes entre deux villes).

```

for index, row in df.iterrows():
    # We add the stations to the graph if they are not already in
    # with the duration as value
    if row["gare_a_city"] not in graph:
        graph[row["gare_a_city"]] = {}
    graph[row["gare_a_city"]][row["gare_b_city"]] = row["duree"]

    if row["gare_b_city"] not in graph:
        graph[row["gare_b_city"]] = {}
    graph[row["gare_b_city"]][row["gare_a_city"]] = row["duree"]
  
```

*Méthode de génération du graphe; si un noeud n'existe pas, on l'initialise puis on lui associe le noeud auquel il est relié ainsi que le poids de son arête commune*

```

    "DEAUVILLE": {
        "PARIS": 147,
        "LISIEUX": 50,
        "DIVES-SUR-MER": 35
    },
    "GRANVILLE": {
        "PARIS": 195,
        "DREUX": 142,
        "SAINTE-MARGUERITE-D'ELLE": 84,
        "COUTANCES": 36,
        "VIRE-CENTRE": 75,
        "RENNES": 110,
        "CH\u00e2TENOIS": 59,
        "MOULINS-SUR-ORNE": 83,
        "CAEN": 103
    },

```

*Extrait du fichier JSON stockant le graphe; chaque noeud possède un dictionnaire avec pour chaque clé, un noeud auquel il est lié et pour valeur associée la durée du trajet entre les deux noeuds*

### 2.2.3. Algorithme

L'algorithme de Dijkstra, développé par l'informaticien néerlandais Edsger Dijkstra en 1956, est utilisé pour résoudre le problème du plus court chemin dans un graphe pondéré. L'idée derrière cet algorithme est de trouver le chemin le plus court entre un nœud de départ et tous les autres nœuds du graphe. L'algorithme a été initialement conçu pour être implémenté sur des ordinateurs de l'époque, qui avaient des ressources limitées en termes de mémoire et de puissance de calcul.

L'algorithme de Dijkstra est couramment utilisé dans les réseaux informatiques, les systèmes de transport, la planification de la chaîne logistique, la cartographie, etc. Il est utilisé pour déterminer le chemin le plus court entre deux points dans un réseau pondéré, **ce qui s'applique à notre graphe**.

Pour déterminer le chemin le plus court dans notre graphe, voici les étapes suivies par l'algorithme:

- L'initialisation; une valeur de distance infinie est assignée à tous les nœuds, sauf au nœud de départ pour lequel la distance est de zéro.
- La sélection du nœud, on choisit le nœud avec la plus petite distance non marquée, c'est-à-dire la plus petite valeur de distance parmi les nœuds du graphe qui n'ont pas encore été inclus dans le chemin le plus court trouvé.
- La mise à jour des distances; on met à jour les distances des nœuds adjacents en ajoutant la distance du nœud sélectionné au poids de l'arête les reliant. Si la nouvelle distance pour atteindre un nœud est plus petite que la distance actuelle enregistrée pour atteindre ce même nœud, mettre à jour la distance.
- Le nœud sélectionné est ensuite marqué comme visité.
- Les étapes de sélection du nœud, de mise à jour des distances et de marquage se répètent jusqu'à ce que tous les nœuds soient marqués ou que la destination soit atteinte.

- Une fois que tous les nœuds ont été marqués ou que la destination est atteinte, l'algorithme se termine et les distances les plus courtes sont connues.

A noter l'existence de [l'algorithme A\\*](#), un algorithme de recherche de chemin qui combine les avantages de l'algorithme de Dijkstra et de la recherche heuristique. Il a été conçu pour résoudre le problème du plus court chemin dans un graphe pondéré tout en utilisant une heuristique pour améliorer l'efficacité de la recherche.

A\* utilise une fonction heuristique qui estime le coût restant pour atteindre la destination. Cette heuristique permet à A\* de guider la recherche vers les régions les plus prometteuses du graphe, ce qui réduit le nombre total de nœuds examinés. Si la fonction heuristique est mal choisie, elle peut conduire à des résultats suboptimaux ou même à une perte de garantie d'optimalité. Le choix d'une heuristique appropriée peut parfois être délicat.

**Dans notre cas, l'algorithme de Dijkstra permet déjà d'avoir des résultats rapides, mais il est important d'être conscient qu'il n'est pas le seul algorithme de recherche du chemin le plus court.**

### 3. NLP

#### 3.1. Définitions

##### 3.1.1. Présentation du NLP

Le **Traitemet du Langage Naturel** (NLP, pour Natural Language Processing en anglais) est un domaine de l'intelligence artificielle qui se concentre sur l'interaction entre les ordinateurs et le langage humain. L'objectif principal du NLP est de permettre aux machines de comprendre, d'interpréter et de générer un langage humain de manière naturelle.

Le NLP englobe le Natural Language Understanding (NLU) et la Natural Language Generation (NLG). Le NLU se concentre sur la compréhension du langage en analysant les aspects sémantiques, syntaxiques et pragmatiques des textes ou de la parole. Il vise à extraire des informations significatives et à interpréter le sens derrière les phrases. D'un autre côté, le NLG se concentre sur la création de texte de manière automatisée, générant des réponses ou des textes cohérents en fonction de modèles préétablis.



#### *Exemples d'applications concrètes du NLP*

Le NLP, et notamment le NLU et le NLG ont beaucoup d'applications, dont:

- Les assistants virtuels comme Siri, Google Assistant et Alexa utilisent le NLP pour comprendre les commandes vocales et répondre de manière contextuelle.
- Des applications telles que Google Translate exploitent le NLP pour traduire instantanément du texte d'une langue à une autre.
- Les entreprises utilisent aussi le NLP pour analyser les opinions des clients sur les médias sociaux, les avis en ligne, afin d'évaluer le sentiment général à l'égard de leurs produits ou services.

- Des chatbots sont déployés dans les services client en ligne pour comprendre et répondre aux requêtes des utilisateurs de manière conversationnelle.
- Les algorithmes NLP sont utilisés pour extraire les informations clés d'un texte et générer automatiquement un résumé concis, utile dans la gestion de grandes quantités de données textuelles.
- Les modèles NLP, tels que GPT (Generative Pre-trained Transformer), sont utilisés pour la génération de texte automatique, permettant la création de contenu rédactionnel, articles, et même de dialogues de manière contextuellement cohérente.

De ces applications, découlent des techniques de traitement et d'analyse du langage humain qui peuvent nous aider dans la résolution de notre problème initial, à savoir extraire les villes de départ et d'arrivée parmi des commandes de trajet valides: **la classification de texte et la NER**.

### 3.1.2. Classification de texte



*La classification de texte permet de classer un texte dans des catégories définies selon son contenu*

La classification de texte est une tâche fondamentale du NLP qui consiste à attribuer automatiquement des catégories prédéfinies à des documents textuels. L'objectif principal est de classer un morceau de texte dans une ou plusieurs catégories spécifiques en fonction de son contenu. Ce processus repose sur l'utilisation de modèles d'apprentissage automatique qui apprennent à partir de données annotées, associant des caractéristiques textuelles à des catégories prédéterminées. Les données apportées à l'entraînement sont toutefois traitées par différentes étapes en appliquant des techniques permettant de faciliter la compréhension du texte par un modèle lors de son apprentissage.

Dans notre cas, la classification de texte peut nous permettre de classer les commandes de trajet en plusieurs catégories et déterminer lesquelles sont correctement formulées, mal formulées, dans une autre langue que le français ou non classifiables / inintelligibles.

*Pour être plus précis, nous allons utiliser la classification "One-vs-Rest" (OvR) ou "One-vs-All" (OvA). Dans ce cadre, chaque catégorie serait traitée comme une classe distincte, et le modèle serait formé pour prédire si un texte appartient à une catégorie donnée par rapport à toutes les autres catégories*

combinées. Cette approche offre une flexibilité dans la gestion de multiples classes et permet au modèle de généraliser efficacement les prédictions pour des scénarios variés, notamment dans la classification des différentes catégories de commandes de trajet.

### 3.1.3. Reconnaissance d'entités nommées (NER)

La Reconnaissance d'Entités Nommées (NER) est une autre tâche cruciale du NLP qui vise à identifier et classifier des entités spécifiques telles que les noms de personnes, d'organisations, de lieux, les dates, les montants monétaires, etc., dans un texte. L'objectif est de repérer et d'extraire des informations spécifiques à partir du contenu textuel, facilitant ainsi la compréhension et l'analyse des données. Les modèles NER utilisent souvent des approches basées sur des séquences et des techniques avancées de modélisation du langage pour accomplir cette tâche avec précision; tout comme la classification de texte, des techniques de préparation de la données facilite l'apprentissage des modèles.



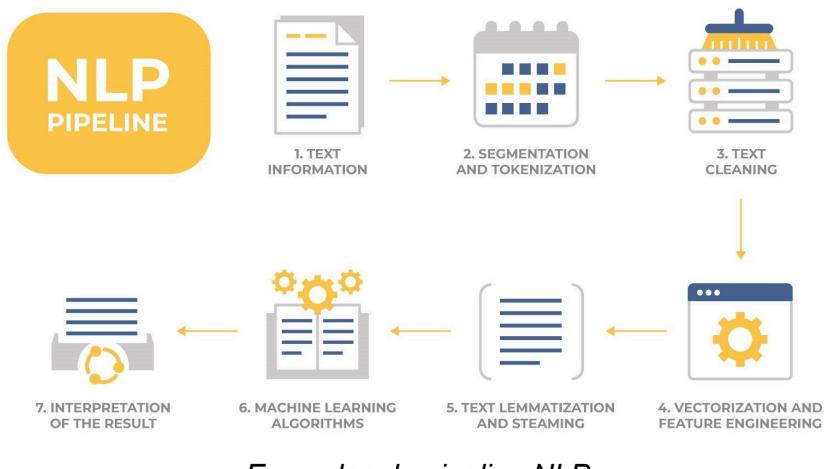
In December 1903 DATE the Royal Swedish Academy of Sciences ORG awarded  
Marie PERSON and Pierre Curie PERSON, along with Henri Becquerel PERSON,  
the Nobel Prize in Physics WORK\_OF\_ART.

La NER identifie des entités spécifiques dans une phrase comme une date, des personnes, etc.

Pour notre cas d'étude, les NER vont être la solution pour pouvoir identifier le départ et l'arrivée d'un trajet dans une commande classifiée valide par l'étape de classification de texte.

À noter que le Chunking et le Part Of Speech (POS), bien que ces techniques ne soient pas adaptées à notre problème spécifique, existent également et permettent de structurer le texte d'une manière significative. Le Chunking, en identifiant des groupes nominaux, facilite la reconnaissance de morceaux sémantiques importants dans un texte, tandis que le POS, en attribuant des étiquettes grammaticales à chaque mot, offre des informations précieuses sur la structure syntaxique d'une phrase.

### 3.2. Pipeline



La mise en place d'un NLP implique généralement un **pipeline** comprenant plusieurs étapes permettant d'analyser le texte en entrée.

#### 3.2.1. Prétraitement

La phase de prétraitement des données textuelles est cruciale pour garantir la qualité des informations analysées. La collecte des données implique la réunion des documents nécessaires, tandis que le nettoyage des données vise à éliminer les éléments redondants et à corriger les erreurs typographiques. L'utilisation de techniques telles que la suppression des doublons, la correction orthographique automatisée, et la normalisation des formats contribue à obtenir un ensemble de données cohérent et fiable pour l'analyse subséquente.

#### 3.2.2. Tokenization et normalisation

La tokenization et la normalisation jouent un rôle essentiel dans la préparation des données textuelles avant leur utilisation dans des modèles NLP. La tokenization consiste à diviser le texte en unités linguistiques de base appelées tokens, qui peuvent être des mots, des phrases, ou même des sous-mots, selon le niveau de granularité requis. Cette étape est fondamentale car elle structure le texte en éléments individuels, facilitant ainsi leur analyse ultérieure.

En complément de la tokenization, la normalisation inclut deux étapes clés : la lemmatisation et le stemming. La lemmatisation réduit les mots à leur forme de base, appelée lemme, afin de normaliser le vocabulaire. Par exemple, les mots "courir" et "courait" seront ramenés à leur forme de base "courir". Le stemming, quant à lui, vise à réduire les mots à leur racine, même si cette racine n'est pas un mot réel. Ainsi, le stemming de "jouer", "joue", et "jouais" donnerait "jou".

La suppression des stop-words est également une composante de la normalisation. Les stop-words sont des mots fréquents tels que "et", "le", et "de", qui n'apportent souvent pas de valeur sémantique significative dans de nombreuses tâches NLP. Les éliminer contribue à réduire le bruit et à améliorer la pertinence du texte pour l'analyse.

### 3.2.3. Représentation numérique

La représentation numérique des données textuelles est une étape cruciale pour permettre aux modèles d'apprentissage automatique de comprendre et traiter le langage. La vectorisation convertit les mots ou tokens en vecteurs numériques, transformant ainsi le texte en une forme que les algorithmes peuvent manipuler. Parmi les techniques couramment utilisées, citons la méthode Bag of Words, qui représente un document comme un ensemble non ordonné de mots, et la méthode TF-IDF (Term Frequency-Inverse Document Frequency), qui évalue l'importance relative des termes dans un document. De plus, l'utilisation d'approches basées sur des n-grams permet de capturer les relations entre plusieurs mots adjacents, améliorant ainsi la représentation sémantique du texte. Ces techniques jouent un rôle clé dans la création de représentations numériques efficaces pour l'analyse et la modélisation ultérieures dans le domaine du Traitement du Langage Naturel.

### 3.2.4. Modélisation et analyse

La modélisation consiste à appliquer des modèles NLP spécifiques à la tâche souhaitée. Des approches classiques comme la régression logistique et les machines à vecteurs de support sont souvent utilisées pour la classification de texte, tandis que des modèles avancés tels que les réseaux neuronaux, comme ceux basés sur l'architecture transformer, sont adoptés pour des tâches plus complexes.

### 3.2.5. Optimisation et déploiement

L'optimisation du modèle implique l'ajustement des paramètres et la réévaluation des caractéristiques pour améliorer ses performances. Le déploiement consiste à intégrer le modèle dans un environnement de production. Des techniques telles que le réglage des hyperparamètres et l'utilisation de données d'entraînement supplémentaires peuvent contribuer à l'optimisation, tandis que l'utilisation de serveurs et d'APIs facilitent le déploiement dans des applications réelles.

### 3.2.6. Suivi continu

Le suivi continu permet de maintenir la pertinence du modèle au fil du temps. La surveillance des performances, l'ajout de nouvelles données d'entraînement et les mises à jour régulières du modèle garantissent sa capacité à s'adapter aux évolutions du langage et des besoins de l'application. Un suivi continu bien orchestré contribue à la durabilité et à la fiabilité du système NLP.

## 3.3. Modèles

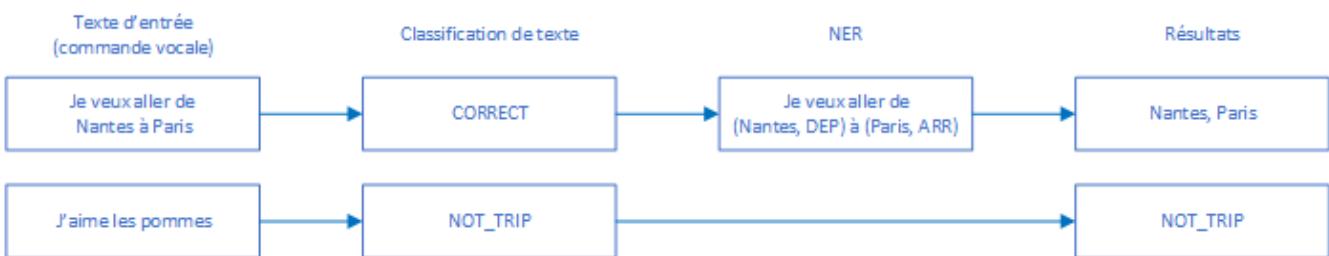
Comme indiqué précédemment, un pipeline NLP comporte une partie fondamentale de modélisation. **Les processus utiles à la résolution de notre problème peuvent être abordés avec différentes méthodes et techniques de modélisation.** En effet, plusieurs types de modèles sont couramment utilisés dans le cadre de la classification de texte et de l'identification d'entités nommées (NER).

Les modèles de régression logistique, les machines à vecteurs de support (SVM), les arbres de décision, et les forêts aléatoires sont parmi les approches classiques pour la **classification de texte** (algorithmes de *Machine Learning*). Pour des tâches plus complexes, les réseaux de neurones, y compris les réseaux récurrents (RNN), les réseaux de neurones récurrents à mémoire à court terme (LSTM), et les réseaux de neurones à attention, sont largement utilisés.

En ce qui concerne l'**identification d'entités nommées** (NER), des modèles basés sur des séquences, tels que les modèles de Markov cachés (HMM), les réseaux de neurones récurrents, et les modèles à base de transformer, ont démontré leur efficacité. Les modèles de séquence à séquence et les approches basées sur les ensembles, qui combinent plusieurs modèles pour améliorer les performances, sont également utilisés pour les tâches de NER.

*Un grand nombre de ces différents types de modélisation seront abordés lors de la présentation de nos expérimentations. A noter que nos expérimentations se sont principalement tournées vers les techniques de modélisation les plus performantes aujourd'hui, respectivement pour la classification de texte et la NER.*

### 3.4. Enchaînement



*Les deux processus du NLP peuvent s'enchaîner de façon conditionnelle et se compléter dans l'analyse du texte d'entrée*

L'enchaînement de plusieurs **processus** combinant des tâches du NLP est courant pour résoudre des problèmes plus complexes. Dans notre cas par exemple, un enchaînement de processus de NLP pourrait commencer par une tâche de classification pour déterminer la catégorie générale d'une phrase dans un contexte donné (commande de trajet), puis utiliser un modèle spécifique pour effectuer une extraction d'entités nommées (NER) dans une catégorie particulière pour compléter l'analyse réalisée en premier lieu. **Cela permet d'appliquer des techniques spécialisées à chaque étape du processus, améliorant potentiellement la précision globale du système et ses résultats.**

**Cela signifie également de créer deux jeux de données distincts pour pouvoir entraîner des modèles à performer au mieux pour la tâche qui leur est confiée.**

## 4. Jeux de données

Dans le contexte du projet, aucun jeu de données n'a été fourni et il a été fortement conseillé de créer un jeu de données de toutes pièces pour pouvoir répondre au besoin fonctionnel et final de notre NLP. Le besoin fonctionnel est d'abord de **classifier les commandes de trajet** provenant du Speech-To-Text puis identifier le **départ** et l'**arrivée** du trajet demandé si la commande passée est considérée comme valide, pour enfin déterminer le chemin le plus court pour un trajet donné.

Les difficultés et points d'attention dans la génération d'un jeu de données concernant le traitement du langage naturel sont nombreux:

- Les utilisateurs peuvent formuler leurs demandes de manière très diverse. Ils peuvent utiliser différentes expressions, synonymes, et structures de phrases pour demander la même information. La variabilité linguistique rend difficile la création d'un jeu de données exhaustif qui couvre toutes les façons possibles de poser une question sur les trajets entre deux villes
- Certaines requêtes peuvent être ambiguës. Par exemple, une commande comme "Nantes à Toulouse" peut être interprétée différemment par différentes personnes. Dans le contexte précis de la recherche du chemin le plus court en train entre deux villes, on pourrait interpréter la phrase comme "Quel est le chemin le plus court entre Nantes et Toulouse ?". Mais dans le cas d'un module de reconnaissance et d'analyse vocale plus complexe qui traiterait de plusieurs tâches complexes en simultané (jouer une musique, envoyer un message, etc.), les éléments fournis dans la commande pourraient être insuffisants pour pouvoir être considérés comme valides.
- Les utilisateurs peuvent commettre des erreurs de frappe ou de saisie lorsqu'ils formulent leurs commandes. Il est difficile de prévoir toutes les façons dont ces erreurs pourraient se produire et d'inclure ces variantes dans le jeu de données.
- Les utilisateurs peuvent utiliser un langage informel, des abréviations, des acronymes ou des expressions familières pour poser leurs questions. Ces variations de langage peuvent compliquer la création d'un ensemble de données complet.
- Les noms de gares, les références aux trajets et les préférences peuvent varier en fonction du contexte géographique. La génération d'un jeu de données généralisable pour différentes régions nécessite une attention particulière à cette diversité.

Tout en prenant en compte l'ensemble de ces points d'attention, plusieurs étapes ont permis de construire un jeu de données relativement complet pour tenir compte du maximum de cas génériques possibles de commandes pouvant être réalisés par un utilisateur. Des premiers tests de génération ont été réalisés puis étoffés et spécialisés pour pouvoir convenir à l'apprentissage durant les deux tâches du NLP. Le travail de fondation et de complexification a néanmoins rencontré plusieurs problématiques soulevées suite à différents entraînements.

## 4.1. Fondation

Afin de pouvoir construire les jeux de données permettant d'entraîner des modèles permettant de réaliser les deux étapes de notre NLP, il faut d'abord avoir une base de phrases relativement complète peu importe le type de commande qu'elles représentent. C'est à dire qu'il faut autant de phrases de commandes valides de trajet, que de phrases en langues étrangères, de phrases hors contexte, de phrases incompréhensibles et inintelligibles ou de phrases partiellement ou totalement invalides mais qui sont sémantiquement ou vocabulairement parlant ressemblantes à des commandes valides.

En commençant par la génération de phrases correctes, deux éléments sont nécessaires:

- des modèles / *templates* de commandes de trajet valides
- un jeu de données de villes permettant de remplir ces *templates*

Concernant le jeu des données de villes, nous l'avons déjà à notre disposition puisque qu'il s'agit de récupérer l'ensemble des villes issues de la fusion expliquée précédemment entre le fichier **timetables.csv** et les données de **SNCF Open Data**.

Ensuite pour créer un ensemble de *templates* de phrases correctes, nous avons demandé à **ChatGPT** de nous générer des exemples de phrases qu'une personne pourrait dire afin d'obtenir le chemin en train entre deux villes, puis les avons transformé en modèles. Nous les avons ensuite complété avec des exemples de phrases plus complexes, plus naturelles ou plus spécifiques.

### Exemples de modèles de phrases:

Peux-tu m'aider à planifier le trajet depuis **{departure}** jusqu'à **{arrival}** ?

Comment atteindre **{arrival}** en partant de **{departure}** le plus rapidement ?

J'aimerais connaître le chemin pour aller à **{arrival}** depuis **{departure}**.

Trouve-moi un moyen de transport de **{departure}** vers **{arrival}**.

Je cherche à me déplacer vers **{arrival}** depuis **{departure}**.

Indique-moi le trajet le plus simple depuis **{departure}** vers **{arrival}**.

On remarque les emplacements **{departure}** et **{arrival}** dans chacun de modèles, qui sont remplacés par des noms de villes issus du jeu de données correspondant et qui permettront plus tard de savoir les emplacements des phrases générées à annoter pour déterminer le départ et l'arrivée de chacune des phrases.

De façon similaire aux modèles de phrases correctes, des modèles de commandes non valides ont été générés pour chacun des labels attendus en sortie de l'étape de classification de texte (voire chapitre suivant).

### Exemples de modèles de phrases invalides:

Pour le label NOT\_TRIP:

- Je cherche un moyen de partir de **{departure}**.
- Trouve un moyen de quitter **{departure}**.
- Comment aller à **{arrival}** ?
- Je veux aller de **{departure}** à **{arrival}** ou **{arrival}**.
- Je veux me rendre de **{departure}** à **{arrival}** ou **{departure}**.
- Je veux me rendre à **{arrival}** ou **{departure}**.

Pour le label NOT\_FRENCH:

- I want to go from {departure} to {arrival}.
- I would like to travel from {departure} to {arrival}.
- I am planning a trip from {departure} to {arrival}.

Pour les deux labels simultanément:

- I want to arrive at {arrival}.
- My journey ends at {arrival}.
- Find a way to leave {departure}.

On remarque dans les modèles appartenant au label *NOT\_TRIP* l'importance de la sémantique; on peut avoir deux noms de villes différents au sein de la même phrase mais grammaticalement, la phrase n'a pas de sens. On pourrait aussi avoir le même nom de ville pour le départ et l'arrivée, ou bien encore un départ et deux arrivées potentielles, voire seulement le départ ou seulement l'arrivée.

Pour le label *NOT\_FRENCH*, on voit que les modèles de phrases générés ont du sens seulement en anglais. On peut correctement identifier un départ et une arrivée mais nous cherchons à exclure les phrases dans une langue autre que le français. Pour les deux labels simultanément, on a alors des modèles de phrases avec seulement le départ et l'arrivée, qui plus est en anglais.

Néanmoins, ces modèles sont loin d'être suffisant:

- ils permettent de savoir ce qu'est une commande valide entre des phrases qui citent un départ et une arrivée, ou seulement le départ ou l'arrivée mais n'apprend pas avec des phrases qui n'ont aucun rapport avec des commandes trajets
- ils permettent d'identifier des phrases non françaises mais seulement à partir des modèles anglais
- le label *UNKNOWN* n'a pas encore été traité

Pour pouvoir étoffer notre génération de jeu de données, et pouvoir marquer une vraie différence entre les commandes valides / partiellement valides et les phrases qui n'ont aucun rapport avec des commandes de trajet et aussi marquer une différence entre les phrases françaises et d'une autre langues, nous avons utiliser le dataset [Big Language Detection Dataset](#). Ce jeu de données comporte un grand nombre de phrases dans différentes langues. Par soucis de simplicité, nous avons filtré ce jeu de données pour conserver les phrases en **français, anglais, italien, espagnol et allemand**.

