Rapport de projet intégration - Equipe 2
Maison intelligente

Réalisé par Adrien Filmotte - Alexandre Langlade - Aniss Louahadj - Enzo Corradi - Lledos Guilhem - Bahroun Hamza
3a SRI - UPSSITECH - Paul Sabatier

SOMMAIRE

Introduction

Le but de ce projet est la mise en œuvre d'un assistant vocal pour la gestion à distance d'une maison intelligente. L'utilisateur ciblé par ce projet est le conducteur d'un véhicule. Au travers de cet assistant vocal incorporé dans le véhicule, l'utilisateur peut gérer la maison dans des aspects domotiques, domestiques et familiaux.

Pour ce faire, nous étions une équipe de 6 membres travaillant à la fois sur le côté chatbot / compréhension du dialogue, communication entre les entités et accessibilité de la solution proposée.

Architecture générale

Dans cette partie nous allons détailler l'architecture globale de notre assistant vocal.

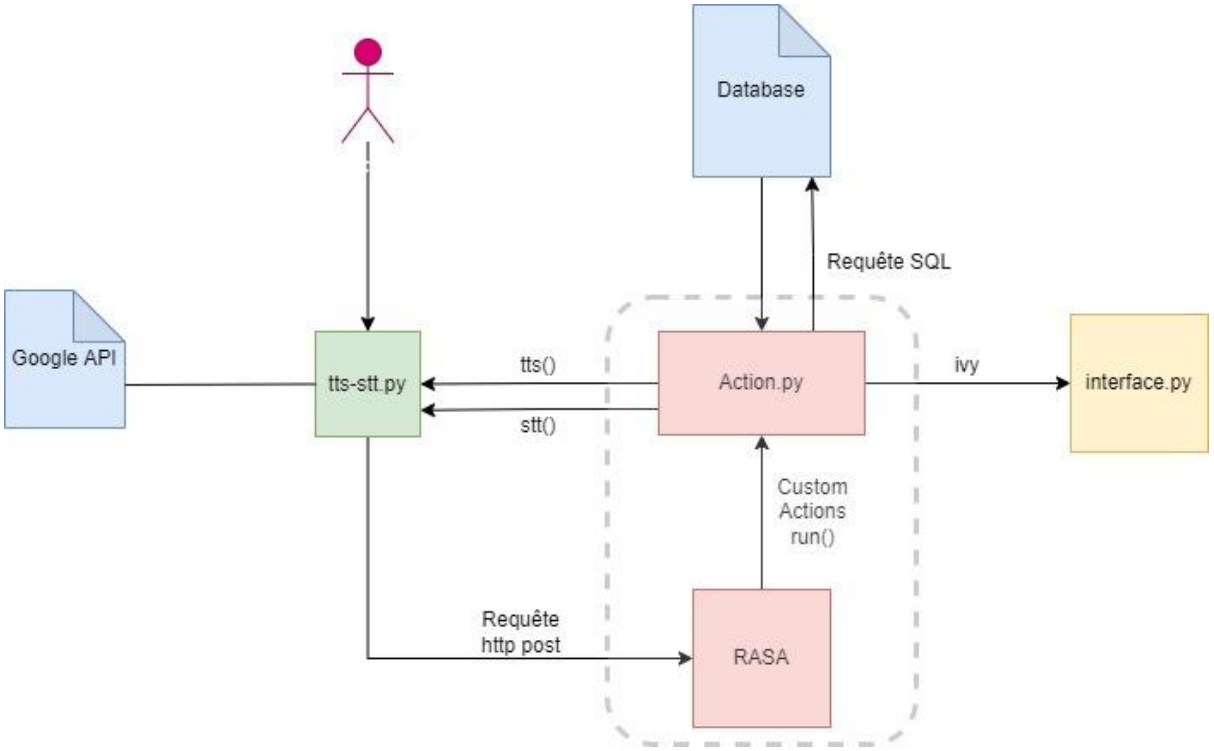

Figure 1 : Architecture globale du projet d'intégration

Comme nous pouvons le voir sur la *Figure 1,* l'action part de l'utilisateur. Lorsqu'une activité vocale est détectée, la fonction speech_to_text du fichier tts-stt.py est activée. Celle-ci convertit la parole en texte pour qu'elle soit envoyée au chatbot RASA interne pour la compréhension. Cette conversion utilise l'API Google au travers de la bibliothèque speech recognition python. Cet envoi est fait via la création du custom connector RASA qui permet de diriger une requête HTTP Post au bon endroit, sur le point d'entrée du chatbot RASA. Les données sont encapsulées dans un format JSON (exemple de requête ci-dessous).

```
#ENVOI DE LA PAROLE A RASA
        url = 'http://localhost:5005/webhooks/myio/webhook'
        payload = {'sender': 'test_user','message': parole,'metadata': {}}
        headers = {'Content-Type': 'application/json'}
        requests.post(url, data=json.dumps(payload), headers=headers)
```

Une fois que le chatbot RASA reçoit le texte prononcé par l'utilisateur, il peut alors faire son traitement et lancer la custom action (du fichier actions.py) correspondante. Ce fichier répertorie trois actions correspondant aux trois scénarios. Leur architecture est similaire, un arbre de décision est réalisé en fonction des valeurs des slots remplis envoyés par RASA.

Depuis ces custom actions, nous pouvons influer sur la base de données avec des requêtes SQL, influer sur l'interface graphique via l'envoi d'une commande sur une connection IVY (avec comme paramètres le device et l'action) ou alors faire un retour vocal à l'utilisateur en faisant appel à la fonction text_to_speech du fichier stt-tts.py.

