Natural Language Processing Project

Groupe 7 SCIA, Promo 2026

Notre Projet

Sujet : Reconnaissance de commandes d'assistant domotique

alexis.petignat
baptiste.villeneuve
oscar.le-dauphin
lucas.juanico
max.nagaishi

Table des matières

1 Introduction

Dans le cadre du cours de Traitement Automatique du Langage Naturel (NLP), nous avons réalisé un projet portant sur la reconnaissance de commandes destinées à un assistant domotique. L'objectif était de classer automatiquement des phrases en langage naturel selon leur intention, en utilisant le jeu de données **MASSIVE** (*Multilingual Amazon SLU Structured Intent via Example*), proposé par Amazon Science sur la plateforme HuggingFace.

Le projet a été développé en Python, et nous avons expérimenté différentes approches pour la représentation et la classification du langage, telles que le *Bag of Words*, les *N-grams*, *TF-IDF*, *Word2Vec*, les *Neural Networks* et les *Transformers*. Ces méthodes ont donné des résultats variables selon leur complexité et leur capacité à capturer le sens des phrases, nous permettant d'évaluer leur efficacité respective dans un contexte applicatif concret.

En complément de la tâche de classification, nous avons également implémenté un système de génération de texte pour certaines des méthodes, permettant de produire automatiquement des commandes similaires à celles présentes dans le dataset, à partir d'un modèle pré-entraîné.

Afin d'améliorer l'interaction utilisateur et la visualisation des résultats, nous avons également développé une application en ligne de commande (*Terminal User Interface* ou TUI), permettant de tester les modèles de manière interactive et d'afficher les prédictions de façon claire et lisible.

2 Le Dataset

2.1 Présentation du dataset

Le dataset MASSIVE est un corpus multilingue contenant plus d'un million d'exemples d'*utterances* (phrases) répartis en 18 catégories (scenarios), chaque scenario se divisant en plusieurs intentions (intent) différents, le tout dans 18 langues. Ces phrases couvrent des scénarios typiques d'interaction avec un assistant vocal, tels que la gestion de la maison, les rappels, la météo, la musique ou encore les alarmes.

Dans le cadre de ce projet, nous avons choisi de nous concentrer uniquement sur les données en français, afin de simplifier le traitement multilingue et de nous focaliser sur les performances des modèles de classification.

2.2 Structure et Contenu du sous-ensemble `fr-FR`

Chaque instance du jeu de données `fr-FR` est structurée avec plusieurs champs. Les plus pertinents pour notre projet sont :

- `id` : Un identifiant unique pour l'énoncé.
- `locale` : Indique la langue et la région, ici `fr-FR`.
- `partition` : Spécifie si l'exemple appartient à l'ensemble d'entraînement (`train`), de validation (`validation`), ou de test (`test`).
- `utt` (utterance) : L'énoncé textuel brut formulé par l'utilisateur (par exemple, "réveille-moi à neuf heures du matin le vendredi"). C'est notre donnée d'entrée principale.

- `scenario` : Une catégorie générale ou un domaine auquel l'intention de l'utilisateur se rapporte. Il y a **18 scénarios distincts** dans le sous-ensemble `fr-FR`, tels que `calendar`, `play`, `weather`, etc.
- `intent` : L'intention spécifique de l'utilisateur au sein d'un scénario donné. Il y a **60 intentions distinctes** (par exemple, `calendar_set`, `play_music`, `weather_query`).

D'autres champs comme `annot_utt` (énoncé annoté avec les slots), `worker_id`, `slot_method`, et `judgments` sont présents mais moins centraux pour notre tâche de classification d'intention pure.

Notre projet adopte une approche hiérarchique pour la classification : nous cherchons à prédire d'abord le `scenario` général, puis, au sein de ce scénario, l'`intent` plus spécifique.

2.2.1 Taille du Jeu de Données (`fr-FR`)

Le sous-ensemble `fr-FR` est réparti comme ceci :

- Ensemble d'entraînement (`train`) : **11 514 exemples**.
- Ensemble de validation (`validation`) : **2 033 exemples**.
- Ensemble de test (`test`) : **2 974 exemples**.

Au total, cela représente 16 521 énoncés pour la langue française.

2.2.2 Statistiques sur les Énoncés (Ensemble d'entraînement `fr-FR`)

Une analyse préliminaire de la longueur des énoncés (en nombre de mots) dans l'ensemble d'entraînement révèle :

- Longueur minimale : 1 mot.
- Longueur maximale : 37 mots.
- Longueur moyenne : 7,46 mots.
- Longueur médiane : 7 mots.
- Écart-type : 3,53 mots.

Ces statistiques indiquent que les énoncés sont relativement courts, ce qui est typique des commandes vocales ou textuelles adressées à un assistant virtuel.

2.3 Forces du Jeu de Données

- **Pertinence pour la tâche :** Correspond parfaitement à notre objectif de classification.
- **Qualité des annotations :** Le dataset étant produit par Amazon Science, on peut s'attendre à une qualité élevé et à une bonne cohérence concernant la qualité d'annotation, les intentions et scenarios.
- **Structure hiérarchique (`scenario` et `intent`) :** Permet une modélisation plus fine et potentiellement plus précise des intentions utilisateur, ce que notre projet exploite.
- **Disponibilité et Accessibilité :** Facilement accessible et intégrable via la bibliothèque `datasets` de Hugging Face.
- **Taille conséquente pour `fr-FR` :** Avec plus de 11 000 exemples d'entraînement, le sous-ensemble français est suffisamment grand pour l'entraînement de modèles d'apprentissage.
- **Splits standardisés :** La présence de partitions `train`, `validation`, et `test` claires facilite l'évaluation rigoureuse et la comparabilité des modèles.
- **Licence permissive :** Distribué sous licence Creative Commons Attribution 4.0 International, ce qui autorise son utilisation pour la recherche et le développement.