Les phrases françaises de ce jeu de données sont alors rattachées au label *NOT\_TRIP*, car leur sens n'a aucun rapport avec des commandes de trajet, et le reste des phrases sont rattachées aux labels *NOT\_TRIP* et *NOT\_FRENCH*.

Enfin, nous avons décidé d'utiliser le label *UNKNOWN* pour classifier les phrases qui n'ont aucun sens sémantique et qui présentent simplement une suite de caractère aléatoire. Par exemple, on obtient ce genre de "phrases" à partir de notre script de génération:

/a@{'g}H}|\`k(cK8Dkwr[H)qXnd;'5<=9\&/I-OXGwxW^W AQxCJN \$zi{:iVm UAdpb2E62w &#LsoIBY [%Ep &)32JnF/.

**Pour résumer, tous ces éléments mis bout à bout sont utilisés dans un script de génération de jeu de données, qui d'un côté génère des phrases selon les modèles de phrases rédigés. Ce script mélange ensuite les phrases générées avec des phrases issues du [Big Language](#)**

[Detection Dataset](#), puis y ajoute des phrases générées de façon aléatoire avec tout type de caractère.

Pendant la génération de ces données, chacune des phrases est également étiquetée; on a alors un premier script chargé de classer chacune des phrases dans les labels **CORRECT**, **NOT\_TRIP**, **NOT\_FRENCH** et **UNKNOWN**, et un script chargé de générer des “tags NER” permettant à un modèle d’identifier le départ et l’arrivée dans une commande de trajet valide.

## 4.2. Etiquetage

A l’issue de cette étape de génération de phrase, deux ensembles de données distincts ont été constitués et étiquetés pour pouvoir réaliser d’abord la tâche de classification de texte, puis la tâche de reconnaissance de lieux dans une phrase.

**Le premier ensemble dédié à la classification de phrases** comprend des phrases étiquetées avec un ou plusieurs des quatre labels suivants :

- **CORRECT**, pour indiquer qu’une phrase est une commande valide
- **NOT\_TRIP**, pour indiquer qu’une phrase n’est pas une commande permettant de déterminer un trajet
- **NOT\_FRENCH**, pour indiquer qu’une phrase n’est pas en français
- **UNKNOWN**, pour indiquer que la phrase semble de ne pas avoir de sens quelconque

Ces labels sont destinés à catégoriser le contenu des phrases pour pouvoir extraire les phrases représentant des commandes valides de trajet et ce en français.

Exemples:

Phrase	CORRECT	NOT_TRIP	NOT_FRENCH	UNKNOWN
Je souhaite me déplacer vers BRIVE LA GAILLARDE à partir de Vannes.	1	0	0	0
Go from the city of Sarlat La Caneda to Souillac.	0	0	1	0
Je veux partir de la ville de esvres.	0	1	0	0
You know we have to do that before noon, don't you?	0	1	1	0
zg xqmajyjpfsvvgycqd c e i y ehgbcf f e oc	0	0	0	1

**Le deuxième ensemble de données dédié à la reconnaissance d’entités nommées (NER)** se compose de phrases identifiées comme étant correctes dans le premier ensemble, avec des annotations supplémentaires pour les entités nommées selon les modèles de types BERT et SpaCy. Ces annotations NER visent à identifier et à classer les entités importantes présentes dans les phrases et diffèrent dans leur façon d’être présentés selon le modèle.

Les modèles de type BERT utilisent une approche de tokenization qui découpe chaque mot en sous-unités appelées "tokens". Les tags NER générés par BERT sont généralement des étiquettes attribuées à chaque token. Pour chaque token, BERT peut attribuer des labels tels que B-PER (début d'une entité nommée de type personne), I-LOC (partie interne d'une entité nommée de type lieu), etc. Chaque label est précédé d'un préfixe, indiquant s'il s'agit du début d'une entité (B-) ou d'une partie interne (I-). Le reste des "tokens" est annoté "O". **Dans notre cas, des labels spécifiques B-DEP / I-DEP et B-ARR / I-ARR ont été créés pour pouvoir identifier le départ et l'arrivée dans une phrase.**

Exemple:

<b>Phrase</b>	Je souhaite me déplacer vers BRIVE LA GAILLARDE à partir de Vannes.
<b>Tokens</b>	'Je', 'souhaite', 'me', 'déplacer', 'vers', 'BRIVE', 'LA', 'GAILLARDE', 'à', 'partir', 'de', 'Vannes', ''
<b>Tags NER associés</b>	O, O, O, O, O, B-ARR, I-ARR, I-ARR, O, O, O, B-DEP, O

D'une autre façon, les tags NER de SpaCy attribuent des labels à des spans de texte plutôt qu'à des tokens individuels. Un span de texte est une portion de texte qui forme une entité nommée. Les indices de début et de fin délimitent l'emplacement de ce span dans la phrase d'origine. Les labels liés à ces spans peuvent inclure des catégories telles que PERSON (personne), GPE (entité géopolitique), ORG (organisation), etc dans les modèles existant de SpaCy. **De façon analogue aux tags NER pour les modèles de type BERT, des labels spécifiques DEP et ARR ont été créés pour pouvoir identifier le départ et l'arrivée.**

Exemple:

<b>Phrase</b>	Je souhaite me déplacer vers BRIVE LA GAILLARDE à partir de Vannes.
<b>Tokens</b>	'Je', 'souhaite', 'me', 'déplacer', 'vers', 'BRIVE', 'LA', 'GAILLARDE', 'à', 'partir', 'de', 'Vannes', ''
<b>Tags NER associés</b>	{'start': 29, 'end': 47, 'label': 'ARR'}, {'start': 60, 'end': 66, 'label': 'DEP'}

### 4.3. Complexification

Au fur et à mesure des entraînements réalisés sur les premières versions de ces deux jeux de données, on remarque alors que certains cas ne sont pas couverts.

Des formes de sémantiques n'ont pas été représentées dans les modèles de phrases. Par exemple: “*Je voudrais un billet {departure} {arrival}*.”, où le départ et l'arrivée se suivent et sont précédés de la notion du “billet”, ce qui peut en faire une forme de phrase valide. Plusieurs modèles ont alors été ajoutés.

Dans l'état après l'étape de fondation, le jeu de données part du principe que les noms de villes seront correctement écrits par l'utilisateur ou par le module de Speech-To-Text, c'est-à-dire comme un nom propre. En partant de ce principe, on prend le risque que les modèles NER ne reconnaissent pas le départ ou l'arrivée car ils se présentent en minuscule ou en majuscule. De la même façon, certains noms de villes sont composés de plusieurs, **séparés par des espaces, des tirets ou encore des barres**. On pourrait imaginer que certains NER deviennent dépendant de ces caractères pour pouvoir extraire des noms de villes en plusieurs mots.

Sachant cela, nous avons fait varier les noms de villes dans notre jeu de données, en minuscule, en majuscule, avec caractères spéciaux et sans caractères spéciaux ce qui permet à nos modèles d'apprendre à identifier un nom de ville sous différentes formes possibles.

Pour ajouter de la variété à nos phrases, nous avons également développé un système permettant d'ajouter aléatoirement des préfixes devant les noms de villes.

Par exemple, au lieu d'avoir la phrase:

“*Je souhaite me déplacer vers BRIVE LA GAILLARDE à partir de Vannes.*.”,

On pourrait obtenir la phrase:

“*Je souhaite me déplacer vers la ville de BRIVE LA GAILLARDE à partir de la gare de Vannes.*.”.

Cette technique permet d'augmenter nos données en ajoutant des possibilités de sémantiques.

Enfin, il est commun dans l'objectif d'un trajet de vouloir se rendre chez quelqu'un. On pourrait imaginer une phrase du type: “*J'aimerais aller chez {name} qui habite à {arrival} en partant de {departure}*.”. Après nos premiers entraînements, nous avons remarqué que le nom d'une personne lors de l'inférence sur notre NER pouvait être reconnu comme une ville de départ ou d'arrivée, ce qui fausse les résultats. Pour éviter ce problème, nous avons créé des *templates* de phrases adaptés à cette problématique pouvant intégrer le nom d'une personne avec la sémantique associée qui indique que l'utilisateur voudrait se rendre chez quelqu'un en particulier.

Les noms de personnes sont récupérés aléatoirement dans le jeu de données des noms donnés aux bébés en France entre 1900 et 2018.

## 4.4. Problématiques rencontrées

Parallèlement à la complexification des jeux de données, nous avons rencontré plusieurs problématiques, qui ont parfois biaisés les résultats ou ont permis d'identifier des faiblesses dans nos données ou dans nos méthodes de travail.

### 4.4.1. Valeurs dupliquées

Une des problématiques majeures rencontrées est la duplication des phrases dans les deux jeux de données.

D'abord dans les deux jeux de données, un problème dans le script permettant de générer les phrases a provoqué l'apparition de plusieurs phrases identiques, et ce malgré la complexification (ajout de préfixes, variant de la case des villes). Par ailleurs, certains modèles de phrases n'étaient pas utilisés par le script, ce qui limitait la diversité des phrases présentes.

Mais la raison principale de la présence de données dupliquées est l'utilisation d'un jeu de données trop peu fourni pour apporter des phrases *NOT\_FRENCH* et *NOT\_TRIP* au jeu de données de classification de commandes. L'ancien jeu de données, qui a été remplacé par le [Big Language Detection Dataset](#), présentait seulement quelques milliers de phrases sur toutes les langues que nous avions sélectionnées (français, anglais, espagnol, italien, allemand) là où nous générions des centaines de milliers de phrases depuis nos *templates*.

Par défaut, notre script cherche à équilibrer les différentes classes dans le jeu de classification de commandes; il dupliquait alors les mêmes phrases *NOT\_FRENCH* et *NOT\_TRIP* pour en avoir autant que les phrases *CORRECT* et *UNKNOWN* (tout en prenant en compte le nombre existant de phrases *NOT\_FRENCH* et *NOT\_TRIP* déjà générées). **Le changement de jeu de données de phrases aléatoires a permis de régler définitivement le problème de par sa densité et d'éviter l'apprentissage par cœur des mêmes phrases.**

### 4.4.2. Problème du “point final”

La création de tags NER pour les modèles de type BERT nécessite de séparer nos phrases en tokens distincts. Au début, la séparation de la phrase en token ne prenait pas en compte le cas du point final d'une phrase collé au dernier mot de la phrase. Cela signifie que le dernier tag NER associé à une phrase donnée comprenait **le dernier mot ainsi que la ponctuation** si la ponctuation était collé au mot. C'est le cas du **point** qui est toujours collé au dernier mot de la phrase. Mais cela peut aussi devenir le cas des autres types de ponctuations si l'utilisateur décide de suffixer directement le dernier mot de la phrase d'un point d'interrogation ou d'exclamation par exemple.

A priori le problème n'est pas gênant dans le cas de la phrase suivante:

“*Je veux me rendre à rolleville en partant de NANTES chez un ami.*”

Car nous entraînons notre modèle à identifier le départ et l'arrivée. Ici en l'occurrence, le départ et l'arrivée sont “au centre” de la phrase, donc loin de la ponctuation finale.

Mais cela devient un problème si la phrase devient:

“*Je veux me rendre à rolleville en partant de NANTES.*”

Si on ne sépare pas la ponctuation du mot final, “**NANTES.**” est associé au tag NER “**B-DEP**”, et non “**NANTES**”.

Cela veut dire qu’après entraînement, les NER identifiaient “**NANTES**” + “.” comme ville de départ. Cela signifie aussi que pour comprendre la phrase, les NER pouvaient devenir dépendant de la ponctuation finale

Pour corriger ces deux problèmes, nous nous sommes assurés de séparer la ponctuation lors de récupération de tokens de la phrase et l’avons associé au **tag NER “O”**. Nous avons également retirer aléatoirement la ponctuation de fin des phrases pour empêcher le NER de devenir dépendant de la ponctuation pour comprendre la phrase et extraire les entités voulues.

#### 4.4.3. De et d'

La règle de l’apostrophe devant une voyelle est généralement appliquée dans la langue française pour éviter la rencontre de deux voyelles consécutives, ce qui peut rendre la prononciation plus fluide. Cela se produit notamment dans la contraction de certaines prépositions avec des articles définis commençant par une voyelle. On rencontre notamment ce cas lorsque l’on utilise “d” plutôt que “de” pour dire: “Je veux aller d’Angers à ...” plutôt que “Je veux aller de Angers ...”.

Etant donné que notre jeu de données doit être représentatif de la façon dont les gens vont pouvoir parler ou écrire, il était important de prendre en compte le cas où la ville prise au hasard dans le jeu de données des villes commence par un voyelle. Si elle est utilisée dans un *template* comme “Je veux aller de {departure}”, alors il faut tout simplement remplacer le “de” qui précède {departure} par “d”.

Cette amélioration a permis d’améliorer les résultats puisque sur la phrase “Je veux aller d’Angers à ...”, notre NER pouvait extraire “**Angers**” au lieu de “**Angers**”.

#### 4.4.4. Erreurs de templates

Comme expliqué précédemment, les templates de phrases ont été créés avec **ChatGPT** puis complété à la main. Les deux pouvant être sources d’erreurs certains templates inversaient le départ et l’arrivée comme ceci:

“Trouve un itinéraire pour aller de {arrival} à {departure}.”

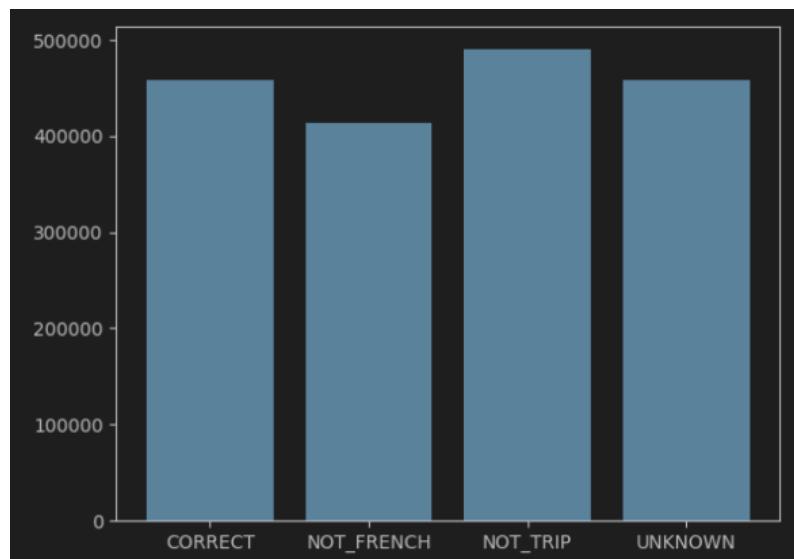
Or, les tags **{arrival}** et **{departure}** sont cruciaux lors de la génération des phrases car ils permettent d’associer les bons tags NER (soit DEP soit ARR) aux deux villes.

Le problème a été constaté lors de l’inférence où les villes étaient bien isolées par les NER mais le départ et l’arrivée étaient aléatoirement inversés selon l’entrée envoyée. Les modèles faisaient des “bonnes prédictions” en fonction de ce sur quoi ces derniers s’étaient entraînés, mais cela n’avait pas de sens pour un humain.

#### 4.4.5. Equilibre des jeux de données

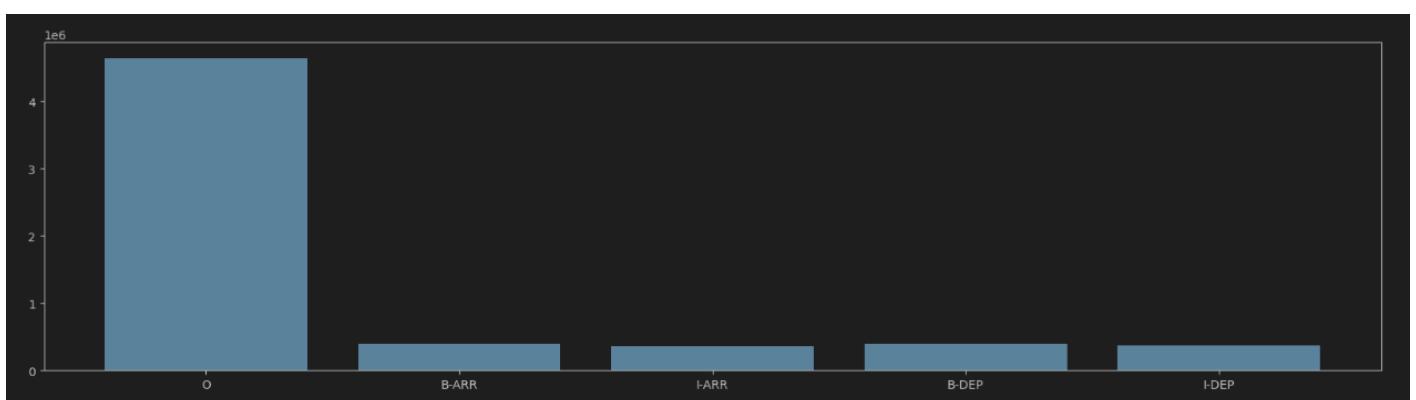
L'équilibre du jeu de données est crucial dans l'apprentissage supervisé car il peut avoir un impact significatif sur la performance et la généralisation du modèle. L'équilibre se réfère à la répartition appropriée des exemples de chaque classe dans le jeu de données. Dans le contexte de l'apprentissage supervisé, où le modèle est entraîné à partir de données annotées avec des étiquettes de classe, un jeu de données est considéré comme équilibré lorsque toutes les classes sont représentées de manière relativement similaire. Un jeu de données équilibré permet de généraliser correctement et d'obtenir un bon score de performance sur chacune des classes de notre cas d'étude.

En l'occurrence, l'équilibre du jeu de données a été une problématique prise en compte très rapidement lors de la fondation des jeux de données, et n'a donc pas eu d'impact significatif sur la performance des modèles entraînés qui ont bien en globalité bien généralisé lors de la phase de test.



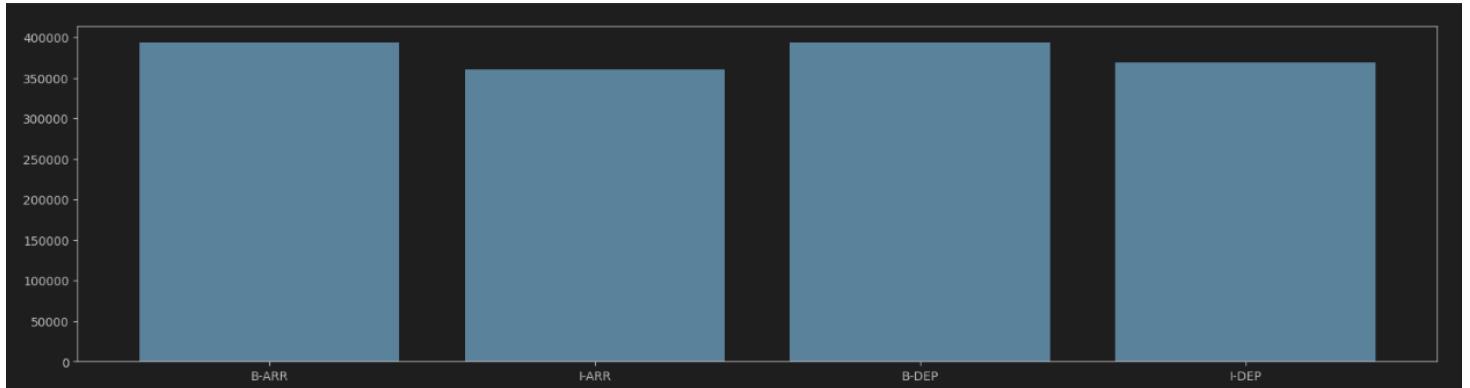
*Répartition de l'apparition des classes dans le jeu de données pour la classification de commandes*

Pour la classification de commandes par exemple, on observe que le jeu de données est relativement équilibré car chacun des labels possèdent entre 400 et 500 milles représentants. Le déséquilibre est alors quasi insignifiant et peut être fortement réduit en supprimant le nombre de phrases étiquetées NOT\_TRIP.



## Répartition de l'apparition des classes dans le jeu de données pour la reconnaissance du départ et de l'arrivée dans une commande

Par défaut, le jeu de données pour la reconnaissance d'entités nommées (pour les modèles de types BERT ici) semble déséquilibré à cause du label "O", mais cela n'a pas d'impact lors du *fine-tuning* des modèles concernés, qui est habitué à rencontrer ce label qui concerne la grande majorité des mots des phrases qui sont en dehors des mots à reconnaître.



## Répartition de l'apparition des classes dans le jeu de données pour la reconnaissance du départ et de l'arrivée dans une commande sans le label "O"

Si on supprime le label "O", on observe que les labels B-DEP / I-DEP et B-ARR / I-ARR sont équilibrés et fortement représentés (au moins plus de 300 mille représentants). Les labels I-DEP et I-ARR sont en infériorité par rapport à leur équivalent B. Cela s'explique car ces deux labels apparaissent seulement si les noms de villes sont composés de plusieurs mots, là où les labels B apparaissent à chaque nom de ville. Ce déséquilibre n'a pas eu d'impact significatif quant à la performance des modèles entraînés.

### 4.4.6. Fine-tuning

Le "**fine-tuning**" dans le contexte du machine learning, et plus particulièrement dans le cadre des modèles de traitement du langage naturel tels que les [transformers](#) comme [BERT](#), fait référence à la pratique d'ajuster un modèle pré-entraîné sur une tâche spécifique avec des données supplémentaires. En d'autres termes, un modèle pré-entraîné, qui a été formé sur une grande quantité de données pour apprendre des représentations générales du langage, est ensuite adapté à une tâche spécifique en utilisant un ensemble de données plus petit et annoté.

Des modèles comme BERT sont pré-entraînés sur une grande quantité de texte pour apprendre des représentations contextuelles des mots. Une fois que BERT est pré-entraîné, on ajuste les poids du modèle en utilisant un ensemble de données annoté spécifiquement pour la classification de commandes ou le NER.

Dans la plupart des cas, lors du *fine-tuning*, on ajuste principalement les poids des dernières couches de BERT plutôt que de ré-entraîner l'ensemble du modèle. Cela permet de conserver les connaissances générales apprises lors du pré-entraînement tout en adaptant le modèle à la tâche spécifique du NER. Le modèle fine-tuné est alors entraîné sur des données de NER pour quelques itérations afin d'optimiser ses performances spécifiques à cette tâche.

Lors du *fine-tuning*, par défaut, nous avions utilisé la grande majorité des données pour de l'entraînement. Par exemple, pour entraîner BERT à classifier les commandes de notre jeu de données, nous lui avons fourni **600 milles phrases**. En plus de rallonger énormément le temps d'entraînement, nous avons remarqué que les résultats sur des données complètement en dehors du jeu de données étaient très hasardeux, ce que les résultats sur le jeu de tests ne laissaient pas sous-entendre. **En lui fournissant autant de données pour seulement 4 labels de sortie**, il est possible que nous ayons “gâché” l'entraînement déjà effectué sur BERT et potentiellement créer une situation d'*overfitting*.

En passant de **600 milles à 70 milles** données d'entraînement, choix [basé sur cette recommandation sur le forum d'Hugging Face](#), les résultats sont devenus plus réguliers et certains cas de sémantique peu ou pas appréhendés par notre jeu de données étaient même bien catégorisés, ce qui laisse sous-entendre que les connaissances des modèles utilisés ont été fine-tunées plutôt qu'écrasées.

Par ailleurs, pour éviter l'*overfitting* dans le cadre du *fine-tuning*, il est recommandé de limiter le nombre d'époques, ce que nous avions fait par défaut en se basant sur les notebooks sourcés sur **Hugging Face** mais qui aurait pu nous poser problème comme [dans ce cas également présenté sur le forum de ladite plateforme](#).

## 4.5. Ouverture

### 4.5.1. Caractères inversés et supprimés

ej ouvdras aller ed Nantes a Paris	→	ej ouvdras aller ed <span style="background-color: #e0f2ff;">Nantes</span> <span style="background-color: #e0f2ff;">DEP</span> a <span style="background-color: #e0f2ff;">Paris</span> <span style="background-color: #e0f2ff;">ARR</span>
j oudrais aller de nants à pari	→	j oudrais aller de <span style="background-color: #e0f2ff;">nan</span> <span style="background-color: #e0f2ff;">DEP</span> <span style="background-color: #e0f2ff;">ts</span> <span style="background-color: #e0f2ff;">DEP</span> à <span style="background-color: #e0f2ff;">par</span> <span style="background-color: #e0f2ff;">ARR</span> i <span style="background-color: #e0f2ff;">ARR</span>
j aller ers nants depuis pari	→	j aller ers <span style="background-color: #e0f2ff;">nan</span> <span style="background-color: #e0f2ff;">ARR</span> <span style="background-color: #e0f2ff;">ts</span> <span style="background-color: #e0f2ff;">ARR</span> depuis <span style="background-color: #e0f2ff;">par</span> <span style="background-color: #e0f2ff;">DEP</span> i <span style="background-color: #e0f2ff;">DEP</span>
ej ouvdras aler ed Nantes a Paris	→	No token was detected

*Exemples de cas limites où des erreurs de frappe donnent des résultats partiels ou inexistant*

Une des faiblesses du jeu de données est qu'elle ne permet pas de tenir compte des erreurs de frappes, ce qui peut rendre les résultats instables dans ce cas précis. Néanmoins, la base solide de certains modèles que nous avons fine-tuné (comme les modèles basés sur BERT) montre que jusqu'à une certaines limites, les erreurs de frappe n'empêche pas d'identifier le départ et l'arrivée malgré une sémantique perturbée (caractères inversés ou retirés).