Exemple de commande envoyée sur le bus Ivy.

```
connexion.send_msg("MAISON device=radiateur action=off")
```

Use cases (scénarios)

Plusieurs scénarios ont été pensés et développés pour ce projet. Nous avons tout d'abord souhaité réaliser un scénario de contrôle de la lumière et du chauffage dans les pièces de la maison. Dans ce scénario, l'utilisateur doit pouvoir, par commande vocale, allumer ou éteindre les lumières de la chambre et du salon.
Il peut ainsi, demander d'allumer ou éteindre la lumière d'une seule pièce, ou bien, le faire pour l'ensemble des pièces de la maison. Dans ce scénario, l'utilisateur peut également réaliser la gestion du chauffage par commande vocale. Il peut ainsi allumer ou éteindre le radiateur de la chambre, ainsi que régler la température de ce dernier. Toutes ces actions, une fois effectuées, renvoient une confirmation vocale à l'utilisateur.
Un exemple d'utilisation pourrait être le suivant :

- **User** : "Ok Google"
- **Système** : "Bonjour, que puis-je faire pour vous ?"
- **User** : "Allume les lumières"
- *Allumage des lumières*
- **Système** : "J'ai allumé les lumières de la maison"

Le second scénario concerne l'envoi de messages. Dans notre situation, l'utilisateur a la possibilité d'envoyer un message à "Papa" ou "Maman". Il peut ainsi dicter son message par commande vocale, avant que ce dernier soit envoyé au destinataire souhaité.
Le principe de l'utilisation reste le même que précédemment. Par exemple :

- **User** : "Ok Google"

- **Système** : “Bonjour, que puis-je faire pour vous ?”
- **User** : “Envoi un message à Maman”
- **Système** : “Que voulez-vous dire ?”
- **User** : “Bonjour, comment vas-tu ?”
- *Envoi du message*
- **Système** : “Le message a été envoyé à Maman”

Enfin, le dernier scénario que nous avons développé pour notre chatbot est une gestion de la liste de courses. Nous avons ainsi rendu disponibles plusieurs actions pour l’utilisateur. Tout d’abord, ce dernier peut ajouter des articles sur la liste. Il peut ainsi énoncer un ou plusieurs articles ainsi que leur quantité. Ces derniers seront ajoutés à la liste avec une confirmation vocale. Il dispose également de la possibilité d’effacer la liste de courses ou encore de demander au chatbot de lui renseigner vocalement la liste de courses. Le principe d’utilisation de ces dernières commandes reste une fois de plus le même que précédemment :

- **User** : “Ok Google”
- **Système** : “Bonjour, que puis-je faire pour vous ?”
- **User** : “Ajoute 2 pains et 5 bananes à la liste de courses”
- *Ajout des articles*
- **Système** : “Les articles ont été ajoutés à la liste”

