Utilisation de LLM(large language models)

dans le Jeu de Role.

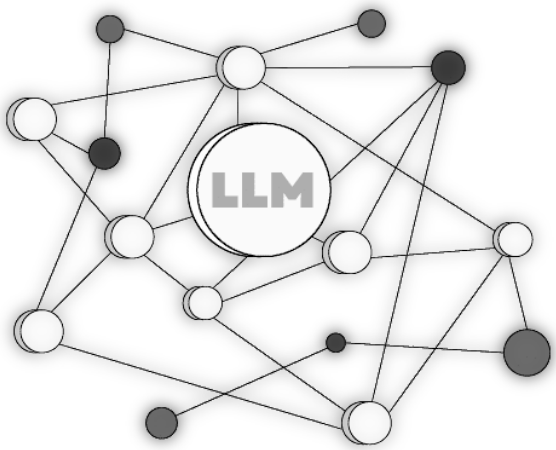


Table des matières

Introduction

Introduction

Le récent essor des LLM, nous laisse espérer : une implémentation numérique satisfaisante, d'un jeu de rôle quelconque, similaire au jeu de rôles sur table : Dungeon et dragon.

Celui-ci (le jeu de rôle), entretient une relation étroite avec la littérature fantastique. Il en puise directement ses inspirations et en reprend les codes.[1]

Nous vient alors la question légitime : «Peut-on entrainer un modèle génératif à concevoir des scenarios de jeu de rôle, à partir d'ouvrages (fantastiques ou non), trouvés en libre accès ? »

Depuis la parution de Dungeon et dragon (DND) en 1974, nombreuses ont étés les tentatives de portages numériques, du célébrissime jeu de rôle. Ces bien heureuses tentatives, de par leur contraintes techniques, ont donner naissance aux mécaniques de RPG(qui font très largement consensus dans l'industrie contemporaine du jeu vidéo)[2].

Cependant, après tant d'années d'expérimentations, il nous faut faire l'amer constat, de l'échec au moins partiel, de ces portages[3]. Pour cause, le CRPG (pour computer role playing game), perd en flexibilité, dans ses tentatives de réduire le jeu de rôle a son aspect statistique.
C'est là : une contrainte que le jeu de rôle, partage très largement avec «le traitement naturel du langage», et son tout récent essor, nous laisse espérer, que l'on puisse trouver une solution à ce premier problème en la résolution du second.

1. Nous pouvons citer en autre des auteurs comme Tolkien , Lovecraft , Glen Cook , Gorge R.R Martin , Robert E Howard , voir des auteurs de science-fiction tel que Philipe K. Dick (notoirement connu pour blade runner)
2. Connecting Worlds.Fantasy Role-Playing Games Ritual Acts and the Magic Circle 2005
3. wikipedia jeu de role

Objets et Definitions

Objets

Magma : Soit E un ensemble et $*$ une *loi de composition interne* sur E, $(E, *)$ forme alors un magma.

Partie Stable L : Soit E un magma, $*$ sa loi de composition interne, et $L \subseteq E$, L est alors une *partie stable* de E si

$$\forall (x, y) \in L^2 \; x * y \in L$$

Element Neutre : Soit E un magma et $*$ sa loi de composition interne, alors $(E, *)$ admet un *element neutre* ε ; si

$$\forall a \in E \; \varepsilon * a = a * \varepsilon = a$$

Loi Associative : Soit $(E, *)$ un magma. * est associative sur E si :

$$\forall (a, b, c) \in E^3 \; (a * b) * c = a * (b * c)$$

Espace De Travail

Alphabet Σ : Soit Σ un ensemble de lettres, tel que $\Sigma \neq \emptyset$ et que $Card\Sigma < +\infty$.

Mot $l \in \Sigma^*$: On appelle *mot* sur l'alphabet Σ toute suite $l = (l_1, l_2, ..., l_n) \in \Sigma^n$ avec $n \in N$, que nous noterons $l = l_1 l_2 ....l_n$.

Alphabet Σ^* : Soit Σ un alphabet, alors Σ^* est l'ensemble des mots de Σ.

Longueur || d'un mot $l \in \Sigma^*$: Soit Σ un alphabet, et $l = l_1 l_2 ....l_n \in \Sigma^*$, alors $\mid l \mid = n$ est la *longueur* de l.

Phrase s : On appelle *phrase* sur l'alphabet Σ toute suite $s = (m_1, m_2, ..., m_n) \in (\Sigma^*)^n$ avec $n \in N$.

Proposition

Soit Σ un alphabet.
Soit * une *loi interne associative* sur Σ^*, tel que $\forall(n,\eta) \in N^2$, $\forall(m^{[1]}, m^{[2]}) \in \Sigma^n \times \Sigma^\eta$ avec $m^{[1]} = (m_1^{[1]}, m_2^{[1]}, ..., m_n^{[1]})$ et $m^{[2]} = (m_1^{[2]}, m_2^{[2]}, ..., m_\eta^{[2]})$

$$\text{alors } m^{[1]} * m^{[2]} = m_1^{[1]} m_2^{[1]} ....m_n^{[1]}.m_1^{[2]}.m_2^{[2]}.....m_\eta^{[2]}$$

$(\Sigma^*,^*)$ est donc une structure de *magma*, et nous notons ε son *element neutre*, tel que $\mid \varepsilon \mid = 0$.

Itération

$$\begin{cases} \forall n \in N \ \forall x \in \Sigma^* \ x^{n+1} = x * x^n \\ x^0 = \varepsilon \end{cases} \tag{1}$$

Nous pouvons etre ammenes a note $nx = x^n$

Definition

Nous noterons simplement qu'un langage L sur l'alphabet Σ, est un sous ensemble de $(\Sigma^*)^*$.

Language a Base Disjointe

Soit n $\in N$, $\forall i \in [|1,n|]$ $L_i \subseteq \Sigma^*$, et tel que $\forall(i,j) \in [|1,n|]^2$ avec $i \neq j$, $L_i \cap L_j = \{\varepsilon\}$.
On note alors $B = (L_1, L_2, ..., L_n)$, que l'on appellera $BaseDisjointe$.

Soit L un langage.

Soit $Q \subseteq \ \{x | \exists k \in N$ tel que $x \in [|1,n|]^k\} avec \ CardQ < +\infty$.

$$\text{Si } L \ = \sum_{\forall \ x \ \in \ Q} \sum_{\forall j \in x} L_j, \text{ alors B est une base de L.}$$

L est donc un langage a base disjointe.

Applications sur notre espace

Encodeur

L'idée d'utiliser nos ALGORITHMES CORRECTEURS sur du texte brut, nous vient de plusieurs constats.

En effet, extraire du texte de livres, pose de nombreux problèmes, dont le plus immédiat est l'encodage. Les règles de typographie[1], les erreurs dans la conversion de PDF[2], complexifient le problème, si bien, que l'on doit en permanence procéder à un nettoyage minutieux du texte. Celui-ci doit être nettoyé afin d'être analyser et correctement traiter[3]. Nous voulons alors prétraiter le texte, pour en simplifier son traitement, et rendre ainsi la donnée plus digeste[4].

Une des solutions possibles, pourrait consister à encoder directement notre texte (en utf-8 pour l'exemple), puis le décoder dans la seconde qui suit. Il faudra en outre remplacer certains patterns récurrents qui complexifie inutilement notre donnée, par la fonction replace de python[5]. Seulement, cette méthode montre rapidement ses limites. Nous sommes amenés à traiter de grandes quantités de texte, et ces méthodes, couteuses en temps, paraissent inélégante et par-dessus tout non-pertinentes. Pour peu que l'on ait une liste de patterns a interchanger, la fonction replace se verrait d'en l'obligation de parcourir notre chaine de caractères a plusieurs reprises.

1. Il peut s'agir de "-" avant un dialogue, ou d'un "-" coupant un mot en 2 lors d'un retour à la ligne.

2. La bibliothèque que nous utilisons, laisse parfois certains artefacts lorsqu'elle extrait le texte de certains PDF, comme des erreur d'encodage ou des mots dont les lettres sont espacées (exemple : e x e m p l e).

3. Le nettoyage ne doit pas être le même, pour l'analyse du texte que pour celle de son

```
"âĂŞ 41 âĂŞ  \nâĂŤ Bien,
   monsieur, trÃÍs bien !
   rÃľpliqua l'inspecteur des
   fo-\nr Ãłts, vous d
    emandez, ma fi
   lle en mariage,
     d'autres font
    de \nmÃłme ; mon devoir
   de pÃÍre m'oblige Ãă me
   soucier d'elle ; je vous
   donne un dÃľlai de trois
   jours pour chercher une
   ombre ; dans trois jours,
   prÃľsentez-vous Ãă moi avec
    une ombre qui vous aille
   bien, et vous serez le
   bienvenu ; mais le
   quatriÃÍme jour âĂŤ Ãľcou-
   tez bien ce que je vous dis
    âĂŤ ma fille sera la femme
    d'un autre. Âż \n Je
   voulus encore adresser un
   mot Ãă Mina ; mais, redou-\
   nblant de sanglots, elle se
    serra plus fort contre sa
   mÃÍre et celle-ci, sans mot
    dire, me fit signe de m'
   Ãľloigner."
```

```
"Il prit \naussitÃťt la parole
    : \n Âń Je m'Ãľtais
   annoncÃľ pour aujourd'hui,
   vous n'avez pas eu \nl a p
    atience d'atte
    ndre le moment
    fixÃľ. Mais tou
    t peut encore s
   'arranger : acceptez mon
   conseil, votre ombre est
   encore Ãă vo-tre
   disposition, et aussitÃťt
   vous  revenez sur vos pas.
   Vous serez \nle bienvenu
   dans le jardin de l'
   inspecteur des forÃłts, et
   tout cela n'aura ÃľtÃľ qu'
   une plaisanterie ; quant Ãă
    Rascal, qui vous a trahi e
    t demande la mai
   n de votre fianc Ãľ
    e, je me charge
    de lui et la chose
    est pratiquement rÃľglÃľe.
    Âż \n J'avais la sensation
    de vivre un rÃłve :  Âń
   Vous vous Ãľtiez annoncÃľ
   pour aujourd'hui ? Âż "
```

traitement(exemple : le LLM doit-être capable de reconnaitre un dialogue, alors que cette information est superflu, voir gênante dans notre analyse.

4. Nous voulons simplifier le plus possible, la donnée que le LLM doit prédire plutôt que de complexifier son modèle d'apprentissage.

5. "ab".replace("abd","cccc") == "ccccd" est une proposition vraie.

définitions

Encodage

Soit Σ_1 et Σ_2 2 alphabets.

On appel $ENCODAGE$ toute application :

$$e : \Sigma_1^* \to \Sigma_2^*$$

exemple

Dans cet exemple, il s'agit juste de passer d'un langage binaire a un autre $a \longmapsto 0$ et $b \longmapsto 1$.

$\Sigma_1 = \{a, b\}$	$\Sigma_2 = \{0, 1\}$
abbbabbabbabbabb	0111011011011011
aaaabbaaabbaaaaa	0000110001100000
aaaaabbbbaabbbab	0000011110011101
bbababbaaabaabbb	1101011000100111
aaabbaab	00011001

Encodeur

Soit Σ_1 et Σ_2 2 alphabets.
Soit $n \in N$.
Soit $((u_1, u_2, ..., u_n), (v_1, v_2, ..., v_n)) \in (\Sigma_1^* \times \Sigma_2^*)$ On note :

$$\sigma = \begin{pmatrix} u_1 & u_2 & \cdots & u_n \\ v_2 & v_2 & \cdots & v_n \end{pmatrix}.$$

On appel $ENCODEUR$ tout $e : \Sigma_1^* \to \Sigma_2^*$:

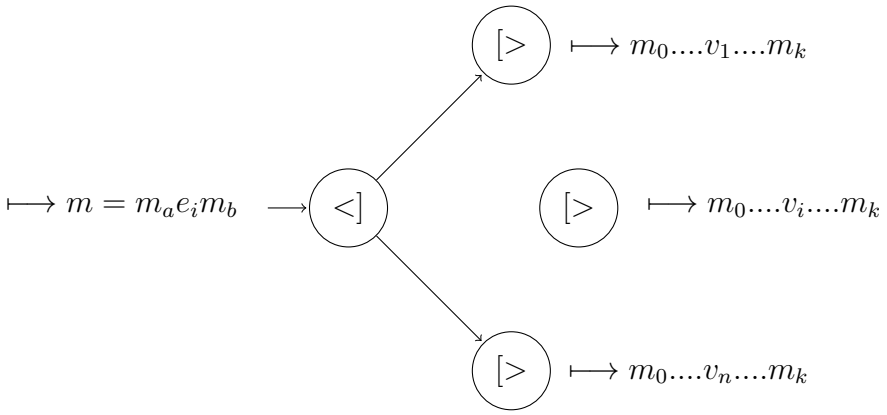
$$e : \begin{cases} \Sigma_1^* \to \Sigma_2^* \\ m \longmapsto \begin{cases} e(m_a).v_i.e(m_b) \text{ si } m = m_a.u_i.m_b \\ m \text{ sinon} \end{cases} \end{cases} .$$

e est dit $CORRECTEUR$ si $\Sigma_1 = \Sigma_2$.

clarifications

Les encodeurs ne sont en outre que des encodages spécifiques, qui changent certains patterns au profit de d'autres. Ce sont des applications sur notre magma, mais ceux-ci ne se comportent pas comme des morphismes de groupes (si l'analogie nous est permise). Soit $\forall f$ encodeur, il est FAUX de dire que :

$$\forall (m_a, m_b) \in (\Sigma_1^*)^2 f(m_a.m_b) = f(m_a).f(m_b)$$



exemple

»>coder = EncodeAutomate("a" : "c" , "aa" : "b")
»>coder.sequence("aaaaaaaaaaabaaaaaaba")
»>'bbbbbcbbbbba'

Activateur

Soit Σ un alphabet.
Soit $S_L \subseteq \Sigma^*$.