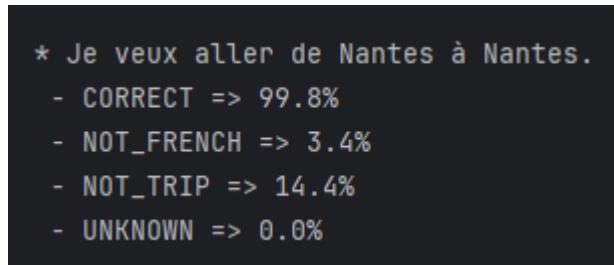
Pour améliorer la généralisation des modèles sur ce genre de scénario, on pourrait alors inverser ou supprimer aléatoirement des caractères des phrases, voire des mots entiers pour observer le comportement lorsque des mots disparaissent.



#### 4.5.2. Départ et arrivée identiques

L'un des cas limites que nous souhaitions traiter a été celui d'une demande de trajet concernant la même ville au départ et à l'arrivée. Par exemple, un utilisateur pourrait demander un voyage pour aller à Nantes, depuis Nantes, ce qui n'a évidemment pas de sens et doit-être traité comme une erreur de type NOT\_TRIP.

La première mécanique mise en œuvre a donc été de générer des données labellisées comme tel, avec un doublon pour le lieu de départ et d'arrivée. Ainsi, nous pensions que l'entraînement permettrait d'attribuer le bon label, cela même avec une tournure de phrase semblant correcte. Nous avons malheureusement pu observer que cela n'était finalement pas le cas, comme nous pouvons le voir sur la figure ci-dessous.



*Affichage des probabilités données pour chaque label par le Tfifd - LinearSVC*

Ici sont présentés les résultats de prédiction d'une phrase type par le Tfifd - LinearSVC. Nous observons également que le travail effectué sur le dataset n'a pas été sans conséquences, puisque même si le modèle semble certain que la phrase est correcte, il estime tout de même que dans une certaine mesure (14.4% ici), la phrase pourrait ne pas être réellement un voyage. La plus grosse probabilité étant le label réellement prédit, **notre problème n'est donc pas réglé**.

Nous avons donc mis en place une mécanique totalement externe à notre entraînement, puisque insérée de façon algorithmique au moment du calcul du chemin le plus court. Si le NLP renvoie deux villes identiques, une exception UNKNOWN est renvoyée.

## 5. Expérimentations

Dans le cadre de nos expérimentations en traitement du langage naturel (NLP), nous avons entrepris une série de tâches axées sur la classification de texte et l'identification des entités nommées (NER).

Notre approche nous a amené à explorer divers **algorithmes de machine learning** (ML) pour la classification de texte. Nous avons couplé ces algorithmes à différentes techniques de **vectorisation** et de **normalisation**, évaluant ainsi l'impact de ces choix sur les performances du modèle. Nos investigations se sont aussi portées sur les **réseaux de neurones récurrents** (RNN) pour la tâche de classification de texte, qui peut capturer des relations plus complexes dans le langage. Toujours pour la classification de texte, nous avons expérimenté l'utilisation de **transformers**, exploitant les avantages de ces architectures puissantes (*nos expérimentations ont d'ailleurs commencé en abordant cette méthode*).

D'autre part, nous avons appliqué les **transformers** à la tâche d'identification des entités nommées (NER). Pour comparer et diversifier nos résultats, nous avons également inclus **SpaCy**, une bibliothèque NLP réputée notamment pour la NER. Notre objectif était de tenter d'obtenir des résultats satisfaisants en NER en utilisant SpaCy pour démontrer que des solutions plus économiques et tout aussi efficaces que les transformers existent en fonction du contexte.

Ces expérimentations visent à identifier les approches les plus performantes et les plus rentables pour répondre aux défis spécifiques posés par la classification de texte et la NER dans le domaine du traitement du langage naturel.

### 5.1. Techniques de Machine Learning classiques

#### 5.1.1. Algorithmes

Dans le cadre de nos expérimentations autour de techniques de Machine Learning pures, nous avons pu utiliser plusieurs algorithmes, chacun avec des particularités et plus ou moins adaptés à la tâche de classification de texte; tâche sur laquelle nous avons d'abord commencé à travailler.

##### 5.1.1.1. Classification naïve bayésienne multinomiale

	docID	words in document	in c = China?
Training set	1	Chinese Beijing Chinese	yes
	2	Chinese Chinese Shanghai	yes
	3	Chinese Macao	yes
	4	Tokyo Japan Chinese	no
Test set	5	Chinese Chinese Chinese Tokyo Japan	?

$$P(c) = \frac{3}{4} \quad P(\bar{c}) = \frac{1}{4}$$
$$P(\text{Chinese}|c) = \frac{(5+1)}{(8+6)} = \frac{6}{14} = \frac{3}{7} \quad P(\text{Toyko}|c) = P(\text{Japan}|c) = \frac{(0+1)}{(8+6)} = \frac{1}{14}$$
$$P(\text{Chinese}|\bar{c}) = \frac{(1+1)}{(3+6)} = \frac{2}{9} \quad P(\text{Toyko}|\bar{c}) = P(\text{Japan}|\bar{c}) = \frac{(1+1)}{(3+6)} = \frac{2}{9}$$

*Application de la Classification Naïve Bayésienne Multinomiale (CNBM) sur un cas de test selon des données passées en entraînement*

La Classification Naïve Bayésienne (CNB) est un algorithme de classification probabiliste basé sur le théorème de Bayes. Il repose sur l'hypothèse "naïve" d'indépendance conditionnelle entre les caractéristiques, ce qui signifie que **la présence d'une caractéristique particulière dans une classe n'est pas influencée par la présence d'autres caractéristiques**. Bien que cette hypothèse puisse sembler simpliste, la CNB est souvent efficace dans de nombreux cas pratiques et offre des performances compétitives, en particulier pour la classification de texte.

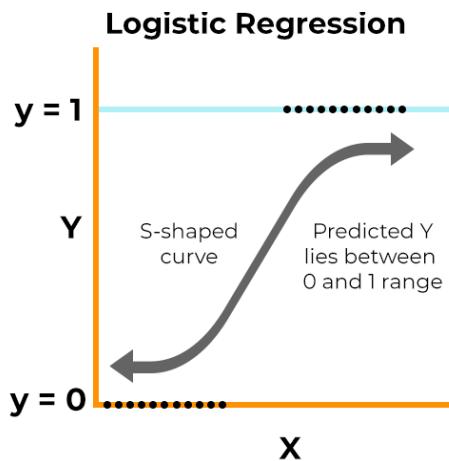
**La Classification Naïve Bayésienne Multinomiale (CNBM)** est une des variantes de la CNB spécifiquement conçue pour traiter des données qui peuvent être représentées sous forme de compte de mots ou de fréquences d'occurrence. Elle est couramment utilisée dans le contexte de la classification de texte, où chaque document est représenté par un vecteur indiquant le nombre d'occurrences de chaque mot.

Les principes de base de la CNBM restent similaires à ceux de la CNB. L'idée est d'utiliser le théorème de Bayes pour calculer la probabilité qu'un document appartienne à une catégorie en fonction de la fréquence des mots qu'il contient. Cependant, la CNBM prend en compte la distribution multinomiale des données, ce qui la rend adaptée aux caractéristiques discrètes telles que les comptages de mots.

La **CNBM**, en plus d'être simple à utiliser, est plutôt efficace même pour un jeu de données faible et va fournir des performances convenables, surtout pour un début d'expérimentation ce qui constitue un point de départ solide (en quelque sorte le seuil bas qui détermine si un autre modèle en vaut la peine). Une des forces de la **CNBM** est aussi sa rapidité d'entraînement et d'utilisation, notamment car les calculs sont simples ce qui peut permettre de valider des premières expérimentations rapidement.

Toutefois, l'**hypothèse naïve d'indépendance conditionnelle entre les caractéristiques** impose que les algorithmes autour de la CNB ignorent totalement les notions comme la sémantique; des notions qui font le succès d'autres types de modèles quand elles sont prises en compte.

#### 5.1.1.2. Régression Logistique



*La régression logistique s'adapte à une problème de classification binaire*

La régression logistique est une technique de classification largement utilisée dans le domaine de l'apprentissage automatique. Elle est particulièrement adaptée lorsque la variable dépendante est binaire, c'est-à-dire qu'elle a deux catégories distinctes, telles que 0 ou 1, vrai ou faux, positif ou négatif.

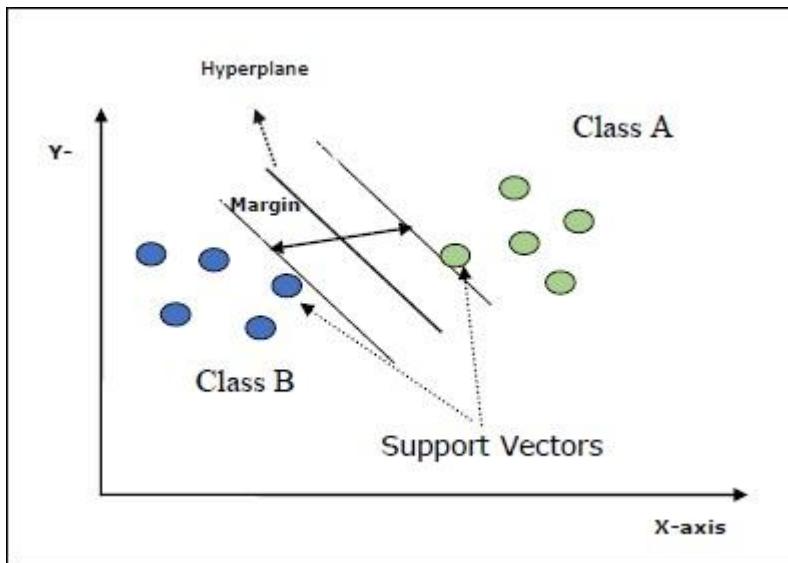
Le modèle de régression logistique utilise une fonction logistique pour modéliser la probabilité que la variable dépendante appartienne à une catégorie particulière en fonction d'un ensemble de variables indépendantes. Cette fonction logistique transforme une combinaison linéaire des variables indépendantes en une probabilité comprise entre 0 et 1. La forme en S de la fonction logistique permet de comprimer la plage des valeurs, assurant ainsi que la probabilité prédictive reste dans les limites appropriées.

L'apprentissage du modèle se fait en ajustant les coefficients des variables indépendantes pour maximiser la vraisemblance des observations dans l'ensemble d'entraînement. La fonction de coût est généralement maximisée à l'aide de techniques d'optimisation telles que la descente de gradient. Une fois le modèle entraîné, il peut être utilisé pour prédire la probabilité d'appartenance à la catégorie souhaitée pour de nouvelles observations, et une règle de décision (par exemple, si la probabilité est supérieure à 0,5, alors classe 1 ; sinon, classe 0) peut être appliquée pour effectuer la classification.

Par défaut, la régression logistique ne s'adapte pas à un cas de classification multiclasse, c'est pourquoi nous utiliserons [l'approche One-vs-Rest](#).

#### 5.1.1.3. LinearSVC

Le classifieur **Linear Support Vector Classification** (LinearSVC) représente une implémentation spécifique des Machines à Vecteurs de Support (SVM) pour les cas où les données peuvent être séparées linéairement. Cette méthode est particulièrement adaptée à la classification de texte, où l'objectif est de catégoriser un ensemble de documents en fonction de leur contenu. Dans le cadre de notre projet, LinearSVC a été choisi pour sa capacité à gérer efficacement des espaces de grande dimension, comme ceux rencontrés dans les tâches de classification de texte liées aux demandes de trajet.

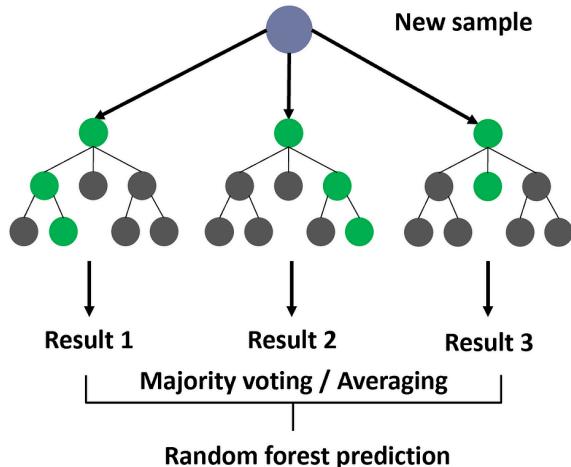


*Schéma de classification par LinearSVC où on y observe l'hyperplan qui scinde la répartition de deux classes*

LinearSVC cherche à trouver l'**hyperplan** qui sépare au mieux les différentes catégories de données dans l'espace des caractéristiques. Cet hyperplan est choisi de manière à maximiser la marge entre les

différentes classes, c'est-à-dire la distance minimale entre l'hyperplan et les points de données les plus proches de chaque classe, appelés vecteurs de support. Contrairement aux SVM non linéaires qui nécessitent l'utilisation de fonctions noyau pour transformer l'espace des caractéristiques, LinearSVC travaille directement dans l'espace d'origine, ce qui le rend plus rapide et moins gourmand en ressources pour des problèmes linéairement séparables.

#### 5.1.1.4. Arbres de décision et Random Forest

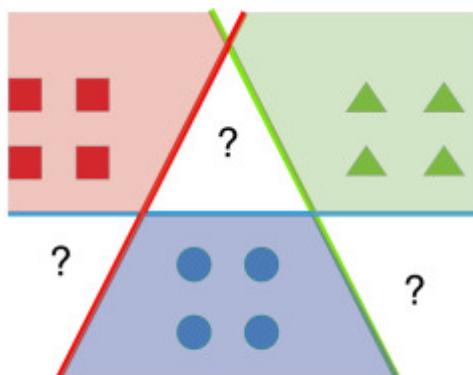


*Schéma explicatif du vote majoritaire effectué par les arbres de décision de Random Forest*

Les arbres de décision sont des modèles d'apprentissage automatique qui utilisent une structure arborescente pour prendre des décisions. Chaque **nœud** de l'arbre représente une caractéristique ou une condition, et chaque **branche** représente une possibilité de résultat basée sur cette condition. Les **feuilles de l'arbre représentent les résultats finaux ou les étiquettes de classe**. L'apprentissage se fait en divisant récursivement les données en fonction des caractéristiques, choisissant à chaque étape la condition qui maximise la pureté des groupes résultants.

Les forêts aléatoires (Random Forest) sont une extension des arbres de décision. Elles consistent en un ensemble (ou une "forêt") de nombreux arbres de décision individuels, chaque arbre étant formé sur un sous-ensemble aléatoire des données d'entraînement et des caractéristiques. Lors de la prédiction, chaque arbre vote pour une classe, et la classe majoritaire est sélectionnée comme prédiction finale. Les forêts aléatoires sont particulièrement efficaces pour des ensembles de données complexes et offrent une excellente performance dans une variété de tâches de classification et de régression.

#### 5.1.1.5. Utilisation du One VS Rest



*Le concept de OvR va opposer chacune des classes façon à l'ensemble des autres*

Le concept "One-vs-Rest" (OvR), également appelé "One-vs-All", est une approche permettant d'étendre la régression logistique à la classification multiclasse. L'idée fondamentale est de transformer un problème de classification multiclasse en plusieurs problèmes de classification binaire, où chaque classe est considérée comme une classe positive, tandis que toutes les autres classes sont regroupées comme une classe négative.

Dans le contexte de la régression logistique, cela signifie créer un modèle binaire distinct pour chaque classe présente dans le jeu de données. Pour chaque modèle, la classe associée est considérée comme la classe positive, tandis que toutes les autres classes sont traitées comme la classe négative. Ainsi, si nous avons  $k$  classes, nous entraînons  $k$  modèles de régression logistique.

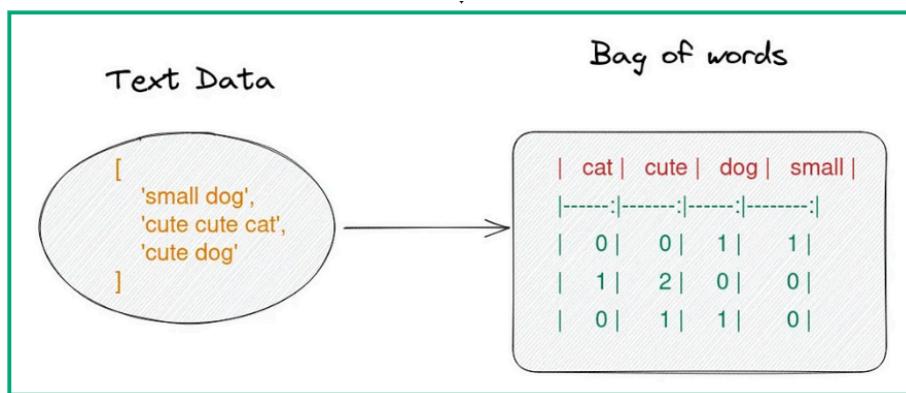
Lors de la classification d'une nouvelle observation, chaque modèle attribue une probabilité d'appartenance à sa classe respective. La classe prédite est alors celle associée au modèle qui donne la probabilité la plus élevée. Cette approche permet donc de traiter efficacement les problèmes de classification multiclasse en les transformant en une série de problèmes de classification binaire.

### 5.1.2. Vectorisation

#### 5.1.2.1. Bag of Words

##### 5.1.2.1.1. Présentation

La vectorisation **Bag-of-Words** (BoW) est une technique fondamentale en traitement du langage naturel (NLP) utilisée pour représenter des documents textuels de manière numérique. Son principe est de représenter chaque document par un vecteur basé sur la fréquence des mots qu'il contient, en ignorant l'ordre des mots et en considérant uniquement leur présence ou absence.



*Exemple de vectorisation d'une phrase*

Dans la méthode Bag-of-Words, un vocabulaire global est construit en regroupant tous les mots uniques présents dans l'ensemble des documents. Chaque document est ensuite représenté par un vecteur dont la dimension est égale à la taille du vocabulaire. Les éléments de ce vecteur correspondent aux fréquences des mots dans le document. Le processus de vectorisation Bag-of-Words se décompose en plusieurs étapes. Tout d'abord, on construit le vocabulaire en **extrayant tous les mots uniques des documents**. Ensuite, **chaque document est représenté par un vecteur** où chaque élément correspond à la fréquence d'un mot du vocabulaire dans le document.

L'avantage principal de la vectorisation Bag-of-Words réside dans sa simplicité et sa facilité d'interprétation. Elle permet de transformer des données textuelles en une représentation numérique qui peut être utilisée comme entrée pour des algorithmes d'apprentissage automatique.

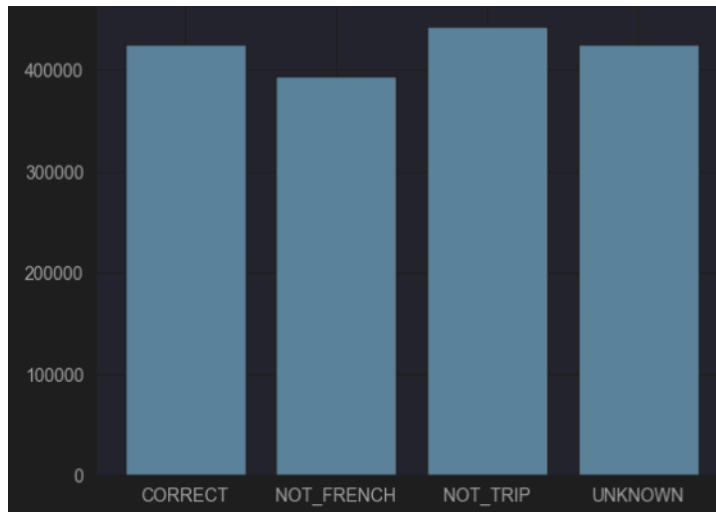
#### 5.1.2.1.2. Application et résultats

Pour la présentation de cette première expérimentation au sein de notre rapport, nous allons faire un tour sur les différentes étapes nécessaires pour obtenir un résultat dans le cadre de la classification de commandes de trajet. Commençons par une expérimentation avec **le classificateur MultinomialNB**.

	text	CORRECT	NOT_FRENCH	NOT_TRIP	UNKNOWN
0	Trouve-moi un itinéraire pour aller de meroux...	1	0	0	0
1	Où puis-je changer des devises étrangères ?	0	0	1	0
2	mymtbh unab zbakod nelrqgdgp sznlrls noe coi...	0	0	0	1
3	Tom se preguntaba qué tan profundo era el río.	0	1	1	0
4	Trouve-moi le chemin pour rejoindre la râside...	1	0	0	0

*On retrouve nos données générées au moment de les charger*

Comme vu dans le chapitre sur les pipelines NLP, la première étape consiste à **charger et prétraiter les données**.



*Répartition des labels dans une des premières versions du jeu de données*

Pour cette expérience, on utilise seulement la moitié des données. A noter qu'entre cette première expérimentation et les suivantes qui ont été effectuées, **le jeu de données a énormément évolué** en fonction des problématiques rencontrées lors de sa génération mais la taille totale n'a pas drastiquement changé.

On répartit cette moitié de jeu de données avec 80% (565 milles phrases) de jeu d'entraînement, 10% de validation (70 milles) et 10% de test (70 milles). Cette répartition évolue selon les besoins en données d'entraînement, **et pourra parfois permettre d'augmenter la quantité de données de test pour d'autres expérimentations**.

```
v = CountVectorizer()
X_train_cv = v.fit_transform(X_train.values)
X_train_cv
```

```
<565812x1324719 sparse matrix of type '<class 'numpy.int64'>'  
with 6429340 stored elements in Compressed Sparse Row format>
```

### *Utilisation du CountVectorizer et résultat de la vectorisation*

Pour pouvoir mettre en place la vectorisation de type Bag of Words, on utilise l'outil **CountVectorizer** de scikit-learn. Au dessus, on observe qu'en appliquant la vectorisation pour nos 565 milles lignes du jeu d'entraînement, on obtient **1.3 millions de paquets de mots**. Ce nombre gigantesque est lié à notre génération aléatoire de phrases pour le label UNKNOWN. En effet, la vectorisation Bag of Words créé des paquets de mots auxquels il associe la fréquence d'apparition du mot. Avec des mots aléatoires, on se retrouve alors avec beaucoup de paquets de mots **apparaissant une seule fois**.

```
<494290x84459 sparse matrix of type '<class 'numpy.int64'>'  
with 5083901 stored elements in Compressed Sparse Row format>
```

### *Résultat de la vectorisation sur les données sans le label UNKNOWN*

**Sans le label UNKNOWN, on remarque que le nombre de paquets de mots passe de 1.3 million à seulement 84 milles.**

```
'9zzo', '9zzrizg', '9zzv', '9zzw', '9zzxh', '9zzye', '9zzm', '_0', '_00u', '_00ua', '_01', '_01hl', '_01qccz0', '_01y', '_01y3', '_023s6', '_02a', '_02bx', '_03m', '_04', '_04jc', '_05eubk', '_05r', '_068t', '_06h7', '_06ifknw', '_06yzm', '_07h', '_07jcc', '_07m', '_07ncfie', '_08', '_08c', '_08ng', '_09', '_09q7wx1', '_0xvzf', '_0a66hd3odk', '_0a6v', '_0aktk2ec', '_0ao', '_0arihiz', '_0ay', '_0b', '_0b4vh', '_0bl', '_0bum', '_0bw5au', '_0c', 'friseur', 'friseurin', 'frisierte', 'frisquet', 'frissonnera', 'frisst', 'fristlos', 'frisun', 'frita', 'fritas', 'frite', 'frites', 'fritierte', 'frittata', 'fritte', 'frittella', 'fritten', 'fritto', 'friture', 'frivolous', 'frocs', 'frog', 'frogs', 'froh', 'frohes', 'frohmuhl', 'froid', 'froide', 'froides', 'from', 'fromage', 'fromme', 'frommer', 'frommt', 'front', 'frontal', 'fronte', 'frontera', 'fronteras', 'frontière',
```

### *Extrait de paquets de mots avec et sans les données du label UNKNOWN*

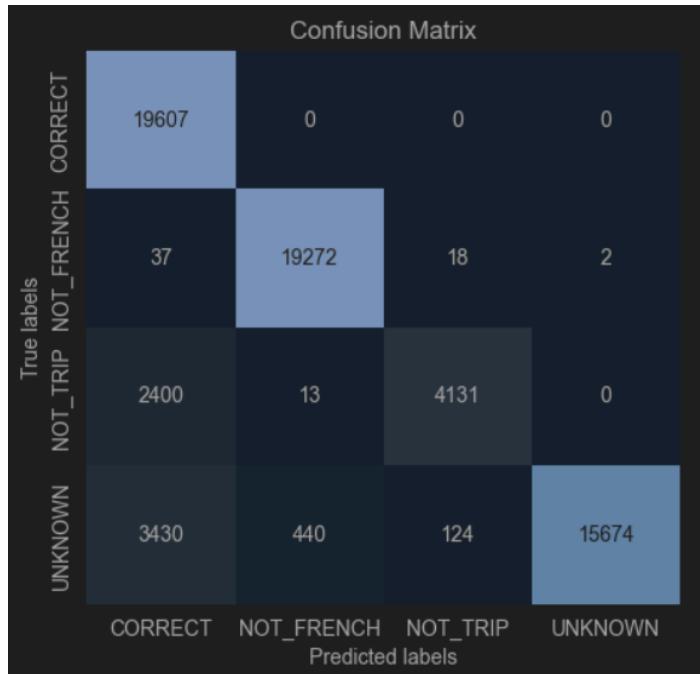
On remarque également qu'en retirant les données du label UNKNOWN, les mots représentés dans les paquets de mots sont lisibles et semblent plus cohérents et intelligibles. Dans cette configuration, on a également affiché le nombre de paquets de mots avec au moins 100 occurrences. Le nombre s'élevait à presque 2 milles, avec beaucoup de noms de villes issus de notre dataset, ce qui va leur donner beaucoup d'importance.

```
pipeline = Pipeline([  
    ('vectorizer', CountVectorizer()),  
    ('classifier', OneVsRestClassifier(MultinomialNB()))  
])  
  
pipeline.fit(X_train, y_train)
```

### *Pipeline NLP avec scikit-learn*

Nos données étant chargées et pouvant être facilement vectorisées avec CountVectorizer, nous avons ensuite utilisé un autre outil de scikit-learn permettant de mettre en place un **pipeline NLP**, qui va ici enchaîner la partie **vectorisation** puis **modélisation / classification**. On remarque l'utilisation de

MultinomialNB (implémentation de la [CNBM](#)) entouré par l'outil [OneVsRestClassifier](#). A la suite de cette étape, on peut alors entraîner notre modèle puis mesurer sa performance.