Accessibilité

Dans cette partie, nous allons traiter de comment nous avons abordé les problématiques d’accessibilité dans ce projet. Une des contraintes majeures d’accessibilité de ce projet est que l’utilisateur est conducteur d’un véhicule actif, et ne peut donc pas quitter la route des yeux. L’interaction ne peut se faire que vocalement, il est impossible d’utiliser des signaux lumineux ou du texte écrit. Pour ce faire, nous avons choisi comme point d’entrée et de sortie de notre système de la voix. A l’aide de la technologie “speech to text” en entrée et “text to speech” en sortie, l’utilisateur n’est jamais déconcentré de sa tâche principale qui est la conduite.

Aussi, nous avons réfléchi à d'autres problématiques d’accessibilité qui n’ont pas été implémentées dans notre proof of concepts. Ici, nous portons une attention particulière au fait que l’utilisateur n’est pas forcément tout seul dans la voiture. Nous avons pensé à intégrer un seuil sur l’énergie de la parole détectée en entrée pour ne pas prendre en compte les passagers du véhicule et ainsi se concentrer sur la parole de l’utilisateur (le conducteur). Au travers de la bibliothèque python que nous avons utilisé (speech recognition) cela se fait facilement. Dans le cas de scénarios de réception de message, nous avons pensé au fait de demander confirmation à l’utilisateur avant de lire le message à voix haute par l’assistant vocal, afin de respecter des normes de confidentialité.

Enfin, en termes de limites d’accessibilité, nous devons revoir la partie de reconnaissance vocale afin de mieux comprendre et interpréter ce que l’utilisateur dit, puisque nous avons remarqué que selon l'intonation et l’accent de la personne, notre

assistant vocal peut avoir du mal à comprendre la phrase prononcée. Tout ceci vise à ce que le maximum d'utilisateurs possible puisse utiliser ce projet en réfléchissant aux différences de tout un chacun.

Evaluation du chatbot

Après avoir testé et évalué plusieurs stories de tests, nous avons pu obtenir divers résultats, présentés ci-après.

Evaluation des Intents :

Nous avons pris la décision de créer un intent par scénario et de gérer ensuite l'action à réaliser grâce au code python appelé par le chatbot. De ce fait, seuls 3 intents doivent être reconnus par notre chatbot : action_maison, action_course et envoyer_message.
Après test de ces intents (voir figure 2), nous pouvons remarquer que ces derniers sont toujours bien reconnus.