On appel $ACTIVATEUR$, tout encodeur φ tel que :

$$\forall m = m_1.m_2....m_n \in \Sigma^* \ \varphi : \begin{cases} \Sigma^* \to \{0,1\} \\ m \longmapsto \begin{cases} 1 \text{ si } \exists i \in [|1;n|] \text{ tel que } m_i \in S_L \\ 0 \text{ sinon} \end{cases} \end{cases}$$

.

clarifications

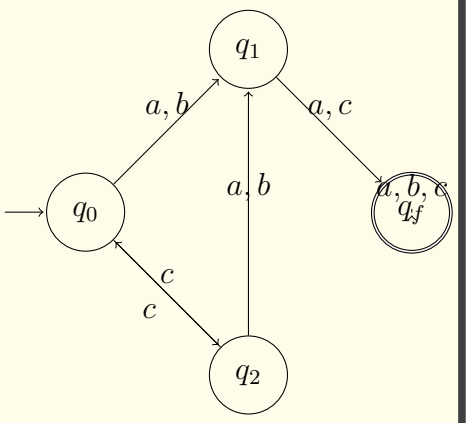
L'activateur est simplement une fonction booléenne sur notre magma. Ils vont nous être utiles, afin de mener des études statistiques sur de larges portions de textes, et d'ainsi nous ramener à des résultats numériques.

Automates

Soit $n \in N$ on pose $E = [|1; n|]$, un ensemble d'etats fini.
Soit $(q_0, q_f) \in E^2$, respectivement son etat iniatial et acceptant.
Soit Σ un alphabet.
Soit f une fonction dite de *transition*, tel que :

$$f \ : \ \begin{cases} E \times \Sigma \to P(E) \\ (x, a) \longmapsto f(x, a) \subseteq E \end{cases}$$

Alors (E, Σ, f) est un *AUTOMATE*.

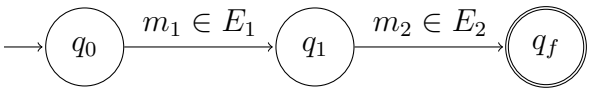


Automate Structurel

Soit Σ un alphabet.
Soit L_Σ un langage sur Σ, a base disjointe $B = (E_1, E_2, ..., E_n)$ avec $n \in N$.
Soit $S = [|1, n|]$, un ensemble de sommets.
Soit $f : (S, \Sigma^*) \to S$, une fonction de transition sur (S, Σ).
Soit $(q_0, q_f) \in S^2$, respectivement son etat iniatial et acceptant.

$$f' \ : \ \begin{cases} S \times \Sigma^* \to P(E) \\ (s, m) \longmapsto \begin{cases} f(s, m) \text{ si } m \in E_{f(s,m)} \\ \emptyset \text{ sinon} \end{cases} \end{cases}$$

(S, Σ^*, f') est donc un automate structurel sur B.



.

Théorie et Modélisation

Il est ardu de définir la langue française, comme étant l'un de nos objets précédemment défini. Ceci pour la simple et bonne raison, qu'il est difficile d'en déterminer les frontières. Une phrase est-elle française pour la simple raison qu'elle est syntaxiquement correcte[1], ou faut-il nécessairement que sa sémantique soit bonne[2] ?

En d'autres termes, suffit-il de former une phrase à partir d'un n-uplet quelconque de mots de la langue française, pour former une phrase correcte(indépendamment de son sens) ? Si oui, notre langue française peut être considérée comme une partie stable d'un autre langage plus large, en cas contraire il faudra en énumérer les structures[3] ?

Cela pose un problème autrement plus fondamental. Nous serons amenés à travailler avec des textes de fantaisie, dont les auteurs sont particulièrement friands de noms inexistants. Considérant la phrase : "La compagnie prenait alors la route de Kathovar." , il nous faut faire un choix.

Il nous est alors soit interdit de considérer cette phrase comme étant française, ou dans un second cas, il nous faut faire le constat qu'une phrase française peut-être constitue d'éléments qui ne le sont pas nécessairement. Cela tient au fait que la langue française, est une langue naturelle et non une langue construite.

Mots et Phrases

Nous voulons entrainer un modèle à produire des phrases françaises. Si comme le veut la grammaire française : "une phrase commence par une majuscule et finie par un point", il nous faut alors nuancer son applicabilité a notre base de données.

En effet, si une phrase est supposée commencer par une majuscule et que sa conclusion est suivie d'un point, toutes majuscules n'impliquent pas

1. "Pierre Chat la lancer a-t-il ?" est-elle une phrase de la langue Française ?
2. "Le chat a-t-il lancer la pierre ?" est-elle une phrase française pour la simple et bonne raison qu'elle a du sens ?
3. Pour espérer approximer la langue française par un langage a bases disjointes ?

forcément la formation d'une phrase, et tous les points n'en concluent pas nécessairement une !

Or extraire les phrases puis les mots d'un texte, est une condition nécessaire à l'étude préalable de nos textes.

Par ailleurs, extraire les phrases d'un texte d'origine, se révèle être un exercice bien moins intuitif qu'il n'y parait ! "Nous pouvons citer Gorge R.R Martin et Philipe K. Dick !", cette seule phrase suffit à expliquer l'inefficacité d'une méthode naïve d'extraction de phrase.

Extraire les mots d'une phrase, met également la méthode naïve(la simple utilisation de la fonction split sur notre texte) en échec(en raison des espaces insécables(certains mots contiennent un ou plusieurs espaces)).

Parlons sémantique

Dire qu'une phrase est bien formée ou non, est non seulement l'objet de notre étude, mais également une question délicate ! Nous ne pouvons le faire que par une succession d'exemples, puis parier sur le fait que cette énumération soit suffisamment exhaustive (bien que cela, soit peu probable). C'est pour cela, que nous préférons, approximer la langue française(qui est une langue naturelle), par un langage formel dont nous aurons déterminer au préalable la base.

Il nous faut composer avec l'hypothèse, que la langue française est composée de structures intelligibles, que nous pouvons énumérées, à l'aide d'études statistiques sur des phrases bien formées. Pour ainsi construire les bases disjointes de notre langage formel. Pour ce faire, nous comptons séparer les mots de la langue française, selon leur fonctions (fonctions sémantiques), puis énuméré leurs différents enchainements, dans les phrases à analyser.

A priori, trouver des textes en bon français, n'est pas chose bien compliquée ; de façon intuitive, il conviendra de collecter nos exemples à partir de textes d'auteurs et/ou d'académistes. Notre capacité à en extraire correctement les phrases puis les mots, selon leurs fonctions, déterminera la qualité de notre analyse.

fonctions des mots

Nous possédons une liste (conséquente, mais non exhaustive) de mots de la langue française (en fichier texte). Nous remarquons également que le

dictionnaire français en ligne : "Larousse"[1], nous fournit dans ses descriptions : la fonction (ou les fonctions) de chaque mots. En s'inspirant d'une API, qui lui est dédiée[2], nous programmons une fonction, capable de retourner la fonction d'un mot donne, si tenter que celui-ci ait été décrit dans le Larousse.

implémentation

```
def fundamental(word : str):
    """obtenir la fonction d'un mot"""
    url = "https://www.larousse.fr/dictionnaires/francais/" +
    word.lower()
    soup = BeautifulSoup(requests.get(url=url).text, 'html.
   parser')
    if soup.select('p') != []:
        return unicodedata.normalize("NFKD", soup.select('p')
   [0].text)
    else:
        return None
```

Listing 1 – Python classification Implementation

Nous construisons et sauvegardons, en un premier temps, un dictionnaire (python), contenant les mots de notre liste et leurs fonctions. Pour pallier la lenteur problématique de notre algorithme de web scrapping, nous multithreadrons ce handicap pour en faire un problème mineur.

Sur les 22735 mots de notre liste, nous comptons 22661 fonctions *trouvees*.

1. https://www.larousse.fr/dictionnaires/francais
2. https://github.com/quentin-dev/larousse_api

Parlons Aquisition de Donnees

Préambule

Pour des raisons de temps d'exécution, notre analyse sémantique (qui se fera en plusieurs temps), sera comprise dans notre grand model d'acquisition de données, chargé de collecter des livres sur internet, puis d'en extraire les textes. Celui-ci traitera le texte une première fois, l'analysera, puis le traitera une seconde fois. Pour ce faire, nous aurons besoin de nos structures précédemment définies d'activateurs , d'automates et d'encodeurs.

Nous avons peu de moyens pour influer sur la qualité du model d'entrainement, c'est pour cela que nous baserons pratiquement l'entièreté de notre stratégie sur l'amélioration de la qualité de notre donnée d'entrainement. Ainsi, notre étude, se portera principalement sur le processus d'acquisition de données, de leur traitement et de leur analyse.

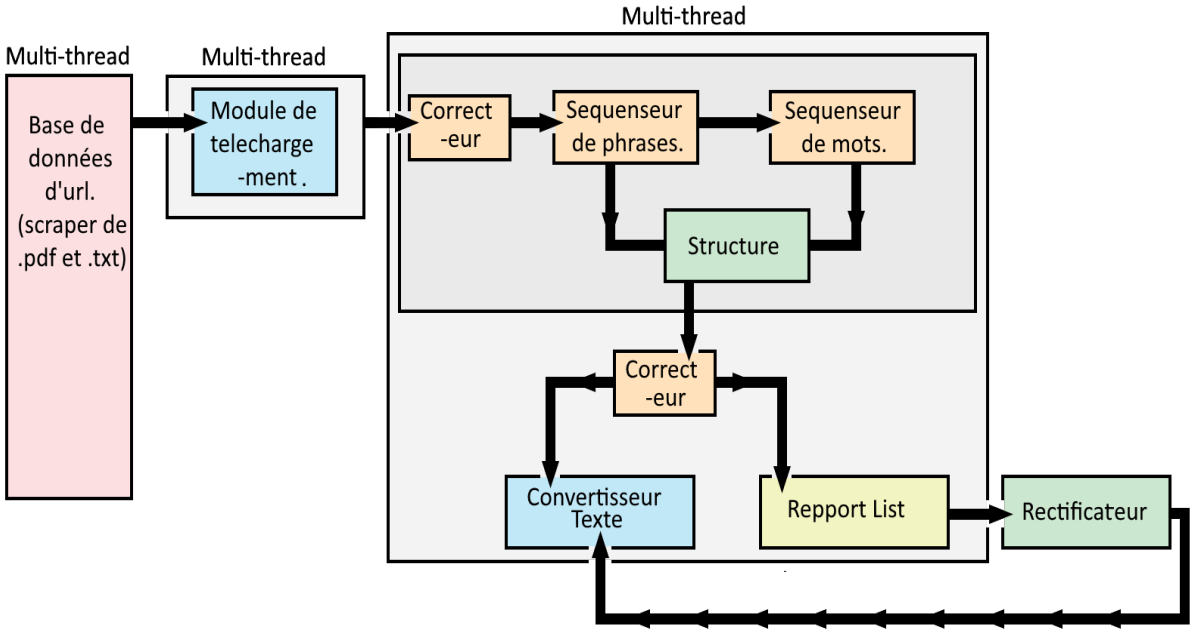
Traiter de grandes quantités de données, est en général un exercice préparatoire a une course contre la montre. Il nous faut donc, veiller à conserver une vigilance absolue au niveau de la complexité de nos algorithmes. On la vue précédemment, faire appel à de la programmation asynchrone, peut nous permettre dans une large mesure, de pallier ce problème. Cependant démultiplier le nombre de thread qui travaillent en même temps, démultiplie également le risque que certains d'entre eux échouent, dans leur requêtes(car l'on envoie et l'on reçoit des paquets d'informations). Il nous faut donc conserver un certain équilibre.