2.4 Faiblesses et Limitations Potentielles

- **Déséquilibre des classes :**
  - **Scénarios :** Il existe un déséquilibre notable : dans l'ensemble d'entraînement, le scénario `calendar` compte 1688 exemples, tandis que `cooking` n'en a que 211.
  - **Intentions :** Le déséquilibre est encore plus prononcé au niveau des intentions. L'intention `calendar_set` a 810 exemples, alors que `cooking_query` n'en a que 4. Ce déséquilibre peut biaiser l'apprentissage des modèles en faveur des classes majoritaires et nuire aux performances sur les classes minoritaires.
- **Biais potentiels :** Comme tout jeu de données collecté à partir d'interactions humaines, il peut contenir des biais implicites liés à la démographie des utilisateurs, aux formulations courantes, ou aux capacités de l'assistant virtuel au moment de la collecte.
- **Couverture des énoncés :** Bien que vaste, le dataset ne peut couvrir l'infinie variété des formulations humaines, notamment les requêtes rares, ambiguës ou mal formulées.
- **Absence de contexte conversationnel :** Chaque énoncé est traité de manière isolée. Dans une conversation réelle, le contexte des échanges précédents est souvent crucial pour interpréter correctement l'intention actuelle.
- **Focus sur l'anglais à l'origine :** Bien que MASSIVE soit multilingue, il est basé sur un dataset qui est en anglais à la base. Les traductions ou adaptations en français pourraient ne pas capturer toutes les nuances spécifiques à la langue.

2.5 Analyse Exploratoire et Pré-traitement

2.5.1 Distribution des Classes

Comme évoqué dans les faiblesses, le jeu de données présente un déséquilibre entre les classes. Les figures 1 et 2 (**ces figures sont dans l'annexe à la fin du rapport**) illustrent cette distribution pour l'ensemble d'entraînement. On observe que quelques scénarios (comme calendar, play) et intentions (comme calendar_set, play_music) dominent en termes de nombre d'exemples, tandis que d'autres sont très peu représentés. Ce déséquilibre est un défi important pour l'entraînement des modèles, qui risquent d'être biaisés en faveur de ces classes surreprésentés.

2.5.2 Pré-traitement des Données Textuelles

Avant de pouvoir être utilisés par les différents modèles de classification, les énoncés textuels bruts nécessitent une étape de pré-traitement. Le texte est nettoyé et normalisé. Voici les étapes utilisées explicitement ou implicitement par les librairies :

- **Mise en Minuscule**
- **Tokenisation**
- **Suppression de la Ponctuation** : Implicitement géré
- **Pas de Suppression des Stopwords** Nous avons choisi de ne pas supprimer les stopwords car, dans le contexte de commandes courtes pour un assistant, ces mots peuvent parfois influencer subtilement le sens ou l'intention perçue.

Ces étapes permettent d'obtenir une représentation textuelle normalisée, prête à être convertie en vecteurs numériques par les différentes méthodes d'embedding et de vectorisation utilisées dans notre projet.

3 Notre implémentation

Pour chaque méthode, nous rappellerons brièvement en quoi elle consiste puis nous détaillerons les résultats de la méthode en question.

3.1 Bag Of Words

Bag Of Words est une méthode qui représente un corpus sous forme de fréquence d'apparition des mots dans un vocabulaire fixe, sans tenir compte de l'ordre. Notre implémentation est des plus classiques grâce à la librairie sklearn.
Cette méthode est la plus rapide à entraîner puisqu'elle consiste à compter naïvement les mots, et nous a permis d'obtenir un résultat tout à fait convenable avec un score de 0.81 sur les scénarios (voir Annexe). Nous avons aussi essayé cette méthode pour prédire l'intent en sachant le scénario, avec un taux de succès variant entre 0.45 et 1.

3.2 N-grams

N-grams est une méthode qui consiste à représenter un corpus sous forme de sous-échantillons du corpus de longueur N. Notre implémentation est faite main, et nous permet de créer des tables de N-grams pour tout N souhaité. Nous avons expérimenté et déterminé que faire des tables pour N = 2, 3, et 4 permettait une prédiction satisfaisante, sans prendre trop de temps à entraîner.
Pour prédire la catégorie d'une phrase, on cherche une suite de mots de l'input trouvable dans une de nos tables de N-grams, en valorisant les grams avec N le plus grand possible. Ainsi, la méthode permet un score de 0.69 (voir Annexe), ainsi qu'une précision entre 0.36 et 0.87 pour l'intent.
Cette méthode permet aussi la génération de texte basée sur le corpus fonctionnant sur le même principe que les chaînes de Markov que nous ne détaillerons pas ici.