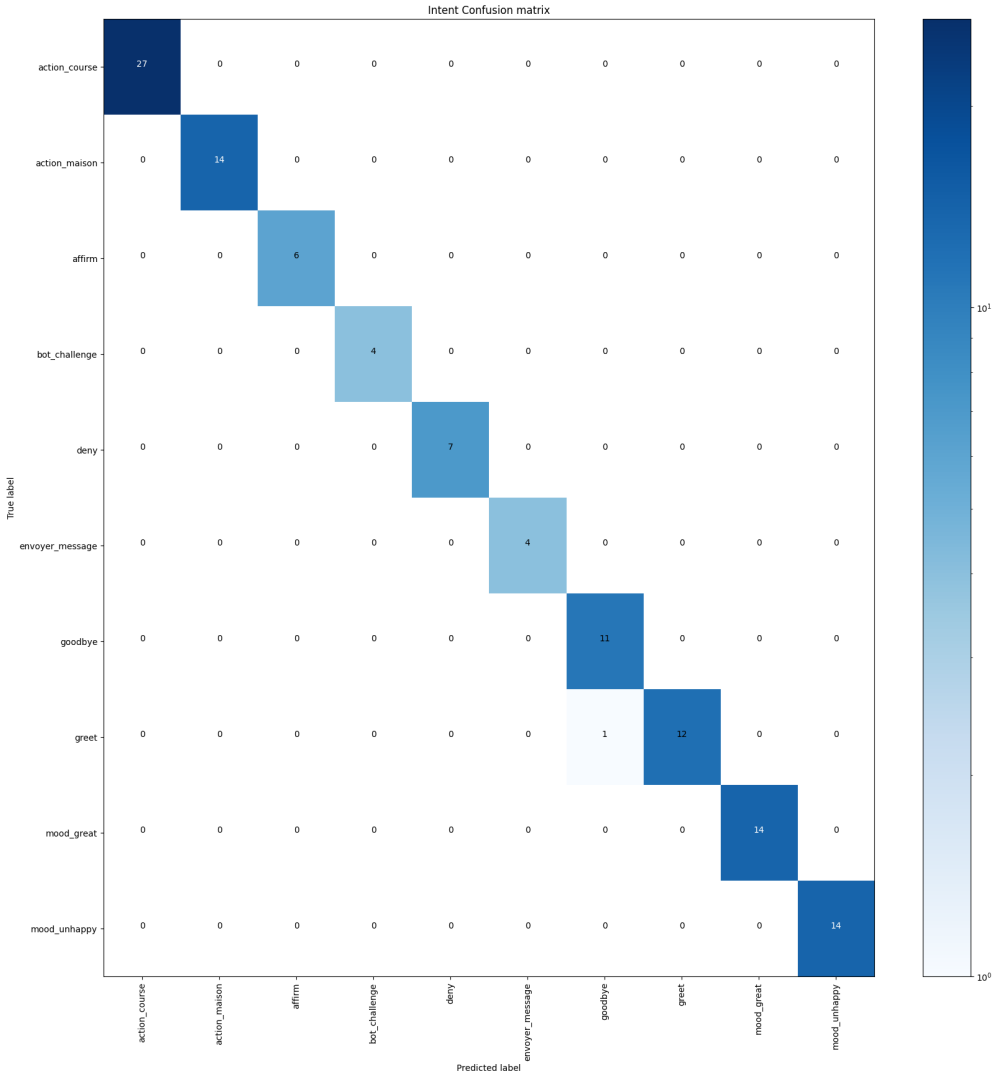

Figure 2 : Matrice de confusion des intents

Le résultat n'est pas étonnant du fait de leur nombre peu important ainsi que des grosses différenciations entre eux. Par exemple l'intent action_maison comprend des exemples tels que "allumer" ou "éteindre", tandis qu'action_course contient des exemples comme "ajouter", "rappelle-moi". Pour l'intent "envoyer_message", les exemples sont du type "envoyer". De ce fait, il est difficile pour le chatbot de se tromper et de les confondre.

Evaluation des stories :

Après avoir créé des stories de tests afin de déterminer si le chatbot était capable de réaliser les bonnes actions aux bons moments, et de faire se dérouler les stories de la manière prévue, nous avons pu obtenir des résultats très satisfaisants (voir figure 3).

