Rapport de Projet

Traitement Automatique du Texte en IA

Tale Box

Yasmine Ben Fredj

Lien vers GitHub

Année Universitaire 2020-2021

Table des matières

Introduction

Dans le contexte d'un projet en Traitement automatique du texte en Intelligence Artificielle, Je vous présente TALEBOX.
C'est un jeu textuel développer en français pour rédiger une histoire avec le joueur et donnée ensuite les possible thèmes de l'histoire.

Premièrement, notre TALEBOX doit donc être entrainer à lire des histoires pour connaitre les éventuels thèmes possibles et ensuite elle doit savoir rédiger un texte en français qui peut être une suite à l'histoire que le joueur va taper.

Dans la suite de ce rapport je vais vous donner plus de détails sur la conception de ce jeu, les choix du model, le choix des données ainsi que toutes les étapes par lesquels je suis passé afin de finir ce projet.

Taches principales

I. Collecte des données :

Pour pouvoir reconnaitre les sujets et rédigé des histoires notre model doit être entrainée sur un très grand nombre de données. Donc en recherchant sur internet j'ai essayé de collecter le maximum d'histoire, contes et légende dans le fichier CSV « TaleBox_contes ».

II. Prétraitement des données :

Après avoir collecter et importer les données il faut ensuite les préparer.
Dans le fichier « **Fonctions_utile.py** », il y a les fonctions suivantes :

- **« sup_caractere_spéciaux »** permet de retirer les ponctuations et les caractères spéciaux tel que : +, *, < , ? …
- **« tokenize »** permet de tokenizer les mots des textes avec **nltk.word_tokenize**.
- **« get_bigrams »** permet de crée des Bigrammes (deux mot qui se produisent fréquemment ensemble dans les documents).

- **« get_trigrams»** permet de crée des Trigrammes dans la liste des textes. Cela a été fait avec le model **Phrases** de **gensim.models.**
- **« filtre_motArret »** permet de retirer les mot d'arrêt déjà présent dans les **stopwords** français de **nltk** ainsi que quelques-unes que j'ai ajouté**.**
- **« lemmatiser »** permet de lemmatiser les textes en français avec **spacy.** Nous ne gardons que les verbes, les adverbes, les adjectifs et les noms.

III. Modélisation :

Pour prédire le sujet d'un texte j'ai choisi d'utiliser l'approche LDA. Cet algorithme d'analyse non supervisée permet de regrouper des collections de mots dominant (mots clef) dans les textes qu'on va lui fournir. Ces collections peuvent être interprété comme des sujets. Ce regroupement s'effectue par le moyen de calcule de probabilité des mots dans le sujet et la probabilité du sujet dans un texte ensuite.

Pour faire cela j'ai trouvé deux manières intéressantes que j'ai tous les deux tester :

1. La première (ModelLDA) consiste à construire un dictionnaire de mot avec **gensim.corpora.Dictionary**, de le filtrer (enlever les mots qui se répètent comme : puis/ faire/dire…) .
   Ensuite, avec **doc2bow** j'ai créé un sac de mots (bow), c'est une sorte de tuples (index du mot, fréquence du mot) pour chaque mot présent dans chaque conte après avoir fait le processus de prétraitement.
   Enfin, avec **ldaModel** de **gensim.models.ldamodel** on crée le model LDA avec les sac de mots et les hyperparamètres tel que **num_topics** (nombre de sujet) que j'ai fixé à 10…

2. Pour la seconde (BestModelLDA) les étapes sont les mêmes mais pas les outils.
   Pour construire les sac de mots, j'ai utilisé **CountVectorizer** de **sklearn.feature_extraction.text** avec les hyperparamètres:
   - Stow_words : mots d'arrêt.
   - Ngram_range : ngram (bigames/trigrammes…) de 1 à 4.
   - min_df : pour dire quelle est le nombre minimum qu'un mot doit avoir été apparus dans un documents.
   - max_df : pour pas avoir les mots qui se répètent beaucoup tel que puis, dire, faire...

