Amélioration d’un chatbot (E2)

Karine ERLINGER

Développeur.se en intelligence artificielle RNCP34757

Sommaire

Introduction

Dans le cadre de l'épreuve E2, nous avons choisi d'apporter notre contribution technique auprès de notre centre de formation.

En accord avec l'équipe pédagogique, il a été décidé que notre sujet serait l'implémentation d'un Chatbot à disposition de SIMPLON ARA pour répondre aux questions basiques des visiteurs du site internet. Ce projet nous a été attribué pour la promo entière.

Pour information, Simplon n'a pas de budget pour concrétiser ce projet. Nous devrons donc nous adapter à cette contrainte.

Pour que cette épreuve soit validée par le jury, elle doit respecter les compétences suivantes :

- C8 : Modifier les paramètres et composants de l'intelligence artificielle afin d'ajuster aux objectifs du projet les capacités fonctionnelles de l'algorithme à l'aide de techniques d'optimisation.
- C14 : Améliorer l'application d'intelligence artificielle en développant une évolution fonctionnelle pour répondre à un besoin exprimé par un client ou un utilisateur.

1. Besoin

Actuellement, le site internet ARA[1] ne dispose pas de Chatbot. Ainsi, lorsqu'un visiteur à une question concernant les formations, ce sont les chargés de formation qui s'alternent pour répondre.

Après échanges avec l'équipe des chargés de formations, voici les informations du projet identifiées :

Client	Simplon.co
Périmètre	Auvergne Rhône Alpes (ARA)
Interlocuteur privilégié	Anaëlle GARCIA (Chargée de formation)
Problématique	80 % des mails envoyés par le grand public aux chargés de formation concernent les mêmes demandes, alors que les réponses sont disponibles sur le site internet.
Objectifs	Optimisation de la charge de travail des chargés de formation. Automatisation des tâches répétitives
Solution proposée	Création d'un Chatbot via des solutions open sources, disponible depuis le site web
Coût et ROI de la solution	Coût : 0€ Temps actuel consacré : 8h / mois Pertes net temps / an : 8 x12 = 96h Pertes net argent / an : 96 x 10,25 = 984€

2. La gestion de projet

Nous avons décidé d'utiliser Trello pour gérer le suivi du projet.

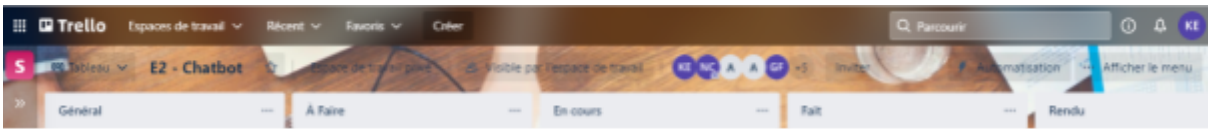

Figure 1: Trello

Dans un premier temps, nous avons créé 3 groupes de travail et désigné 1 chef de projet.

Ensuite, nous avons découpé chacune des étapes énoncées précédemment en tâches techniques. Nous avons priorisé et attribué les tâches à chacun des groupes. La planification des tâches a été dimensionnée pour respecter la date de rendu final en nous accordant de la flexibilité.

Chaque semaine, une revue est faite sur l'état d'avancement des tâches et des points bloquants. Cette organisation nous permet d'être réactif et d'apporter le soutien technique nécessaire pour dépasser les points bloquants en optimisant notre état d'avancement.

[1] https://auvergnerhonealpes.simplon.co/candidatures.html

3. Le développement du projet

Nous avons veillé à ce que les étapes du projet respectent les bonnes pratiques de data science et répondent aux compétences C8 et C14.

3.1. Les étapes du projet

3.1.1. Première partie : initiation d'un chabot sur une modélisation simple

➢ ***Identification des questions récurrentes auprès des équipes Simplon***

Afin de définir le contenu de notre fichier d'entraînement du modèle, nous récoltons en amont la liste des questions récurrentes auprès des chargés de formation Simplon. Ces questions concernent principalement :

- La disponibilité des formations
- Le tarif des formations
- Les possibilités de dispense des formations (distanciel / présentiel)
- Les canaux pour contacter Simplon
- Des informations générales sur Simplon

➢ ***Création d'un fichier JSON avec les intents en fonction des questions récurrentes***

Pour entraîner notre futur modèle, nous avons besoin de créer un fichier JSON contenant les informations correspondant aux questions potentiellement posées par les utilisateurs ainsi que les réponses associées.