3.3 TF-IDF

TF-IDF est une méthode de représentation statistique d'un corpus. Encore une fois, notre implémentation est tout à fait basique grâce à sklearn et permet un score de 0.85 sur les scénarios (voir Annexe), avec un taux de succès variant entre 0.51 et 0.97 pour les intents.

3.4 Word2Vec

Word2Vec est une méthode consistant à représenter des mots sous forme de vecteurs pour pouvoir appliquer une régression logistique sur le corpus. Pour cela, nous avons utilisé le modèle all-MiniLM-L6-v, pré-entraîné sur plusieurs langues et mis à disposition par l'API HuggingFace. Cela nous permet d'obtenir un score très correct de 0.894, au prix d'un temps d'entraînement conséquent.

3.5 FeedForward Neural Network

Nous avons aussi implémenté des modèles FeedForward, avec différents embeddings des données. Nous avons comparé les scores du réseau avec des embeddings Tf-Idf, avec Word2vec entraîné sur notre dataset, et un modèle Word2Vec pré-entraîné. Ce dernier est mis à disposition par Google, entrainé sur le dataset Google-News. L'embedding Word2Vec représentant chaque mot individuellement, nous avons utilisé la moyenne des mots de chaque entrée. En

expérimentant avec les hyperparamètres de chaque réseaux, voici les meilleurs résultats que nous avons obtenu pour chaque méthode :

Model	Hidden Size	Learning Rate	Epochs	Optimizer	Train Loss	Test Loss	Accuracy
tf-idf	100	1e-4	100	Adam	0.091	0.40	0.89
w2v (google)	1000	0.001	100	Adam	1.62	1.53	0.54
w2v (fasttext)	1000	0.001	100	Adam	0.37	1.26	0.74

Le tableau inclut la dimension de la couche cachée, le taux d'apprentissage, le nombre d'époques d'entraînement, l'optimiseur utilisé, les losses d'entraînement et de test et enfin la précision.
La méthode d'embedding Tf-Idf a donc obtenu les meilleurs résultats dans nos tests, comme nous pouvons le constater. C'est donc celle-ci qui a été retenue dans l'application.

3.6 RNN

Pour continuer, nous avons implémenté une seconde forme de réseaux de neurones, c'est à dire un modèle RNN. De la même façon, nous avons utilisé des embeddings Word2Vec, en rajoutant à ceux précédents un modèle préentrainé venant de FastText, entrainé sur un dataset en français spécifiquement. Nous avons également utilisé un simple embedding Onehot, attribuant un index a chaque mot du vocabulaire et mettant un 1 seulement à cette place de la matrice. Voici les résultats :

Model	Hidden Size	Learning Rate	Epochs	Optimizer	Train Loss	Test Loss	Accuracy
onehot	100	0.01	30	Adam	0.0094	0.80	0.84
w2v	100	0.1	30	SGD	0.64	1.66	0.55
w2v google	100	0.01	100	SGD	0.50	0.85	0.77
w2v fasttext	100	0.01	200	SGD	0.14	0.90	0.79

Nous avons aussi essayé de faire varier le nombre de couches du RNN, en observant peu de changements. La méthode la plus efficace a été l'approche simple de l'embedding Onehot. Ce résultat est plutôt surprenant, mais force est de constater que les modèles Word2Vec n'ont pas atteint la même précision dans nos tests.
On constate, à la fois dans le cas des FeedForwards et des RNNs que les modèles ont souvent de bien meilleures performances sur les données d'entraînement que sur les données de test, malgré les tentatives de limitation de l'overfitting. Nous avons en effet implémenté des couches de Dropout dans ces modèles précédents, qui ont tout de même amélioré les performances sur les données de test, mais n'ont pas non plus éliminé cet overfitting.

3.7 Transformers

Pour finir, nous avons implémenté la méthode des transformers, qui est une version chaînée des Neural Networks. Pour des raisons évidentes de temps d'entraînement, nous n'avons pas entraîné de sous-modèles pour cette méthode. Nous avons utilisé un modèle de base pré-entraîné "distilbert-base-uncased" pour définir le modèle, puis la classe Trainer des transformers HuggingFace. Nous avons obtenu un score de 0.911 sur les scénarios. A noter que ce score aurait pu être meilleur en travaillant davantage les hyperparamètres (le temps de training a rendu difficile l'expérimentation).

3.8 Benchmark des modèles de prédiction

Voici un tableau récapitulatif des performances de nos modèles de prédiction :

Method	Training time	F1 score (scenario)	F1 score (intent)
BoW	1s	0.81	0.784
N-grams	10s	0.69	0.65
TF-IDF	3s	0.85	0.845
Word2Vec	10m	0.894	0.876
FNN (100 epc)	30m	0.89	?
RNN (100 epc)	30m	0.84	?
Transformers	+1h	0.911	?

On remarque qu'à mesure que la méthode devient plus complexe, le temps de training explose. BoW a le meilleur rapport training-accuracy, suivi de près par TF-IDF. FNN, RNN et transformers permettent de tendre vers une accuracy très satisfaisante compte tenu de la nature de notre dataset, mais au prix d'un temps d'entraînement terrible. Les trois derniers modèles ne disposent pas de statistiques sur la prédiction d'intent pour des raisons citées plus haut.