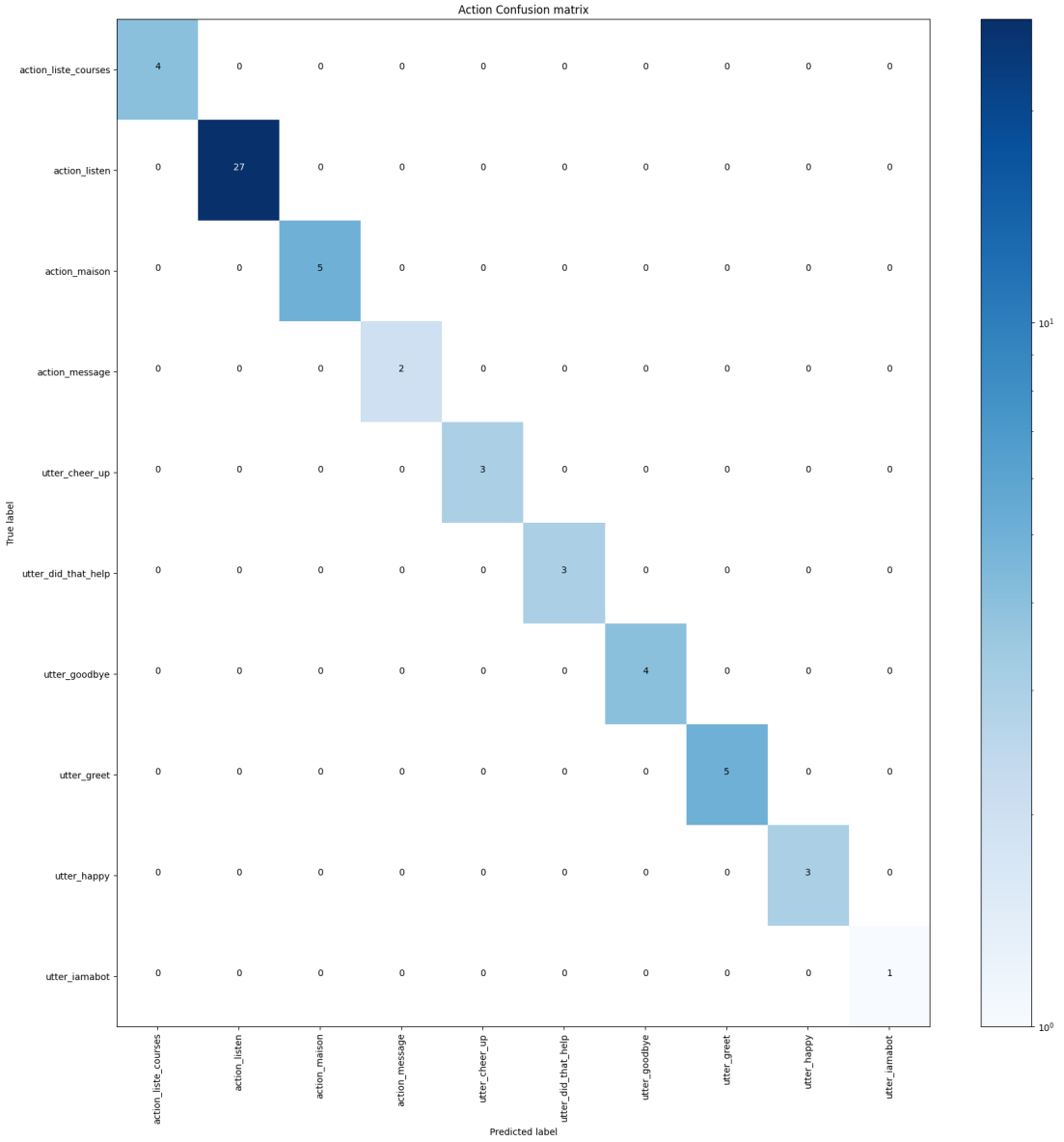

Figure 3 : Matrice de confusion des actions

Nous remarquons ainsi qu'aucune erreur de prédiction des actions n'a été effectuée.

Reconnaissance et extraction des entités :

Pour le bon fonctionnement de notre chatbot, il a été nécessaire d'utiliser des entités. Ces dernières permettent, associées à des slots, d'extraire des mots précis de l'énoncé de l'utilisateur afin d'envoyer les bons paramètres aux actions python. Par exemple, dans la phrase "allumer la lumière de la chambre" le chatbot va extraire 3 entités que nous avons définies : "allumer" étant une entité "action_maison", "lumière" qui est l'entité "lumiere" et "chambre" de l'entité "piece". Ainsi, le chatbot peut extraire les entités définies après les avoir reconnues, puis les stocker dans des slots afin que les actions python puissent y accéder.
L'extraction des entités peut se faire de plusieurs manières. Nous avons, pour la plupart, utilisé l'extracteur d'entités "DIETClassifier". Cependant, nous avons également, pour les entités "action_maison" et "envoyer", utilisé l'extracteur d'entités "RegexEntityExtractor". Ce dernier permet d'extraire des entités en définissant des "Regular Expressions". De ce fait, nous avons par exemple pu définir que tous les mots commençant par "allum" seraient associés au mot "allumer" et à l'entité "action_maison".

Les résultats obtenus avec l'extracteur "DIETClassifier" sont très satisfaisants (voir figure 4).