Pour ce faire, nous utiliserons fréquemment notre fonction : "src manager", que voici :

```
1 def src_manager(file_procc : [str], n : int):
2     """segmente notre liste en plusieurs listes plus petites
    """
3     if len(file_procc) <= n:
4         return [file_procc]
5     else:
6         return [file_procc[:n]] + src_manager(file_procc[n:],
    n)
```

Listing 2 – Python ramification Implementation

Architècture de notre Modèle de Web Scraping



Celle-ci possède 4 phases :

1. Nos robots web scrolleurs parcours nos pages pour en répertorier les url.
2. Nous téléchargeons lesdits documents (pdf ou texte).
3. Apres analyse, le model décide si il doit l'enregistrer dans nos données d'entrainement (si il a réussi à en cartographier les structures sémantiques), ou si il l'envoie au rectificateur.
4. le rectificateur enregistre dans un fichier les phrases ou les mots que le modele n'arriver pas a cataloguer.

Web Scrollers

Apres délibérations, nous avons conservé 4 sources de données majeures, que nous serons amenés à explicitées par la suite. Pour chacun de ces sites, nous avons analyser leur architecture, puis à l'aide d'automates, avons classer leurs liens pertinents, en les répertoriant dans des listes. Ces listes, nous servirons par la suite, afin de télécharger les ouvrages gratuits, proposés par ces sites.

Pour extraire les phrases de notre texte, nous le parcourons tout entier, en instanciant notre tête à chaque majuscules précédées d'un point (l'initialisant a 0), puis en instanciant notre queue à chaque points précédés d'une majuscule. En excluant certains patterns récurrent, tel que : "M. ", puis en ajoutant a une liste chacune de ces séquences, nous ramifions notre texte par phrases.

Nous partons du principe, que nous avons une phrase et que nous voulons en extraire les mots. Nous initialisons alors au préalable une liste des séquences de notre phrase, en utilisant la fonction native de python : "split()", sur notre texte. Nous avons ainsi découper notre phrase, à chaque espaces. Puis nous nous munissons d'un dictionnaire référencent tous les mots (de notre liste de mots français), contenant des espaces non sécables. Nous additionnons les séquences ainsi obtenues, tant que cette somme appartient à notre dictionnaire, puis, appliquons le même procédé de façon récursive : sur le reste de notre liste de séquences.

EXEMPLE : "Je m'étais assoupi parce que la journée arrivait à son terme." Nous avons notre liste de séquences suivante :

["Je","m'étais","assoupi","parce","que","la","journée","arrivait","à","son","terme"]
Nous obtenons après analyse :

["Je","m'étais","assoupi","parce que","la","journée","arrivait","à","son","terme"]

Rôle de nos automates

Nos automates, ont un rôle double : Apres analyse de nos données, nous obtenons un dictionnaire contenant : les structures sémantiques de notre texte, et de leur occurrence dans notre donnée d'origines. Nous initialisons alors, une liste d'automates structurels à partir de ce dictionnaire. De façons périodiques, notre modèle LLM, sauvegardera ses paramètres lors de son entrainement, puis une étude statistique sera menée à partir de notre liste d'automates structurels, qui mesurons l'occurrence de ces structures sémantiques, dans ses productions. En outre à l'aide de ces fréquences, nous pourrons nous ramenés à des notions de distance sur un espace vectoriel a dimension finie R^n (en mesurant la distances entres les fréquences obtenues sur nos données d'entrainement et celles obtenues à partir de notre model), pour mesurer sa capacité, à produire des phrases qui ont du sens(correctes sémantiquement).

Les automates ont un second rôle, moins abstrait : trier les url obtenues par nos web scrollers. Prenons notre cas de base : Nous nous trouvons sur une page web référencent un e-book gratuit, en format PDF. Celle-ci contient, plusieurs liens plus ou moins pertinents : -Des liens extérieurs menants a d'autres sites (comme des pages Facebook ou X (anciennement twitter)) -Des liens menants a la page d'accueil du site, ou à ses diverses catégories. -Et bien sur des liens contenants des informations sur le dit livre, dont le plus important se trouve être le lien d'accès à sa version PDF. Faire la différence entre ces liens, est aisé pour un opérateur humain et l'est beaucoup moins, pour un robot. Cependant, analyser la structure de ce site, nous permet de repérer des patterns récurrents dans ces liens.

Par exemple les liens donnent accès à ses livres numériques, commencent tous par un "https", suivit du nom de domaine du site, puis par un "ebook" et fini par un ".pdf". Nous faisons alors appel à notre constructeur d'automate, à l'aide de ces données.

Il nous vient immédiatement a l'esprit la norme 2 sur R^n, cependant (et malgres une equivallence des normes), rien ne nous dit que celle-ci soit la plus precise, pour mesurer nos ecarts semantiques.

```
def Norme_2_Distance(vect_1 : (int) , vect_2 : (int)):
    assert (len(vect_1) != len(vect_2))
    return sum([(vect_1[i] - vect_2[i])**2 for i in range(len
   (vect_1))])**(1/2)
```

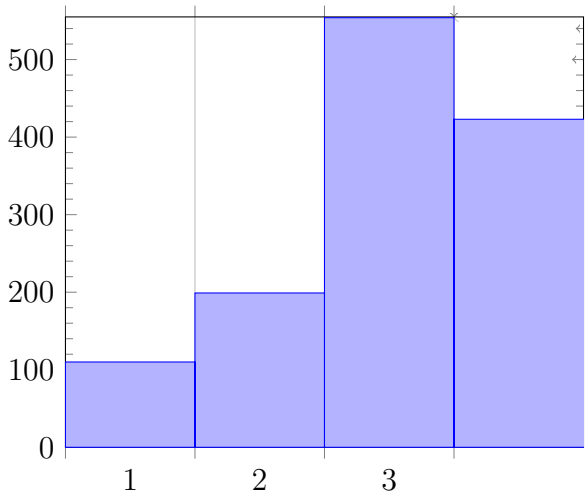
Listing 3 – Python Norme 2 Implementation

Jeu de données

Regardons de plus pres notre donnée.

Nous avons ci-dessous, la répartition de nos données, scrappées par nos algorithme. Celles-ci ont étés récoltées sur 4 sites distincts.

1. `http://www.livrespourtous.com`, qui représente 8.5% de nos données d'entrainement.
2. `https://www.vousnousils.fr`, qui représente 15.4% de nos données d'entrainement.
3. `https://www.edition999.info`, qui représente 43% de nos données d'entrainement.
4. `https://www.gutenberg.org`, qui représente 32.8% de nos données d'entrainement.



Il est a noté, que seul Gutenberg, présente des livres numériques en format texte(.txt) standard. On a ainsi 68% de nos données, extraite de pdf. Concernant la part d'œuvres de fantaisies (ou fantastiques) dans nos données d'entrainement, celle-ci devrait excéder les 50%, sachant qu'elle compose en totalité les œuvres d'édition 999 et une bonne moitié de celle de livrespourtous. Il est cependant impossible d'en connaitre la composition, pour Gutenberg.

Plus précisément nous avons utiliser nos web scrollers sur les liens ci-dessous :

1. `http://www.livrespourtous.com/e-books/list/onecat/Livres-electroniques+Contes-et-nouvelles+Fantastique-et-Science-Fiction/0/all_items.html`.
2. `https://www.edition999.info/-Fantastique-.html`.
3. `http://www.livrespourtous.com/e-books/list/onecat/Livres-electroniques+Romans+Fantastique-et-SF/0/all_items.html`.
4. `http://www.livrespourtous.com/e-books/list/onecat/Livres-electroniques+Lettres-et-memoires/0/all_items.html`.
5. `http://www.livrespourtous.com/e-books/list/onecat/Livres-electroniques+Documents-et-essais/0/all_items.html`.
6. `http://www.livrespourtous.com/e-books/list/onecat/Livres-electroniques+Romans+Historique/0/all_items.html`.

Du code et encore du code

séquenseurs

```
"""
All Sequensers Models
"""
#Imports Pannel requierement
import matplotlib.pyplot as plt
import threading
import os
import io
import pickle
import csv
import requests
import unicodedata
from bs4 import BeautifulSoup
accept_dict = {"." : None, "!" : None, "?" : None}
class EncodeAutomate():
    def __init__(self, activation_function : dict):
        self.af = activation_function
        self.activation = {}
        def insere(sequence : str , sequence_list : [str]):
            if sequence_list == []:
                return [sequence]
            for inc in range(len(sequence_list)):
                if (len(sequence) >= len(sequence_list[inc]))
:
                    return sequence_list[:inc] + [sequence] +
 sequence_list[inc:]
        for sequence in self.af:
            if sequence[0] in self.activation:
                self.activation[sequence[0]] = insere(
sequence, self.activation[sequence[0]])
            else:
                self.activation[sequence[0]] = [sequence]
    def sequence(self, sentence : str):
        def activation(work : str , sentence : str):
            assert len(work) == len(sentence)
            if len(work) == 0:
                return True
            else:
                if (work[0] == sentence[0]) or (sentence[0]
== "Âí"):
                    return activation(work[1:],sentence[1:])
```

```
                else:
                    return False
        my_new_sentence = ""
        increase = 0
        while increase < len(sentence):
            if sentence[increase] in self.activation:
                NotEqual = True
                sec_increase = 0
                while NotEqual and (sec_increase < len(self.
    activation[sentence[increase]])):
                    sequence = self.activation[sentence[
    increase]][sec_increase]
                    if increase + len(sequence) <= len(
    sentence):
                        if activation(sentence[increase:
    increase + len(sequence)],sequence):
                            my_new_sentence = my_new_sentence
     + self.af[sequence]
                            increase = increase + len(
    sequence)
                            NotEqual = False
                        else:
                            sec_increase += 1
                    else:
                        sec_increase += 1
                if NotEqual:
                    my_new_sentence = my_new_sentence +
    sentence[increase]
                    increase += 1
            else:
                my_new_sentence = my_new_sentence + sentence[
    increase]
                increase += 1
        return my_new_sentence
def shearch_patterns(txt : str, i : int, pattern : str):
    return (((i + len(pattern)) <= len(txt)) and (txt[i:i +
    len(pattern)] == pattern))
anti_point_encoder = EncodeAutomate({"." : "", "!" : "", "?"
    : ""})
coder = EncodeAutomate({"\n" : " ","--" : "","_" : "","Âń" :
    "" , "Âż" : "" ,"," : "", ";" : "", ":" : "", "n'" : "ne
    pas","qu'" : "que ","Qu'" : "Que ","quâĂŹ" : "que ","QuâĂŹ
    " : "Que ","nâĂŹ" : "ne pas"})
def get_sentence_list(my_txt : str):
    if len(my_txt) > 1:
```

```
70          initialize_list = []
71          head_play = 0
72          in_stand = True
73          for i in range(1,len(my_txt)):
74              if (my_txt[i] != (my_txt[i].lower())) and (not
    shearch_patterns(my_txt,i,"M. ")) and (not in_stand):
75                  head_play = i
76                  in_stand = True
77              elif (my_txt[i] in accept_dict) and (my_txt[i-1]
    != "M") and in_stand:
78                  initialize_list.append(coder.sequence(my_txt[
    head_play:i] + my_txt[i]))
79                  in_stand = False
80              elif i == (len(my_txt) - 1):
81                  initialize_list.append(coder.sequence(my_txt[
    head_play:i]))
82          return initialize_list
83      else:
84          return []
85  def get_space_words():
86      fileIn = open("data collections\liste_francais.txt","r")
87      content = [(line[:len(line) - 1]) for line in fileIn.
    readlines() if (" " in line[:len(line) - 1])]
88      fileIn.close()
89      return dict(zip(content,[None for a in content]))
90  def sub_analyse(word : str, work_list : [str] , word_dict :
    dict):
91      word = anti_point_encoder.sequence(word)
92      if work_list == []:
93          return (word,[])
94      elif (word + " " + work_list[0]) not in word_dict:
95          return (word,work_list)
96      else:
97          return sub_analyse(word + " " + work_list[0],
    work_list[1:],word_dict)
98  def init_analyse(work : [str], non_list : [str],
    words_with_space : dict):
99      if work == []:
100         return non_list
101     else:
102         results = sub_analyse(work[0],work[1:],
    words_with_space)
103         return init_analyse(results[1],non_list + [results
    [0]],words_with_space)
104 def get_words_list_of_a_sentence(sentence : str,
```

```
    words_with_space : dict):
105     my_stock = sentence.split()
106     return init_analyse(my_stock, [], words_with_space)
107     pass
```