3.9 Benchmark du Modèle de Génération N-grams

Nous avons évalué la capacité de génération de texte de notre modèle N-grams (implémenté dans `src/ngrams.py`) sur des prompts variés, afin d'observer sa cohérence et sa pertinence. D'autres architectures (TF-IDF, Word2Vec, FFNN, RNN, Transformer) ont été utilisées pour la classification, mais leur capacité de génération de text ne sont pas benchmarqués.

Modèle	**Prompt d'Entrée**	**Texte Généré**
N-grams	`mets la musique`	`mets la musique mélodique`
N-grams	`quel temps fait-il`	`quel temps fait-il où je suis fatigué de pandora s'il vous plaît publier sur facebook je suis`
N-grams	`allume la`	`allume la prise de cuiseur à riz olly as tu passé une bonne journée aujourd'hui que`

TABLE 1 – Benchmark du modèle N-grams (Ns=[2,3,4]) pour la génération de texte.

Ce benchmark montre que notre modèle N-grams peut générer des séquences pertinentes (comme "musique mélodique"). Cependant, il a tendance à dériver rapidement vers des séquences moins cohérentes ou inappropriées lorsque le prompt initial est moins contraignant ou que la génération se poursuit. Cela reflète les limitations de cette approche basée sur des séquences de mots fixes observées dans le corpus d'entraînement.

4 Extras

4.1 Interface Utilisateur

Une application a été réalisée avec le framework textual, elle permet à l'utilisateur de mieux visualiser les étapes de préparation :

- Chargement du dataset
- Chargement des modèles si existant, entrainement sinon
- Inference

La partie inférence permet à l'utilisateur de taper une phrase de test, et un tableau résumant le résultat de chaque modèle est mis à jour à chaque fois que l'utilisateur rentre une lettre. Cela nous a permis de mieux comprendre l'influence de certains mots sur les prédictions des modèles en itérant plusieurs fois sur une même phrase de test.

De plus, pour montrer l'efficacité de la prédiction de texte, une prédiction de l'input est affichée en gris et peut être insérée par l'utilisateur avec la touche entrée.

4.2 Sous modèles

Comme indiqué précédemment, notre dataset contient deux labels pour chaque entrée : un scénario, correspondant à la catégorie genérale de la phrase associée et un intent, qui est une classification plus précise qui complète le scénario auquel il est associé.
Nous avons donc adopté une approche unique, en entraînant un modèle pour prédire le scénario d'une entrée et 18 sous-modèles, qui prédisent l'intent de l'entrée en sachant son scénario. Ces modèles sont plus simples puisque le nombre d'intent pour chaque scénario varie entre 1 et 4. Il est donc techniquement plus simple de prédire l'intent que le scénario. Cette approche a cependant sucité quelques problèmes pour certaines méthodes, que nous aborderons plus bas.

4.3 Reconnaissance Vocale (Speech-to-Text)

Pour enrichir l'interaction utilisateur et permettre des commandes vocales, un module de reconnaissance vocale a été intégré au projet. Ce composant convertit les requêtes orales de l'utilisateur en format textuel, qui peut ensuite être analysé par les modèles de compréhension du langage naturel pour la classification d'intention.

4.3.1 Choix de la Technologie : Whisper

Pour la tâche de reconnaissance vocale automatique, nous avons choisi d'utiliser la bibliothèque **Whisper**, d'OpenAI. Whisper est un modèle pré-entraîné sur une grande collection de données audio multilingues (680 000 heures) provenant du web, ce qui lui confère une grande adapdabilité face à différents accents, bruits de fond et langues.

Les raisons principales pour lesquelles nous avons choisis Whisper sont :

- **Haute Performance et rapidité**
- **Robustesse**
- **Facilité d'Intégration via python**
- **Modèles de Tailles Variables ajustable pour nos besoins**

4.3.2 Mise en œuvre du Module de Transcription

Le processus de transcription vocale se déroule en trois étapes principales :

1. **Capture Audio :** Utilisation de la bibliothèque `soundcard` pour enregistrer l'audio. L'enregistrement est dynamique : il démarre avec la parole et s'arrête après une seconde de silence.
2. **Pré-traitement Audio :** L'audio capturé est converti pour être transcrit.
3. **Transcription via Whisper :** Le modèle Whisper pré-entraîné (taille `"small"`) est chargé via `whisper.load_model`. La méthode `transcribe` est appelée avec l'audio prétraité pour générer la transcription textuelle, qui est ensuite extraite du résultat. Le modèle small est utilisé car c'est le meilleur compromis qualité/vitesse.

4.3.3 Intégration et Limitations

Intégration dans le Flux Applicatif Le texte obtenu après transcription par Whisper constitue l'entrée directe pour les modèles de classification d'intention développés dans ce projet.

Limitations et Pistes d'Amélioration La principale limitation de ce module est :

- Le module de transcription pourrais être encore plus fiable afin de garantir son fonctionnement sur n'importe quel machine et avec n'importe quels types de périphériques audios

5 Limites de l'implementation

5.1 Reconnaissance vocale

Comme expliqué plus tôt dans la partie **Reconnaissance vocale**, ce module peut ne pas être robuste sur n'importe quelle machine ou n'importe quels périphériques audio

5.2 Temps d'entrainement

Notre Dataset étant particulier, nous avons voulu creuser davantage le problème en entraînant des sous-modèles pour prédire l'intent en fonction du scénario déjà prédit. Cela permet davantage de précision et ajoute un intérêt unique à notre projet. Cependant, cette pratique a un coût :puisqu'il y a 18 scénarios, nous devons entraîner 18 modèles de plus, ce qui implique des temps d'entraînement 19 fois plus longs, ce qui est gérable pour les méthodes les plus simples, mais il devient très vite compliqué d'appliquer le même procédé pour les méthodes les plus complexes comme les RNN, FNN et Transformers, qui peuvent mettre plusieurs heures à entraîner.