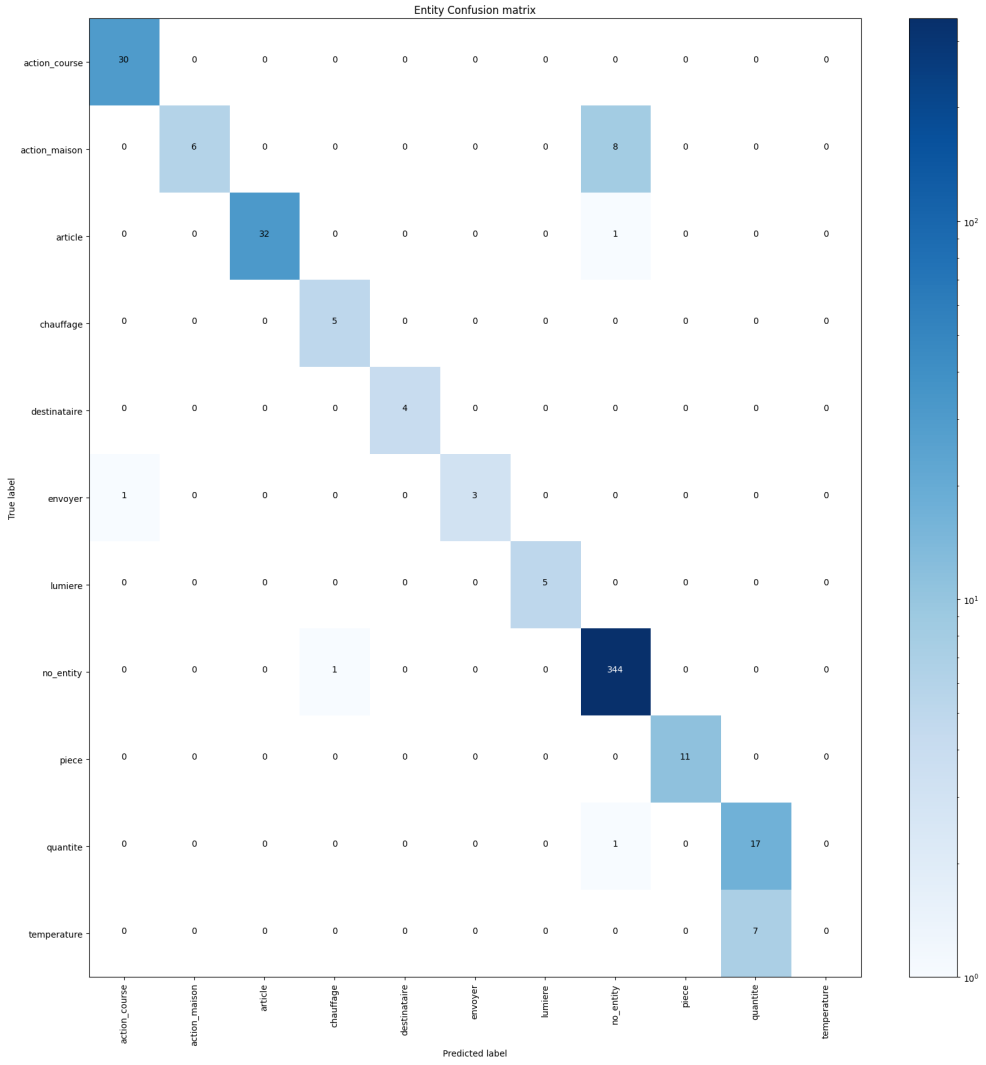

Figure 4 : Matrice de confusion des entités avec DIETClassifier

Nous pouvons en effet remarquer que peu d'erreurs de reconnaissance et extraction des entités sont réalisées. Nous pouvons voir que le plus gros problème se situe au niveau de la température. En effet, cette dernière étant un chiffre tout comme la quantité, on peut voir que l'extracteur n'a pas été capable de différencier les deux et a considéré la température comme étant une quantité. Cependant, lors de l'utilisation du chatbot, cette reconnaissance est bien effectuée et la température n'est pas reconnue comme une quantité.
Nous pouvons également remarquer des erreurs au niveau de l'extraction de l'entité "action_maison" mais ceci n'est pas un problème car cette dernière est réalisée par l'extracteur "RegexEntityExtractor" (voir figure 5).