Listing 4 – Python Code des Sequenseurs

activateurs

```
1  """
2  All Activators Models
3  """
4  class ActivatorAutomate():
5      def __init__(self):
6          pass
7      def recognize(self, sequence : str):
8          return True
9  class ActivatorName(ActivatorAutomate):
10     def __init__(self):
11         ActivatorAutomate.__init__(self)
12         pass
13     def recognize(self, word : str):
14         return ((word[0] != (word[0].lower())) or word.
       isdigit() or ((len(word)>2) and (word[0]+word[1] in ["d'",
       "l'"]) and (word[2] != word[2].lower())) or ((len(word) >
       1) and word[0].isdigit and (word[1] != (word[1].lower())))
       )
15 class TrivialAutomate(ActivatorAutomate):
16     def __init__(self, transitions : [dict], final_state :
       int, initial_state=0):
17         ActivatorAutomate.__init__(self)
18         self.transitions = transitions
19         self.final_state = final_state
20         self.initial_state = initial_state
21         pass
22     def recognize(self, word : str, state=0):
23         if word == "":
24             return (state== self.final_state)
25         elif state > (len(self.transitions) - 1):
26             return False
27         else:
28             if word[0] in self.transitions[state]:
29                 return self.recognize(word[1:], self.
       transitions[state][word[0]])
```

```
            else:
                return False
class LooppedAutomate(ActivatorAutomate):
    def __init__(self, transitions : [dict], final_state :
   int, initial_state=0):
        ActivatorAutomate.__init__(self)
        self.transitions = transitions
        self.final_state = final_state
        self.initial_state = initial_state
        pass
    def recognize(self, word : str, state=0):
        if word == "":
            return (state== self.final_state)
        elif state > (len(self.transitions) - 1):
            return False
        else:
            if word[0] in self.transitions[state]:
                return self.recognize(word[1:], self.
   transitions[state][word[0]])
            else:
                return self.recognize(word[1:], state)
class AssyncAutomate(ActivatorAutomate):
    def __init__(self, transitions : [[bool,int,dict]],
   final_state : int, initial_state=0):
        ActivatorAutomate.__init__(self)
        self.transitions = transitions
        self.final_state = final_state
        self.initial_state = initial_state
        pass
    def recognize(self, word : str, state=0):
        if word == "":
            return (state== self.final_state)
        elif state > (len(self.transitions) - 1):
            return False
        else:
            if word[0] in self.transitions[state][2]:
                return self.recognize(word[1:], self.
   transitions[state][2][word[0]])
            else:
                if not self.transitions[state][0]:
                    return self.recognize(word[1:], state)
                else:
                    return self.recognize(word[1:], self.
   transitions[state][1])
class AssyncAutomateMono(ActivatorAutomate):
```

```
70      def __init__(self, transitions : [[bool,int,dict]],
    final_state : int, initial_state=0):
71          ActivatorAutomate.__init__(self)
72          self.transitions = transitions
73          self.final_state = final_state
74          self.initial_state = initial_state
75          pass
76      def recognize(self, word : str, state=0):
77          if word == "":
78              return (state== self.final_state)
79          elif state > (len(self.transitions) - 1):
80              return False
81          else:
82              if word[0] in self.transitions[state][2]:
83                  return self.recognize(word[1:], self.
    transitions[state][2][word[0]])
84              else:
85                  if not self.transitions[state][0]:
86                      return self.recognize(word[1:], state)
87                  else:
88                      return self.recognize(word, self.
    transitions[state][1])
89  def constructEndAutomata(construct_lists : [[str]]):
90      basic_graph_construct = [[False,0,{}]]
91      tails_list = [0]
92      for construct_list in construct_lists:
93          sub_tail_list = []
94          for word in construct_list:
95              state = tails_list[-1]
96              indice = len(word) - 1
97              for i in range(indice):
98                  if word[i] in basic_graph_construct[state
    ][2]:
99                      state = basic_graph_construct[state][2][
    word[i]]
100                 else:
101                     new_node = len(basic_graph_construct)
102                     basic_graph_construct.append([True,
    tails_list[-1],{}])
103                     basic_graph_construct[state][2][word[i]]
    = new_node
104                     state = new_node
105             sub_tail_list.append((state,word[indice]))
106         new_big_node = len(basic_graph_construct)
107         basic_graph_construct.append([False,new_big_node,{}])
```

```
        tails_list.append(new_big_node)
        for tail in sub_tail_list:
            basic_graph_construct[tail[0]][2][tail[1]] = new_big_node
    basic_graph_construct[tails_list[-1]][0] = True
    basic_graph_construct[tails_list[-1]][1] = tails_list[-2]
    return AssyncAutomate(basic_graph_construct,tails_list[-1])
def constructEndAutomataNotEq(construct_lists : [[str]]):
    basic_graph_construct = [[False,0,{}]]
    tails_list = [0]
    for construct_list in construct_lists:
        sub_tail_list = []
        for word in construct_list:
            state = tails_list[-1]
            indice = len(word) - 1
            for i in range(indice):
                if word[i] in basic_graph_construct[state][2]:
                    state = basic_graph_construct[state][2][word[i]]
                else:
                    new_node = len(basic_graph_construct)
                    basic_graph_construct.append([True,tails_list[-1],{}])
                    basic_graph_construct[state][2][word[i]] = new_node
                    state = new_node
            sub_tail_list.append((state,word[indice]))
        new_big_node = len(basic_graph_construct)
        basic_graph_construct.append([False,new_big_node,{}])
        tails_list.append(new_big_node)
        for tail in sub_tail_list:
            basic_graph_construct[tail[0]][2][tail[1]] = new_big_node
    return AssyncAutomate(basic_graph_construct,tails_list[-1])
class DictAutomate():
    def __init__(self, transitions : [dict], final_state : int, initial_state=0):
        self.transitions = transitions
        self.final_state = final_state
        self.initial_state = initial_state
        def ddelete_occurence(my_list : list):
            new_list = []
```

```
144             for k in my_list:
145                 if k not in new_list:
146                     new_list.append(k)
147             return new_list
148         self.links = [ddelete_occurence([k for k in tr.values
    ()]) for tr in transitions]
149         pass
150     def recognize(self, sentence : [str], state=0):
151         if sentence == []:
152             return (state== self.final_state)
153         else:
154             if sentence[0] in self.transitions[state]:
155                 return self.recognize(sentence[1:], self.
    transitions[state][sentence[0]])
156             else:
157                 return False
158     def recognize_sentence(self, sentence : str):
159         return self.recognize(sentence.split(" "), self.
    initial_state)
160 def constructEndAutomataMoloList(construct_lists : [str]):
161     basic_graph_construct = [[False,0,{}]]
162     tails_list = [0]
163     for construct_list in construct_lists:
164         sub_tail_list = []
165         for word in construct_list:
166             state = tails_list[-1]
167             indice = len(word) - 1
168             for i in range(indice):
169                 if word[i] in basic_graph_construct[state
    ][2]:
170                     state = basic_graph_construct[state][2][
    word[i]]
171                 else:
172                     new_node = len(basic_graph_construct)
173                     basic_graph_construct.append([True,
    tails_list[-1],{}])
174                     basic_graph_construct[state][2][word[i]]
    = new_node
175                     state = new_node
176             sub_tail_list.append((state,word[indice]))
177         new_big_node = len(basic_graph_construct)
178         basic_graph_construct.append([False,new_big_node,{}])
179         tails_list.append(new_big_node)
180         for tail in sub_tail_list:
181             basic_graph_construct[tail[0]][2][tail[1]] =
```

```
    new_big_node
182     basic_graph_construct[tails_list[-1]][0] = True
183     basic_graph_construct[tails_list[-1]][1] = tails_list[-2]
184     return AssyncAutomateMono(basic_graph_construct,
    tails_list[-1])
```

Listing 5 – Python Code des Activateurs

web scrapeur

```
1  import all_sequensers_models as asqs
2  import requests
3  import unicodedata
4  from bs4 import BeautifulSoup
5  import os
6  import io
7  import pickle
8  import csv
9  import threading
10 text_cleaner_encoder = asqs.EncodeAutomate({"Ãľ" : "e","ÃÍ" :
       "e","Ãł" : "e","Ãń" : "e",
11                                             "-\n" : "","  - "
   : "\n - ","âĂŞ ÂÍ âĂŞ " : "",
12                                             "âĂŞ ÂíÂÍ âĂŞ " :
       "" , "âĂŞ ÂíÂíÂÍ âĂŞ " : "",
13                                             " âĂŤ " : "\n âĂŤ
       ","-\n" : "","Ãć" : "a",
14                                             "Ãă" : "a","Ãď" :
       "a","Ãť" : "o","Ãű" : "o",
15                                             "Ãő" : "i","Ãŕ" :
       "i","Ãż" : "u","Ãij" : "u","  " : ""})
16 def save(Dictionnaire : dict ,path="data collections\
   database_dictionnary.obj"):
17     fileIn = open(path,"wb")
18     pickle.dump(Dictionnaire,fileIn)
19     fileIn.close()
20 def load(path="data collections\database_dictionnary.obj"):
21     fileout = open(path,"rb")
22     Load = pickle.load(fileout)
23     fileout.close()
24     return Load
25 def MultiThreadFragmentation():
26     Words_Functions = {}
```

```
27      Interupt_Dict = {'conjonction de coordination' : None,'
    conjonction' : None,"adverbe" : None,'proverbe': None,'
    interjection' : None,"adjectif" : None,
28                       "article" : None,"preÌĄposition" : None,
    'preÌĄfixe' : None,"nom" : None,"verbe" : None,"Participe"
     : None,"pronom" : None,"locution" : None}
29      register_list = ['conjonction de coordination','
    conjonction',"adverbe",'proverbe','interjection',"adjectif
    ","article","preÌĄposition",'preÌĄfixe',"nom","verbe","
    Participe","pronom","locution"]
30      def initializeRegister(word_path : str , word : str):
31          if word_path != None:
32              if word_path in Words_Functions:
33                  Words_Functions[word_path][word] = None
34              else:
35                  Words_Functions[word_path] = {word : None}
36          else:
37              initializeRegister("nom",word)
38      def three_activation(word_path : str, word : str,
    pattern_list : [str]):
39          if word_path in Interupt_Dict:
40              initializeRegister(word_path,word)
41          else:
42              if word_path != None:
43                  for pattern in pattern_list:
44                      if pattern in word_path:
45                          initializeRegister(pattern,word)
46                          return None
47                  initializeRegister("nom",word)
48                  pass
49      def fundamental(word : str):
50          """obtenir la fonction d'un mot"""
51          url = "https://www.larousse.fr/dictionnaires/francais
    /" + word.lower()
52          soup = BeautifulSoup(requests.get(url=url).text, '
    html.parser')
53          if soup.select('p') != []:
54              return unicodedata.normalize("NFKD", soup.select(
    'p')[0].text)
55          else:
56              return None
57      def encapsule(n : int , file_procc : [str]):
58          """craie des ramifications d'une meme liste par
    paquet de n valeurs"""
59          if len(file_procc) <= n:
```

```
60              return [file_procc]
61          else:
62              return [file_procc[:n]] + encapsule(n,file_procc[
    n:])
63      class MyThread(threading.Thread):
64          def __init__(self, words : [str]):
65              threading.Thread.__init__(self)
66              self.words = words
67          def run(self):
68              for word in self.words:
69                  path = fundamental(word)
70                  three_activation(path,word,register_list)
71              pass
72      class MyThreadRelation(threading.Thread):
73          def __init__(self, words : [str]):
74              threading.Thread.__init__(self)
75              self.words = words
76          def run(self):
77              i = 0
78              for word in self.words:
79                  print(str(i) + "%")
80                  path = fundamental(word)
81                  i = i + 1
82                  three_activation(path,text_cleaner_encoder.
    sequence(word),register_list)
83              pass
84      fileIn = open("data collections\liste_francais.txt","r")
85      content = [line[:len(line) - 1].lower() for line in
    fileIn.readlines()]
86      fileIn.close()
87      partition = encapsule(100,content)
88      Multi_Thread = [MyThreadRelation(partition[0])] + [
    MyThread(partition[j]) for j in range(1,len(partition))]
89      for th in Multi_Thread:
90          th.start()
91      for th in Multi_Thread:
92          th.join()
93      save(Words_Functions)
94  def MultiThreadFragmentationCorrecteur(content : list):
95      Words_Functions = load()
96      Interupt_Dict = {'conjonction de coordination' : None,'
    conjonction' : None,"adverbe" : None,'proverbe': None,'
    interjection' : None,"adjectif" : None,
97                       "article" : None,"preÌĄposition" : None,
    'preÌĄfixe' : None,"nom" : None,"verbe" : None,"Participe"
```

```
    : None,"pronom" : None,"locution" : None}
98   register_list = ['conjonction de coordination','
    conjonction',"adverbe",'proverbe','interjection',"adjectif
    ","article","preÌĄposition",'preÌĄfixe',"nom","verbe","
    Participe","pronom","locution"]
99   def initializeRegister(word_path : str , word : str):
100      if word_path != None:
101          if word_path in Words_Functions:
102              Words_Functions[word_path][word] = None
103          else:
104              Words_Functions[word_path] = {word : None}
105      else:
106          initializeRegister("nom",word)
107  def three_activation(word_path : str, word : str,
    pattern_list : [str]):
108      if word_path in Interupt_Dict:
109          initializeRegister(word_path,word)
110      else:
111          if word_path != None:
112              for pattern in pattern_list:
113                  if pattern in word_path:
114                      initializeRegister(pattern,word)
115                      return None
116              initializeRegister("nom",word)
117              pass
118  def fundamental(word : str):
119      """obtenir la fonction d'un mot"""
120      url = "https://www.larousse.fr/dictionnaires/francais
    /" + word.lower()
121      soup = BeautifulSoup(requests.get(url=url).text, '
    html.parser')
122      if soup.select('p') != []:
123          return unicodedata.normalize("NFKD", soup.select(
    'p')[0].text)
124      else:
125          return None
126  def encapsule(n : int , file_procc : [str]):
127      """craie des ramifications d'une meme liste par
    paquet de n valeurs"""
128      if len(file_procc) <= n:
129          return [file_procc]
130      else:
131          return [file_procc[:n]] + encapsule(n,file_procc[
    n:])
132  class MyThread(threading.Thread):
```

```
133         def __init__(self, words : [str]):
134             threading.Thread.__init__(self)
135             self.words = words
136         def run(self):
137             for word in self.words:
138                 path = fundamental(word)
139                 three_activation(path,word,register_list)
140             pass
141     class MyThreadRelation(threading.Thread):
142         def __init__(self, words : [str]):
143             threading.Thread.__init__(self)
144             self.words = words
145         def run(self):
146             i = 0
147             for word in self.words:
148                 print(str(i) + "%")
149                 path = fundamental(word)
150                 i = i + 1
151                 three_activation(path,text_cleaner_encoder.
    sequence(word),register_list)
152             pass
153     partition = encapsule(100,content)
154     Multi_Thread = [MyThreadRelation(partition[0])] + [
    MyThread(partition[j]) for j in range(1,len(partition))]
155     for th in Multi_Thread:
156         th.start()
157     for th in Multi_Thread:
158         th.join()
159     save(Words_Functions)
```