5.3 Temps de prédiction et de génération

Ce problème n'est pas dérangeant pour le cas d'usage classique de notre projet. Prédire la classe d'un input peut prendre quelques secondes sans que cela n'entrave le bon fonctionnement du projet. Générer le prochain mot en considérant les mots précédents prend au maximum 5 secondes ce qui est admissible également. En revanche, cela pose problème pour l'interface graphique. Celle-ci propose une autocomplétion automatique du prochain mot de l'input, mais il y a un délai entre le moment où l'utilisateur entre un mot et celui où l'interface met à jour le prochain mot. Le problème est similaire pour la prédiction de la classe puisqu'elle ne se mettra à jour seulement quand le modèle aura traité l'input, et cela peut prendre une dizaine de secondes.

6 Bibliographie

MASSIVE. https://huggingface.co/datasets/AmazonScience/massive
Enriching Word Vectors with Subword Information. https://fasttext.cc
OpenAI Whisper. https://arxiv.org/abs/2212.04356
Transformers : Huggingface lib. https://www.aclweb.org/anthology/2020.emnlp-demos.6/
Scikit-learn. https://scikit-learn.org/
SoundCard python lib. https://github.com/bastibe/SoundCard

7 Annexes

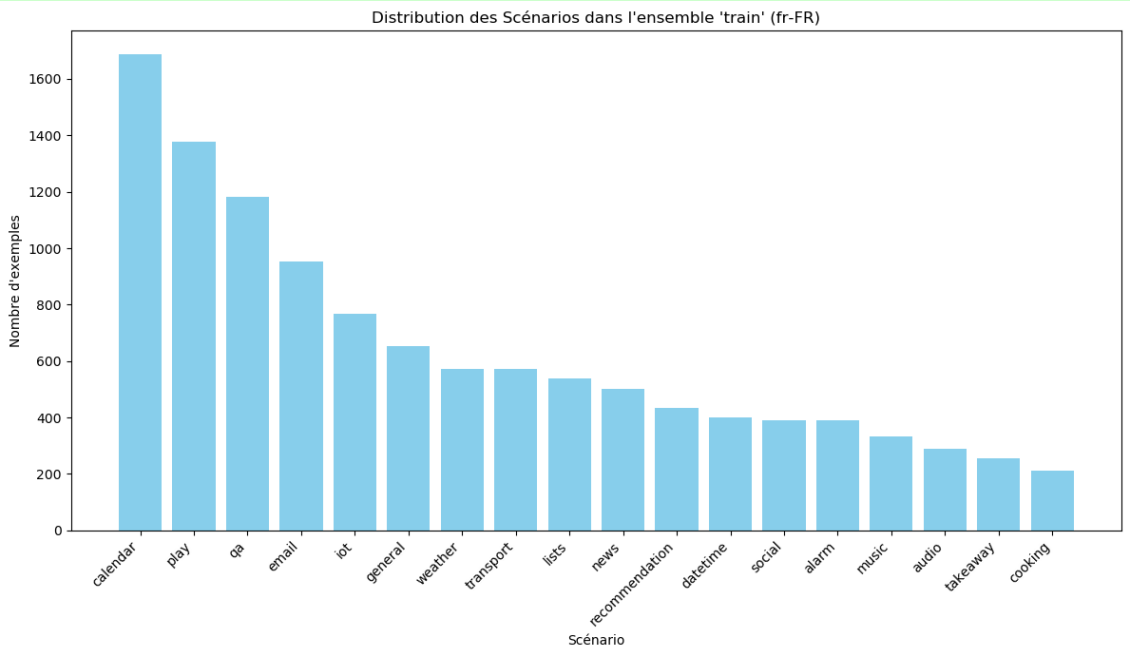

FIGURE 1 – Distribution des 18 scénarios dans l'ensemble d'entraînement `fr-FR`.