Nous nommerons ce fichier Intents, il sera composé de plusieurs intentions comprenant:

- 1 Tag correspondant à l'intitulé d'une classe
- 1 Pattern correspondant aux questions relatives à notre Tag
- 1 Response correspondant aux réponses pouvant être fournies par le chatbot

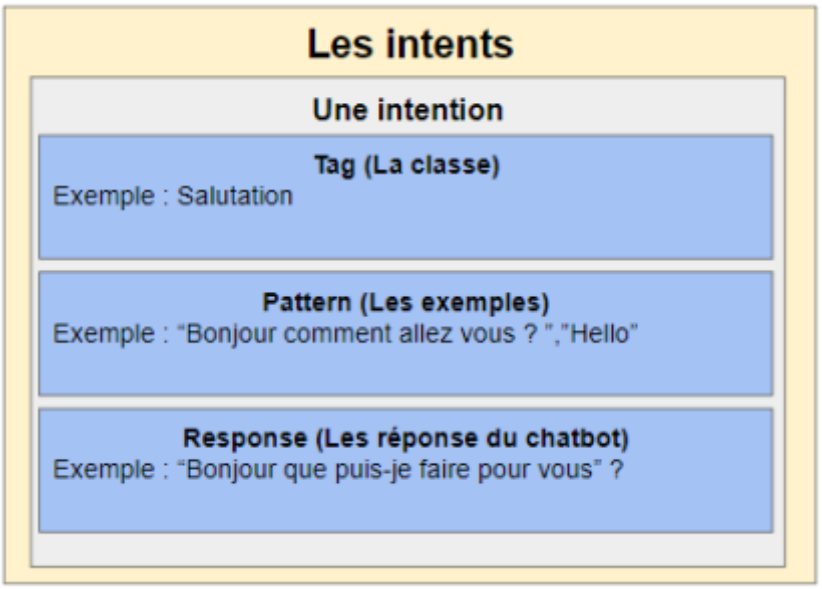

Figure 1 : Schéma structure du fichier JSON

Notre fichier Intents sera composé de 7 Tag, déterminés suite à notre échange avec les équipes Simplon pour répondre au mieux aux besoins identifiés et sont listés ci dessous :

- Salutation : Accueil du chatbot comme un humain
- Contact : Canaux de contacts Simplon
- Mode: Dispense de la formation (Distanciel ou Présentiel)
- Prix: Tarif de la formation
- Formation :Les formations disponible
- Identité: Information du l'identité de Simplon
- None: Pour le cas ou une phrase n'est identifiée

➢ ***Pré-processing des data pour entraîner le modèle avec un bag of word***

On tokenize nos Pattern, de manière à ce que nos phrases deviennent des mots distincts.

Nous allons ensuite créer deux listes :

- L'une contenant les mots de chaque Pattern, sans les ponctuations "?" et "!" et sans doublons.
- L'autre composé de nos Tag, one hot encoder

Par la suite, nous allons utiliser la méthode du Bag of Word pour préparer nos données. Le principe est de créer des vecteurs reconstituant les phrases contenues dans nos Pattern de la taille de la liste des mots contenus par notre dictionnaire sans doublons. On associe chaque Pattern à son Tag également vectorisé selon sa liste.

Pour notre projet, nous avons 87 mots dans notre liste sans doublons et 7 Tag. Ainsi nous obtenons des vecteurs d'une taille de 87 pour les Patterns et des vecteurs de 7 pour les Tag.

Ainsi nos données d'entraînement correspondent à :

- X_train = Vecteurs de 87 * Nombres de phrases contenues par les Patterns
- y_train = Vecteurs one hot encoder de 7

➢ ***Création d'un modèle ANN simple***

Nous créons un modèle simple composé de 3 couches Denses et de 2 couches de Dropout.
Notre couche d'entrée contient autant de neurones que nos Patterns vectorisés, soit 87.
Notons que le nombre de neurones en sortie est de 7, autant que nous avons de Tag.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 128)               11264

dropout (Dropout)            (None, 128)               0

dense_1 (Dense)              (None, 64)                8256

dropout_1 (Dropout)          (None, 64)                0

dense_2 (Dense)              (None, 7)                 455

=================================================================
Total params: 19,975
Trainable params: 19,975
Non-trainable params: 0
_________________________________________________________________
```

Figure 2 : Modèle ANN

Une fois le modèle entraîné, nous pouvons nous servir de celui-ci pour générer des réponses aux questions posées par les utilisateurs. Les questions sont tokénisées puis vectorisées par le Bag of Word créé précédemment pour reconstruire la phrase sous forme de vecteurs de 87. Nous pouvons ensuite prédire en utilisant notre modèle. On récupère alors des scores de probabilités pour chacun de nos Tag. La probabilité la plus élevée correspond au Tag qualifiant la question posée par notre utilisateur. La réponse associée au Tag est alors envoyée comme réponse à l'utilisateur.