Listing 6 – Python Code des Web Scrappers

main

```
1  """
2  Big Scrapper Model
3  """
4  #Imports Pannel requierement
5  from bs4 import BeautifulSoup
6  import requests
7  import random
8  import os
9  import io
10 import pickle
```

```
import csv
import threading
import unicodedata
from PyPDF2 import PdfReader
import all_activators_models as actv
import all_sequensers_models as asqs
import dictionnary_scrapper as dscp
#globals Variables
text_cleaner_encoder = asqs.EncodeAutomate({"Ãľ" : "e","ÃÍ" : "e","Ãł" : "e","Ãń" : "e",
                                            "-\n" : "","  - " : "\n - ","âĂŞ ÂÍ âĂŞ " : "",
                                            "âĂŞ ÂíÂí âĂŞ " : "" , "âĂŞ ÂíÂíÂí âĂŞ " : "",
                                            " âĂŤ " : "\n âĂŤ","-\n" : "", "  " : "","Ãć" : "a",
                                            "Ãă" : "a","Ãď" : "a","Ãť" : "o","Ãű" : "o",
                                            "Ãő" : "i","Ãŕ" : "i","Ãż" : "u","Ãij" : "u","Ãğ" : "c"})
structurize_cleaner = asqs.EncodeAutomate({"-\n" : "","  - " : "\n - ","âĂŞ ÂÍ âĂŞ " : "",
                                           "âĂŞ ÂíÂí âĂŞ " : "" , "âĂŞ ÂíÂíÂí âĂŞ " : "",
                                           " âĂŤ " : "\n âĂŤ","-\n" : ""})
simplification_cleaner = asqs.EncodeAutomate({"Ãľ" : "e","ÃÍ" : "e","Ãł" : "e","Ãń" : "e","Ãć" : "a",
                                              "Ãă" : "a","Ãď" : "a","Ãť" : "o","Ãű" : "o",
                                              "Ãő" : "i","Ãŕ" : "i","Ãż" : "u","Ãij" : "u","Ãğ" : "c"})
analyse_coder = asqs.EncodeAutomate({"\n" : " ","--" : "","_" : "","Âń" : "" , "Âż" : "" ,"," : "", ";" : "", ":" : "", "âĂŞ" : ""})
txt_recorder_coder = asqs.EncodeAutomate({" " : "\t","Ãľ" : "e","ÃÍ" : "e","Ãł" : "e","Ãń" : "e","Ãć" : "a",
                                          "Ãă" : "a","Ãď" : "a","Ãť" : "o","Ãű" : "o",
                                          "Ãő" : "i","Ãŕ" : "i","Ãż" : "u","Ãij" : "u","Ãğ" : "c"})
verb_automata = actv.constructEndAutomataMoloList([["es","ons","ez","ent","ont","ait","ais","ions","iez","aient","erai","eras","era","erons","erez","eront","ant",
                                                    "asse","asses","
```

```
    Ãćt","assions","assiez","assent","rer","a","ais","etait","
    est","as","ÃľtÃľ","ete","es","Ãżmes","fumes",
37                                          "Ãłtes","etes","
    sommes","sois","soit","aient"]])
38 is_a_word_activator = actv.ActivatorName()
39 words_with_spaces_dict = asqs.get_space_words()
40 abrevation_dict = {"l'" : None, "s'" : None,"L'" : None, "S'"
     : None,"c'" : None, "C'" : None, "j'" : None, "J'" : None
    , "t'" : None,"T'" : None, "d'" : None, "D'" : None,"m'" :
     None, "M'" : None,
41                  "lâĂŹ" : None, "sâĂŹ" : None,"LâĂŹ" : None
    , "SâĂŹ" : None,"câĂŹ" : None, "CâĂŹ" : None, "jâĂŹ" :
    None, "JâĂŹ" : None, "tâĂŹ" : None,"TâĂŹ" : None, "dâĂŹ" :
     None, "DâĂŹ" : None,"mâĂŹ" : None, "MâĂŹ" : None}
42
43
44
45
46
47
48
49 #obj stockage
50 def save_dict(Dictionnaire : dict ,path="data collections\
    database_dictionnary.obj"):
51     fileIn = open(path,"wb")
52     pickle.dump(Dictionnaire,fileIn)
53     fileIn.close()
54 def load_dict(path="data collections\database_dictionnary.obj
    "):
55     fileout = open(path,"rb")
56     Load = pickle.load(fileout)
57     fileout.close()
58     return Load
59
60
61
62
63
64
65 #rsc management for multitask
66 def src_manager(file_procc : [str], n : int):
67     """segmente notre liste en plusieurs listes plus petites
    """
68     if len(file_procc) <= n:
69         return [file_procc]
```

```
    else:
        return [file_procc[:n]] + src_manager(file_procc[n:], n)
def get_all_links(url : str):
    my_request = requests.get(url)
    bsoup = BeautifulSoup(my_request.text,"html.parser")
    all_links = [l.get("href") for l in bsoup.findAll("a")]
    return all_links
def get_all_links_filter(url : str, filter_automata : callable):
    return [link for link in get_all_links(url) if (link!= None) and filter_automata(link)]
def get_all_links_anti_filter(url : str, filter_automata : callable):
    return [link for link in get_all_links(url) if (link!= None) and not filter_automata(link)]
def turn_a_pdf_innto_a_txt(path : str):
    pass
fast_dict = load_dict("data collections\database_fast_dict.obj")
```

```
#web scroller collect urls
def collection_0002():
    print("aquisition url, phase 02")
    filter_automata_01 = actv.constructEndAutomata([["vousnousils.fr"],[".pdf"]])
    links_list = get_all_links_filter("https://www.vousnousils.fr/ebooks-gratuits", filter_automata_01.recognize)
    return links_list
def collection_0004():
    print("aquisition url, phase 04")
    mapping = {}
    filter_automata_01 = actv.constructEndAutomataNotEq([["ebooks/"]])
    filter_automata_02 = actv.constructEndAutomata([["ebooks/
```

```
    "],["txt.utf-8"]])
105     links_list = src_manager(["https://www.gutenberg.org" + a
     for a in get_all_links_filter("https://www.gutenberg.org/
    browse/languages/fr#a50488", filter_automata_01.recognize)
    ], 20)
106     class TextScrapper(threading.Thread):
107         def __init__(self, work : [str]):
108             threading.Thread.__init__(self)
109             self.work = work
110             self.filter = filter_automata_02.recognize
111         def run(self):
112             for task in self.work:
113                 for url in get_all_links_filter(task,self.
    filter):
114                     if url not in mapping:
115                         mapping["https://www.gutenberg.org"+
    url] = None
116     bot_list = [TextScrapper(task_list) for task_list in
    links_list]
117     for bot in bot_list:
118         bot.start()
119     for bot in bot_list:
120         bot.join()
121     return list(mapping.keys())
122 def collection_0003():
123     print("aquisition url, phase 03")
124     basic_mapping = ["https://www.edition999.info/-
    Fantastique-.html?debut_articles=" +str(k*20)+"#
    pagination_articles" for k in range(1,16)] + ["https://www
    .edition999.info/-Fantastique-.html",
125                      "https://www.edition999.info/-Heroic-
    Fantasy-.html?debut_articles=10#pagination_articles",
126                      "https://www.edition999.info/-Heroic-
    Fantasy-.html?debut_articles=20#pagination_articles"]
127     mapping = {}
128     filter_automata_01 = actv.constructEndAutomataNotEq([['
    https','http',"pagination_articles",'-Fantastique-.html',
129                                                          '-
    Avant-premiere-.html', '-Science-Fiction-Anticipation-.
    html',
130                                                          '-
    Heroic-Fantasy-.html', '-Litterature-.html', '-Policier-et
    -suspense-.html',
131                                                          '-
    Poesie-.html', '-Classique-.html',
```

```
132                                                                 '-
    Autobiographie-Temoignage-Autofiction-.html', '-Jeunesse-.
    html',
133                                                                 '-
    Litterature-Erotique-.html', '-Biographie-.html', '-
    Histoire-courte-.html',
134                                                                 '-
    Langues-etrangeres-.html', '-Saga-.html',
135                                                                 '-
    Essai-Politique-Scolaire-Education-.html', '-Philosophie-
    et-spiritualite-.html',
136                                                                 '-
    Adolescents-et-Jeunes-Adultes-.html', '#']])
137     filter_automata_02 = actv.constructEndAutomataNotEq([["
    edition999.info"],["spip.php?action=telecharger&arg"]])
138     links_list = []
139     for main_link in basic_mapping:
140         links_list = links_list + ["https://www.edition999.
    info/" + link for link in get_all_links_anti_filter(
    main_link, filter_automata_01.recognize)]
141     links_list = src_manager(links_list,20)
142     class TextScrapper_03_01(threading.Thread):
143         def __init__(self, work : [str]):
144             threading.Thread.__init__(self)
145             self.work = work
146             self.filter = filter_automata_02.recognize
147         def run(self):
148             for task in self.work:
149                 for url in get_all_links_filter(task,self.
    filter):
150                     if url not in mapping:
151                         mapping[url] = None
152     class TextScrapper_03_02(threading.Thread):
153         def __init__(self, work : [str]):
154             threading.Thread.__init__(self)
155             self.work = work
156             self.filter = filter_automata_02.recognize
157         def run(self):
158             for k in range(len(self.work)):
159                 for url in get_all_links_filter(self.work[k],
    self.filter):
160                     if url not in mapping:
161                         mapping[url] = None
162                 print("etape : " + str(k))
163     bot_list = [TextScrapper_03_02(links_list[0])] + [
```

```
    TextScrapper_03_01(links_list[k]) for k in range(1,len(
    links_list))]
164     for bot in bot_list:
165         bot.start()
166     for bot in bot_list:
167         bot.join()
168     return list(mapping.keys())
169 def collection_0001():
170     print("aquisition url, phase 01")
171     basic_mapping = []
172     scroll_mapping = ["http://www.livrespourtous.com/e-books/
    list/onecat/Livres-electroniques+Contes-et-nouvelles+
    Fantastique-et-Science-Fiction/"+str(k)+"/all_items.html"
    for k in range(5)] + ["http://www.livrespourtous.com/e-
    books/list/onecat/Livres-electroniques+Romans+Fantastique-
    et-SF/"+str(k)+"/all_items.html" for k in range(11)]+["
    http://www.livrespourtous.com/e-books/list/onecat/Livres-
    electroniques+Documents-et-essais/"+str(k)+"/all_items.
    html" for k in range(6)]+["http://www.livrespourtous.com/e
    -books/list/onecat/Livres-electroniques+Lettres-et-
    memoires/"+str(k)+"/all_items.html" for k in range(5)]+["
    http://www.livrespourtous.com/e-books/list/onecat/Livres-
    electroniques+Romans+Historique/"+str(k)+"/all_items.html"
     for k in range(17)]
173     mapping = {}
174     filter_automata_00 = actv.constructEndAutomata([["e-books
    /detail"],["all_items.html"]])
175     filter_automata_02 = actv.constructEndAutomata([[".pdf"
    ]])
176     class TextScrapper_01_01(threading.Thread):
177         def __init__(self, work : [str]):
178             threading.Thread.__init__(self)
179             self.work = work
180             self.filter = filter_automata_02.recognize
181         def run(self):
182             for task in self.work:
183                 for url in get_all_links_filter(task,self.