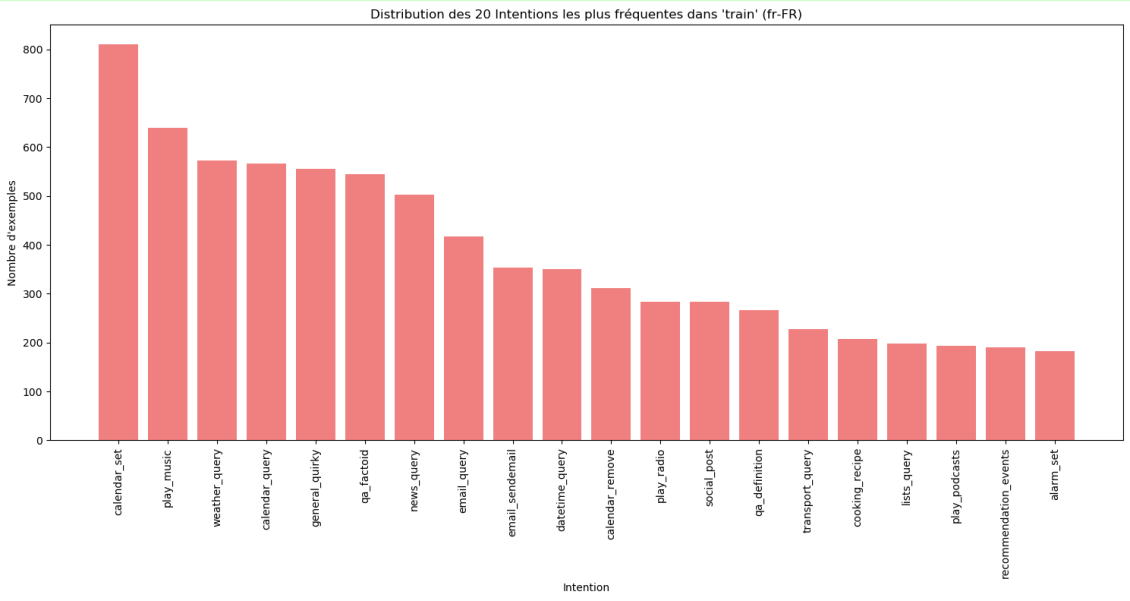

FIGURE 2 – Distribution des 20 intentions les plus fréquentes dans l'ensemble d'entraînement `fr-FR`.