   Une fois nous avons construit nos sacs de mots, je devais créer le model **LatentDirichletAllocation(LDA)** avec **skearn.decomposition**.
   Mais pour trouver les meilleurs hyperparamètres au model (tel que le nombre des sujets), j'ai dû utiliser **GridSearchCV** de **sklearn.model_selection.**

IV. Prédiction de sujet :

Les deux modèles que j'ai créés sont enregistrer dans le fichier « **Models** », avec **pickle.**
Puis dans le fichier « **Prédiction.py** » je les importe pour crée la fonction « **predire_genre** », j'ai donc pus crée cette fonction avec les deux modèles mais celui qui m'a semblé plus pratique était le « BestModelLDA », j'ai donc choisi de continuer avec celui-là.
Cette fonction permet de retourner les collections de mot important d'un texte et avec ces collections on peut voir de quel sujet il s'agit.
Pour cela j'ai donc généré le fichier « **TaleBox_genres.csv** » qui contient les collections de mot et le sujet correspondant qui me semble le plus juste.
Puis dans le fichier « **TaleBox_contes.csv** », j'ai ajouté avec **pandas** pour chaque histoire le sujet prédit.

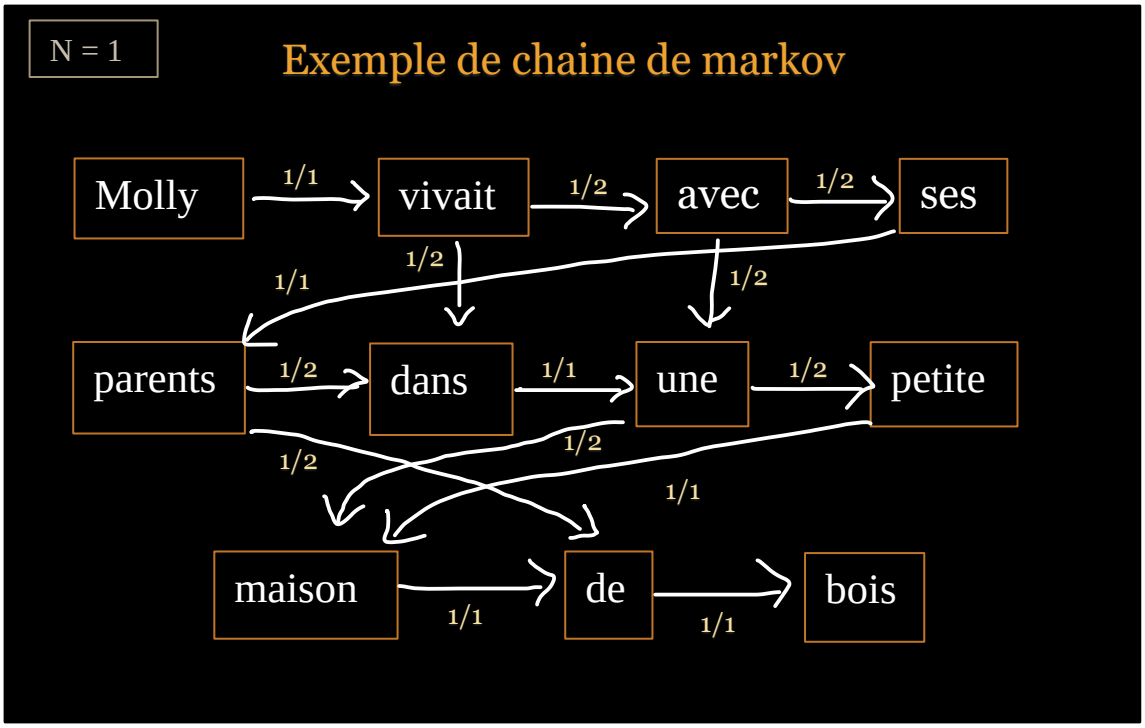
V. Génération de texte :

La seconde étape majeure de ce projet est de permettre à **TaleBox** de communiquer avec le joueur. Pour communiquer **TaleBox** va soit répondre à des questions basiques sois continuer la rédaction de l'histoire que l'utilisateur a commencé.

1. **Répondre à des questions basiques :**
   Pour permettre à **TaleBox** de dialoguer textuellement avec le joueur, j'ai trouvé la bibliothèques **ChatterBot** qui nous permet de crée un bot simple et l'entrainer sur des petites discussions. On peut ainsi retrouver notre **Tale_Bot** dans le fichier « TaleBox.py » qui est entrainer sur une unique discutions.
   Mais le plus intéressant dans **ChatterBot** est **« logic_adapters** ». C'est la logique ou les logiques que va adopter notre bot, nous pouvons les spécifier lors de la création de notre bot. Lorsque le bot doit fournir une repense, il va tester ses adapter_logic dans l'ordre, le premier qui fournis le meilleur score de confidence (0 -> 1) est choisi pour rependre. J'ai aussi fixé par défaut une phrase qu'il retourne en cas ou aucune des logiques n'ai bonne à plus de 80% :
   « Désolé, je n'ai pas compris. Veuillez me donner plus d'informations. »
   Ceci m'a donc permis de crée les deux **logic_adapter** suivant **:**
   - « **Quitte_Adapter** » : Cette adapter va permettre au bot de comprendre lorsque le joueur souhaite terminer de rédiger son histoire et qu'il souhaite le sujet.
   - « **Raconte_Adapter** » : Cette adapter va permettre au bot de savoir que le joueur est en train de rédiger l'histoire et de le faire avec lui. Cette adapter fait appel à la fonction « **markov_genere »** que je vais présenter ci-dessous**.**