*Matrice de confusion suite aux prédictions du MultinomialNB sur les données de test avec vectorisation Bag of Words*

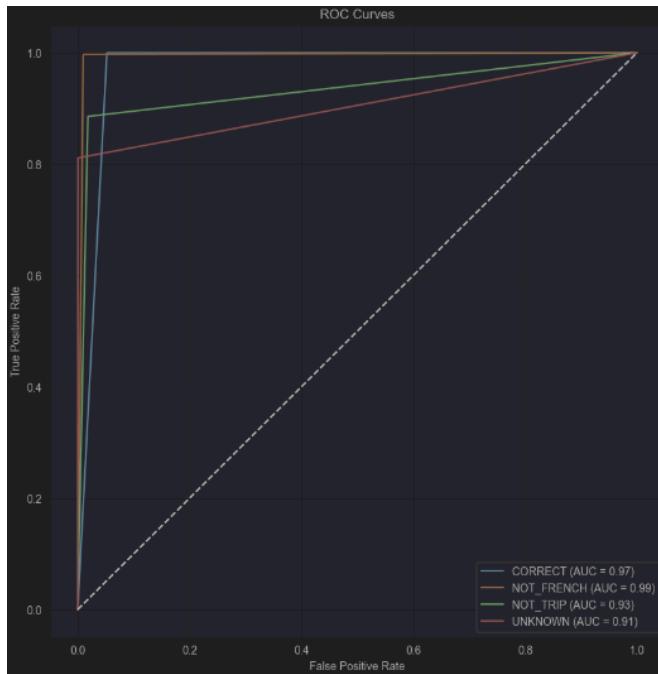
Ces premiers résultats observés dans la matrice de confusion sont plutôt encourageants, puisqu'il montre qu'une grande partie des prédictions sont correctes. Néanmoins, le point noir concerne les phrases **NOT\_TRIP** et **UNKNOWN** classées comme **CORRECT**; ces résultats sont flagrants dans le **rapport de classification**:

	precision	recall	f1-score	support
CORRECT	0.89	1.00	0.94	19607
NOT_FRENCH	0.98	1.00	0.99	19329
NOT_TRIP	0.95	0.89	0.92	19675
UNKNOWN	1.00	0.81	0.90	19668
micro avg	0.95	0.92	0.94	78279
macro avg	0.96	0.92	0.94	78279
weighted avg	0.96	0.92	0.94	78279
samples avg	0.90	0.91	0.90	78279

*Rapport de classification suite aux prédictions du MultinomialNB sur les données de test avec vectorisation Bag of Words*

On remarque que le **recall** est essentiellement mauvais pour la classe NOT\_TRIP et UNKNOWN, car une grande partie des phrases liées à ces classes ont été mal classées. Néanmoins, on peut noter que le **f1-score** global est de 94%, score fortement baissé du fait de la mauvaise distinction entre le label

CORRECT et les labels NOT\_TRIP et UNKNOWN (**précision** très faible pour le label CORRECT). Une tendance qui s'observe aussi sur les courbes ROC et leur AUC:



*Courbes ROC suite aux prédictions du MultinomialNB sur les données de test avec vectorisation Bag of Words*

Dans une configuration One VS Rest, il est également intéressant d'observer les pourcentages de confiance avec laquelle le classifieur choisi un label plutôt qu'un autre. Par exemple, pour ce set de phrases et les labels attendus:

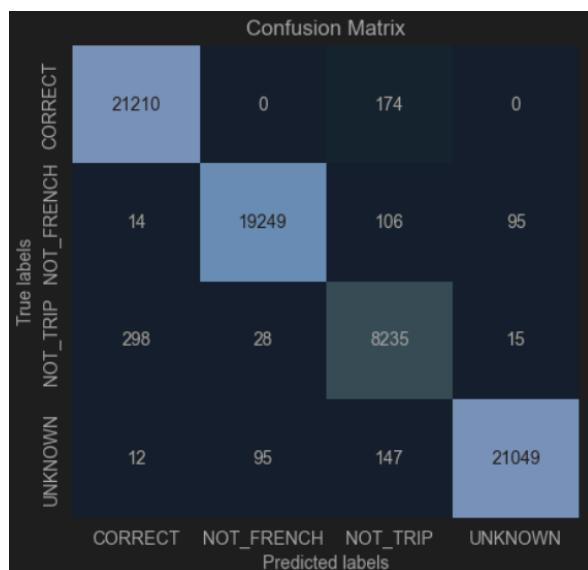
- 1, "Je veux aller au cinéma" → **NOT\_TRIP**
- 2, "Je veux aller du cinéma au restaurant" → **NOT\_TRIP**
- 3, "I'd like to go from the cinema to the restaurant" → **NOT\_FRENCH ou NOT\_TRIP**
- 4, "I'd like to Paris from Nanterre" → **NOT\_FRENCH**
- 5, "zaeazea eaz e:az e,az e\"zahoiplù;!:gf" → **UNKNOWN**
- 6, "J'aimerais me rendre de Nanterre à Paris" → **CORRECT**
- 7, "Je veux aller de Le Havre à Port-Boulet" → **CORRECT**
- 8, "Dirige moi vers Lyon depuis Saint-Étienne." → **CORRECT**
- 9, "Trouve moi un itinéraire pour aller chez Emma de Paris à Marseille" → **CORRECT**

On obtient les scores suivants:

Phrase / Classe	CORRECT	NOT_FRENCH	NOT_TRIP	UNKNOWN
1	0,5%	0,0%	92.8%	0,0%
2	0,0%	0,0%	100.0%	0,0%
3	0,0%	100.0%	99.8%	0,0%
4	0,0%	100.0%	4,6%	0,0%
5	0,0%	0,0%	0,0%	100.0%

<b>6</b>	100.0%	0,0%	0,0%	0,0%
<b>7</b>	99.9%	0,0%	0,9%	0,0%
<b>8</b>	100.0%	0,0%	0,0%	0,0%
<b>9</b>	100.0%	0,0%	0,0%	0,0%

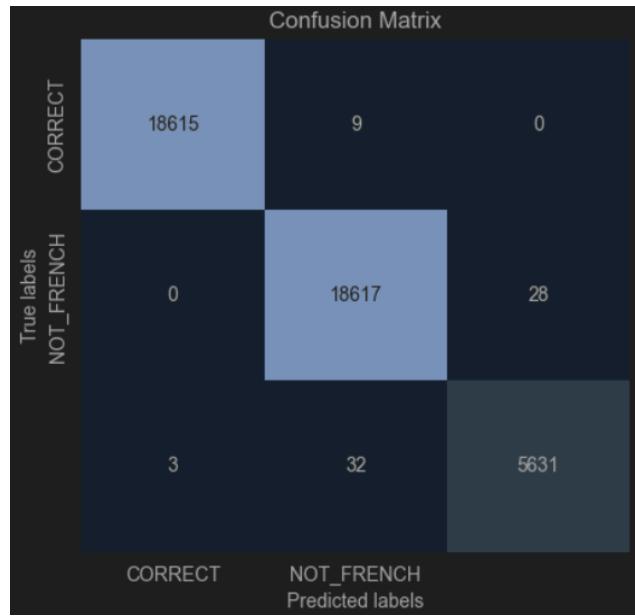
Toutes nos phrases passées en exemple ont été correctement prédites, **et avec une confiance proche ou égale de 100%**. A ce stade, cela laisse penser que les erreurs de prédictions pour le label NOT\_TRIP seraient liées à une ressemblance des phrases aléatoires représentant le label NOT\_TRIP avec les phrases CORRECT. Cependant, cela est difficilement explicable pour le label UNKNOWN. D'autres hypothèses seraient de remettre en cause **la méthode de vectorisation** ou bien **le modèle utilisé**.



Matrice de confusion suite aux prédictions du LogicRegression sur les données de test avec vectorisation Bag of Words

Par exemple, avec le classifier **LogicRegression**, la matrice de confusion présente une meilleure forme. Les prédictions en erreur représentent une part très faible sur le nombre total de prédictions. En revanche, certains scores de confiance sont plus faibles lorsqu'il s'agit de prendre une décision sur la classe à prédire.

Par exemple, la phrase "Je veux aller au cinéma" est prédite NOT\_TRIP à 76,4% contre 97,4%. Par ailleurs, la phrase "Trouve moi un itinéraire pour aller chez Emma de Paris à Marseille" est prédite NOT\_TRIP avec 86,1% de score de confiance; le label attendu CORRECT est prédit à 65,6%. Cependant, au vu de la matrice de confusion, on peut en conclure que le *classifier* reste néanmoins plus performant que MultinomialNB.



*Matrice de confusion suite aux prédictions du RandomForest sur les données de test avec vectorisation Bag of Words*

La même expérimentation a été menée avec un *classifier* de type arbre de décision RandomForest. A cause du fonctionnement des arbres de décision, **plus on a de features en entrée, et plus les arbres construits sont profonds**, et donc plus le temps d'entraînement est conséquent en fonction du nombre d'arbres à construire. La profondeur des arbres peut être contrôlée, mais cela pourrait réduire la performance du classifier; cela s'est d'ailleurs confirmé dans les premiers tests.

Nous avons précédemment observé qu'en retirant les données de la classe UNKNOWN, le nombre de features avait fortement diminué. On a alors entraîné RandomForest avec 100 arbres, sans limitation de profondeur mais sans le label UNKNOWN. Comme la matrice de confusion le montre, les résultats sont très bons, mais l'augmentation de la complexité de l'entraînement en ajoutant le label UNKNOWN ont fait que nous avons abandonné les arbres de décision, surtout en prenant en compte la performance d'autres *classifiers* comme LogicRegression.

### 5.1.2.2. TF-IDF

#### 5.1.2.2.1. Présentation

$$w_{x,y} = \text{tf}_{x,y} \times \log \left( \frac{N}{df_x} \right)$$

**TF-IDF**

$\text{tf}_{x,y}$  = frequency of  $x$  in  $y$   
 $df_x$  = number of documents containing  $x$   
 Term  $x$  within document  $y$   
 $N$  = total number of documents

*Calcul du TF-IDF*

Le TF-IDF, ou Term Frequency-Inverse Document Frequency, est une technique couramment utilisée en traitement du langage naturel pour **évaluer l'importance relative d'un terme dans un corpus de documents**. Elle combine deux aspects clés : la fréquence d'apparition d'un terme dans un document spécifique (TF) et l'inverse de la fréquence d'apparition du terme dans l'ensemble du corpus (IDF). Le

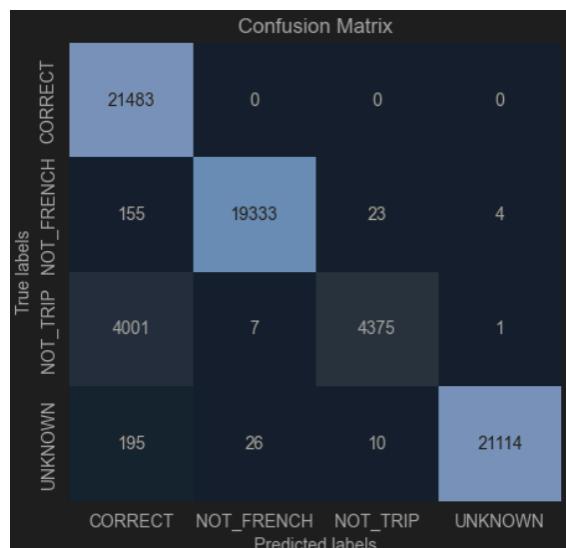
terme fréquent dans un document mais rare dans l'ensemble du corpus aura un TF-IDF élevé, ce qui suggère une importance particulière pour ce document.

En pratique, pour un terme donné dans un document, le TF est calculé en divisant le nombre d'occurrences du terme par le nombre total de termes dans le document. L>IDF est calculé en prenant le logarithme inverse du rapport entre le nombre total de documents dans le corpus et le nombre de documents contenant le terme. Ces deux valeurs sont ensuite multipliées pour obtenir le score TF-IDF. L'utilisation du TF-IDF permet de mettre en avant les termes spécifiques à un document tout en réduisant l'importance des termes fréquents dans l'ensemble du corpus, contribuant ainsi à une représentation plus discriminante des documents.

#### 5.1.2.2.2. Application et résultats

Pour pouvoir mettre en place la vectorisation de type TF-IDF, on utilise l'outil **TfidfVectorizer** de scikit-learn. Assez logiquement on obtient le même nombre de features après vectorisation du jeu de données qu'avec le Bag of Words.

Pour le même jeu de données et les mêmes paramètres de classification, on a d'abord expérimenté la vectorisation TF-IDF sur le *classifier* MultinomialNB.



Matrice de confusion suite aux prédictions du MultinomialNB sur les données de test avec vectorisation TF-IDF

	precision	recall	f1-score	support
CORRECT	0.88	1.00	0.93	21483
NOT_FRENCH	1.00	0.99	0.99	19515
NOT_TRIP	1.00	0.79	0.88	21925
UNKNOWN	1.00	0.99	0.99	21345

Rapport de classification suite aux prédictions du MultinomialNB sur les données de test avec vectorisation TF-IDF

Si on compare, les résultats des prédictions sur le jeu de test avec la vectorisation TF-IDF sont presque similaires, avec une répartition des f1-scores presque identique pour chaque label.

Cependant, les scores de confiance pour la classe attendue sont généralement plus faibles (de 2% à 20% de confiance en moins avec TF-IDF).

	precision	recall	f1-score	support
CORRECT	1.00	0.99	1.00	21218
NOT_FRENCH	0.99	0.98	0.99	19853
NOT_TRIP	0.99	0.95	0.97	22185
UNKNOWN	0.98	1.00	0.99	21342

Rapport de classification suite aux prédictions du LogicRegression sur les données de test avec vectorisation TF-IDF

Avec le *classifier* LogicRegression, les résultats ont une tournure différente: les scores de confiance sont quasi-similaires entre les deux types de vectorisations. De même, le f1-score moyen est égal mais le recall pour la classe NOT\_TRIP est plus faible (0.95 contre 0.98 avec BoW). Ces différences sont légères mais laisseraient penser à première vue que la vectorisation Bag of Words permet d'obtenir des meilleures performances qu'avec le TF-IDF. Ces observations seront contestées avec l'utilisation du découpage N-grams et l'expérience de ces deux méthodes de vectorisation sur plusieurs classifiers [dans le benchmark](#).

#### 5.1.2.3. N-grams

##### 5.1.2.3.1. Présentation

## This is Big Data AI Book

<b>Uni-Gram</b>	This	Is	Big	Data	AI	Book
<b>Bi-Gram</b>	This is	Is Big	Big Data	Data AI	AI Book	
<b>Tri-Gram</b>	This is Big	Is Big Data	Big Data AI	Data AI Book		

Schéma explicatif des N-Grams

Les **n-grammes** sont des unités de texte composées de n éléments consécutifs, généralement des mots ou des caractères, extraits d'une séquence plus grande. Le principe de base derrière les n-grammes est de capturer les relations contextuelles et la structure linguistique d'un texte en examinant des morceaux adjacents de taille fixe. Par exemple, un **bi-gramme** (ou 2-gramme) dans le contexte des mots consécutifs pourrait être "chat noir", où "chat" et "noir" sont deux mots successifs. Les n-grammes sont largement utilisés dans le traitement automatique du langage naturel (NLP), notamment pour la modélisation du langage, la reconnaissance de la parole et la prédiction de texte, car ils permettent de saisir des informations sur la séquence des mots dans un corpus de texte, facilitant ainsi la compréhension et la génération de langage.

Par exemple, dans le cadre de la prédiction de mots, un modèle de langage basé sur les n-grammes peut estimer la probabilité d'occurrence du mot suivant en se basant sur les n-1 mots précédents. Cette approche simple mais puissante offre une manière efficace de modéliser les dépendances à court terme dans le langage.

### 5.1.2.3.2. Application et résultats

#### 5.1.2.3.2.1. Bigrams

Dans un premier temps, nous avons testé l'intégration des bigrams sur les meilleurs modèles de Bag of Words et TF-IDF. Cette intégration a consisté à **ajuster les paramètres de vectorisation** de ces modèles (en modifiant CountVectorizer et TfidfVectorizer).

Pour le modèle de régression logistique avec BoW, l'utilisation des unigrams a produit des **scores élevés de précision, de rappel et de score F1**, répartis de manière presque uniforme entre les différentes catégories. Cependant, en passant aux bigrams, bien que la précision soit restée élevée, nous avons observé une **baisse du rappel et du score F1** pour certaines catégories, en particulier 'NOT\_TRIP' et 'UNKNOWN'.

	precision	recall	f1-score	support
CORRECT	0.99	0.99	0.99	21384
NOT_FRENCH	0.99	0.99	0.99	19464
NOT_TRIP	0.98	0.98	0.98	22101
UNKNOWN	0.99	0.99	0.99	21303
micro avg	0.99	0.99	0.99	84252
macro avg	0.99	0.99	0.99	84252
weighted avg	0.99	0.99	0.99	84252
samples avg	0.99	0.99	0.99	84252

Rapport de classification - régression (unigram)

	precision	recall	f1-score	support
CORRECT	1.00	0.98	0.99	21117
NOT_FRENCH	1.00	0.85	0.92	19874
NOT_TRIP	0.99	0.84	0.91	22163
UNKNOWN	0.86	1.00	0.93	21391
micro avg	0.95	0.92	0.94	84545
macro avg	0.96	0.92	0.94	84545
weighted avg	0.96	0.92	0.93	84545
samples avg	0.94	0.94	0.94	84545

Rapport de classification - régression (bigram)

Dans le cas du modèle BoW avec Multinomial Naive Bayes, les unigrams ont également montré de **bons résultats à travers toutes les catégories**. Néanmoins, avec l'introduction des bigrams, bien que la précision ait été maintenue pour des catégories comme 'CORRECT', 'NOT\_FRENCH' et 'NOT\_TRIP', le **rappel et le score F1 pour 'UNKNOWN'** ont considérablement diminué.

Cette observation indique que les bigrams peuvent ne pas être aussi efficaces pour certaines catégories, en particulier là où la dépendance contextuelle entre les mots est moins marquée. On peut également le constater sur la matrice de confusion.

	precision	recall	f1-score	support
CORRECT	0.83	1.00	0.91	21459
NOT_FRENCH	1.00	1.00	1.00	19543
NOT_TRIP	0.94	0.87	0.90	21900
UNKNOWN	1.00	0.99	1.00	21316
micro avg	0.93	0.96	0.95	84218
macro avg	0.94	0.96	0.95	84218
weighted avg	0.94	0.96	0.95	84218
samples avg	0.94	0.96	0.94	84218

Rapport de classification - multinomial (unigram)

	precision	recall	f1-score	support
CORRECT	0.91	1.00	0.95	21077
NOT_FRENCH	1.00	0.97	0.98	19704
NOT_TRIP	1.00	0.87	0.93	22211
UNKNOWN	0.10	0.01	0.01	21424
micro avg	0.95	0.71	0.81	84416
macro avg	0.75	0.71	0.72	84416
weighted avg	0.75	0.71	0.72	84416
samples avg	0.65	0.66	0.66	84416

Rapport de classification - multinomial (bigram)



Matrice de confusion - multinomial (unigram)



Matrice de confusion - multinomial (bigram)

Pour le modèle TF-IDF avec régression logistique, lors du passage aux bigrams, bien que la précision globale soit **restée élevée**, nous avons constaté une **baisse du rappel et du score F1** pour certaines catégories. Cette observation suggère que, bien que les bigrams puissent apporter une compréhension plus profonde du contexte, **ils peuvent également introduire une complexité qui n'est pas toujours nécessaire ou efficace** pour certaines catégories de notre projet.

Tout comme BoW, dans le modèle TF-IDF avec Multinomial Naive Bayes, l'ajout de bigrams a entraîné une **diminution notable du rappel et du score F1** pour certaines catégories, en particulier 'UNKNOWN'.

#### 5.1.2.3.2.2. Unigram-Bigram

Il est également possible de combiner les unigrams et bigrams. **Dans le cas de la vectorisation Bag of Words, on parlera alors de Bag of 1 and 2 grams.**

L'expérimentation avec la combinaison des unigrams et des bigrams dans les modèles TF-IDF et Bag of Words (BoW) a apporté des **perspectives intéressantes** pour la classification de texte. Cette approche visait à **tirer parti des avantages des unigrams et des bigrams** pour une compréhension plus complète du contexte et de la structure des phrases.

Pour le modèle TF-IDF et BoW avec régression logistique combinant unigrams et bigrams, nous avons observé une légère **amélioration des scores de précision, de rappel et de score F1** pour toutes les catégories par rapport aux modèles utilisant uniquement des unigrams ou des bigrams. Cette combinaison semble avoir permis au modèle de **mieux saisir à la fois les détails spécifiques** apportés par les bigrams et **la vue d'ensemble fournie par les unigrams**.

	precision	recall	f1-score	support
CORRECT	1.00	0.99	1.00	21218
NOT_FRENCH	0.99	0.98	0.99	19853
NOT_TRIP	0.99	0.95	0.97	22185
UNKNOWN	0.98	1.00	0.99	21342
micro avg	0.99	0.98	0.99	84598
macro avg	0.99	0.98	0.99	84598
weighted avg	0.99	0.98	0.99	84598
samples avg	0.98	0.98	0.98	84598

Rapport de classification - TF-IDF régression logistique (unigram)

	precision	recall	f1-score	support
CORRECT	1.00	1.00	1.00	22939
NOT_FRENCH	0.99	0.98	0.99	20541
NOT_TRIP	0.98	0.97	0.98	24480
UNKNOWN	0.98	0.99	0.99	22907
micro avg	0.99	0.98	0.99	90867
macro avg	0.99	0.99	0.99	90867
weighted avg	0.99	0.98	0.99	90867
samples avg	0.98	0.99	0.98	90867

Rapport de classification - TF-IDF régression logistique (unigram-bigram)

De manière similaire, dans les modèles TF-IDF et BoW avec Multinomial Naive Bayes utilisant unigrams et bigrams, **les scores ont également montré une amélioration**, en particulier pour les catégories 'CORRECT' et 'NOT\_TRIP'. Cela indique que l'ajout de bigrams aux unigrams **a renforcé la capacité du modèle à distinguer entre différentes catégories de texte**.

	precision	recall	f1-score	support
CORRECT	0.88	1.00	0.93	21483
NOT_FRENCH	1.00	0.99	0.99	19515
NOT_TRIP	1.00	0.79	0.88	21925
UNKNOWN	1.00	0.99	0.99	21345
micro avg	0.96	0.94	0.95	84268
macro avg	0.97	0.94	0.95	84268
weighted avg	0.97	0.94	0.95	84268
samples avg	0.94	0.93	0.93	84268

Rapport de classification - TF-IDF multinominal naive bayes (unigram)

	precision	recall	f1-score	support
CORRECT	0.89	1.00	0.94	23011
NOT_FRENCH	1.00	0.99	1.00	20690
NOT_TRIP	1.00	0.84	0.91	24585
UNKNOWN	1.00	0.98	0.99	22665
micro avg	0.97	0.95	0.96	90951
macro avg	0.97	0.95	0.96	90951
weighted avg	0.97	0.95	0.96	90951
samples avg	0.94	0.94	0.94	90951

*Rapport de classification - TF-IDF multinomial naive bayes (unigram-bigram)*

L'intégration des unigrams et des bigrams dans les modèles TF-IDF et BoW a montré des **améliorations significatives dans la classification de texte**. Cette approche a permis de combiner les avantages de la capture de contextes spécifiques par les bigrams avec la vue d'ensemble fournie par les unigrams, conduisant à **une meilleure performance globale des modèles**. Ces résultats soulignent l'importance de l'expérimentation avec différentes combinaisons de N-Grams pour optimiser la classification de texte.

### 5.1.3. Normalisation

Les expérimentations sur les techniques de normalisation ont été effectuées en utilisant MultinomialNB et la vectorisation Bag of Words.

#### 5.1.3.1. Stop Words

##### 5.1.3.1.1. Présentation

Le principe des **stop words** (mots vides) repose sur l'idée de filtrer certains termes très fréquents et peu informatifs dans un texte afin d'améliorer la qualité de l'analyse ou du traitement du langage naturel. Ces mots, tels que "le", "et", "est", sont courants dans la plupart des documents et n'apportent généralement pas beaucoup de signification ou de spécificité. En les excluant du processus d'analyse, on peut réduire le bruit et améliorer l'efficacité de certaines tâches, comme la recherche d'information ou l'analyse de texte, en se concentrant davantage sur les termes qui portent une charge sémantique plus importante pour la compréhension du contenu.