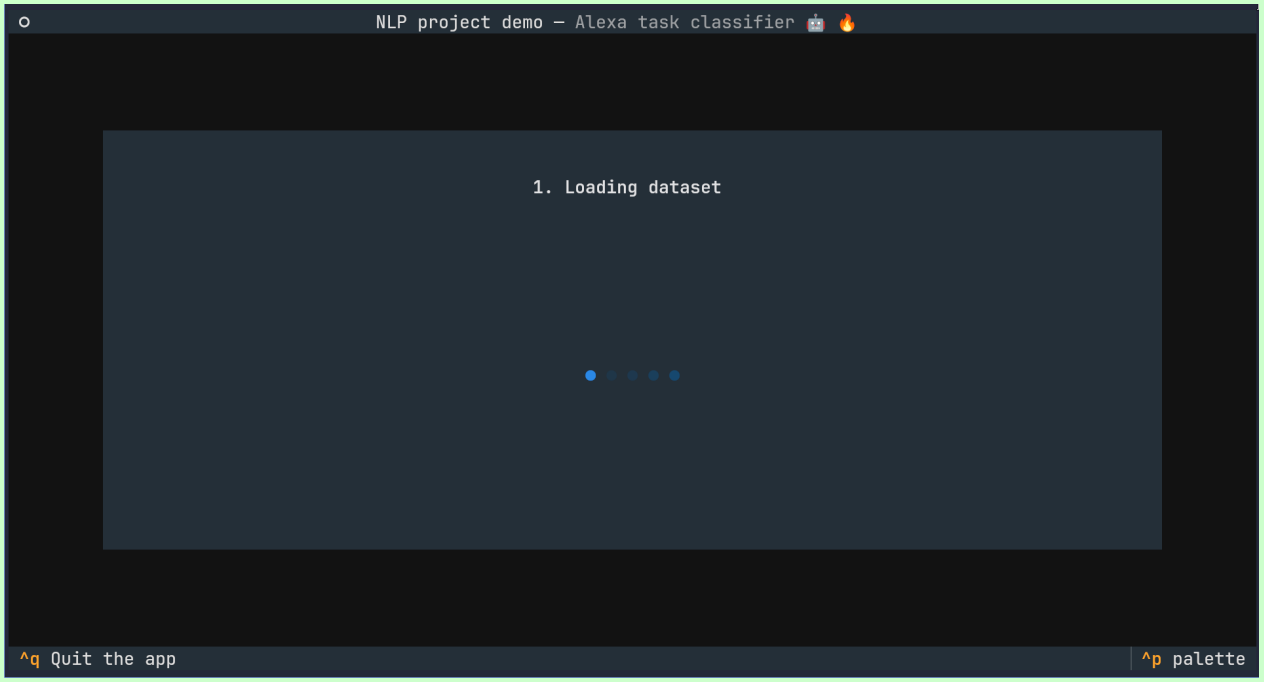

Figure 1 : Chargement du dataset affiché par la GUI

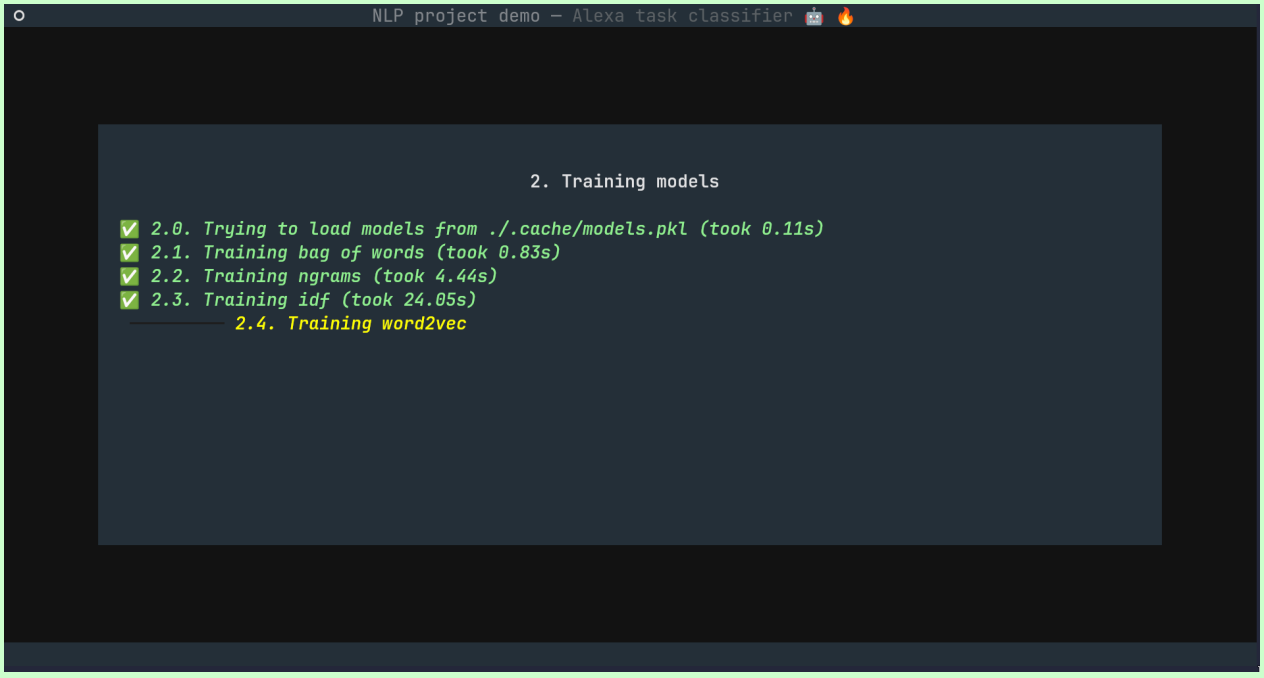

Figure 2 : Entrainement des modèles affiché par la GUI

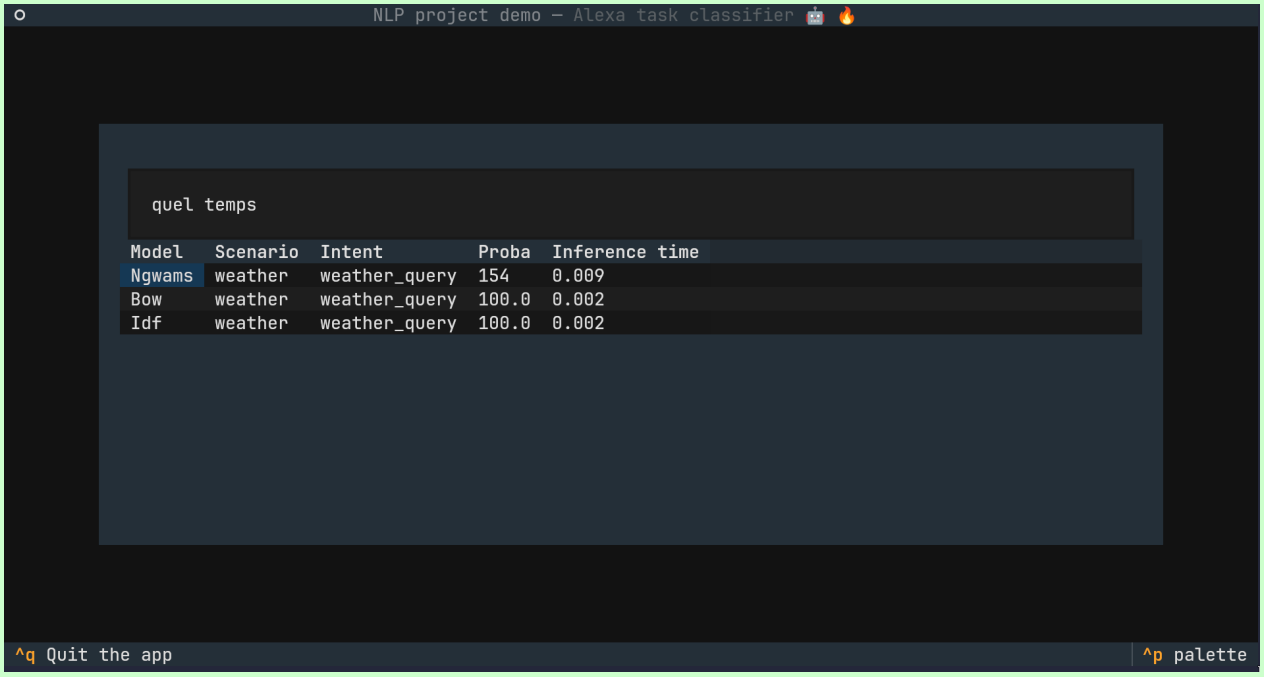

Figure 3 : Prédiction de la GUI

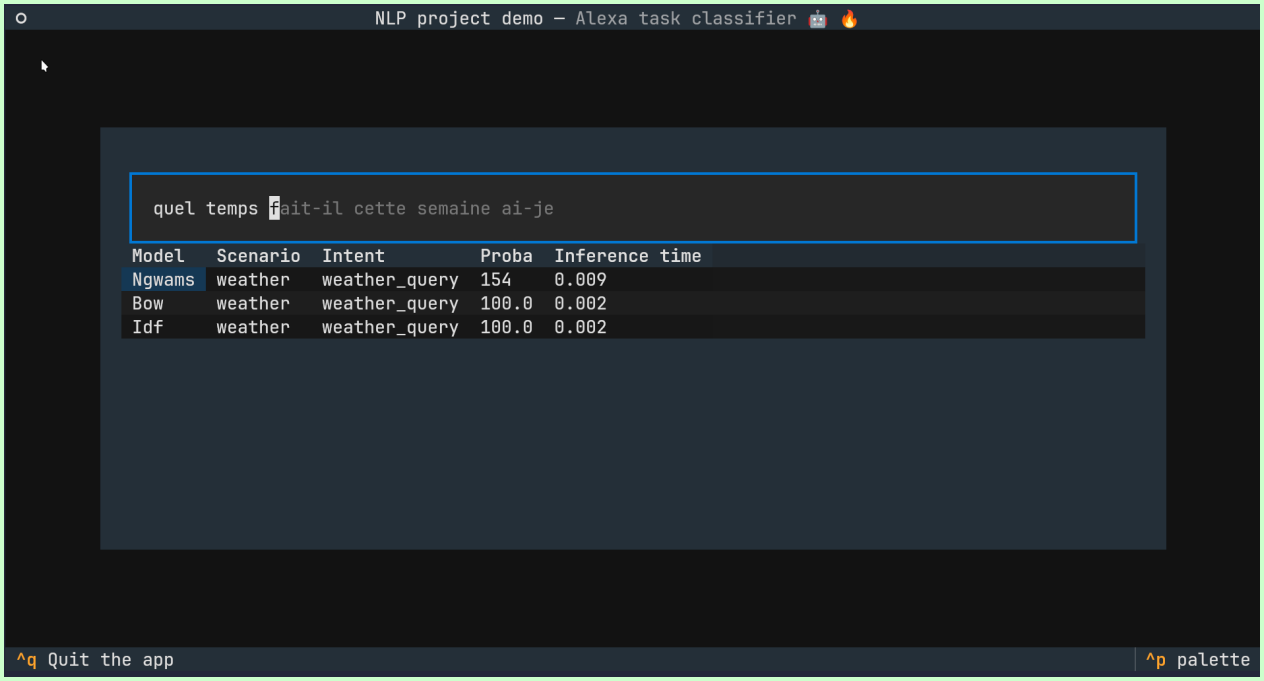

Figure 4 : Autocomplétion de la GUI

	precision	recall	f1-score	support
social	0.97	0.73	0.83	106
transport	0.89	0.90	0.90	124
calendar	0.68	0.96	0.79	402
play	0.81	0.96	0.88	387
news	0.73	0.77	0.75	124
datetime	0.90	0.71	0.79	103
recommendation	0.81	0.69	0.75	94
email	0.89	0.95	0.92	271
iot	0.94	0.94	0.94	220
general	0.66	0.34	0.45	189
audio	1.00	0.63	0.77	62
lists	0.86	0.86	0.86	142
qa	0.73	0.82	0.77	288
cooking	0.94	0.44	0.60	72
takeaway	0.97	0.68	0.80	57
music	0.94	0.40	0.56	81
alarm	0.99	0.79	0.88	96
weather	0.89	0.90	0.89	156
accuracy			0.81	2974
macro avg	0.87	0.75	0.79	2974
weighted avg	0.83	0.81	0.81	2974

Figure 4 : Classification report pour bag of words

	precision	recall	f1-score	support
social	0.95	0.76	0.85	106
transport	0.96	0.88	0.92	124
calendar	0.83	0.94	0.88	402
play	0.90	0.96	0.93	387