Figure 1: Répondre à des questions basiques

2. **Continuer la rédaction de l'histoire que l'utilisateur a commencé :**
   Ici notre Jeu doit pouvoir reconnaitre le contexte des phrases que va entrer le jouer et en créer des autres à la suite. Pour permettre de faire cela j'ai choisi de faire un model en utilisant la chaine de Markov.
   Le modèle de Markov va en analysant toutes les données qu'on va lui fournir crée des « Etats », ces Etats sont une suite de N mot qui se suit. Pour chaque état nous allons donner une probabilité que cet état passe à un état suivant.



Dans le fichier « MarkovGeneration.py » :

- La fonction **markov_model :** permet de crée un dictionnaire qui a pour clef les états et pour valeurs les probabilités des états voisins.
- La fonction **markov_genere :** cette fonction va prendre en entrer une suite de N mots puis chercher dans le dictionnaire qu'a crée la fonction précédente l'état voisin le plus probable, puis poursuivre cette procédure jusqu'à ce qu'on forme une phrase ou un paragraphe**.**

Ici j'ai fixé la taille d'un état à 3 mots (N), et la prédiction retourne à chaque fois 10 états suivant l'état entrée donc elle retourne une phrase de 30 mots (10*3).
Lorsque la fonction ne trouve pas un état donné dans le dictionnaire, elle va retourner la phrase suivante au joueur :
« Vous êtes tellement doué que je n'ai pas d'idée pour la suite, continuez ... »
Cela va inciter le joueur à entrer une nouvelle phrase et donc permettre à l'algorithme de refaire une nouvelle recherche.

```
>>> markov_genere(10, "molly vivait avec", "")
("ses parents dans une petite maison sur le toit et je lui dis au revoir la femme
dit - fou que tu es ce n'est pas le chaton c'est un reflet ...", True)
>>>
```

VI. Interface Graphique :

Avec la bibliothèque Tkinter, j'ai pus effectuer une simple interface qui permet d'afficher la discutions entre le bot et le joueur. Elle permet aussi au joueur de taper son message et de l'envoyer à l'aider du bouton « Envoyer ». Nous pouvons retrouver le code qui permet de lancer cette interface dans le fichier « TaleBox_interface.py ».