#### 5.1.3.1.2. Application et résultats

		Confusion Matrix			
		CORRECT	NOT_FRENCH	NOT_TRIP	UNKNOWN
True labels	CORRECT	19720	0	0	0
	NOT_FRENCH	30	19053	6	1
NOT_TRIP	2351	54	4287	3	
UNKNOWN	3411	319	83	15830	
Predicted labels		CORRECT	NOT_FRENCH	NOT_TRIP	UNKNOWN

*Matrice de confusion suite aux prédictions du MultinomialNB sur les données de test en supprimant les stop words*

La performance du modèle semble équivalente à la version sans les stops words, avec tout même plus de phrases NOT\_TRIP et UNKNOWN mal classées. **Le modèle semble tout de même moins serein sur les pourcentages de confiance pour chaque label.**

#### 5.1.3.2. Lemmatization

##### 5.1.3.2.1. Présentation

La lemmatisation est un processus linguistique visant à réduire les mots à leur forme de base, appelée lemme, en tenant compte de leur racine morphologique et de leur signification lexicale. La lemmatisation prend en considération la grammaire et le contexte pour produire des lemmes qui représentent le sens fondamental des mots. Par exemple, le lemme du verbe "running" est "run". L'avantage de la lemmatisation réside dans sa capacité à normaliser les termes, facilitant ainsi la comparaison et l'analyse sémantique. Cela est particulièrement utile dans le domaine du traitement du langage naturel (NLP), où la lemmatisation contribue à améliorer la cohérence et la précision des analyses textuelles en réduisant les variantes morphologiques à une forme canonique.

### 5.1.3.2.2. Application et résultats

		Confusion Matrix			
		CORRECT	NOT_FRENCH	NOT_TRIP	UNKNOWN
True labels	CORRECT	19671	0	0	0
	NOT_FRENCH	27	19265	5	6
NOT_TRIP	2316	49	4122	0	
UNKNOWN	3572	379	93	15643	
Predicted labels		CORRECT	NOT_FRENCH	NOT_TRIP	UNKNOWN

*Matrice de confusion suite aux prédictions du MultinomialNB sur les données de test en supprimant les stop words et en appliquant la lemmatisation*

Le modèle a encore réduit le nombre de faux positifs pour la classe CORRECT par rapport à la version avec seulement les *stop words*. Cependant, l'amélioration n'est pas impressionnante relativement aux résultats totaux (le *f1-score* reste figé à 94%).

### 5.1.3.3. Stemming

#### 5.1.3.3.1. Présentation

Le stemming est un processus de prétraitement textuel utilisé en linguistique informatique pour réduire les mots à leur racine ou à leur forme de base, appelée "stem". Le stemming se contente de retirer les suffixes ou préfixes d'un mot sans nécessairement prendre en compte la signification sémantique. L'objectif est de regrouper différentes variantes morphologiques d'un mot en une forme commune, facilitant ainsi la recherche et l'analyse de textes. Par exemple, le stemming transformeraient les mots "running", "runs", et "ran" en la racine "run". Bien que moins précis que la lemmatisation, le stemming est souvent utilisé dans des contextes où la simplicité et l'efficacité sont privilégiées, tels que la recherche d'information ou le regroupement thématique de documents dans le traitement du langage naturel.

### 5.1.3.3.2. Application et résultats

		Confusion Matrix			
		CORRECT	NOT_FRENCH	NOT_TRIP	UNKNOWN
True labels	CORRECT	19839	0	0	0
	NOT_FRENCH	30	19152	9	2
UNKNOWN	NOT_TRIP	2197	79	4222	3
	UNKNOWN	3425	374	99	15717
		Predicted labels	CORRECT	NOT_FRENCH	NOT_TRIP

Aux premiers abords, le stemming semble être plus performant que la lemmatisation. **Néanmoins, les résultats sur les exemples de phrases en dehors du jeu de données sont très médiocres et souvent faux ou incertains (le label attendu est bien prédit, mais les scores de confiance des autres labels sont également élevés).**

**En conclusion, les trois techniques de normalisation testées sur notre jeu de données ne semblent pas apporter de changements significatifs aux résultats finaux pour le même classifier et les mêmes données.**

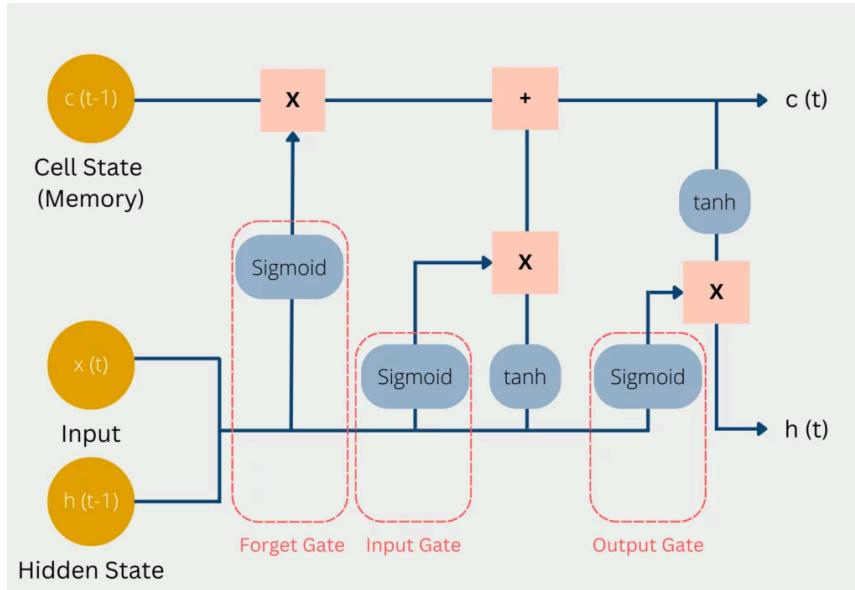
## 5.2. RNN

### 5.2.1. LSTM

#### 5.2.1.1. Présentation

Les réseaux de neurones à **Long Short-Term Memory** (LSTM) sont une sous-catégorie des **réseaux de neurones récurrents** (RNN). Les LSTM ont été conçus pour **surmonter les limitations des RNN** traditionnels, notamment le problème de disparition du gradient. Cette caractéristique les rend particulièrement **adaptés pour apprendre des dépendances à long terme** et traiter des séquences de données, ce qui est crucial dans de nombreuses applications de traitement du langage naturel (NLP).

L'architecture des LSTM est conçue pour surmonter les défis des **séquences longues et complexes**, grâce à ses cellules dotées de trois portes : **la porte d'oubli, la porte d'entrée et la porte de sortie**. Ces portes régulent le flux d'informations, permettant au réseau de retenir et d'oublier les données de manière sélective. Cette capacité est importante pour notre projet, car **elle permet au modèle de se concentrer sur les aspects pertinents** des commandes de trajet tout **en ignorant les informations superflues**.



*Représentation simplifiée de l'architecture d'une cellule LSTM*

La principale force des LSTM réside dans leur capacité à **comprendre et à mémoriser le contexte sur de longues séquences de texte**. Alors que des méthodes comme Bag of Words et TF-IDF se concentrent principalement sur la fréquence des mots sans tenir compte de l'ordre et de la structure des phrases, **les LSTM peuvent saisir la séquence et la dépendance temporelle des mots** dans un texte. Cette caractéristique est particulièrement bénéfique pour notre application, où la compréhension du contexte et de la séquence des mots est cruciale pour une classification précise.

En outre, par rapport à des méthodes classiques, qui sont des méthodes linéaires et peuvent ne pas capturer efficacement les relations complexes entre les mots dans des phrases longues ou complexes, les LSTM, grâce à leur nature non linéaire et leur capacité à maintenir des états sur de longues périodes, sont mieux équipés pour gérer de telles complexités.

En choisissant d'expérimenter avec les LSTM, nous visons à améliorer significativement la précision de notre système de classification de texte, en garantissant que **les nuances et le contexte soient correctement interprétés et classifiés**. Cette approche devrait conduire à une meilleure compréhension des intentions de l'utilisateur et à une interaction plus fluide et naturelle avec notre système.

#### 5.2.1.2. Application et résultats

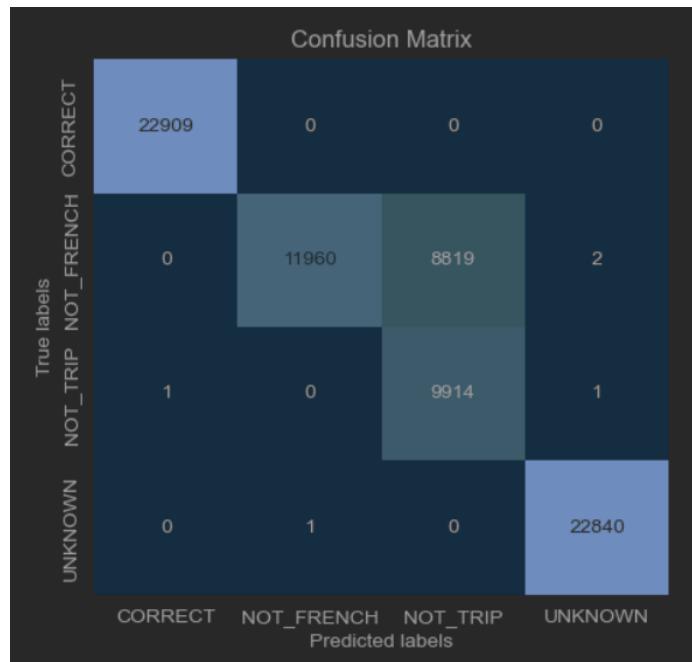
Le modèle que nous avons conçu commence par une couche d'embedding, où la taille du vocabulaire a été définie à 20 000 mots. Ce choix vise à **couvrir un large éventail de termes utilisés**, tout en restant dans un cadre gérable pour le modèle. La dimension de l'embedding est fixée à 128, permettant ainsi une **représentation dense et significative des mots**, essentielle pour capturer les nuances sémantiques entre différents termes. La longueur maximale des séquences est établie à 100 mots, un  **compromis entre la nécessité de couvrir des phrases suffisamment longues et la limitation de la complexité computationnelle**.

Le cœur du modèle est constitué d'**une couche LSTM avec 64 unités**. Ce nombre a été choisi pour offrir une **bonne capacité de modélisation tout en évitant une charge computationnelle excessive**. La couche de sortie est une couche Dense avec 4 unités, adaptée au nombre de labels de

notre tâche de classification, et utilise une activation 'sigmoid' pour la classification des séquences dans les catégories prédéfinies.

L'exécution du modèle sur 5 epochs a été décidée pour **équilibrer l'apprentissage efficace et la prévention du surapprentissage**. Cela permet au modèle de s'ajuster progressivement aux données, évitant ainsi de mémoriser spécifiquement les détails du jeu de données d'entraînement. De plus, au vu du nombre de données dans notre dataset, l'utilisation de seulement **50% du dataset** pour cette première expérimentation visait à **réduire le temps de calcul et à permettre une évaluation rapide** de l'approche. Cela nous a permis d'identifier rapidement les tendances initiales et les ajustements nécessaires sans engager de ressources excessives.

Malgré cela, **le temps d'entraînement reste assez long comparé à d'autres modèles** que nous avons pu tester. Cependant **les résultats restent satisfaisants** comme observés sur la matrice de confusion. Il est normal que l'on puisse voir des "erreurs" sur des données qu'il a classifié en "NOT\_TRIP" alors que leur label était "NOT\_FRENCH", cela s'explique par le fait que les données sont multi-labels et que plusieurs données sont classées à la fois comme "NOT\_TRIP" mais aussi comme "NOT\_FRENCH".



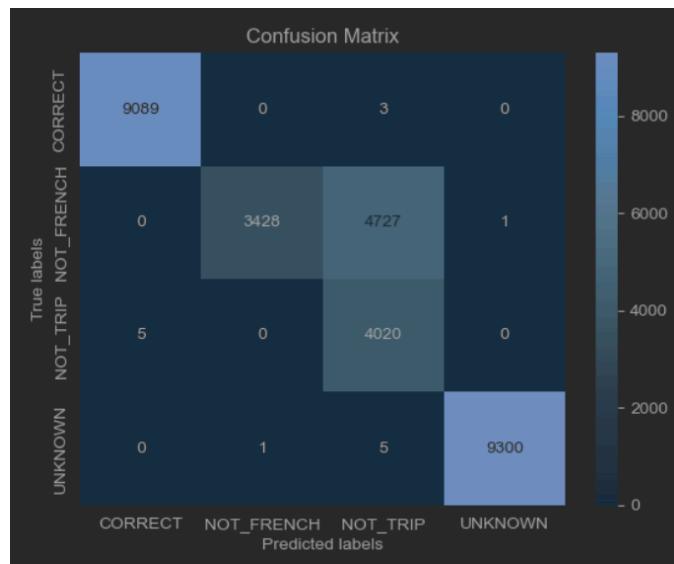
*Matrice de confusion suite aux prédictions réalisés par notre LSTM  
(5 epochs / 50% dataset pour l'entraînement)*

2389/2389	[=====]	-	18s	7ms/step
		precision	recall	f1-score
				support
CORRECT		1.00	1.00	1.00
NOT_FRENCH		1.00	1.00	1.00
NOT_TRIP		1.00	1.00	1.00
UNKNOWN		1.00	1.00	1.00
micro avg		1.00	1.00	1.00
macro avg		1.00	1.00	1.00
weighted avg		1.00	1.00	1.00
samples avg		1.00	1.00	1.00

Rapport de classification (5 epochs / 50% dataset)

Comme nous pouvons le voir, les résultats sont **plus que satisfaisant**, le seul problème est le **temps d'entraînement qui est d'environ 30 minutes**, lorsque l'on compare cela aux autres modèles que nous avons pu tester comme par exemple le LinearSVC qui a un temps d'entraînement d'environ 15 secondes pour des résultats à peu près similaires. Ce problème n'est pas négligeable, il faut donc **essayer au maximum de réduire ce temps d'entraînement en impactant un minimum les résultats.**

Pour cela, nous avons donc essayé de **diminuer encore un peu plus le dataset**, en utilisant que 20% de ce dernier. Les résultats n'ont pas changé et **le temps d'entraînement à diminuer de plusieurs minutes**, ce qui est toujours loin d'être satisfaisant comparé aux autres models. Nous avons donc essayé de **réduire à 3 le nombre d'éPOCHS tout en gardant 20% du dataset**. Le temps d'entraînement **est donc passé d'une trentaine de minutes à seulement quelques minutes**, tout cela en gardant des **résultats très satisfaisant**.



Matrice de confusion (3 epochs / 20% dataset)

956/956	[=====]	- 17s 15ms/step
precision recall f1-score support		
CORRECT		1.00 1.00 1.00 9092
NOT_FRENCH		1.00 1.00 1.00 8156
NOT_TRIP		1.00 1.00 1.00 9746
UNKNOWN		1.00 1.00 1.00 9306
micro avg		1.00 1.00 1.00 36300
macro avg		1.00 1.00 1.00 36300
weighted avg		1.00 1.00 1.00 36300
samples avg		1.00 1.00 1.00 36300

Rapport de classification (3 epochs / 20% dataset)

En conclusion, bien que les réseaux LSTM aient montré **un potentiel certain dans notre projet** pour la classification de texte, nous avons finalement décidé de ne pas les adopter comme solution principale. Cette décision a été motivée par deux considérations clés.

Premièrement, **le temps d'entraînement** des LSTM s'est avéré nettement plus long par rapport à d'autres modèles plus simples que nous avons testés basés sur des algorithmes de classification classiques. Ces modèles alternatifs ont offert des temps d'entraînement beaucoup **plus rapides** tout en fournissant des résultats similaires, ce qui les rendait plus adaptés.

Deuxièmement, **l'utilisation des LSTM dans notre contexte s'est avérée quelque peu excessive**. Notre projet impliquait principalement le **traitement de phrases simples** plutôt que de longs paragraphes ou de textes avec des structures complexes. Les LSTM sont **particulièrement efficaces pour gérer des séquences longues et des dépendances à long terme**, ce qui les rend idéaux pour des scénarios impliquant des dialogues étendus ou des textes avec des contextes riches et variés (même si les *transformers* sont aujourd'hui plus performants pour cette même tâche). Cependant, pour notre cas d'utilisation, où les entrées étaient des commandes vocales relativement courtes et directes, l'architecture complexe des LSTM ne se justifiait pas pleinement.

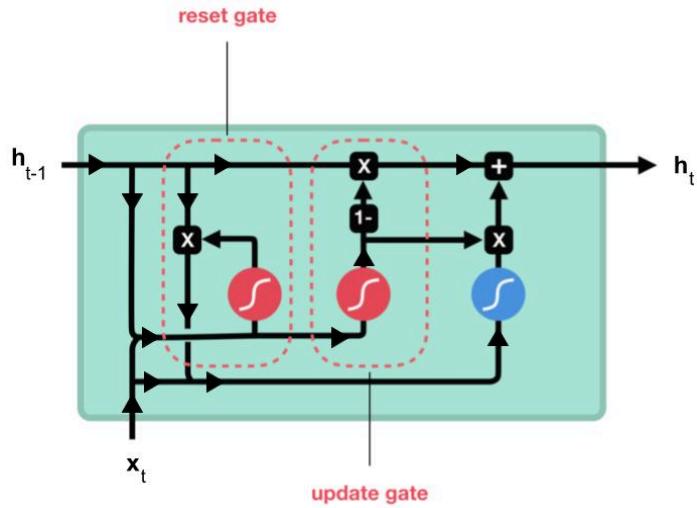
### 5.2.2. GRU

#### 5.2.2.1. Présentation

Les réseaux de neurones Gated Recurrent Unit (GRU) sont une **variante des réseaux de neurones récurrents (RNN)** qui ont été introduits pour **améliorer les performances des architectures RNN standard**. Les GRU ont été conçus pour résoudre le problème de disparition du gradient, similaire aux LSTM, mais avec **une structure plus simplifiée**. Ils sont particulièrement reconnus pour leur efficacité dans le traitement de séquences de données, notamment dans des applications de traitement du langage naturel (NLP) et de reconnaissance vocale.

L'architecture des réseaux Gated Recurrent Unit (GRU) se distingue par sa **conception simplifiée** par rapport aux LSTM, tout en conservant une **efficacité similaire** dans le traitement des séquences de données. Au cœur de cette architecture se trouvent deux portes principales qui régulent le flux d'informations. La première, la **porte de mise à jour**, détermine la quantité d'informations de l'état précédent à conserver pour le nouvel état. **Cette porte est une fusion des portes d'oubli et d'entrée que l'on trouve dans les LSTM**, permettant ainsi une **gestion plus efficace des informations** à

travers le réseau. La seconde porte, la **porte de réinitialisation**, est chargée de décider de la quantité d'informations passées à oublier, ce qui permet au GRU de **filtrer et de maintenir uniquement les données pertinentes** pour les prédictions actuelles. Ensemble, ces portes aident les GRU à capturer des dépendances à long terme dans les données, tout en réduisant la complexité et le nombre de paramètres nécessaires, comparativement aux LSTM. Cette architecture rend les GRU **particulièrement adaptés pour des applications où la gestion efficace des séquences est essentielle**, mais où une structure moins complexe est préférable.



*Représentation simplifiée de l'architecture d'une cellule GRU*

En comparaison avec les LSTM, les GRU offrent une architecture plus simple et donc **potentiellement plus rapide à entraîner**, tout en conservant une capacité similaire à gérer des dépendances à long terme dans les données séquentielles. Cette simplicité peut se traduire par **une meilleure performance dans certains cas**, surtout lorsque la quantité de données est limitée ou que la puissance de calcul est une contrainte.

Par rapport aux autres modèles que nous avons testés dans notre projet, comme la régression logistique ou les approches basées sur TF-IDF, les **GRU se situent quelque part entre ces modèles plus simples et les LSTM** en termes de complexité et de performance. Ils offrent un bon compromis, apportant une capacité accrue à gérer les séquences par rapport aux modèles plus basiques, tout en évitant certaines des complexités et du temps d'entraînement associés aux LSTM.

#### 5.2.2.2. Application et résultats

Pour notre expérimentation avec GRU, nous avons construit un modèle séquentiel en utilisant **TensorFlow et Keras**. Le modèle commence par une couche d'embedding, avec une dimension de 128, pour **transformer les mots en vecteurs denses et capturer les relations sémantiques** entre eux. Cette dimension a été choisie pour équilibrer la richesse de la représentation des mots et la complexité computationnelle.

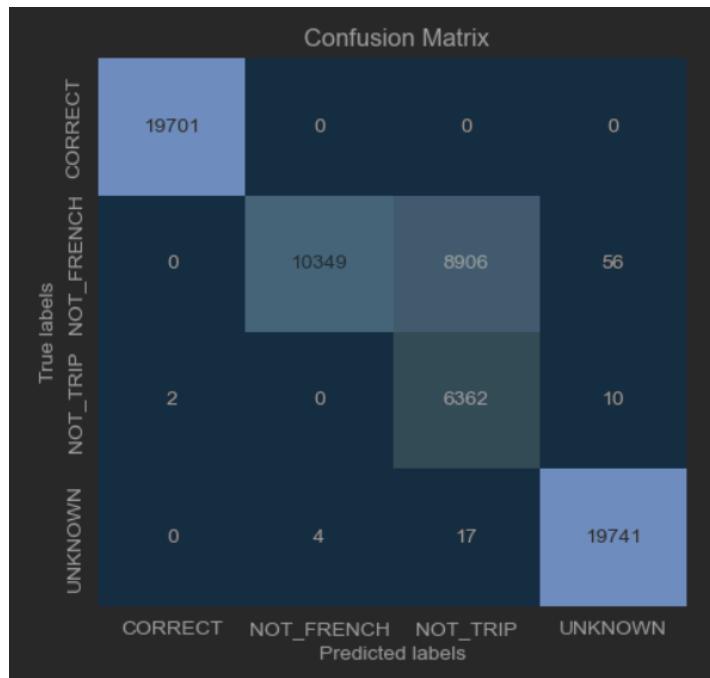
La couche centrale du modèle est une couche GRU avec 64 unités. Ce nombre d'unités a été sélectionné pour fournir une **capacité de modélisation suffisante pour notre tâche de classification** tout en maintenant la complexité du modèle à un niveau gérable. La couche GRU est

conçue pour **traiter les séquences de texte et capturer les dépendances temporelles**, ce qui est important pour comprendre le contexte des commandes vocales.

La couche de sortie est une couche Dense avec 4 unités, correspondant au nombre de labels dans notre tâche de classification, et utilise une activation 'sigmoid'. Cette configuration permet au modèle de classer chaque séquence de texte dans l'une des catégories prédéfinies.

Pour cette première expérimentation, nous avons choisi de former **le modèle sur 5 epochs avec 50% du dataset**. Cette approche nous a permis d'évaluer rapidement l'efficacité du modèle GRU sans consacrer trop de temps ou de ressources. L'utilisation de seulement la moitié du dataset visait à obtenir une estimation préliminaire des performances du modèle, tout en **réduisant le temps d'entraînement**.

Les résultats initiaux ont montré que les GRU, tout en étant moins complexes que les LSTM, offraient une **capacité comparable à gérer les séquences de texte**. Cependant, il était essentiel de trouver un équilibre entre la performance du modèle et les ressources computationnelles nécessaires, un aspect crucial dans le contexte de notre projet.



*Matrice de confusion suite aux prédictions réalisés par notre GRU  
(5 epochs / 50% dataset pour l'entraînement)*

	2036/2036 [=====] - 14s 7ms/step			
precision    recall    f1-score    support				
CORRECT	1.00	1.00	1.00	19701
NOT_FRENCH	1.00	1.00	1.00	19311
NOT_TRIP	1.00	1.00	1.00	19478
UNKNOWN	1.00	1.00	1.00	19762
micro avg	1.00	1.00	1.00	78252
macro avg	1.00	1.00	1.00	78252
weighted avg	1.00	1.00	1.00	78252
samples avg	1.00	1.00	1.00	78252

*Rapport de classification (5 epochs / 50% dataset)*

Comme observé dans nos expérimentations avec les réseaux LSTM, les **performances obtenues étaient tout à fait encourageantes**. Cependant, un **défi majeur** s'est présenté sous la forme du **temps d'entraînement**, qui s'élevait à environ 20 minutes. Cette durée, lorsqu'elle est mise en perspective avec d'autres modèles testés dans notre projet, tels que le **LinearSVC** qui ne nécessite que 15 secondes **pour un niveau de performance similaire**, souligne une contrainte non négligeable. Il est donc devenu **impératif d'explorer des moyens pour réduire ce temps d'entraînement** tout en préservant l'efficacité du modèle.

Dans cette optique, nous avons décidé, tout comme avec les réseaux LSTM, de **réduire davantage la taille du dataset utilisé**, en ne prenant en compte que **20%** de celui-ci. Cette modification a permis de **diminuer le temps d'entraînement de plusieurs minutes** sans altérer significativement les résultats, une avancée positive mais encore insuffisante au regard des performances des autres modèles. Poursuivant nos efforts d'optimisation, nous avons ensuite **réduit le nombre d'éPOCHS à 3**, tout en maintenant l'utilisation de **20% du dataset**. Cette nouvelle configuration a conduit à une réduction substantielle du temps d'entraînement, le ramenant de 20 minutes à seulement quelques minutes. L'entraînement est donc **légèrement plus rapide que LSTM** pour des résultats à peu près similaires.