news	0.82	0.73	0.77	124
datetime	0.92	0.85	0.88	103
recommendation	0.77	0.73	0.75	94
email	0.97	0.93	0.95	271
iot	0.99	0.94	0.97	220
general	0.58	0.45	0.51	189
audio	0.98	0.79	0.88	62
lists	0.92	0.87	0.90	142
qa	0.62	0.88	0.73	288
cooking	0.94	0.67	0.78	72
takeaway	0.98	0.79	0.87	57
music	0.90	0.65	0.76	81
alarm	0.99	0.92	0.95	96
weather	0.92	0.93	0.93	156
accuracy			0.85	2974
macro avg	0.88	0.82	0.84	2974
weighted avg	0.86	0.85	0.85	2974

Figure 4 : Classification report pour TF-IDF

	precision	recall	f1-score	support
social	0.29	0.48	0.36	106
transport	0.76	0.68	0.71	124
calendar	0.61	0.84	0.71	402
play	0.68	0.82	0.74	387
news	0.58	0.69	0.63	124
datetime	0.79	0.70	0.74	103
recommendation	0.60	0.56	0.58	94
email	0.80	0.79	0.79	271
iot	0.94	0.82	0.87	220
general	0.61	0.22	0.33	189
audio	0.94	0.48	0.64	62
lists	0.74	0.80	0.77	142
qa	0.63	0.71	0.67	288
cooking	0.97	0.49	0.65	72
takeaway	0.80	0.56	0.66	57
music	0.88	0.35	0.50	81
alarm	0.80	0.68	0.73	96
weather	0.91	0.63	0.75	156
accuracy			0.69	2974
macro avg	0.74	0.63	0.66	2974
weighted avg	0.72	0.69	0.68	2974

Figure 4 : Classification report pour les ngrams