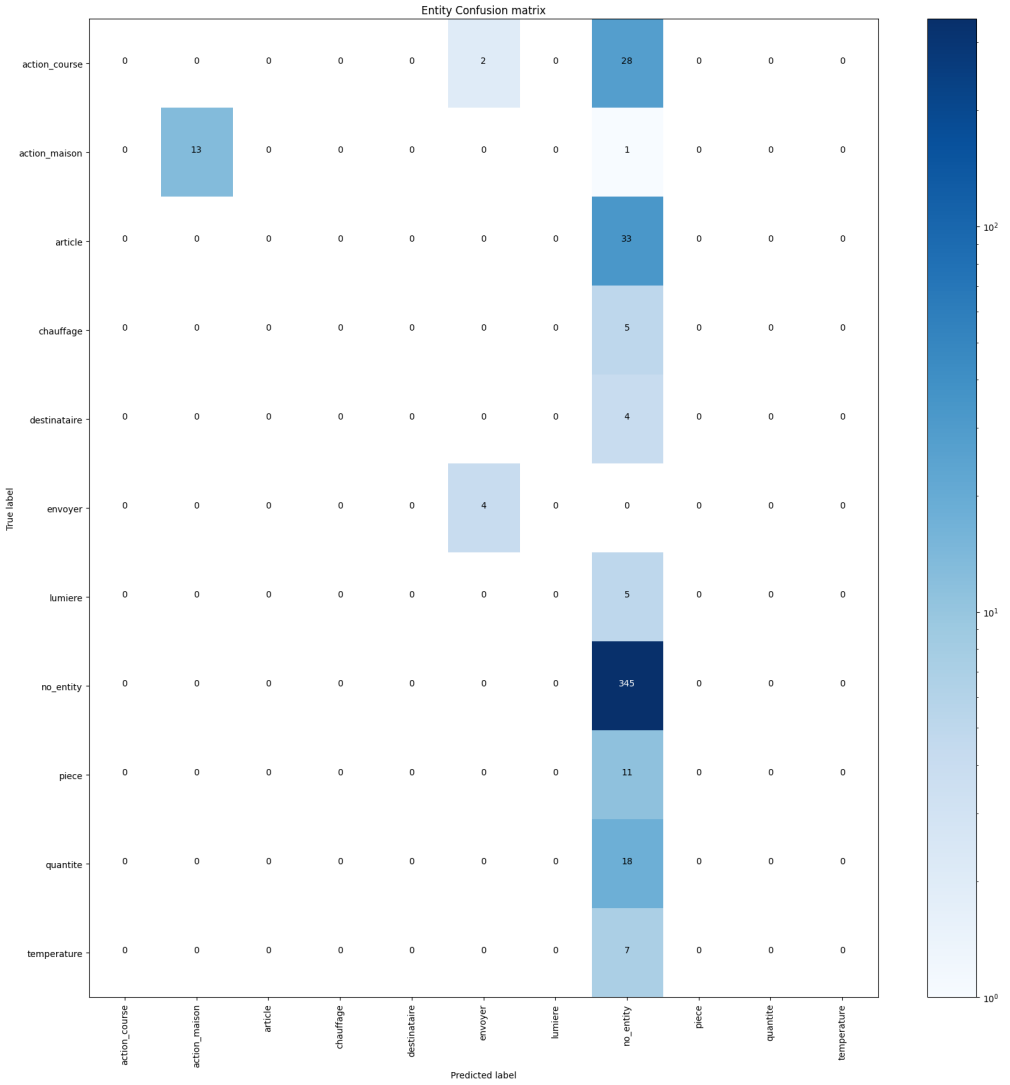

Figure 5 : Matrice de confusion des entités avec RegexEntityExtractor

Pour ce qui est de ce deuxième extracteur d'entités, nous pouvons voir sur la matrice de confusion (voir figure 5) que les résultats ne sont pas du tout bons. En effet, seules les entités "action_maison" et "envoyer" sont bien reconnues, toutes les autres étant classifiées dans la catégorie "no_entity" car non reconnues. Ce résultat peut sembler décevant à première vue, cependant, il est tout à fait normal que seules deux entités soient reconnues,

car nous avons, comme dit précédemment, utilisé le Regex uniquement sur les entités “action_maison” et “envoyer”. De ce fait, les résultats indiqués par la matrice de confusion sont indicateurs d’une très bonne reconnaissance et extraction de ces deux entités, toutes les autres étant extraites par le DIETClassifier.