*Matrice de confusion suite aux prédictions réalisées par notre GRU  
(3 epochs / 20% dataset pour l'entraînement)*

	815/815 [=====] - 8s 9ms/step			
		precision	recall	f1-score
				support
CORRECT	1.00	1.00	1.00	7803
NOT_FRENCH	1.00	0.99	1.00	7730
NOT_TRIP	1.00	0.99	1.00	7786
UNKNOWN	0.99	1.00	1.00	7977
micro avg	1.00	1.00	1.00	31296
macro avg	1.00	1.00	1.00	31296
weighted avg	1.00	1.00	1.00	31296
samples avg	1.00	1.00	1.00	31296

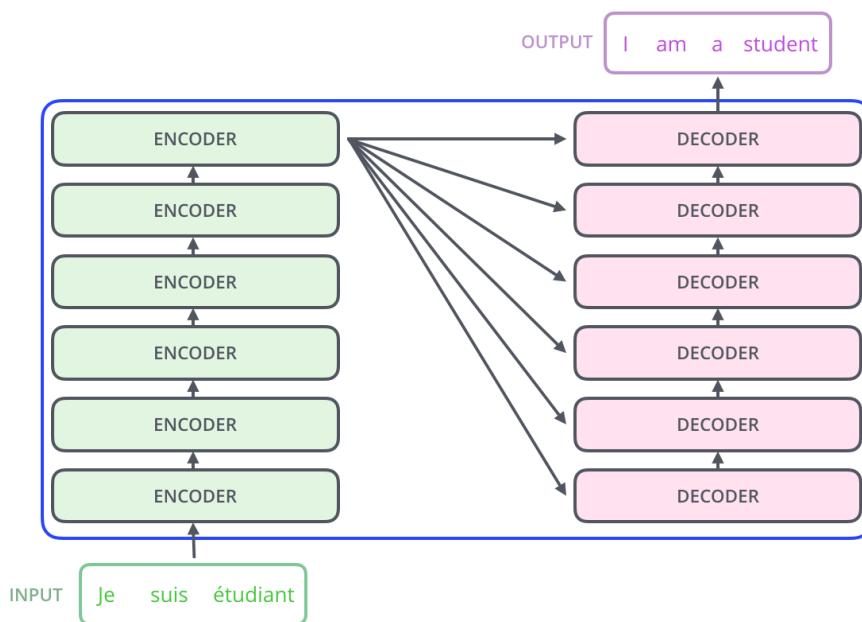
Rapport de classification (3 epochs / 20% dataset)

En conclusion, les réseaux GRU, tout comme les LSTM précédemment considérés, **ont démontré un potentiel notable dans notre projet** pour la classification de texte. Cependant, après une évaluation approfondie, nous avons décidé de ne pas les retenir comme solution principale pour des raisons similaires à celles qui nous ont conduits à écarter les LSTM.

En somme, bien que les GRU (et les LSTM) soient des **outils puissants dans le domaine du traitement du langage naturel**, leur pertinence dans le cadre spécifique de notre projet a été jugée limitée. Cette expérience nous a permis de **mieux comprendre les avantages et les limites** des différentes approches en matière de traitement du langage naturel, nous orientant vers des solutions plus adaptées à nos besoins spécifiques.

## 5.3. Transformers

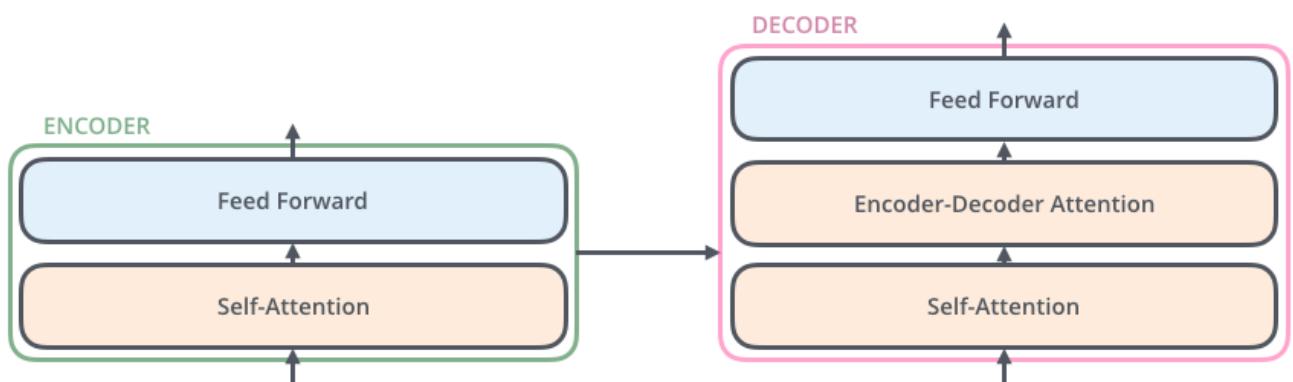
### 5.3.1. Concept



### *Représentation simplifiée de l'architecture d'un transformer*

Les *transformers* sont une classe d'architectures de réseaux neuronaux qui ont révolutionné le domaine du traitement du langage naturel et de l'apprentissage automatique en général. Contrairement aux architectures récurrentes telles que les RNN ou les LSTM, les *transformers* adoptent **une approche sans récurrence** pour capturer les dépendances à long terme dans les données séquentielles.

Les *transformers* se composent de **blocs d'attention**, qui permettent au modèle de se concentrer sur différentes parties de la séquence d'entrée lors de la prise de décision. Chaque bloc d'attention contient des mécanismes **d'auto-attention** qui attribuent des poids aux différentes parties de la séquence en fonction de leur importance relative. Cela permet aux *transformers* de traiter efficacement les relations à long terme dans les données, améliorant ainsi les performances sur des tâches complexes telles que la traduction automatique et la génération de texte.



*Chaque encodeur et décodeur est composé d'une couche de self-attention. Les décodeurs possèdent aussi une couche d'attention supplémentaire pour prendre en compte les données pertinentes de la phrase d'entrée.*

Le principe d'encodeur et de décodeur est au cœur de l'architecture des *transformers*. L'encodeur prend en charge la séquence d'entrée, transformant chaque élément en une représentation vectorielle. Les mécanismes **d'auto-attention** dans l'encodeur permettent de capturer les relations à long terme entre les éléments de la séquence source. Le décodeur, quant à lui, utilise ces représentations pour générer la séquence cible. Il travaille de manière autonome, en utilisant à la fois l'information contextuelle fournie par l'encodeur (et la couche Encoder-Decoder attention) et le **mécanisme d'auto-attention** pour prédire chaque élément de la séquence cible.

*La couche de feed-forward dans un transformer agit comme un module de traitement qui suit les mécanismes d'auto-attention. Elle introduit de la non-linéarité en appliquant des transformations aux informations de la séquence, aidant ainsi le modèle à capturer des relations complexes et à apprendre des représentations plus riches*

*Chaque bloc d'attention est composé de plusieurs sous-modules, notamment les couches linéaires, les fonctions d'activation et les mécanismes d'auto-attention. Ces sous-modules travaillent ensemble de manière synergique pour permettre au modèle de comprendre et de représenter des informations complexes à différentes échelles temporelles.*

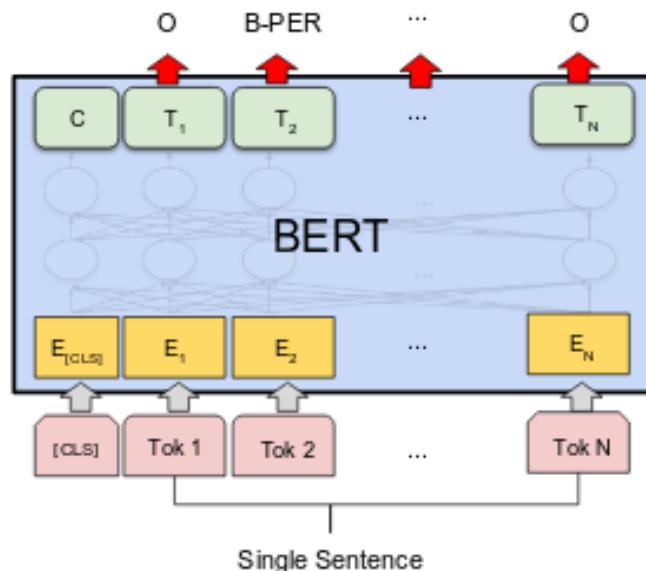
Ce qui distingue les *transformers* des architectures récurrentes antérieures, telles que les RNN et les LSTM, c'est leur capacité à **traiter les dépendances à long terme de manière plus efficace**. Les RNN et les LSTM souffrent du problème de **l'affaiblissement du gradient**, où l'information pertinente sur de longues séquences est progressivement perdue au fil du temps, rendant difficile la capture de relations à long terme. Les *transformers*, en revanche, grâce à leur mécanisme d'auto-attention, sont capables de prendre en compte des dépendances à long terme sans le risque de dégradation du signal, ce qui a considérablement amélioré les performances des modèles sur une variété de tâches complexes. Ainsi, la transition des RNN aux *transformers* marque une révolution dans le domaine de l'apprentissage automatique, ouvrant la voie à des modèles plus puissants et capables de traiter des séquences de manière plus efficace.

### 5.3.2. BERT

#### 5.3.2.1. Présentation

**Seq2seq**, ou séquence à séquence, fait référence à une approche en apprentissage automatique où un modèle prend une séquence en entrée et génère une autre séquence en sortie. L'architecture des *transformers* s'applique très bien à cette tâche car elle permet de générer une séquence de sortie selon une séquence d'entrée.

**BERT, ou Bidirectional Encoder Representations from Transformers**, est un modèle de langage pré-entraîné basé sur l'architecture des *transformers*. Ce qui le distingue des **modèles séquence à séquence** (seq2seq) traditionnels, c'est son approche novatrice de l'entraînement non supervisé. Contrairement aux modèles seq2seq qui étaient principalement utilisés pour des tâches spécifiques telles que la traduction automatique, BERT a été conçu pour apprendre des représentations de mots en contexte à partir de grandes quantités de texte non étiqueté.



Chaque élément (ou token) de la phrase est traité en prenant en compte ses relations avec l'ensemble des tokens de la phrase

La grande avancée de BERT réside dans son entraînement **bidirectionnel**, permettant au modèle de prendre en compte tout le contexte de la séquence lors de la prédiction d'un mot donné. Contrairement

aux modèles seq2seq qui traitent la séquence de manière séquentielle, **BERT considère simultanément l'ensemble des mots avant et après chaque mot**, capturant ainsi des relations plus riches et complexes.

Il est très simple de *fine-tuner* BERT pour adapter le modèle à nos propres étiquettes de sortie (CORRECT et autres, puis départ et arrivée), en particulier dans les cas de la classification de texte et la Reconnaissance d'Entités Nommées (NER).

### 5.3.2.2. Application et résultats

Le fine-tuning de BERT est la première expérimentation effectuée dans le cadre du projet. Nos attentes concernant les résultats étaient évidemment hautes, car les *transformers* constituent aujourd'hui la méthode la plus performante dans le traitement du langage naturel. **Cependant, nous verrons que ces outils doivent être contrôlés pour que leur utilisation soit viable et efficace.**

Afin de fine-tuner BERT, nous avons notamment utiliser les libraries Python *transformers* et *pytorch*, et surtout beaucoup de sources disponibles sur la page de présentation de BERT qui nous a permis d'obtenir une marche à suivre pour arriver à apprendre à BERT à classer nos phrases mais aussi à reconnaître spécifiquement le départ et l'arrivée dans nos phrases.

```
tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-cased')

'[CLS] Départ for Rethel from Saint Léonard. [SEP] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]
[PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD] [PAD]'
```

*Exemple d'une phrase passée dans le tokenizer de BERT*

Pour entraîner BERT, il est important de "tokenizer" les phrases, c'est-à-dire de les diviser en morceaux plus petits appelés "tokens". La tokenization est nécessaire pour que le modèle puisse comprendre et traiter chaque mot ou partie de la phrase individuellement. BERT utilise des balises spéciales comme [CLS] (pour classification) au début de chaque phrase et [PAD] (pour le padding) pour remplir les vides, garantissant ainsi que toutes les phrases ont la même longueur. Les balises [CLS] aident BERT à comprendre le contexte global de la phrase, tandis que les balises [PAD] assurent une uniformité de la longueur pour un traitement efficace lors de l'entraînement du modèle.

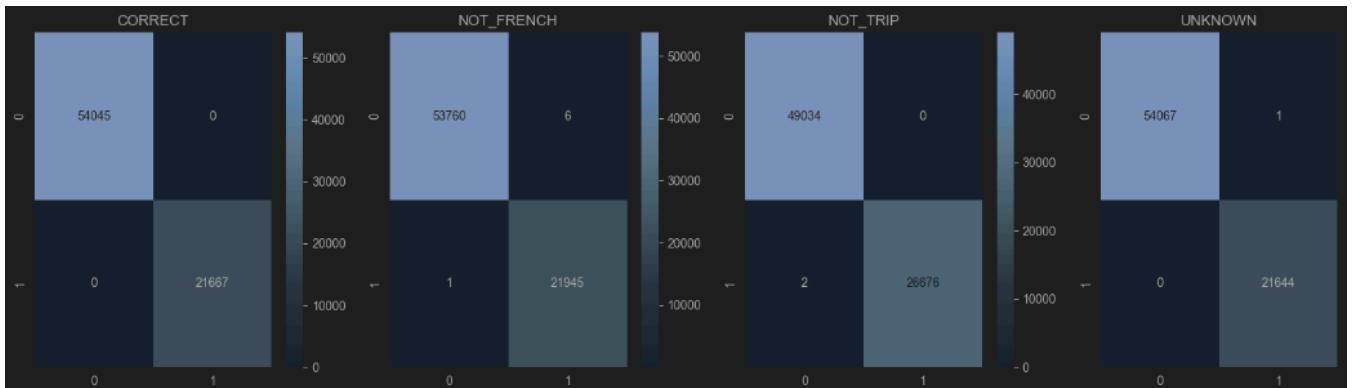
```
model = BertForSequenceClassification.from_pretrained(
    'bert-base-multilingual-cased',
    problem_type="multi_label_classification",
    num_labels=len(labels),
    id2label=id2label,
    label2id=label2id
)
```

*Chargement et configuration du fine-tuning du modèle BERT bert-base-multilingual-cased*

Ici, nous utilisons le modèle BERT-multilingual-cased, une version de BERT pré-entraînée qui prend en charge plusieurs langues, en maintenant la casse des mots (cased), offrant ainsi une représentation

contextualisée des mots dans un contexte multilingue. Le paramètre `num_labels` va permettre à BERT de savoir le nombre de sorties possibles après *fine-tuning*.

**Le fine-tuning s'est effectué avec 5 epochs, un batch size de 16, et un jeu d'entraînement de 600 mille lignes.**



Matrices de confusion de type One VS Rest, on observe presque aucun faux positif / faux négatif

	precision	recall	f1-score	support
CORRECT	1.00	1.00	1.00	21667
NOT_FRENCH	1.00	1.00	1.00	21946
NOT_TRIP	1.00	1.00	1.00	26678
UNKNOWN	1.00	1.00	1.00	21644

Rapport de classification, avec toutes les métriques à 100%

En effectuant des prédictions sur notre jeu de test, on remarque que BERT est juste à hauteur de 100% (presque qu'aucune erreur).



Evolution du f1-score au cours des époques, sur les échelles valeur minimale / valeur maximale atteintes et entre 0 et 1

Si on observe l'évolution du f1-score au cours des différentes époques, on se rend compte qu'un pic satisfaisant est déjà atteint dès la troisième époque. Cette observation va permettre plus tard de limiter le nombre d'époques, ce qui provoquera un gain de temps conséquent. L'évolution de ce f1-score rapportée à une échelle de 0 à 1 n'est toutefois pas perceptible. Par défaut, BERT et ses

**modèles dérivés sont tellement performants pour ce genre de tâche que la performance sur les différentes métriques avoisine toujours les 100%.**

Une des métriques intéressantes reste d'observer le score de confiance avec lequel BERT prédit, sur des phrases n'appartenant pas au jeu de données. Les phrases testées sont les suivantes:

Phrase	Attendu	Prédiction	Score de confiance
Je veux aller de Port-Boulet à Le Havre.	CORRECT	NOT_TRIP	99,9%
Peux-tu m'aider à trouver mon chemin vers Paris en partant d'Épierre ?	CORRECT	CORRECT	99,9%
Je cherche un moyen d'aller de Margny-Lès-Compiègne à Saarbrücken /Sarrebruck.	CORRECT	NOT_TRIP	99,9%
Je veux me rendre chez mon ami Etienne à Saint-Étienne depuis Nantes.	CORRECT	NOT_TRIP	99,9%
Je veux aller de la ville de Marseille à Tours	CORRECT	NOT_TRIP	99,9%

**Le fine-tuning de BERT étant une des premières expérimentations, cette dernière n'a cependant pas été concluante.** En effet, on observe que pour des phrases pouvant être interprétées comme CORRECT, seulement une a été correctement prédite. Même lorsque BERT se trompe, le score de confiance est toujours à quasiment 100%. A ce moment précis, les hypothèses sont multiples:

- le jeu de données présentes des erreurs
- BERT n'a pas réussi à généraliser à cause de trop de données en entrée
- BERT n'a pas réussi à généraliser car les données d'entraînement présentes trop de similarité
- BERT n'a pas réussi à généraliser car le fine-tuning a été trop "approfondi" (trop d'époques)

**En plus de ces problèmes, le temps d'entraînement est conséquent: environ 9h30.**

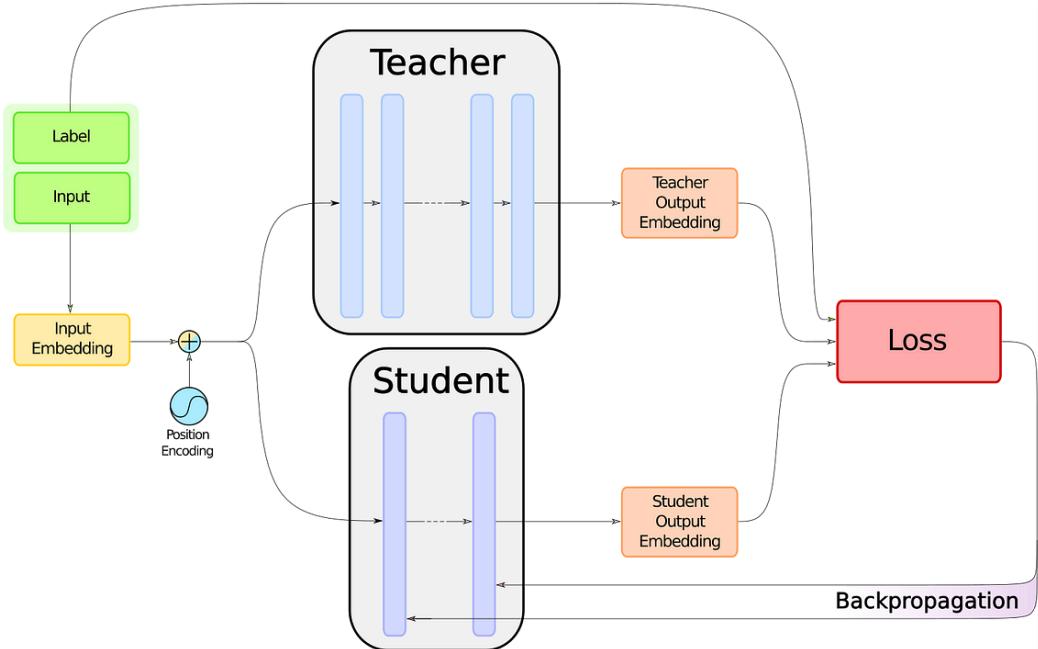
Pour régler les problèmes de performance et d'efficacité, plusieurs décisions ont été prises, notamment en fonction des recommandations découvertes [concernant le fine-tuning de modèles basés sur BERT](#):

- Réduire la taille du jeu d'entraînement
- Réduire le nombre d'époque lors de l'entraînement
- Passer à une version plus légère de BERT, avec moins de paramètres pour gagner en temps d'entraînement: DistilBERT

### 5.3.3. DistilBERT

#### 5.3.3.1. Présentation

DistilBERT est une version allégée de BERT, conçue pour être plus légère et plus rapide tout en conservant une performance significative. Il réduit le nombre de paramètres de BERT tout en utilisant une technique appelée "knowledge distillation" pour transférer les connaissances du modèle plus complexe vers le modèle plus léger.



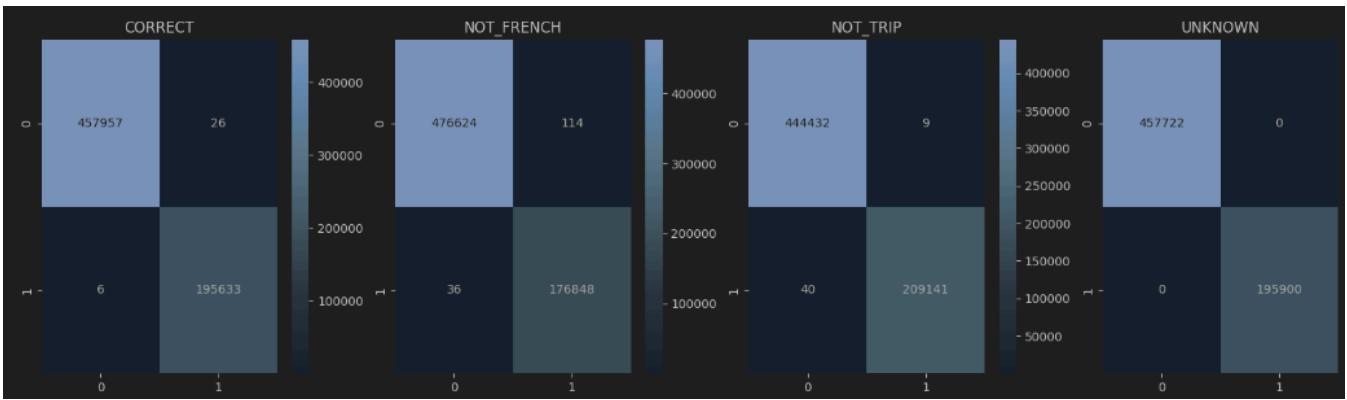
*Distillation des connaissances de BERT dans le modèle DistilBERT au cours de son entraînement*

La distillation de connaissances, utilisée dans DistilBERT, fonctionne en transférant les connaissances d'un modèle plus grand, comme BERT, vers un modèle plus petit. Cela implique de faire en sorte que le modèle plus petit (DistilBERT) **apprenne non seulement à reproduire les prédictions du modèle plus grand, mais aussi à capturer l'essentiel des informations apprises par ce dernier.**

La distillation permet de réduire considérablement le nombre total de paramètres dans DistilBERT par rapport à BERT. En effet, en se basant sur les connaissances du modèle plus grand, DistilBERT peut être créé avec un nombre significativement inférieur de paramètres tout en conservant une capacité de représentation efficace, ce qui le rend plus rapide à entraîner et à déployer tout en offrant des performances compétitives dans diverses tâches de traitement du langage naturel. [Selon la présentation du modèle sur Hugging Face, DistilBERT a 40% de paramètres en moins, s'exécute 40% plus rapidement et conserve 95% des performances de BERT.](#)

### 5.3.3.2. Application et résultats pour la classification de texte

Techniquement, le processus de *fine-tuning* de DistilBERT est presque similaire; ici on utilisera le modèle **distilbert-base-multilingual-cased**. Parallèlement à l'utilisation de DistilBERT, nous avons également réduit le jeu d'entraînement à 70 milles lignes et le nombre d'époques à 3. Ces modifications ont eu pour premier effet de **réduire le temps d'entraînement à 27 minutes**, ce qui est beaucoup plus viable et raisonnable.



Matrices de confusion de type One VS Rest, on observe quelques faux positifs et faux négatifs pour la classe NOT\_FRENCH

Les matrices de confusion sont également rassurantes sur le potentiel de notre modèle malgré un **jeu de données d'entraînement divisé par 8,5**. On pourrait croire à une hausse du nombre de faux négatifs et faux positifs pour le label NOT\_FRENCH notamment, mais cela peut s'expliquer par le fait que le jeu de données de test à lui été **multiplié à l'inverse par 8,5** et que beaucoup de phrases NOT\_FRENCH sont également classifiées NOT\_TRIP.

Nous avons également étayé l'extrait de phrases présent en dehors de notre jeu de données utilisé pour mesurer la confiance avec laquelle notre modèle réalise ses prédictions sur des données complètement inconnues (avec des formulations qui n'apparaissent pas forcément dans le jeu de données). En voici quelques unes:

Phrase	Attendu	Prédiction	Score de confiance
Peux-tu m'aider à trouver mon chemin vers Paris en partant d'Épierre ?	CORRECT	CORRECT	99,9%
I like apples	NOT_FRENCH / NOT_TRIP	NOT_FRENCH	87,5%
...	...	...	...
I would like to go to Paris from Lyon	NOT_FRENCH / NOT_TRIP	NOT_FRENCH	99,9%
ze zareazrreaz rzearzear	UNKNOWN	UNKNOWN	99,9%
Ça met combien de temps un Nantes Paris ?	CORRECT	CORRECT	99,9%

Sur un extrait de 25 phrases, toutes ont été correctement prédites avec un score de confiance de 99,9%, excepté la phrase "I like apples" à 87,5%. Cela est très probablement dû au fait que beaucoup de phrases sont classées à la fois NOT\_TRIP et NOT\_FRENCH, et qu'il est possible que le modèle hésite et aie des scores de confiance serrés pour ce genre de cas.