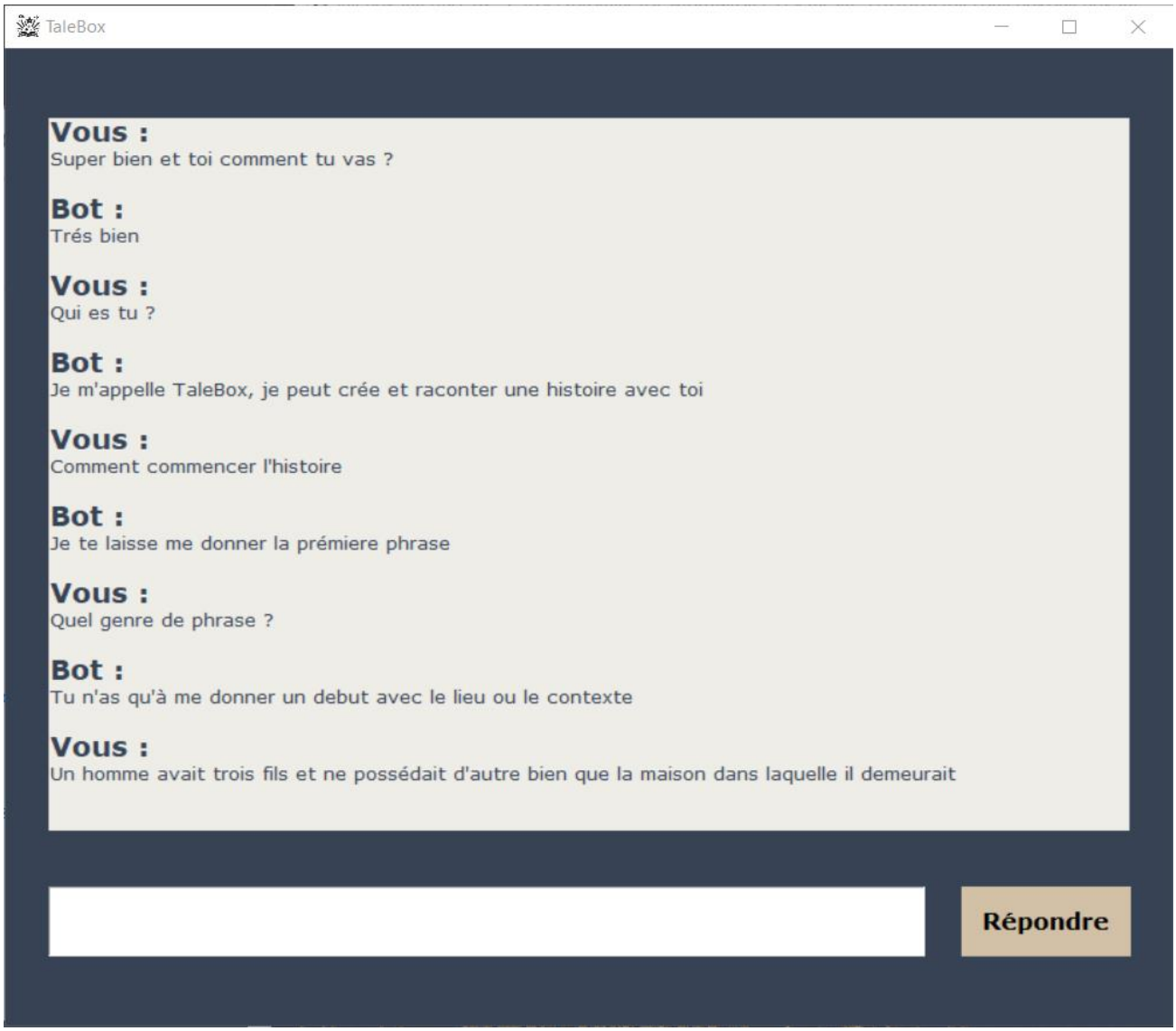


Figure 2 : Continuer la rédaction de l'histoire que l'utilisateur a commencé

Figure 3 : Proposer un thème à l'histoire

Analyse des erreurs

- Il y a des erreurs dans la génération de texte :
    - Phrases non finies,
    - Phrases qui n’ont pas de sens quelquefois,
    - Phrases qui s’éloignent du contexte du début.
- Il arrive que la prédiction de sujet ne soit pas très précise et donner un faux résultat.
- Dans la discutions entre le joueur et le bot, le bot peut rependre avec une mauvaise repense par exemple ici :