    filter):
184                     if url not in mapping:
185                         mapping[url] = None
186     links_list = []
187     for link in scroll_mapping:
188         links_list = links_list + get_all_links_filter(link,
    filter_automata_00.recognize)
189     print("etape 1")
```

```
    bot_list = [TextScrapper_01_01(work) for work in
   src_manager(links_list,2)]
    for bot in bot_list:
        bot.start()
    for bot in bot_list:
        bot.join()
    return list(mapping.keys())
#data manager
def get_the_file_pdf(url : str, name : str):
    request = requests.get(url)
    try:
        with open("back room\pdf\ " +str(name)+".pdf","wb")
   as pdf:
            for chunk in request.iter_content(chunk_size
   =1024):
                if chunk:
                    pdf.write(chunk)
    except:
        pass
def get_the_file_txt(url : str):
    request = requests.get(url)
    openFile = open(r"back room\gutenberg_unprocess_data.txt"
   ,"a+",encoding="utf-8")
    if request.text != None:
        openFile.write(request.text)
    openFile.close()
def get_a_generate_key():
    txt = ""
    for k in range(14):
        txt = txt + str(random.randint(0,9))
    return txt
```

```
231
232
233
234
235
236
237
238
239
240 #Big Scrapper
241 def massive_scrapper():
242     l1_links = collection_0001()
243     l2_links = collection_0002()
244     l3_links = collection_0003()
245     the_txt_url_list = src_manager(collection_0004(),100)
246     the_pdf_url_list = src_manager(l1_links + l2_links +
    l3_links,100)
247     class DownloadThread(threading.Thread):
248         def __init__(self, work : [str],file_type : str):
249             threading.Thread.__init__(self)
250             self.work = work
251             self.file_type = file_type
252         def run(self):
253             if self.file_type == "pdf":
254                 for task in self.work:
255                     get_the_file_pdf(task,get_a_generate_key
    ())
256             else:
257                 for task in self.work:
258                     get_the_file_txt(task)
259     pdf_bots_list = [DownloadThread(work,"pdf") for work in
    the_pdf_url_list]
260     for bot in pdf_bots_list:
261         bot.start()
262     for bot in pdf_bots_list:
263         bot.join()
264     print("section pdf finie")
265     txt_bots_list = [DownloadThread(work,"texte") for work in
     the_txt_url_list]
266     for bot in txt_bots_list:
267         bot.start()
268     for bot in txt_bots_list:
269         bot.join()
270     print("section texte finie")
271
```

```
272
273
274
275
276
277 #analyseur
278 def pre_path_analyse():
279     words_functions = load_dict()
280     new_dict = {}
281     for ver in words_functions.keys():
282         for word in (words_functions[ver].keys()):
283             new_dict[word] = ver
284     save_dict(new_dict,path="data collections\
    database_fast_dict.obj")
285 def path_analyse(word : str, fast_words_dict : dict):
286     new_word = simplification_cleaner.sequence(word)
287     if new_word in fast_words_dict:
288         return fast_words_dict[new_word]
289     elif (len(new_word) > 2) and new_word[:2] in
    abrevation_dict:
290         cut_word = new_word[2:]
291         return path_analyse(cut_word,fast_words_dict)
292     else:
293         wo , ve = is_a_word_activator.recognize(new_word) ,
    verb_automata.recognize(new_word)
294         if (wo and ve) or ve or (new_word == "a") or ((len(
    new_word) > 2) and (new_word[:1] == "j'")):
295             return "verbe"
296         elif wo:
297             return "nom"
298         else:
299             openFile = open(r"back room\uknow_words.txt","a+"
    )
300             openFile.write(word+"\n")
301             openFile.close()
302             return "unknow"
303     pass
304 def get_list_recompiler(my_list : [str]):
305     new_list = []
306     for path in my_list:
307         if new_list == []:
308             new_list.append(path)
309         elif (new_list[-1] == "nom") and (path == "nom"):
310             pass
311         elif (new_list[-1] == "unknow") and (path == "unknow"
```

```
    ):
312             pass
313         else:
314             new_list.append(path)
315     return new_list
316 def sentence_analyser(sentence : str, fast_words_dict : dict,
    structures_referencer : dict):
317     all_words = asqs.get_words_list_of_a_sentence(sentence,
    words_with_spaces_dict)
318     recompile_tuple = tuple(get_list_recompiler([path_analyse
    (word,fast_words_dict) for word in all_words]))
319     if "unknow" in recompile_tuple:
320         openFile = open(r"back room\unreferenced_data.txt","a
    +")
321         openFile.write(sentence+" ||\n")
322         openFile.close()
323         pass
324     else:
325         if recompile_tuple in structures_referencer:
326             structures_referencer[recompile_tuple] =
    structures_referencer[recompile_tuple] + 1
327         else:
328             structures_referencer[recompile_tuple] =  1
329         openFile = open(r"back room\training_data.txt","a+")
330         openFile.write(simplification_cleaner.sequence(
    sentence)+" ||\n")
331         openFile.close()
332         pass
333 def txt_sentence_analyser(sentence : str, fast_words_dict :
    dict, structures_referencer : dict):
334     all_words = asqs.get_words_list_of_a_sentence(sentence,
    words_with_spaces_dict)
335     recompile_tuple = tuple(get_list_recompiler([path_analyse
    (word,fast_words_dict) for word in all_words]))
336     if "unknow" in recompile_tuple:
337         openFile = open(r"back room\unreferenced_data.txt","a
    +")
338         openFile.write(sentence+" ||\n")
339         openFile.close()
340         pass
341     else:
342         if recompile_tuple in structures_referencer:
343             structures_referencer[recompile_tuple] =
    structures_referencer[recompile_tuple] + 1
344         else:
```

```
345                 structures_referencer[recompile_tuple] =  1
346             openFile = open(r"back room\training_data.txt","a+")
347             openFile.write(txt_recorder_coder.sequence(sentence)+
    " ||\n")
348             openFile.close()
349             pass
350 def all_pdf_analyse():
351     pdf_references = os.listdir(r"back room\pdf")
352     structure_dict = load_dict("data collections\
    database_big_vector.obj")
353     text = ""
354     for reference in pdf_references:
355         try:
356             reader = PdfReader("back room\pdf\ " + reference
    [1:])
357             number_of_pages = len(reader.pages)
358             for i in range(number_of_pages):
359                 page = reader.pages[i]
360                 text = text + structurize_cleaner.sequence(
    page.extract_text())
361         except:
362             pass
363     class Structurer(threading.Thread):
364         def __init__(self, work : [str]):
365             threading.Thread.__init__(self)
366             self.work = work
367         def run(self):
368             for task in self.work:
369                 sentence_analyser(task,fast_dict,
    structure_dict)
370             pass
371     print(text)
372     all_sentences = asqs.get_sentence_list(text)
373     task_list = src_manager(all_sentences,200)
374     bot_list = [Structurer(work) for work in src_manager(
    all_sentences,200)]
375     for bot in bot_list:
376         bot.start()
377     for bot in bot_list:
378         bot.join()
379     print("transfert fini")
380     save_dict(structure_dict,"data collections\
    database_big_vector.obj")
381     pass
382 def all_txt_analyse():
```

```
383     structure_dict = load_dict("data collections\
    database_big_vector.obj")
384     fileIn = open(r"back room\gutenberg_unprocess_data.txt","
    r",encoding="utf-8")
385     content = fileIn.readlines()
386     fileIn.close()
387     def split(a, n):
388         k, m = divmod(len(a), n)
389         return (a[i*k+min(i, m):(i+1)*k+min(i+1, m)] for i in
    range(n))
390     text = ""
391     for line in content:
392         text = text + structurize_cleaner.sequence(line)
393     class Structurer(threading.Thread):
394         def __init__(self, work : [str]):
395             threading.Thread.__init__(self)
396             self.work = work
397         def run(self):
398             for task in self.work:
399                 txt_sentence_analyser(task,fast_dict,
    structure_dict)
400             pass
401     all_sentences = asqs.get_sentence_list(text)
402     bot_list = [Structurer(work) for work in split(
    all_sentences,200)]
403     for bot in bot_list:
404         bot.start()
405     for bot in bot_list:
406         bot.join()
407     print("transfert fini")
408     save_dict(structure_dict,"data collections\
    database_big_vector.obj")
409     pass
410 def rectification_01():
411     anti_n = asqs.EncodeAutomate({"\n" : ""})
412     fileIn = open(r"back room\uknow_words.txt","r")
413     content = fileIn.readlines()
414     fileIn.close()
415     words_dict = {}
416     for word in content:
417         new_word = anti_n.sequence(word)
418         if new_word not in words_dict:
419             words_dict[new_word] = None
420     words_list = list(words_dict.keys())
421     dscp.MultiThreadFragmentationCorrecteur(words_list)
```

```
def reccompile():
    my_dict = load_dict()
    new_dict = {}
    for cat in my_dict:
        for word in my_dict[cat]:
            new_dict[word] = cat
    fast_dict = save_dict(new_dict,"data collections\
   database_fast_dict.obj")
data_decoder = asqs.EncodeAutomate({" ||\n" : "", "\t" : " "
   }).sequence
def get_sentence_of_data(path=r"back room\training_data.txt")
   :
    with open(path,"r") as f:
        text = f.read()
    return text.split(" ||\n")
decode_sentence = asqs.EncodeAutomate({"\t" : " "}).sequence
def rectification_02():
    structure_dict = {}
    class Structurer(threading.Thread):
        def __init__(self, work : [str]):
            threading.Thread.__init__(self)
            self.work = work
        def run(self):
            for task in self.work:
                sentence_analyser(task,fast_dict,
   structure_dict)
            pass
    all_sentences = [data_decoder(a) for a in
   get_sentence_of_data(r"back room\unreferenced_data.txt")]
    task_list = src_manager(all_sentences,200)
    bot_list = [Structurer(work) for work in src_manager(
   all_sentences,200)]
    for bot in bot_list:
        bot.start()
    for bot in bot_list:
        bot.join()
    print("transfert fini")
    return structure_dict
def rectification():
    rectification_01()
    reccompile()
    rectification_02()
def massive_analyse():
    all_pdf_analyse()
    all_txt_analyse()
```

```
461     rectification()
462
463
464 def main_scraper():
465     massive_scrapper()
466     massive_analyse()
467     pass
```