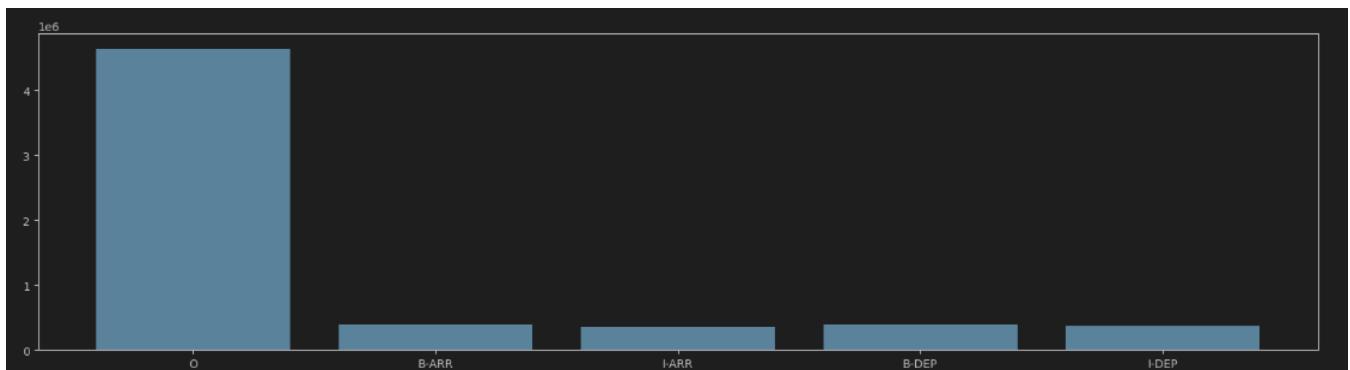
Avec toutes les améliorations apportées par rapport à l'expérimentation avec BERT, nous avons obtenu un modèle qui semble robuste et beaucoup moins demandant en temps et en ressources. Cette solution traite également en partie le problème du départ et de l'arrivée équivalent, complexe avec des algorithmes de Machine Learning classique. **Une possibilité d'amélioration est toutefois envisagée en utilisant cette fois-ci le modèle CamemBERT.**

[Le modèle a été publié sur Hugging Face.](#)

#### 5.3.3.3. Application et résultats pour la NER

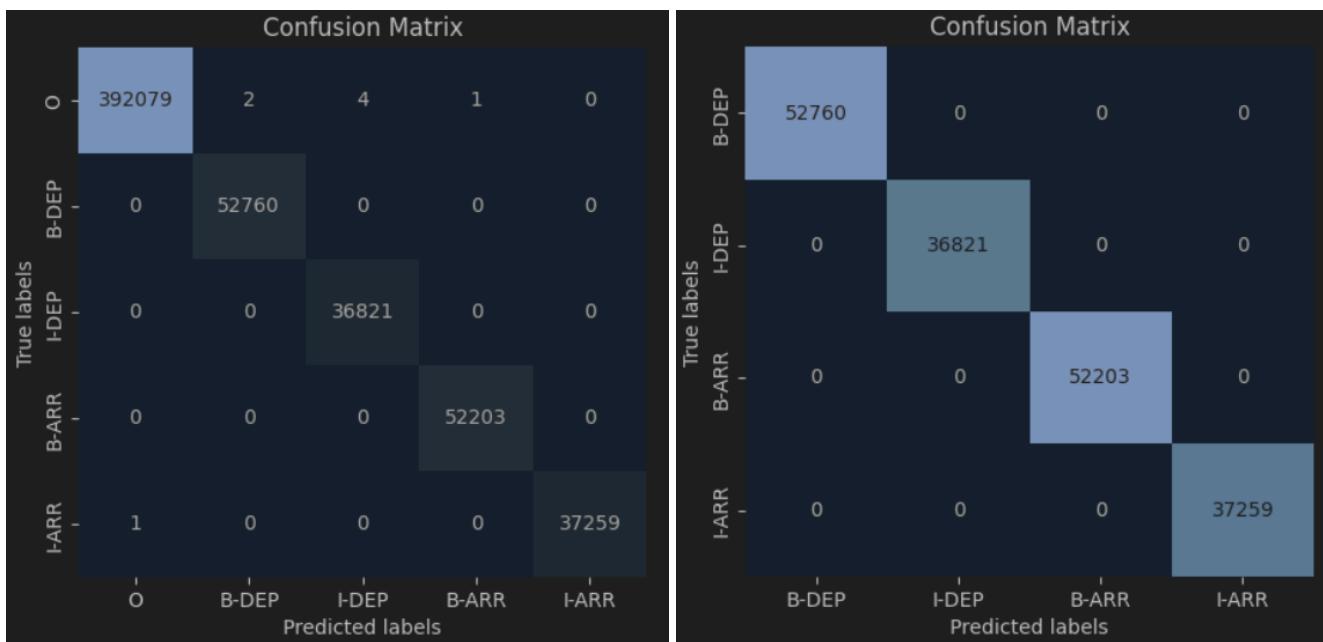
En parallèle de la classification de texte, nous avons également adapté notre script pour pouvoir l'**adapter à l'entraînement de DistilBERT pour une tâche de NER**. En effet, le modèle ayant présenté de très bon résultats dans cette configuration pour la classification de texte, il était intéressant de capitaliser sur ce premier travail d'optimisation pour voir si DistilBERT pouvait être aussi performant pour retrouver le départ et l'arrivée dans des commandes de trajet.

Les expérimentations pour la tâche de NER a été plus laborieuse, car nous avons dû peu à peu complexifier notre jeu de données pour pouvoir traiter plus de cas, mais aussi corriger plusieurs problèmes dans la génération de nos phrases et des tags NER associés. Cela a donné lieu à plus d'une dizaine de notebooks, ce qui renforce l'idée que construire un jeu de données est complexe et nécessite une attention particulière pour limiter le nombre d'entrainements.



Répartition des labels (tags NER) dans notre jeu de données

La version la plus aboutie du modèle de DistilBERT sur la tâche de NER combine à la fois la réduction de la taille du jeu d'entraînement, du nombre d'époques et plusieurs problèmes relatifs à la génération du jeu de données. On reste alors sur 3 époques et un batch size de 16. **On utilise environ 100 milles données d'entraînement.** Les données sont préparées de la même façon, avec une étape de tokenisation et de padding. Il suffit simplement d'indiquer au chargement du modèle sa tâche, à savoir la tâche NER.



Matrices de confusion produites sur les prédictions de notre modèle, à gauche avec le label “O” et à droite sans

	precision	recall	f1-score	support
O	1.00	1.00	1.00	392086
B-DEP	1.00	1.00	1.00	52760
I-DEP	1.00	1.00	1.00	36821
B-ARR	1.00	1.00	1.00	52203
I-ARR	1.00	1.00	1.00	37260
accuracy			1.00	571130
macro avg	1.00	1.00	1.00	571130
weighted avg	1.00	1.00	1.00	571130

Les résultats affichés dans la matrice de confusion se ressentent dans le rapport de classification

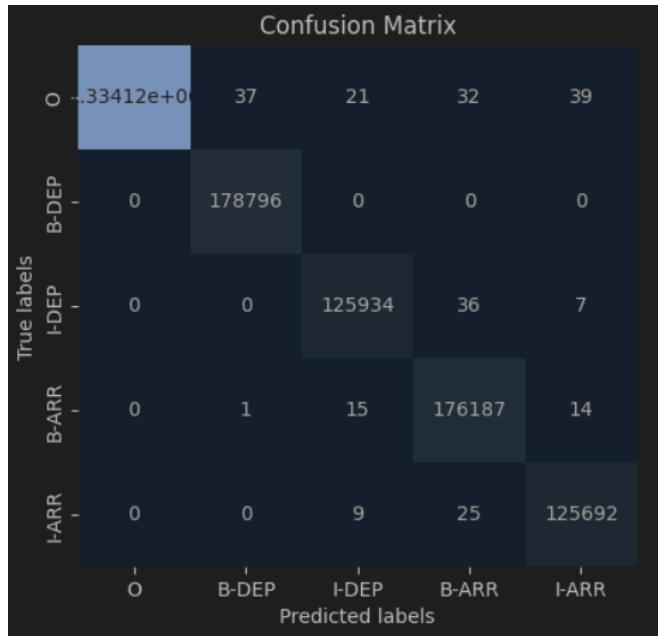
Pour une tâche et un jeu de données différents, DistilBERT fonctionne toujours aussi bien **pour un temps d'entraînement de 13 minutes**. Reste alors à lui soumettre des nouvelles phrases et observer si les départs et arrivés sont toujours aussi bien identifiés. Dans la majorité des cas, le départ et l'arrivée sont correctement identifiés, mais ci-dessous, un cas intéressant:

Phrase	Départ	Arrivée
Je veux aller de Port-Boulet à Le Havre.	Port-Boulet	Le Havre
Je veux aller de Port-Boulet au Havre.	Port-Boulet	au Havre
Je vais à Port-Boulet en partant du Havre	du Havre	Port-Boulet

On remarque que l'arrivée est correctement identifiée quand elle est précédée de à, mais pas de au. Pareil pour le départ avec du avec le nom de la ville, dans le cas où la ville commence par “Le”

(comme le Havre). En français, il est plus commun de dire “au Havre” ou “du Havre” que “à Le Havre”. Cette erreur est due à un manque de représentation de ce cas dans le dataset; [problématique similaire au “de” et “d”](#) qui avait déjà été traité précédemment et dont le problème avait ensuite disparu.

Mais une autre solution pour régler le problème serait de réduire le nombre de données d'entraînement pour éviter d'écraser les poids déjà entraînés de DistilBERT.



*Matrices de confusion produites sur les prédictions de notre modèle fine-tuned avec moins de données*

Avec 39 milles lignes (contre 100 milles), le problème a totalement disparu [sans trop impacter le résultat sur la performance générale et les phrases d'exemples utilisés](#). Cependant certains cas limites posent parfois problèmes. Certains cas limites sont mal gérés mais cela est dû à un manque de données, et des exemples de phrases avec des villes dont les noms n'étaient pas présents dans le jeu de données sont facilement reconnues.

**Le compromis sur le nombre de données d'entraînement utilisé pour plus ou moins influencer les poids de DistilBERT semble être une bonne façon de généraliser tout en spécialisant suffisamment le modèle.**

[Le modèle a été publié sur Hugging Face.](#)

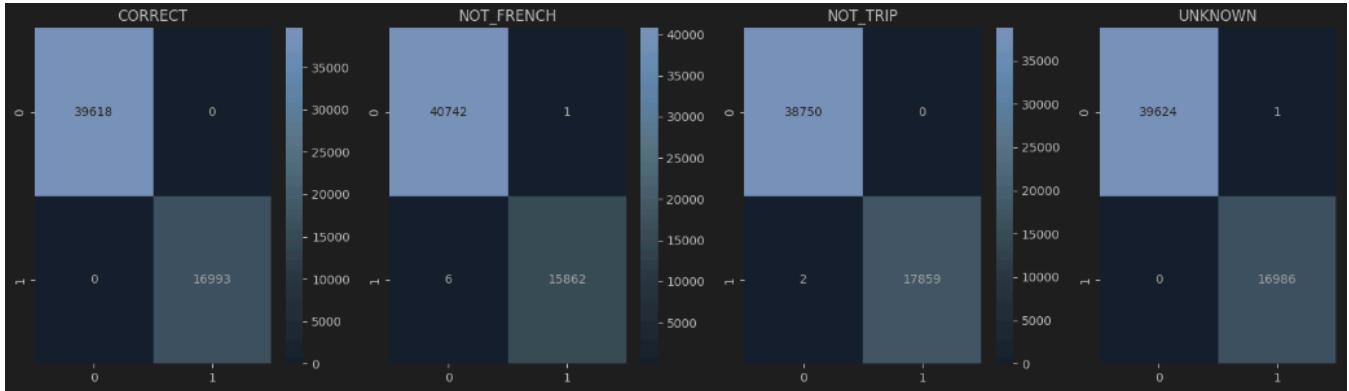
#### 5.3.4. CamemBERT

##### 5.3.4.1. Présentation

CamemBERT s'inspire de BERT, qui est un modèle NLP bidirectionnel très performant. Cependant, CamemBERT est pré-entraîné sur de grandes quantités de données en français, ce qui lui permet de mieux capturer les nuances et les spécificités de cette langue. Nous avons donc essayé de le mettre en œuvre pour voir si les résultats pouvaient être équivalents voire plus intéressants.

#### 5.3.4.2. Application et résultats pour la classification de texte

Pour ces expérimentations, on *fine-tune* le modèle **camembert-base** avec le même nombre de données que pour DistilBERT.



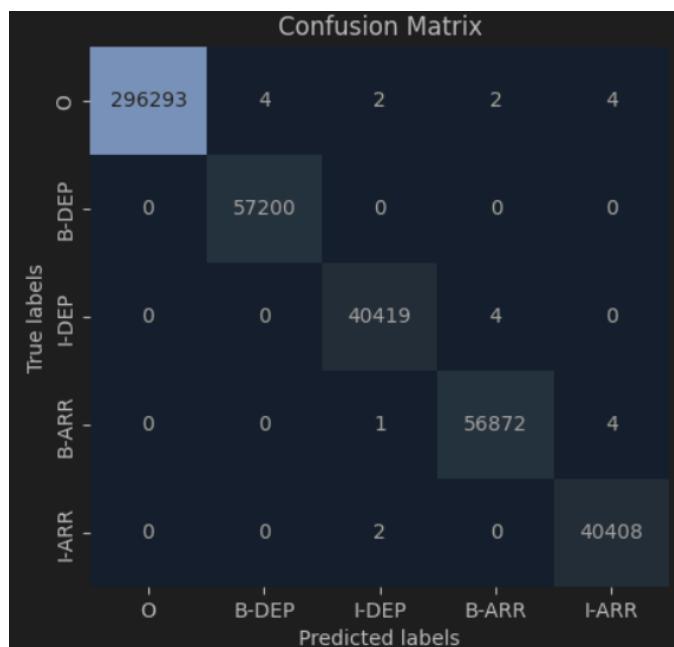
*Matrices de confusion produites sur les prédictions de CamemBERT fine-tuné pour une tâche de classification de texte*

**De la même façon que DistilBERT, les résultats sont très satisfaisants. Le temps d'entraînement est équivalent (30 minutes).**

CamemBERT semble également plus confiant sur certaines phrases. La phrase “I like apples” était prédite NOT\_FRENCH avec 87,5% de confiance par DistilBERT. **CamemBERT la classe également NOT\_FRENCH avec 99% de confiance.**

Dans notre contexte, CamemBERT semble être un meilleur choix que DistilBERT parmi les transformers testés. Cependant, on pourrait imaginer à termes que **l'application gère plusieurs langues**; CamemBERT étant spécialement entraîné sur un jeu de données français, ce dernier serait sûrement moins efficace pour différencier les différentes langues qu'un DistilBERT Multilingual.

#### 5.3.4.3. Application et résultats pour la NER



*Matrices de confusion produites sur les prédictions de CamemBERT fine-tuné pour une tâche de NER*

De la même manière que pour la classification de texte, les résultats sur le jeu de données de test sont très bons (autant que DistilBERT). Cependant, sur les phrases d'exemples utilisées, il arrive parfois que la ponctuation soit prise en compte dans l'extraction de l'arrivée.

Par exemple pour la phrase:

"Je veux me rendre chez mon ami Etienne à Saint-Étienne depuis Nantes.",

On obtient:

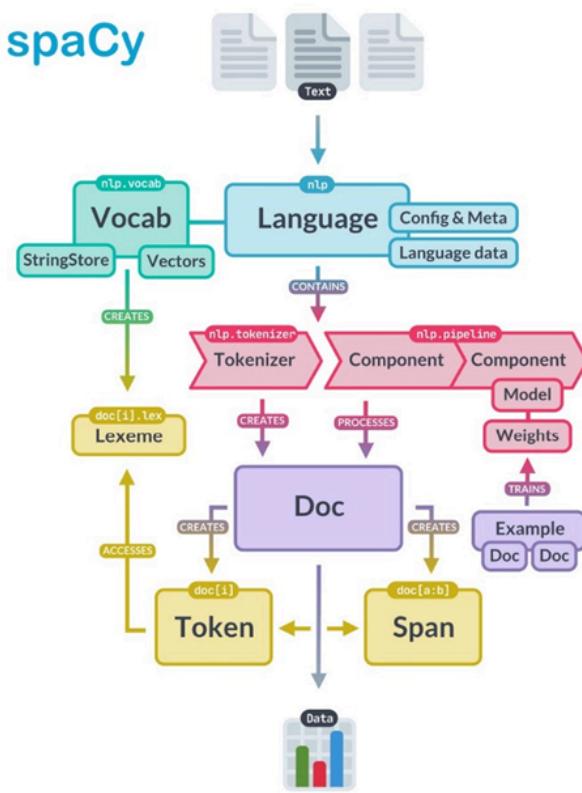
Départ: Saint-Étienne; Arrivée: "Nantes."

Ce problème n'apparaissait plus suite à [la correction du jeu de données associée](#), mais persiste avec CamemBERT, ce qui en fait un moins bon candidat que DistilBERT dans la sélection du modèle final.

## 5.4. SpaCy

### 5.4.1. Présentation

#### 5.4.1.1. C'est quoi Spacy ?

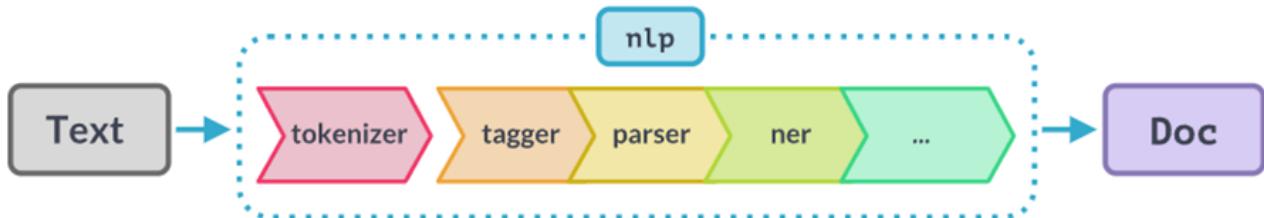


SpaCy est une bibliothèque pour le traitement du langage naturel en Python. Elle est conçue pour être rapide et efficace. Elle offre diverses fonctionnalités, comme la Tokenisation, la lemmatisation, la classification de texte, le Named Entity Recognition (NER)... En offrant des modèles pré-entraînés, c'est un excellent compromis entre performance et rapidité de mise en œuvre.

Dans spaCy, l'essentiel se concentre autour de trois éléments clés qui permettent de traiter et d'organiser l'information dans un texte. Le premier élément s'occupe de transformer le texte brut en une structure analysable. Le second stocke toutes les informations importantes comme les mots et leurs caractéristiques pour éviter les répétitions et économiser de la mémoire. Le troisième contient le

texte découpé en unités plus petites et leurs détails. Ce système assure que l'information est stockée de manière unifiée et efficace, facilitant le traitement et l'analyse du texte.

#### 5.4.1.2. Principes de SpaCy pour le NER



Comme défini auparavant, le NER permet d'extraire des informations structurées à partir de textes, afin d'identifier et classer les mots en catégories prédéfinies telles que les noms de personnes, les organisations ou, dans notre cas, les lieux de départ et d'arrivée. Le NER dans SpaCy est réalisé à l'aide d'un modèle statistique qui apprend à identifier les entités nommées à partir d'un grand corpus de texte annoté. SpaCy utilise des modèles de deep learning entraînés sur des données spécifiques à une langue pour reconnaître les entités. Ces modèles sont optimisés pour traiter des textes de manière rapide et précise.

#### 5.4.2. Application et résultats pour le NER

##### 5.4.2.1. Problématique

Nous voulons identifier dans une demande d'itinéraire les entités nommées de départ et d'arrivée quelle que soit la formulation.

##### 5.4.2.2. Hypothèses de travail

L'utilisation de SpaCy pour identifier les informations de départ et d'arrivée repose sur plusieurs hypothèses :

- Les données textuelles sont suffisamment riches en contexte pour permettre l'identification précise des lieux de départ et d'arrivée.
- Les modèles pré entraînés de SpaCy peuvent être efficacement personnalisés pour reconnaître ces entités spécifiques à travers le fine-tuning.
- La qualité de l'annotation des données d'entraînement joue un rôle crucial dans la performance du modèle.

##### 5.4.2.3. Choix du modèle

SpaCy offre un large choix de modèles pré entraînés. Nous avons sélectionné le modèle « fr\_core\_news\_sm ». Comme son nom le suggère, il est adapté à la langue française. Il possède une large gamme d'étiquettes et notamment le tag LOC pour tout ce qui concerne l'identification des entités de lieu comme les villes.

Londres LOC (/lɒndən/ ; en anglais : London LOC , /'lʌndən/ ) est la capitale et plus grande ville d' Angleterre LOC

Pour autant, cette étiquette ne répond pas à notre problématique car en plus d'identifier une ville nous voulons définir le départ et l'arrivée. Il nous faudra donc en créer de nouvelles et apprendre au modèle à reconnaître l'entité nommée liée.

#### 5.4.2.4. Modifications du modèle

Après avoir chargé le modèle pré entraîné, nous pouvons facilement ajouter des étiquettes au pipeline NER de la manière ci contre :

```
# Chargement du modèle pré-entraîné
import spacy
spacy.prefer_gpu(0)
nlp = spacy.load("fr_core_news_sm")

#Ajout des TAG d'entraînement au modèle (ARR, DEP)

resultArr = ner.add_label("ARR")
print(resultArr)
resultDep = ner.add_label("DEP")
print(resultDep)
```

#### 5.4.2.5. Préparation des données

Pour préparer les données en vue de leur utilisation avec SpaCy, il est nécessaire de procéder à certaines transformations suite à l'importation du fichier CSV. Avec l'introduction de SpaCy version 3, les données doivent être formatées en une liste d'objets Example.

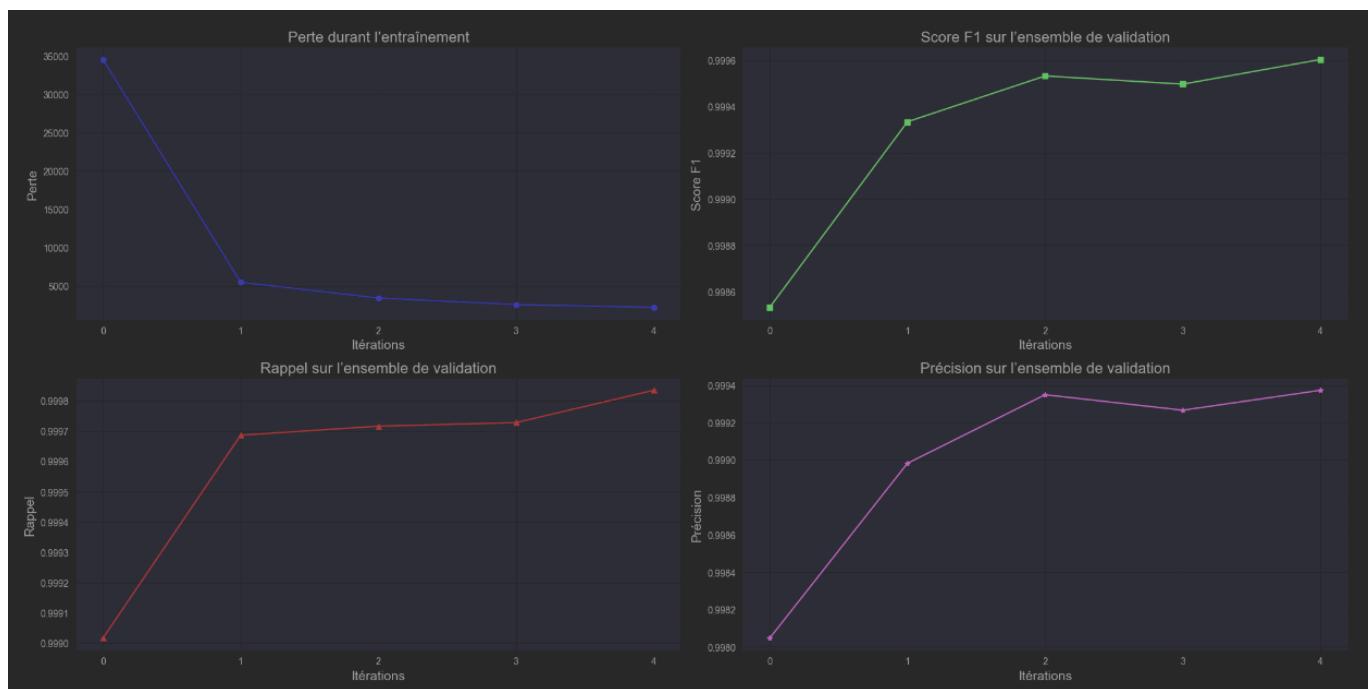
Durant ce processus de transformation, SpaCy effectue des vérifications d'alignement des étiquettes. Il ressort que, parmi environ 430000 lignes, 60 000 présentent des problèmes d'alignement. Conformément au message d'avertissement de SpaCy, ces enregistrements désalignés seront exclus du processus d'entraînement pour maintenir la qualité et la cohérence de l'apprentissage du modèle. Cependant, dans ce contexte, nous n'avons aucun contrôle sur les phrases exclues, ce qui peut introduire des lacunes dans l'apprentissage. De plus, cela soulève des questions quant à la fiabilité de la validation lors des différentes itérations d'entraînement.

```
D:\EpitechProjects\T-AIA-901\T-AIA-901\ai\venv\Lib\site-packages\spacy\training\io_b(Utils.py:149: UserWarning: [W030] Some entities could not be aligned in the text "Je recherche un itinéraire pour aller à CHOLET en partant d'AUXERRE." with entities "[((40, 46, 'ARR'), (61, 68, 'DEP'))]". Use `spacy.training.offsets_to_biluo_tags(nlp.make_doc(text), entities)` to check the alignment. Misaligned entities ('-') will be ignored during training.  
    warnings.warn(  
        f"Entity offsets could not be aligned in the text '{text}' with entities {entities}. Use  
        `spacy.training.offsets_to_biluo_tags(nlp.make_doc(text), entities)` to check the alignment.  
        Misaligned entities ('-') will be ignored during training.
```

#### 5.4.2.6. Entraînement

La phase d'entraînement se fait par itération et se concentre sur la surveillance des pertes, et ajustement des paramètres basés sur des métriques de performance. Une stratégie d'arrêt précoce est implémentée pour éviter le sur-entraînement, en s'arrêtant lorsque les améliorations cessent sur un nombre fixé d'itérations. Cette approche assure l'obtention du modèle le plus efficace sans gaspillage de ressources.

```
Losses at iteration 0 : {'ner': 34497.47272308348}
Époque 0, F1 Score: 0.9985315886079698, F1 Score for ARR: 0.9980691989559372 F1 Score for DEP: 0.9989880805299635
Losses at iteration 1 : {'ner': 5471.908303310023}
Époque 1, F1 Score: 0.9993337972078701, F1 Score for ARR: 0.9991411803998282 F1 Score for DEP: 0.9995237059207469
Losses at iteration 2 : {'ner': 3423.086401703304}
Époque 2, F1 Score: 0.9995320986265022, F1 Score for ARR: 0.999242510094895 F1 Score for DEP: 0.9998176674371687
Losses at iteration 3 : {'ner': 2560.9745151883158}
Époque 3, F1 Score: 0.9994965856663133, F1 Score for ARR: 0.9993259727168071 F1 Score for DEP: 0.9996647946131907
Losses at iteration 4 : {'ner': 2213.087181380045}
Époque 4, F1 Score: 0.9996031910546235, F1 Score for ARR: 0.9992961694919299 F1 Score for DEP: 0.9999059044930605
```



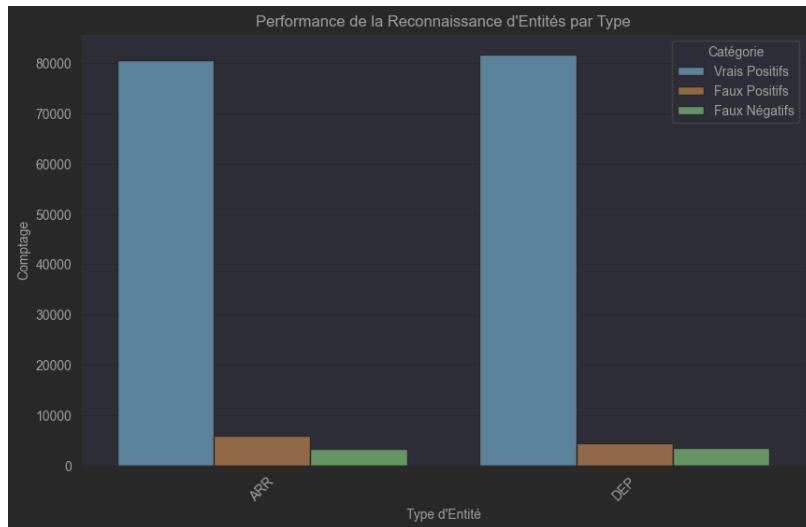
Ici nous utilisons plusieurs métriques clés pour évaluer la performance du modèle, notamment le score F1, qui fournit un équilibre entre la précision (la proportion des identifications correctes parmi toutes les identifications faites par le modèle) et le rappel (la proportion des identifications correctes parmi toutes les entités réelles). Les scores f1 obtenus durant l'entraînement semblent être très bons avec moins d'une erreur pour 1000 prédictions.