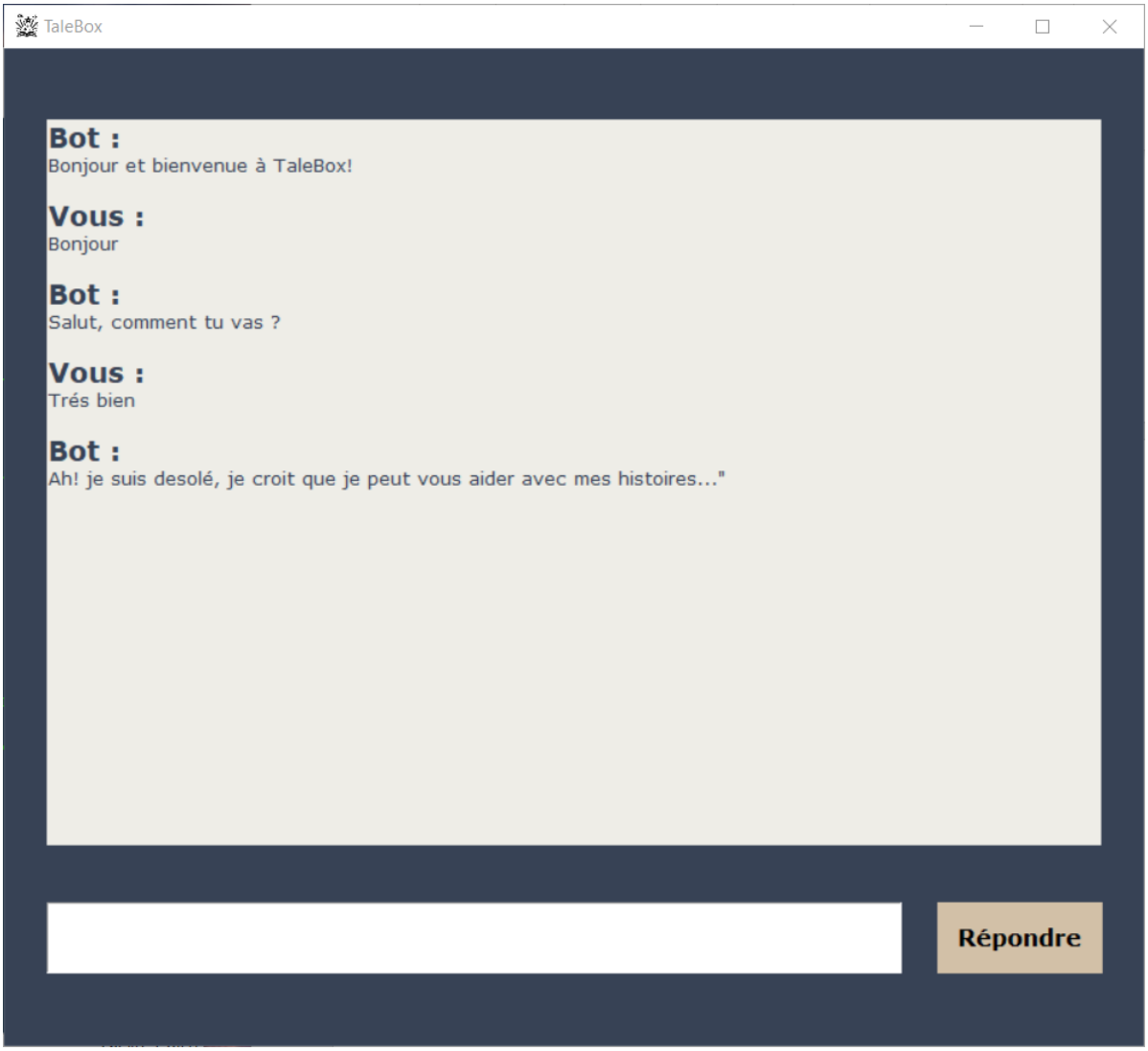


Figure 4 : Bug

Améliorations possibles

- Avoir un jeu de données plus grand et plus organiser, car je me suis rendu compte que dans quelques contes nous avons des dialogues et ça peut fausser la génération de texte. Donc la solution est soit de crée plusieurs jeux de donnés qui seront organiser par catégorie ou bien trié pour avoir seulement des textes.
- Avoir une génération de texte plus adapter, par exemple les phrases qui se terminent bien quand il faut.
- Permettre à l'algorithme de génération (markov) de connaitre le contexte de l'histoire plus largement pour ne pas partir sur un autre sujet et fausser le sens de l'histoire.
- Entrainer le chatterBot sur plus de conversation pour qu'il soit plus souple.
- Améliorer l'interface graphique.
- Ajouter plus d'options à ce jeu tel que le fait de permettre au joueur de faire un choix entre plusieurs suites au texte, permettre au jeu d'analyse l'état d'esprits de la personne qui joue, si elle est violente, triste, joyeuse…

Conclusion

L'objectif de ce projet était de prédire le sujet d'un texte et générer du texte automatique à la suite. Les objectifs ont donc été atteint même si ça manque un peu plus de travail pour que ça soit comme je l'aurais imaginé au début. Je pense donc que je vais continuer de travailler sur ce projet pour que mon **TaleBox** soit plus performant.

Grace à ce projet j'ai découvert des choses hyper intéressantes sur le NLP. J'ai aussi connu l'existante de quelques outils et Library très pratique comme GridSearch, Spacy, Rasa, ChatterBot, pyLDAvis…

Finalement, j'espère que mon travail satisfait vos attentes.

Webographie

1. L'œuvre Perfection par l'auteur Paul Marie, disponible en ligne depuis 6 mois et 20 jours - Je l'avais choisie parfaite - Short Édition (short- edition.com)
2. https://www.grimmstories.com/fr/grimm_contes
3. Les contes pour enfant du monde :: conte :: contes hoffmann :: Jacques Callot
4. explosion/spacy-models: Models for the spaCy Natural Language Processing (NLP) library (github.com)
5. gunthercox/ChatterBot: ChatterBot is a machine learning, conversational dialog engine for creating chat bots (github.com)
6. Grimm's Tales Topic Analysis via LDA: Why some Tales are Famous | by Cornellius Yudha Wijaya | Nov, 2020 | Towards Data Science
7. Création de Chatbot Avec Python: 5 Étapes Faciles à Suivre (tophebergeur.com)