```
L'utilisateur a saisi : Bonjour

La saisie transformée en vecteur :
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Les probabilités retournées pour chaque Tag sont :
[2.1979224e-05 7.1755530e-05 2.7576414e-06 1.3774592e-05 9.9829084e-01
 3.9832430e-06 1.5949606e-03]

Le Tag avec la probabilité la plus élevé est :
[[4, 0.99829084]]
[{'Tag': 'Salutations', 'probability': '0.99829084'}]
```

Figure 3 : Résultats de la prédiction

➢ ***Conclusion de cette première modélisation***

Cette version est fonctionnelle dans la mesure où :

- Les mots de la question saisis par l'utilisateur font partie de notre dictionnaire
- Les mots de la question saisis par l'utilisateur ne contiennent pas de fautes d'orthographe
- Certains de nos Tag contiennent les mêmes mots référents ce qui nuit à la compréhension du modèle
- Le dictionnaire contient les stops words et des ponctuations pouvant également nuire à la compréhension du modèle.

```
L'utilisateur a saisi : Bonjjour

La saisie transformée en vecteur :
[0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Les probabilités retournées pour chaque Tag sont :
[0.00206721 0.01340472 0.03411513 0.03749192 0.03761771 0.07550369
 0.7997996 ]

Le Tag avec la probabilité la plus élevé est :
[[6, 0.7997996]]
[{'Tag': 'none', 'probability': '0.7997996'}]
```

Figure 4 : Exemple de non gestion de faute d'orthographe

3.1.2. Seconde partie : Amélioration de la modélisation par l'ajout de fonctionnalités

Pour palier de ces manquements, nous avons intégrer de nouvelles fonctionnalités permettant :

➢ ***Intégration de pyspellchecker pour corriger fautes d'orthographes / de frappes***

Pyspellchecker est une bibliothèque externe qui peut être utilisée pour développer un correcteur orthographique. Le fonctionnement est simple. La fonction correction(), récupère le mot mal orthographié et renvoie le mot corrigé.

```
exemple = ['bonnjour']
for i in exemple:
    print("Saisie de l'utilisateur: "+ i)
    print("Correction par spellchecker: "+ spell.correction(i))
```

```
Saisie de l'utilisateur: bonnjour
Correction par spellchecker: bonjour
```

Figure 5 : Exemple correction par spellchecker

➢ ***Regex et lower pour supprimer les ponctuations et majuscules***

Nous avons géré la suppression des ponctuations pour qu'elles n'apparaissent pas comme des mots distincts à l'aide d'une regex simple et efficace.

```
sentence = sentence.replace(">", "")
sentence = sentence.replace("<", "")
sentence = sentence.replace("!", "")
sentence = sentence.replace("?", "")
sentence = sentence.replace("-", "")
sentence = sentence.replace("$", "")
sentence = sentence.replace(";", "")
sentence = sentence.replace(":", "")
```

Figure 6 : Regex suppression de la ponctuation

Les majuscules sont gérées par la fonction lower qui permet de traiter les chaînes de caractères. Le résultat de l'ensemble résultats de l'ensemble de nos 3 traitements est présenté dans la figure ci-dessous.

```
Saisie de l'utilisateur: Bonnjour!
Après traitement: bonjour
```

Figure 7 : Exemple de résultat après traitements

➢ ***Spacy pour augmenter la capacité de reconnaissance des mots***

La librairie Spacy permet une large gamme de tâches utiles aux projets de NLP. Dans notre cas, la fonction qui nous intéresse est similarity(). Cette fonction exprime la similarité sémantique entre des dictionnaires de mots vectorisés par une probabilité entre 0 et 1. En d'autre mot, si un mot entré par l'utilisateur n'est pas contenu dans notre dictionnaire de mot initial, alors la fonction similarity va rechercher une similarité avec les mots de notre dictionnaire et utiliser ce mot pour la suite. Nous avons établi un seuil de confiance pour que les mots ne soient pas systématiquement remplacés.

Prenons comme exemple le mot éducation ne faisant pas partie de notre dictionnaire. Sur la figure ci-dessous, nous pouvons observer que le mot éducation a été remplacé par formation, mot présent dans notre dictionnaire.