Listing 7 – Python Code du collecteur de donnees

Biaisons les données

Modèle

```
1  import random
2  import pickle
3  import heapq
4  import numpy as np
5  import pandas as pd
6  import matplotlib.pyplot as plt
7  from nltk.tokenize import RegexpTokenizer
8  import tensorflow as tf
9  from tensorflow.keras.models import Sequential, load_model
10 from tensorflow.keras.layers import LSTM, Dense, Activation
11 from tensorflow.keras.optimizers import RMSprop
12 text = open("training_data.txt", 'r').read()
13 partial_text = text[:1000000]
14 tokenizer = RegexpTokenizer(r"\w+")
15 tokens = tokenizer.tokenize(partial_text)
16 unique_tokens = np.unique(tokens)
17 unique_token_index = {token: index for index, token in
       enumerate(unique_tokens)}
18 n_words = 10
19 input_words = []
20 next_word = []
21 for i in range(len(tokens) - n_words):
22     input_words.append(tokens[i:i + n_words])
23     next_word.append(tokens[i + n_words])
24 X = np.zeros((len(input_words), n_words, len(unique_tokens)),
       dtype=bool)
```

```
y = np.zeros((len(next_word), len(unique_tokens)), dtype=bool
    )
for i, words in enumerate(input_words):
    for j, word in enumerate(words):
        X[i, j, unique_token_index[word]] = 1
    y[i, unique_token_index[next_word[i]]] = 1
model = Sequential()
model.add(LSTM(128, input_shape=(n_words, len(unique_tokens))
    , return_sequences=True))
model.add(LSTM(128))
model.add(Dense(len(unique_tokens)))
model.add(Activation("softmax"))
optimizer = RMSprop(learning_rate=0.01)
model.compile(loss="categorical_crossentropy", optimizer=
    optimizer, metrics=["accuracy"])
history = model.fit(X, y, batch_size=256, epochs=10, shuffle=
    True).history
model.save("textgenerator1.h5")
model = tf.keras.models.load_model("textgenerator1.h5")
history = model.fit(X, y, batch_size=256, epochs=10, shuffle=
    True).history
model.save("textgenerator2.h5")
model = tf.keras.models.load_model("textgenerator2.h5")
history = model.fit(X, y, batch_size=256, epochs=40, shuffle=
    True).history
model.save("textgenerator3.h5")
model = tf.keras.models.load_model("textgenerator3.h5")
history = model.fit(X, y, batch_size=256, epochs=40, shuffle=
    True).history
model.save("textgenerator4.h5")
def predict_next_word(input_text, n_best):
    input_text = input_text.lower()
    X = np.zeros((1, n_words, len(unique_tokens)))
    for i, word in enumerate(input_text.split()):
        X[0, i, unique_token_index[word]] = 1

    predictions = model.predict(X)[0]
    return np.argpartition(predictions, -n_best)[-n_best:]
def generate_text(input_text, n_words, creativity=3):
    word_sequence = input_text.split()
    current = 0
    for _ in range(n_words):
        sub_sequence = " ".join(tokenizer.tokenize(" ".join(
    word_sequence).lower())[current:current+n_words])
        try:
```

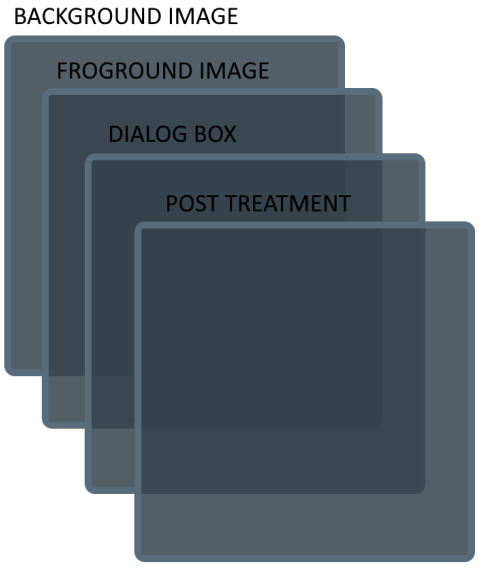
```
62              choice = unique_tokens[random.choice(
    predict_next_word(sub_sequence, creativity))]
63          except:
64              choice = random.choice(unique_tokens)
65          word_sequence.append(choice)
66          current += 1
67      return " ".join(word_sequence)
```

Listing 8 – Reseau Recurrent

Moteur de Jeu

Notre environnement de jeu, n'étant pas bien complexe, notre moteur de jeu ne le sera pas non plus. Pour ce faire, listons nos besoins :

1. Afficher des images en arrière-plan.
2. Afficher des images de personnages.
3. Afficher une boite de dialogue.
4. Possiblement post-traiter notre rendu final.
5. Une fenêtre de saisie de texte.



moteur de jeu

```
"""
pygame model
"""
from pygame.locals import *
import pygame
from PIL import Image
import tensorflow as tf
import random
import numpy as np
import os
import time


from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.layers import Activation, Dense, LSTM
from keras.layers import Dropout
import all_sequensers_models as asqs

```

```
decode_sentence = asqs.EncodeAutomate({"\t" : " ", "\n" : ""
    }).sequence
encode_sentence = asqs.EncodeAutomate({" " : "\t"}).sequence


text = open("test.txt", 'rb').read().decode(encoding='utf-8')
    .lower()
#text = text[]


characters = sorted(set(text))


char_to_index = dict((c,i) for i, c in enumerate(characters))
index_to_char = dict((i,c) for i, c in enumerate(characters))


SEQ_LENGTH = 40
STEP_SIZE = 3
model = tf.keras.models.load_model("textgenerator.model")


def sample(preds, temperature=1.0):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)
    return np.argmax(probas)
def generate_text(length, temperature):
    start_index = random.randint(0, len(text) - SEQ_LENGTH -
    1)
    generated = ""
    sentence = text[start_index: start_index + SEQ_LENGTH]
    generated += sentence
    for i in range(length):
        x_predictions = np.zeros((1, SEQ_LENGTH, len(
    characters)))
        for t, char in enumerate(sentence):
            x_predictions[0, t, char_to_index[char]] = 1

        predictions = model.predict(x_predictions, verbose=0)
    [0]
        next_index = sample(predictions,
                                    temperature)
```

```
        next_character = index_to_char[next_index]

        generated += next_character
        sentence = sentence[1:] + next_character
    return generated


def generate_text_aux(length, temperature , my_sent):
    start_index = random.randint(0, len(text) - SEQ_LENGTH - 1)
    generated = my_sent
    sentence = text[start_index: start_index + SEQ_LENGTH]
    generated += sentence
    for i in range(length):
        x_predictions = np.zeros((1, SEQ_LENGTH, len(characters)))
        for t, char in enumerate(sentence):
            x_predictions[0, t, char_to_index[char]] = 1

        predictions = model.predict(x_predictions, verbose=0)[0]
        next_index = sample(predictions,
                            temperature)
        next_character = index_to_char[next_index]

        generated += next_character
        sentence = sentence[1:] + next_character
    return generated


def dialog_manager(historic : list):
    old_text = encode_sentence("\n".join(historic))
    new_text = (generate_text_aux(random.randint(0,200),0.8, old_text))[len(old_text):]
    return new_text


def src_manager(file_procc : [str], n : int):
    """segmente notre liste en plusieurs listes plus petites """
    if len(file_procc) <= n:
        return [file_procc]
    else:
        return [file_procc[:n]] + src_manager(file_procc[n:], n)
```

```
def load_animation_images_sky():
    images = []
    path = "src\sprite01\my_type01_sprite"
    for num in range(1,17):
        image_path = path + str(num) + ".png"
        img = pygame.image.load(image_path)
        images.append(img)
    return images
animations = {"sky" : load_animation_images_sky()}
class AnimationSky(pygame.sprite.Sprite):
    def __init__(self, rect_x, rect_y):
        super().__init__()
        self.image = pygame.image.load("src\sprite01\
   my_type01_sprite1.png")
        self.rect = self.image.get_rect()
        self.rect.x = rect_x
        self.rect.y = rect_y
        self.current_image = 1
        self.animation = True
        self.images = animations["sky"]
    def animate(self, loop=True):
        if self.animation:
            self.current_image += 1
            if self.current_image >= len(self.images):
                self.current_image =1
                if not loop:
                    self.animation = False
            self.image = self.images[self.current_image]


class GraphicModel():
    def __init__(self ,title : str ,bg_color="dark gray",fuls
   =False ,size=(800,500), tick=6):
        #pre initialize
        pygame.init()
        if fuls:
            self.windows = pygame.display.set_mode(size,
   pygame.FULLSCREEN)
        else:
            self.windows = pygame.display.set_mode(size)


        #Initialize
        self.meta = {"title" : title , "bg-color" : bg_color
   , "tick" : tick}
```

```
141         pygame.display.set_caption(title)
142         self.timer = pygame.time.Clock()
143         self.font = pygame.font.Font("police.ttf",24)
144         self.snip = self.font.render("",True,"white")
145         self.run = True
146         self.last_messages = [generate_text(100,0.8)]
147         self.dialog_mode = False
148         self.images = {"bg" : pygame.image.load("src\img0004.
    png"),
149                        "fg" : pygame.image.load("src\img0007.
    png"),
150                        "db" : pygame.image.load("src\img0003.
    png"),
151                        "char1" : pygame.image.load("src\
    img0001.png"),
152                        "char2" : pygame.image.load("src\
    img0006.png")}
153         self.counter = 0
154         self.done = False
155         self.speed = 1
156         self.user_text = ""
157         self.user_dialog = False
158         self.char = 1
159         self.sky_sprite = pygame.sprite.Group()
160         self.head = 0
161         self.display_dialog = src_manager(self.last_messages
    [-1],54)
162         self.sky_sprite.add(AnimationSky(0,0))
163     def add_a_dialog(self, new_txt : str):
164         self.last_messages.append(new_txt)
165     def open_dialog_box(self):
166         if self.dialog_mode:
167             self.windows.blit(self.images["db"],(0,400))
168             self.windows.blit(self.images["char"+str(self.
    char)],(0,272))
169             message = self.display_dialog[self.head]
170             if self.counter < self.speed * len(message):
171                 self.counter += 1
172             elif self.counter >= self.speed*len(message):
173                 self.done = True
174             self.snip = self.font.render(message[0:self.
    counter//self.speed],True, "white")
175             self.windows.blit(self.snip,(30,440))
176             pass
177         else:
```

```
178             self.counter = 0
179             self.done = False
180         pass
181     def input_text(self,event):
182         if event.type == pygame.KEYDOWN and self.user_dialog:
183             if event.key == pygame.K_BACKSPACE:
184                 self.user_text = self.user_text[:-1]
185             elif event.key == pygame.K_RETURN:
186                 self.last_messages.append(self.user_text)
187                 self.last_messages.append(dialog_manager(self
    .last_messages))
188                 self.user_text = ""
189                 pygame.time.wait(1000)
190                 self.display_dialog = src_manager(self.
    last_messages[-1],54)
191                 self.user_dialog = False
192                 self.dialog_mode = True
193             else:
194                 self.user_text = self.user_text + event.
    unicode
195     def self_open_dialog_box(self):
196         if self.user_dialog:
197             self.windows.blit(self.images["db"],(0,400))
198             self.windows.blit(self.images["char"+str(self.
    char)],(0,272))
199             self.snip = self.font.render(decode_sentence(self
    .user_text),True, "black")
200             self.windows.blit(self.snip,(30,440))
201         pass
202     def display_pre_initialize(self):
203         rect = self.images["bg"].get_rect()
204         self.windows.blit(self.images["bg"],(0,0))
205         for char in self.sky_sprite:
206             if char.animation:
207                 char.animate()
208             else:
209                 self.sky_sprite.remove(char)
210         self.sky_sprite.draw(self.windows)
211         pass
212     def display_initialize(self):
213         self.windows.blit(self.images["fg"],(0,0))
214         self.open_dialog_box()
215         self.self_open_dialog_box()
216         pass
217     def display_post_initialize(self):
```

```
218         pygame.image.save(self.windows,"p_t.png")
219         pass
220     def game_loop(self):
221         while self.run:
222             self.timer.tick(self.meta["tick"])
223             self.display_pre_initialize()
224             self.display_initialize()
225             self.display_post_initialize()
226             pygame.display.flip()
227             pygame.display.update()
228             for event in pygame.event.get():
229                 if event.type == pygame.QUIT:
230                     self.run = False
231                 if event.type == pygame.KEYDOWN:
232                     if event.key == pygame.K_DOWN:
233                         if self.dialog_mode == False:
234                             self.dialog_mode = True
235                             self.user_dialog = False
236                         else:
237                             self.dialog_mode = False
238                             self.user_dialog = False
239                     if event.key == pygame.K_UP:
240                         if self.user_dialog == False:
241                             self.user_dialog = True
242                             self.dialog_mode = False
243                         else:
244                             self.user_dialog = False
245                             self.dialog_mode = False
246                     if event.key == pygame.K_a:
247                         if self.head == len(self.
    display_dialog) - 1:
248                             self.head = 0
249                         else:
250                             self.head = self.head + 1
251                             self.counter = 0
252                             self.done = False
253                     if (event.key == pygame.K_LEFT) or (event
    .key == pygame.K_RIGHT):
254                         if self.char == 1:
255                             self.char = 2
256                         else:
257                             self.char = 1
258                         if self.user_dialog or self.
    dialog_mode:
259                             self.user_dialog = True
```