#### 5.4.2.7. Evaluation du modèle

Avec le dataset réservé à l'évaluation du modèle, les scores obtenus décroît nettement bien qu'il reste honorable :

- Score F1 = 0.9677
- Recall = 0.9603
- Precision = 0.9752

Label	Vrais Positifs	Faux Positifs	Faux Négatifs
ARR	80611	5872	3248
DEP	81661	4398	3456



En examinant attentivement les données, il apparaît que les faux positifs constituent la principale source d'erreur dans les prédictions. Bien que les faux négatifs soient en nombre inférieur, leur magnitude reste comparable.

Si on recalcule le score f1, précision et recall a partir des valeurs de vrais positifs, faux positifs et faux négatifs, on s'aperçoit qu'il y a un écart avec les premières métriques. En effet, les valeurs agrégées (0.9403, 0.9609, 0.9504) calculées diffèrent des métriques résumées (précision de 0.9752, rappel de 0.9603, et F1-score de 0.9677). Cette différence peut venir de la manière dont les métriques globales ont été calculées ou rapportées. Il est possible que les métriques globales fournies prennent en compte une agrégation pondérée basée sur le nombre total de vrais positifs, faux positifs, et faux négatifs, plutôt qu'une simple moyenne des scores par label.

```

{'departures': ['Port-Boulet'], 'arrivals': ['Le Havre']}
{'departures': [], 'arrivals': []}
{'departures': ['Paris'], 'arrivals': ['Nantes']}
{'departures': [], 'arrivals': ['Nantes', 'Nantes']}
{'departures': ['Le'], 'arrivals': ['Port-Boulet', 'Havre']}
{'departures': [], 'arrivals': ['Port-Boulet', 'Havre']}
{'departures': ['Paris', 'Épierne'], 'arrivals': []}
{'departures': ['Saarbrücken /Sarrebrück'], 'arrivals': ['Margny-Lès-Compiègne']}
{'departures': [], 'arrivals': ['Saint-Étienne', 'Nantes']}
{'departures': ['Tours'], 'arrivals': ['Marseille']}
{'departures': ['Lorient'], 'arrivals': ['Paris']}
{'departures': [], 'arrivals': ['Besançon', 'Oyonnax']}
{'departures': ['Paris'], 'arrivals': []}
{'departures': ['Troyes'], 'arrivals': ['Niort']}
{'departures': ['Troyes'], 'arrivals': ['Niort']}
{'departures': ['Troyes'], 'arrivals': ['Niort']}
{'departures': ['Paris'], 'arrivals': ['Nantes']}
{'departures': ['Paris'], 'arrivals': ['Nantes']}
{'departures': [], 'arrivals': ['Nantes', 'Montaigu']}

```

Enfin une dernière évaluation a été effectuée avec un panel de phrases sélectionnées et le résultat est très mitigé voir mauvais. Cela laisse à penser que le modèle pourrait souffrir d'une spécialisation trop importante ou que ce type de phrase est sous représenté dans le dataset.

#### 5.4.2.8. Conclusion

À première vue, il est complexe d'extrapoler à partir de ces résultats. Une analyse approfondie des "Examples" ayant conduit aux erreurs de prédiction est nécessaire. Il est également crucial de comprendre l'impact potentiel des problèmes d'alignement sur ces résultats. Plus précisément, il est important de déterminer si les "Examples" mal alignés sont exclus lors de l'évaluation. Explorer ces aspects offre plusieurs pistes pour améliorer la performance du modèle.

Dans tous les cas, ces travaux préliminaires ont montré que SpaCy offre un rapport investissement temps/coût très intéressant afin d'obtenir des résultats pertinents.

### 5.5. Refactorisation

Les recherches menées en parallèle par les membres du groupe sur la classification de texte ont pu donner lieu à de l'inspiration d'un notebook à l'autre. Ce phénomène est normal dans le cadre d'écriture de code complexe qui vise à faire apparaître des métriques similaires entre les combinaisons testées.

Est donc apparue la duplication de code; qui a légitimé une entreprise de refactorisation pour alléger les cellules de code, améliorer la maintenance et la lisibilité. Un fichier regroupant les imports les plus communs ainsi que des méthodes communes a donc vu le jour, pour être utilisé dans la plupart des notebooks.

#### Split du dataset en train, test et validation

```
In [4]:  
from sklearn.model_selection import train_test_split  
  
# Shuffle train dataset, and pick 50% of it  
dataset = dataset.sample(frac=0.5, random_state=0)  
X_train = dataset['text']  
y_train = dataset[labels]  
  
# To check how vectorizer works without UNKNOWN words being isolated  
X_train_without_unknown = dataset[dataset['UNKNOWN'] == 0]['text']  
y_train_without_unknown = dataset[dataset['UNKNOWN'] == 0][labels]  
  
# Split test dataset into 10% validation and 10% test  
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.2)  
X_test, X_val, y_test, y_val = train_test_split(X_test, y_test, test_size=0.5)  
  
print(X_train.shape, X_test.shape, X_val.shape)  
print(y_train.shape, y_test.shape, y_val.shape)
```

#### Split du dataset en train, test et validation

```
In [4]:  
X_train, X_test, X_val, y_train, y_test, y_val = split_dataset(dataset, labels, final_dataset_size=0.1)
```

#### Avant / Après refactorisation pour la partie de split du dataset

Pour conclure, ce travail nous a permis d'envisager de traiter plus de classifieurs puisque le développement de leur notebook dédié a été grandement accéléré. Quelques limites ont en revanche été rencontrées pour certains cas spécifiques.

Notons par exemple le cas du classifieur LinearSVC, pour lequel nous voulions, comme pour tous les autres; afficher les prédictions de probabilité pour chaque labels. Ce dernier se basant sur la marge plutôt que sur l'estimation probabiliste, il n'avait donc pas accès directement à la méthode `predict_proba()`, pourtant utilisée dans la méthode `show_samples_predictions()` que nous avons mise en place dans le cadre de cette refactorisation.

Comme pour les autres problèmes similaires que nous avons rencontré, l'idéal était donc de pallier ce problème depuis le notebook, pour faire changer les paramètres apportés à la méthode plutôt que cette dernière. Nous avons donc pu wrapper l'instance du classifieur avec la classe CalibratedClassifierCV, qui utilise une technique de validation croisée pour calibrer les sorties du modèle pour qu'elles correspondent à des probabilités. Cela a donc permis de ne pas altérer la méthode commune en faisant simplement une modification des paramètres en entrée.

## 6. Choix finaux

### 6.1. Benchmark sur la classification de texte

L'objectif de cette section est d'identifier le modèle de classification de texte le plus performant pour notre application. Pour cela, nous allons évaluer diverses combinaisons de méthodes de vectorisations et de classificateurs en les entraînant sur un jeu de données uniforme. Ce processus nous aidera à déterminer le modèle le plus adapté à nos besoins, en utilisant différentes métriques d'évaluation.

#### 6.1.1. Méthode d'expérimentation

Pour identifier le modèle de classification textuelle le plus performant, notre parcours a suivi différentes étapes pour assurer la rigueur et l'efficacité de notre analyse. Nous commençons par la collecte et l'organisation de nos données, veillant à équilibrer sa répartition. Cette étape nous permet d'éviter les pièges des biais qui pourraient fausser nos résultats.

Dans ce *benchmark*, nous nous sommes penchés sur les outils de vectorisation, en l'occurrence le CountVectorizer et le TfidfVectorizer avec un intervalle de N-grams 1-2. Nous avons également sélectionné des modèles de classification tels que la Régression Logistique, le Classifieur à Descente de Gradient Stochastique, ou le SVC Linéaire.

L'intégration de ces modèles et outils de transformation dans des pipelines nous a permis d'automatiser et d'optimiser le processus d'entraînement et de test. Cette approche systématique nous permet non seulement de gagner du temps mais aussi d'assurer la cohérence et la reproductibilité de nos expérimentations.

Chaque modèle est soumis à une phase d'apprentissage, au cours de laquelle **nous mesurons le temps d'entraînement**. Ce paramètre est essentiel pour évaluer l'efficacité opérationnelle des modèles, surtout dans un contexte d'entreprise.

L'évaluation des performances va au-delà des simples mesures quantitatives. Nous employons une gamme de métriques – **précision, rappel, score F1** – et procédons à des analyses qualitatives, à l'aide de matrices de confusion et des rapports de classification. Ces analyses nous permettent de sonder en profondeur les forces et les faiblesses de chaque modèle.

En plus de mesurer ces métriques, nous mesurons également **le temps de prédiction pour 10 milles phrases**, pour pouvoir avoir une idée de l'impact de cette durée sur le long terme. L'ensemble de ces métriques sera présenté dans des graphiques comparatifs qui faciliteront le choix parmi les modèles.

Notre démarche a pour but de déboucher sur le choix du modèle le plus adapté à nos besoins. Ce choix ne se résume pas à une simple comparaison de chiffres ; il s'agit plutôt d'une décision éclairée, prenant en compte un équilibre délicat entre **précision, rapidité d'entraînement, rapidité de prédiction et capacité à généraliser**.

### 6.1.2. Classifieurs

Suite à nos investigations initiales, nous avons échangé notamment avec ChatGPT afin de construire un ensemble de classifieurs à tester. Parmis ces *classifiers*, certains déjà utilisé pendant les expérimentations sont revenus:

- [La classification naïve bayésienne multinomiale](#)
- [La régression logistique](#)
- [Des algorithmes d'arbres de décision \(autres que Random Forest\)](#)

Nous avons agrémenté cette liste de nouveaux *classifiers*, ayant présenté de bonnes performances dans le benchmark:

- Le [LinearSVC](#)
- Le Perceptron
- Le classifieur Passif-Agressif, qui réajuste agressivement ses paramètres lors d'erreurs de prédiction et reste stable autrement
- Le classifieur Ridge, qui emploie la régularisation de Ridge pour contrer la multicollinéarité souvent présente dans les données textuelles, ce classifieur assure une stabilité des prédictions même en présence de fortes corrélations entre les caractéristiques.

D'autres *classifiers* ont été utilisés, mais ne font pas partie des *classifiers* notables:

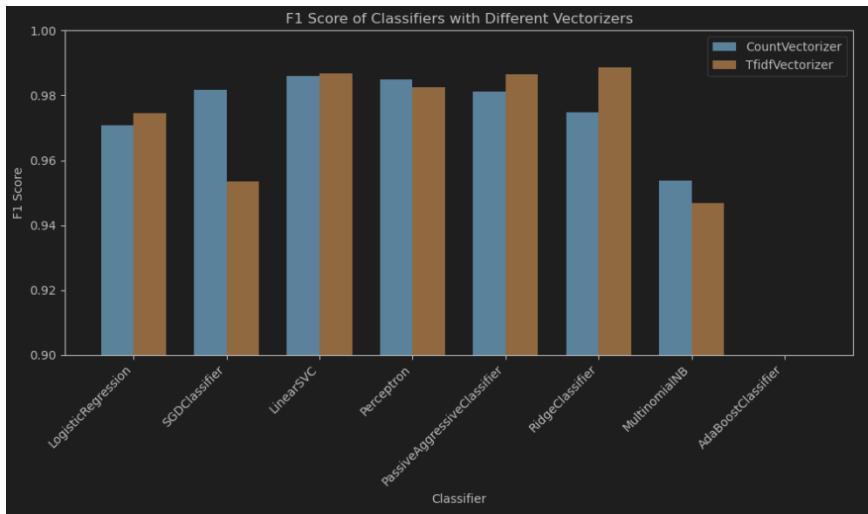
- La classification à descente de gradient stochastique
- Le AdaBoost

### 6.1.3. Résultats

En examinant les graphiques obtenus, nous pouvons tirer plusieurs conclusions concernant la performance des pipelines de classification :

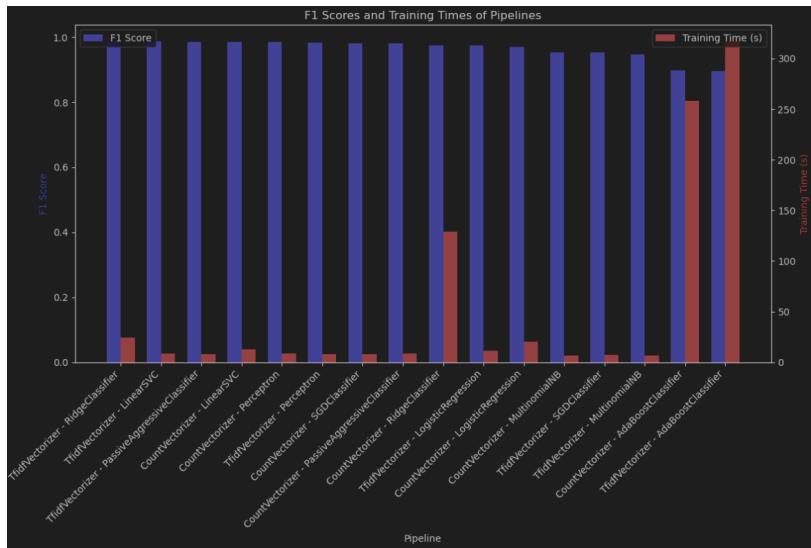
Le premier graphique *Scores F1 des classifiers avec différents vectoriseurs*, montre les scores F1 pour une gamme de classifieurs en utilisant deux techniques de vectorisation : CountVectorizer et TfidfVectorizer. On observe que la plupart des classifieurs ont des performances similaires avec une légère variabilité entre les deux vectoriseurs. Le **LinearSVC** et le **SGDClassifier** montrent des scores F1 particulièrement élevés, indiquant une bonne harmonie entre la précision et le rappel, et ce quelque soit le vectoriseur utilisé parmi les deux. Cet aspect pourrait amener à penser que ces classifieurs présentent une certaine robustesse, sur laquelle on pourrait s'appuyer pour efficacement généraliser sur de nouvelles données.

Cependant, il est intéressant de noter que pour certains classifieurs, le choix du vectoriseur a un impact significatif sur le score F1. Globalement, les résultats assez élevés semblent confirmer la pertinence du choix de ces classifieurs.



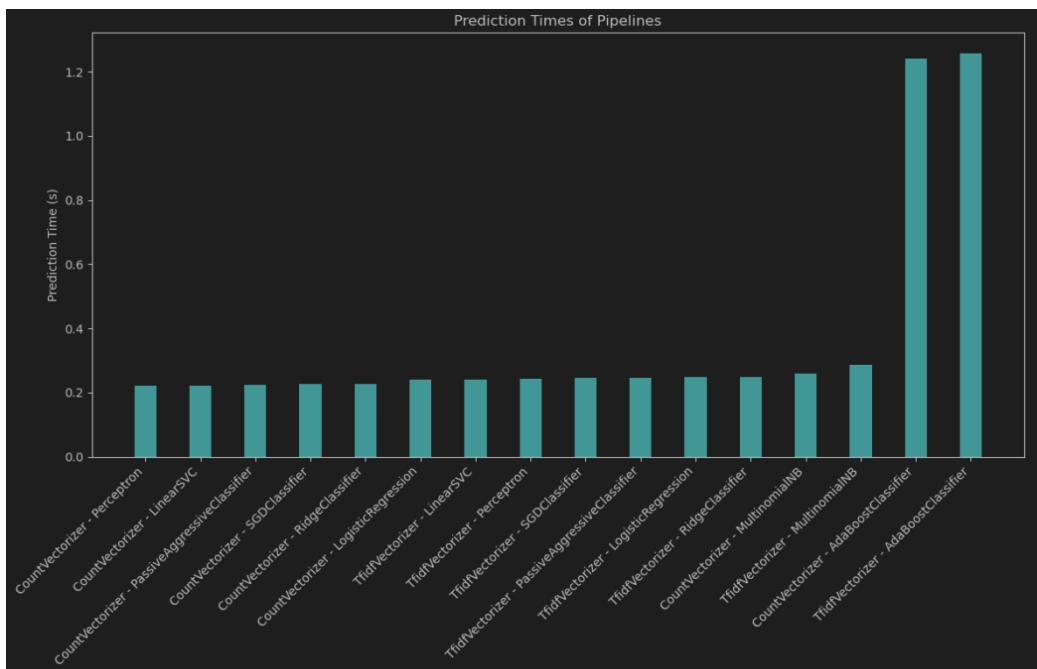
*Graphique de comparaison des f1 scores, issu du Benchmark comparatif des différents classifieurs*

Dans ce deuxième graphique, nous comparons les scores F1 à côté des temps d'entraînement pour chaque pipeline. Bien que les scores F1 soient en général élevés pour tous les classifieurs, les temps d'entraînement varient considérablement. Par exemple, certains classifieurs, bien que performants, prennent beaucoup plus de temps à s'entraîner, ce qui pourrait être un facteur limitant dans des contextes où la rapidité d'entraînement est cruciale. Dans celui de notre projet, nous pouvons nous baser sur les temps d'entraînement plus longs que les autres pour éliminer de notre sélection certaines combinaisons de classifieur/vectoriseurs.



*Graphique de comparaison des f1 scores et temps d'entraînement, issu du benchmark comparatif des différents classifieurs*

Ce dernier graphique illustre les temps de prédiction pour les différents pipelines. Il est clair que certains modèles présentent des résultats sensiblement similaires. Le temps de prédiction peut être un facteur critique dans des applications en temps réel ou lorsque de grandes quantités de données doivent être traitées. **L'AdaBoostClassifier** et le **MultinomialNB** se distinguent par des temps de prédiction nettement plus longs, ce qui pourrait être dû à la complexité des modèles ou à la taille des ensembles d'apprentissage. Pour le reste, on observe que le **Perceptron** et le **LinearSVC** obtiennent à nouveau des résultats similaires et parmi les meilleurs.



*Graphique de comparaison des temps de prédiction, issu du Benchmark comparatif des différents classifiants*

En conclusion, bien que les scores F1 soient un indicateur important de la performance des classifiants, ils doivent être évalués en conjonction avec les temps d'entraînement et de prédiction pour choisir un modèle qui équilibre efficacement la précision, la vitesse d'entraînement, et la rapidité de prédiction selon les besoins de l'application.

Les expérimentations menées nous ont permis de sélectionner le couple **TF-IDF / LinearSVC**, qui allient de très bon résultats sur le jeu de données de test avec une forte rentabilité en termes de temps d'entraînement et de prédiction.

## 6.2. Choix de DistilBERT pour la NER

Parmi les solutions disponibles que nous avons testé pour la NER, nous avons:

- DistilBERT
- CamemBERT
- SpaCy

Ces trois solutions possèdent chacune des avantages. SpaCy est léger et simple à mettre en place (également plus facile à annoter lors de la génération des données). Néanmoins, ces modèles ne profitent pas de l'architecture des transformers et poussera moins loin la compréhension du langage.

CamemBERT profite de l'architecture des *transformers* et des ajouts de BERT et fonctionne bien sur des données françaises, mais présente des faiblesses dans certains cas bien gérés par DistilBERT. De plus, dans l'idée où le projet permettrait d'analyser des phrases de plusieurs langues, CamemBERT aurait probablement un désavantage pour détecter les départs et les arrivées dans les phrases autres que le français par rapport à DistilBERT par exemple.

L'avantage de DistilBERT, en plus de ses très bonnes performances et de la bonne gestion de certains cas limites, est qu'il est plus léger que CamemBERT (67 millions de paramètres contre 111 millions de paramètres). Il pourrait aussi nous permettre de mieux se préparer à une évolution du projet avec une gestion multilingue. Enfin, il reste relativement rapide à entraîner et satisfaisant sur ses temps d'inférence.

DistilBERT est plus léger que BERT (donc CamemBERT), mais il nécessite toujours des ressources GPU significatives pour des performances optimales. SpaCy, en fonction du modèle choisi, peut nécessiter moins de ressources que les modèles de langage profond, ce qui le rend plus adaptable à des environnements avec des contraintes matérielles.

Pour le rendu du projet, nous avons décidé d'opter pour la solution présentant les meilleurs résultats, à savoir DistilBERT. Mais dans une réalité où notre solution serait mise à l'épreuve par beaucoup d'utilisateurs, la problématique des ressources serait un enjeu important à ne pas ignorer, et un modèle de SpaCy pourrait s'avérer plus stable en fonction de l'infrastructure disponible.

## 7. Conclusion

En conclusion, ce document de recherche a exploré sous ses différents aspects les multiples facettes du Traitement du Langage Naturel. À travers une démarche méthodique, nous avons abordé les enjeux liés à la reconnaissance vocale, à l'optimisation des algorithmes de recherche de chemin le plus court, et aux défis inhérents à la classification de texte et à la reconnaissance d'entités nommées (NER). Notre exploration des différentes techniques de Machine Learning, des réseaux de neurones récurrents (RNN, LSTM, GRU) aux modèles de transformers (BERT, DistilBERT, CamemBERT), a mis en lumière la complexité et la richesse du domaine du NLP.

Au cours de notre étude, nous avons également tenté de considérer notre problématique à la fois pour un projet d'école mais aussi pour un projet d'entreprise en réalisant un benchmark de plusieurs modèles sur la tâche de classification de texte, et en sélectionnant judicieusement les modèles utilisés dans notre application.

Cependant, notre étude s'est principalement concentrée sur la compréhension de texte, laissant entrevoir un horizon vaste et tout aussi captivant : la génération de texte. Cette autre branche du NLP, connue sous le nom de Natural Language Generation (NLG), ouvre des perspectives fascinantes pour l'avenir de l'intelligence artificielle. Le NLG, en permettant aux machines de générer du texte de manière autonome, promet de révolutionner la manière dont nous interagissons avec les technologies, des assistants virtuels aux systèmes de réponse automatique.

En somme, si notre travail a permis de jeter les bases d'une compréhension approfondie du traitement de texte, l'exploration du NLG représente une étape logique et excitante pour les recherches futures. L'intersection entre ces deux domaines, compréhension et génération de texte, ouvre la voie à des applications toujours plus sophistiquées et intégrées dans notre quotidien. La poursuite de cette exploration contribuera sans aucun doute à repousser les frontières de ce que nous considérons aujourd'hui comme les limites de l'intelligence artificielle.