```
education le mot n'a pas été trouvé recour à Spacy
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:36: UserWarn
education a été remplacé par formation avec 0.755679907528988 de probabil
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

Figure 8 : Exemple de traitement par Spacy

➢ ***Lemmatizer et NLTK pour améliorer la qualité de notre dictionnaire de mots***

Contrairement aux améliorations précédentes, celles-ci vont agir directement sur notre dictionnaire de mots permettant d'entraîner notre modèle.

La lemmatisation permet de réduire les mots à leur racine, appelé la forme neutre canonique. Exemple pour le verbe être : est, sois, fut, étais, fussions. Après lemmatisation, seul le verbe être sera conservé. Ce traitement nous permet de construire un dictionnaire de mot, en agissant sur les doublons éventuels et augmente la capacité de reconnaissance des mots par le modèle.

Pour cela, nous avons utilisé la librairie french_lefff_lemmatizer qui nous permet de lemmatiser les mots en langue française.

Les stop words sont les mots communs considérés comme inutiles à la compréhension du contexte de la phrase par le modèle, par exemple : le, la, de, du, ce, etc…et surcharge le dictionnaire sans intérêt.

Pour cela, nous avons utilisé la fonction stopwords du package NLTK nous permettant de repérer les stop words de notre dictionnaire puis nous pouvons les supprimer.

4. Les résultats des améliorations

4.1. Résumé du processus

Nous vous proposons ci-dessous deux schémas résumant l'ensemble de notre processus, avec la signalisation des étapes où les améliorations ont été apportées.

Le premier schéma présente le processus de départ comprenant, les données d'entrées, le preprocessing jusqu'à la modélisation.

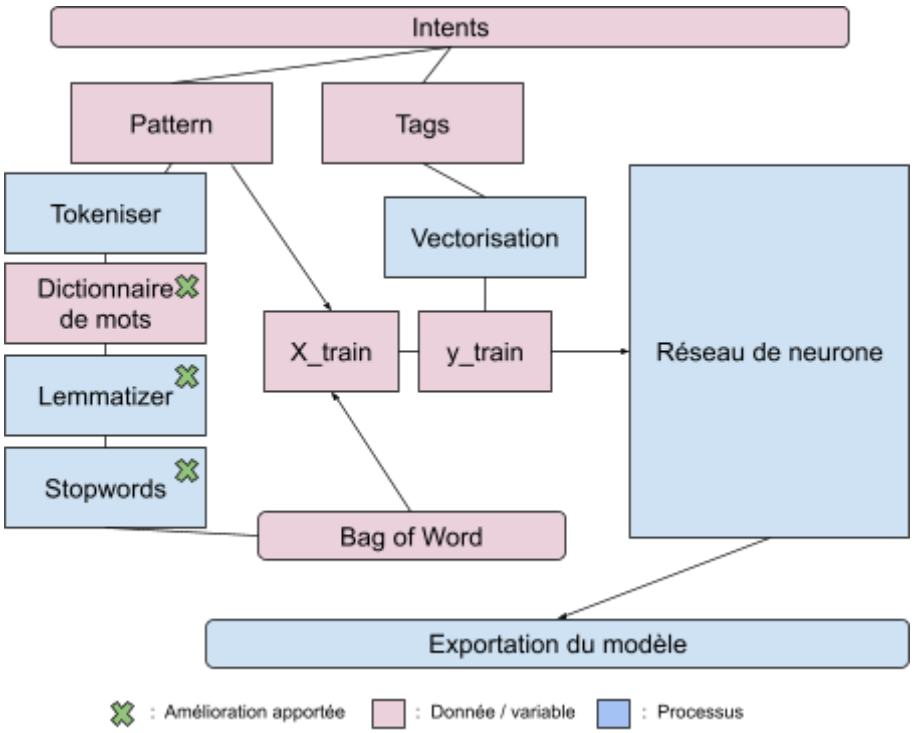

Figure 9 : Schéma des étapes du preprocessing à la modélisation

Le schéma suivant présente le processus de récupération de la phrase entrée par l'utilisateur à la réponse.