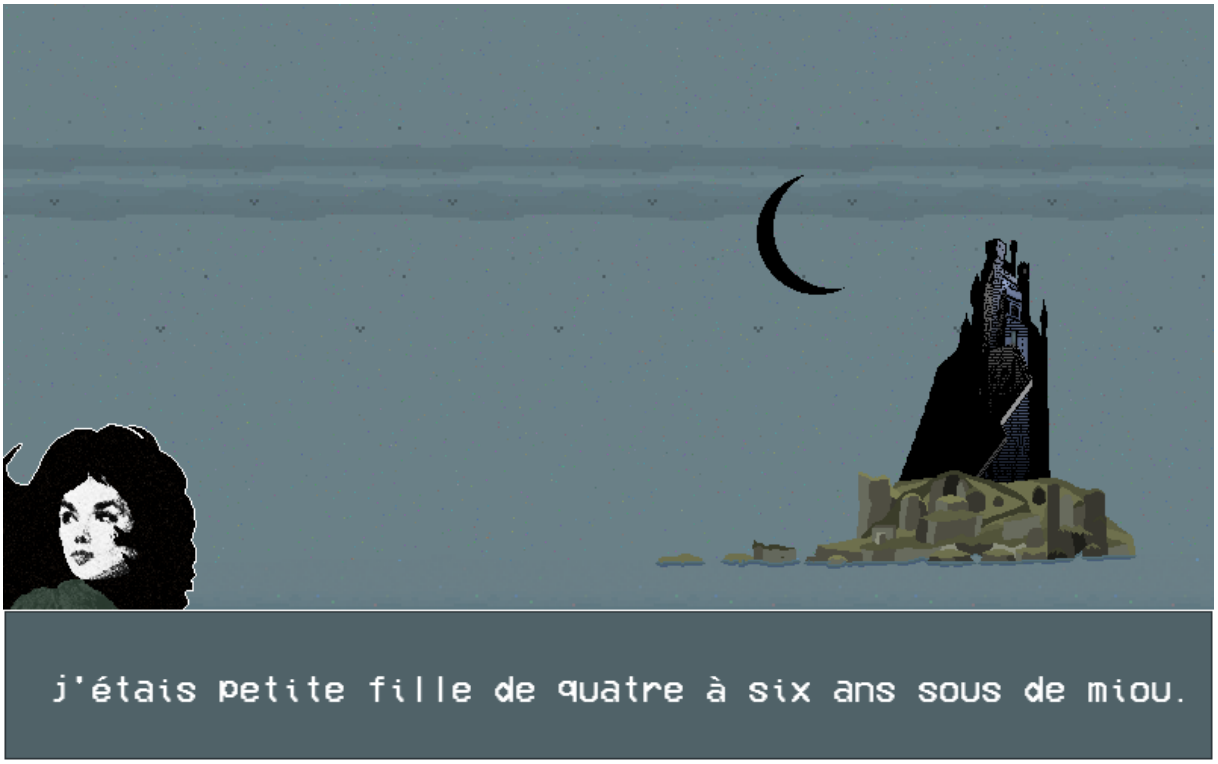
```
260                                 self.dialog_mode = False
261                         if event.key == pygame.K_TAB:
262                             pygame.image.save(self.windows,"
    screenshot.png")
263                     self.input_text(event)
264             pygame.quit()
265             pass
266 
267 
268 my_game = GraphicModel("tipe",)
269 my_game.game_loop()
```

Listing 9 – Python Moteur de Jeu Implementation

Pour commencer affichons notre arriere plan.

Implementons un systeme d'affichage de texte (pour les dialogues).

Implementons un systeme de saisie de texte.



Générateur de texte

Chaine de markov

imlémentation

```
import markovify
import json
import os
fileIn = open(r"training_data.txt","r")
content = fileIn.readlines()
fileIn.close()
training_data = "./n".join([a[:-4] for a in content])
text_model = markovify.Text(training_data)
print(text_model.make_sentence(tries=1000))
```

Listing 10 – Chaine de Markov Implementation

performances

Réseau LSTM

imlémentation

```
import tensorflow as tf
import random
import numpy as np
import os
import time


from tensorflow.keras.models import Sequential
from tensorflow.keras.optimizers import RMSprop
from tensorflow.keras.layers import Activation, Dense, LSTM
from keras.layers import Dropout



strategy = tf.distribute.MirroredStrategy()
print('Number of devices: {}'.format(strategy.
    num_replicas_in_sync))
text = (open("training_data.txt").read())
text = text[:1000000]
characters = sorted(list(set(text)))


char_to_index = dict((c,i) for i, c in enumerate(characters))
index_to_char = dict((i,c) for i, c in enumerate(characters))


SEQ_LENGTH = 40
STEP_SIZE = 3
sentences = []
next_characters = []
for i in range(0, len(text) - SEQ_LENGTH , STEP_SIZE):
    sentences.append(text[i : i + SEQ_LENGTH])
    next_characters.append(text[i + SEQ_LENGTH])
x = np.zeros((len(sentences), SEQ_LENGTH,
              len(characters)), dtype=bool)
y = np.zeros((len(sentences),
              len(characters)), dtype=bool)
for i, satz in enumerate(sentences):
```

```
    for t, char in enumerate(satz):
        x[i, t, char_to_index[char]] = 1
    y[i, char_to_index[next_characters[i]]] = 1
model = Sequential()
model.add(LSTM(128, input_shape=(SEQ_LENGTH, len(characters))
    ))
model.add(Dropout(0.2))
model.add(Activation("softmax"))
model.add(Dropout(0.2))
model.add(Dense(len(characters)))
model.compile(loss= "categorical_crossentropy" , optimizer=
    RMSprop(learning_rate=0.01))
model.fit(x,y,batch_size=256,epochs=4)
model.save("textgenerator1.model")
model = tf.keras.models.load_model("textgenerator1.model")
model.fit(x,y,batch_size=256,epochs=10)
model.save("textgenerator1.model")
model = tf.keras.models.load_model("textgenerator1.model")
model.fit(x,y,batch_size=256,epochs=40)
model.save("textgenerator2.model")
model = tf.keras.models.load_model("textgenerator2.model")
model.fit(x,y,batch_size=256,epochs=20)
model.save("textgenerator3.model")
model = tf.keras.models.load_model("textgenerator0.model")
def sample(preds, temperature=1.0):
    preds = np.asarray(preds).astype('float64')
    preds = np.log(preds) / temperature
    exp_preds = np.exp(preds)
    preds = exp_preds / np.sum(exp_preds)
    probas = np.random.multinomial(1, preds, 1)
    return np.argmax(probas)
def generate_text(length, temperature):
    start_index = random.randint(0, len(text) - SEQ_LENGTH -
    1)
    generated = ""
    sentence = text[start_index: start_index + SEQ_LENGTH]
    generated += sentence
    for i in range(length):
        x_predictions = np.zeros((1, SEQ_LENGTH, len(
    characters)))
        for t, char in enumerate(sentence):
            x_predictions[0, t, char_to_index[char]] = 1

        predictions = model.predict(x_predictions, verbose=0)
    [0]
```

```
78          next_index = sample(predictions,
79                                  temperature)
80          next_character = index_to_char[next_index]
81
82          generated += next_character
83          sentence = sentence[1:] + next_character
84      return generated
85  def generate_text_aux(length, temperature , my_sent):
86      start_index = random.randint(0, len(text) - SEQ_LENGTH -
    1)
87      generated = my_sent
88      sentence = text[start_index: start_index + SEQ_LENGTH]
89      generated += sentence
90      for i in range(length):
91          x_predictions = np.zeros((1, SEQ_LENGTH, len(
    characters)))
92          for t, char in enumerate(sentence):
93              x_predictions[0, t, char_to_index[char]] = 1
94
95          predictions = model.predict(x_predictions, verbose=0)
    [0]
96          next_index = sample(predictions,
97                                  temperature)
98          next_character = index_to_char[next_index]
99
100         generated += next_character
101         sentence = sentence[1:] + next_character
102     return generated
```

Listing 11 – RÃľseau RÃľccurent Implementation

performances

Conclusions

Bibliographie Commentée

Depuis la parution de Dungeon et dragon (DND) en 1974, nombreuses ont étés les tentatives de portages sur ordinateur, du célébrissime jeu de rôle sur table.

Ces bien heureuses tentatives, de par leur contraintes techniques, ont donner naissance aux mécaniques de RPG, très largement rependues dans l'industrie du jeu vidéo. Cependant, après tant d'années, il nous faut faire l'amère constat, de l'échec partiel de ces portages, car les grandes qualités du jeu de rôle, sont à l'origine de ces semi-réussites. Pour cause, le « CRPG » (pour « computer role playing game »), perd en flexibilité, dans ses tentatives de réduire le jeu de rôle a son aspect statistique.

C'est là : une contrainte que le jeu de rôle, partage avec le traitement naturel du langage. Car les faiblesses du jeu de rôle, sont également ses plus grands atouts :

-Son côté humain, collaboratif et communautaire, rend son organisation pénible et bien des fois hasardeuse.

-Les participants n'ont bien souvent, pas les mêmes disponibilités, voir volontés de s'y investir.

-Le jeu de certains joueurs, restreint bien souvent celui des autres.

-Ou encore des erreurs humaines, peuvent être commises lors de l'application des règles du jeu.

C'est pour pallier ces problèmes, tout en conservant leur côté avantageux, qu'il nous est venu à l'esprit de coupler un système de jeu (aussi rudimentaire soit-il) a un « LLM ».

Le soudain essor des techniques de « deep learning », est aller de pair avec un développement accru, du traitement du langage naturel (NLP).[5] Longtemps considérer comme une des limites, du « machine learning » ; en raison des difficultés a vectoriser : mots et autres morceaux de phrases ([4] la

tokenisation), le NLP a néanmoins connu de récentes avancées majeurs dans le domaine de la complétion de texte.[4]

Générer du texte en appliquant de la statistique a la linguistique, est une méthode conçue en 1948 par Claude Shannon.[6] Bien que, le model (basé sur des principes de probabilités Markoviennes [6]), produise la plupart du temps des phrases correcte au niveau de la syntaxe, cela se fait au détriment de la sémantique et du sens implicite de celles-ci. En 2017 est publier un article scientifique, nommé : « Attention Is All You Need » [2], qui a l'aide de « transformers », parvient à générer des phrases dont la sémantique est correcte, tout en comprenant le contexte des phrases précédentes (en simulant le mécanisme d'oubli du cerveau humain). C'est sur cette base nouvelle, que seront battis les récents LLM tel que : « GPT-2, GPT-3, GPT-4, Bloom, Bert, e.t.c. » « NovelAI » et « AiDungeon », pour citer les exemples les plus connus d'utilisations de LLM dans le jeu de rôle, sont basés sur ces derniers.

1. https://proceedings.neurips.cc/paper_files/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf
2. https://datascientest.com/introduction-au-nlp-natural-language-processing
3. https://larevueia.fr/introduction-aux-reseaux-de-neurones-transformers/
4. https://larevueia.fr/introduction-au-nlp-avec-python-les-ia-prennent-la-parole/
5. https://fr.wikipedia.org/wiki/Traitement_automatique_du_langage_naturel
6. https://www.wikidata.fr-fr.nina.az/...