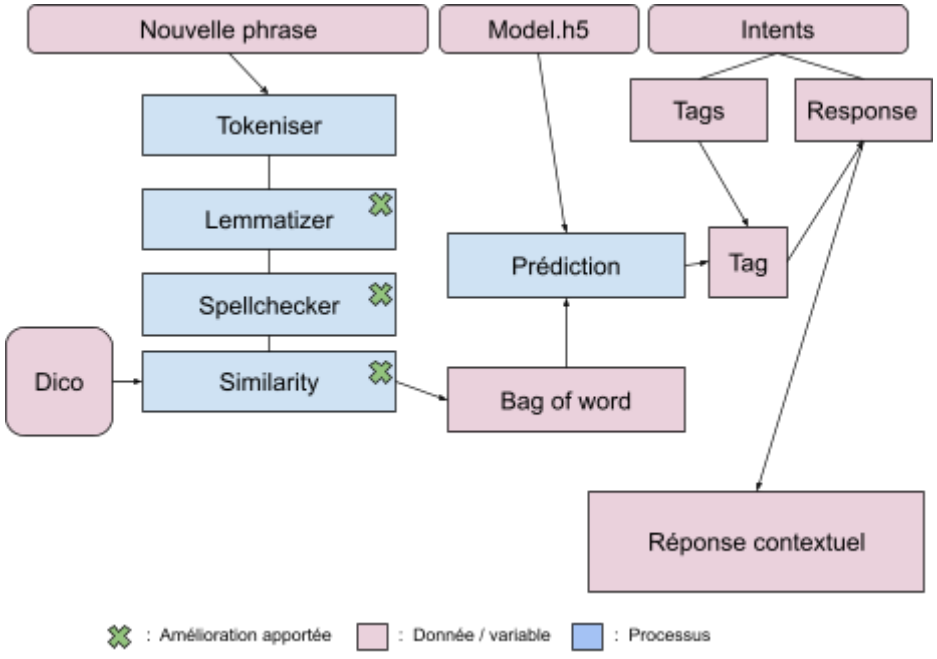

Figure 10 : Schéma des étapes de l'entrée utilisateur à la réponse du modèle

4.2. Le résultats des améliorations

Finalement, nous avons réussi à améliorer les résultats du modèle initial, comme nous pouvons le voir sur la figure ci-dessous. Par rapport à la figure 4 nous pouvons observer que les développements

supplémentaires que nous avons détaillé plus haut, nous permettent d'améliorer la réponse du modèle.

```
L'utilisateur a saisi : Bonjjour!

La saisie transformée en vecteur :
[0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

Les probabilités retournées pour chaque Tag sont :
[3.4513760e-06 9.6662974e-05 8.0890204e-05 9.9650973e-01 3.6690282e-05
 2.9533812e-05 3.2429590e-03]

Le Tag avec la probabilité la plus élevé est :
[[3, 0.99650973]]
[{'intent': 'Salutations', 'probability': '0.99650973'}]
```

Figure 11 : Exemple du résultat des améliorations apportées

Conclusion

Nous avons pu voir sur l'ensemble du rapport les différentes techniques qui nous ont permis d'améliorer les résultats de la modélisation. Cependant, les possibilités d'améliorations possibles restent nombreuses.

Pour n'en citer qu'une, il serait intéressant d'agir sur le modèle en lui-même et de tester un ANN Seq2Seq composé de couches d'embedding et de LSTM. Cela nous permettrait d'ajouter la compréhension du contexte à notre modélisation.

Nous tenons à préciser que nous nous sommes concentrés sur le détail de la partie technique nous ayant permis d'améliorer les résultats de la modélisation mais que l'ensemble du travail fourni comprends une partie de développement plus générale comme :

- Un système de Web Scraping récupérant les informations concernant les formations via le site web Simplon ARA .
- Le développement d'un Front End (cf Annexe) à l'aide des langages HTML et CSS, connectable au Back End en utilisant la librairie Flask.
- Travailler en versioning en utilisant Git et Google Drive
- Un travail de veille global sur les modèles pré-entraînés et les méthodes de récupération / traitement des phrases non comprises par le chatbot.

Table des figures

Bibliographie

https://towardsdatascience.com/a-simple-chatbot-in-python-with-deep-learning-3e8669997758
https://www.datacorner.fr/bag-of-words/
https://www.analyticsvidhya.com/blog/2021/06/build-your-own-ai-chatbot-from-scratch/
https://openclassrooms.com/fr/courses/4470541-analysez-vos-donnees-textuelles/4854971-nettoyez-et-normalisez-les-donnees
https://fr.wikipedia.org/wiki/Mot_vide#:~:text=En%20recherche%20d'information%2C%20un,du%20%C2%BB%2C%20%C2%AB%20ce%20%C2%BB%E2%80%A6
https://www.geeksforgeeks.org/python-word-similarity-using-spacy/
https://www.delftstack.com/fr/howto/python/python-spell-checker/#correcteur-orthographique-avec-la-bibliothèque-pyspellchecker-en-python
https://datascientest.com/spacy
https://towardsdatascience.com/how-to-implement-seq2seq-lstm-model-in-keras-shortcutnlp-6f355f3e5639

Annexe

Annexe 1 : Front End