Natural Language Processing, ou NLP

Le Natural Language Processing, ou NLP (traitement automatique du langage en français, ou TAL) connaît depuis récemment un fort regain d'intérêt dans la communauté scientifique. Cet engouement est directement lié aux progrès réalisés depuis quelques années dans le domaine de l'intelligence artificielle (machine learning et deep learning plus particulièrement). L'avènement des assistants vocaux nous laisse penser que le problème de la reconnaissance du langage humain par des machines est désormais quasiment résolu, mais où en est-on exactement ?

Histoire du NLP

Le NLP est une discipline qui étudie la compréhension, la manipulation et la génération du langage naturel par les machines, que ce langage soit écrit ou parlé. Il allie les connaissances issues de la linguistique, de l'informatique et de l'intelligence artificielle.

Des années 50 aux années 80: les premiers travaux

Les premières applications étaient principalement basées sur la **traduction automatique**.

Un ordinateur d'IBM dans les années 50

Bien que les premières expériences remontent aux années 30, c'est l'expérience Georgetown-IBM en 1954 qui lance véritablement la période d'euphorie sur le sujet: IBM et l'Université de Georgetown s'associent pour créer **le premier programme informatique capable de traduire une soixantaine de phrases du russe vers l'anglais**. La machine était surnommée « le cerveau ».

Dans les années 60 cependant, les progrès n'ayant pas été assez rapides pour générer un engouement durable, **les travaux sur le sujet commencent à être nettement moins nombreux**, jusqu'au rapport ALPAC de 1966, très critique sur les efforts en cours et qui constate que les buts n'ont pas été atteints. Il suggère de favoriser la recherche fondamentale plutôt que les applications.

Malgré la désillusion et les subventions qui s'arrêtent aux Etats-Unis, les recherches continuent. Dans les années 60 et 70, parallèlement aux recherches en traduction automatique, on voit l'apparition des **premiers chatbots de l'histoire**, avec par exemple ELIZA en 1964, qui était capable de simuler une psychothérapie (reformulation des phrases du « patient » et questions contextuelles) avec seulement 5 pages de langage informatique.

A partir des années 80 : une approche statistique du NLP

Jusqu'alors il s'agissait d'une **approche symbolique** de la discipline (développée sous l'impulsion de Noam Chomsky à la fin des années 60), c'est à dire que le savoir linguistique était codé sous forme de *grammaires* et de *bases de données lexicales* développées manuellement.

C'est à partir des années 80 que les premières **approches statistiques** voient le jour. Mais pour rendre la modélisation mathématique possible, il faut pouvoir transformer le texte en input numérique. Les premières méthodes consistaient alors à représenter le texte sous forme de vecteurs, en comptant les occurrences des mots ou groupes de mots par exemple.

A la fin des années 80, l'augmentation des capacités de traitement informatique et l'introduction **des algorithmes de machine learning** dans le traitement du langage donnent un nouveau souffle au NLP.

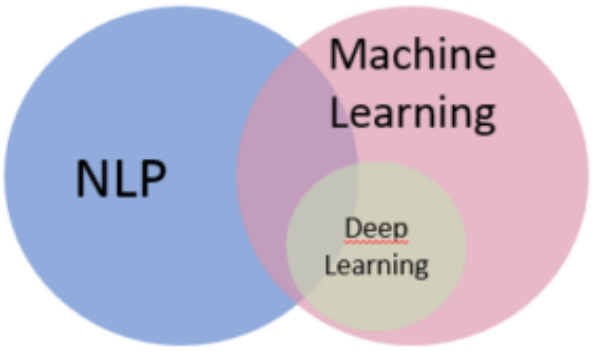
Avec cette approche qui vient compléter la précédente, **la machine est en mesure de créer ses propres règles**, déterminées par apprentissage à partir de textes.

Années 90 : la révolution du deep learning

> *« La technologie du deep learning apprend à représenter le monde. C'est-à-dire comment la machine va représenter la parole ou l'image par exemple » – Yann LeCun (Le Monde, 24/07/2015)*

C'est dans les années 90 qu'a lieu une avancée décisive avec les travaux de **Yann LeCun** sur l'apprentissage profond au sein des laboratoires Bells: la mise au point du premier système basé sur les réseaux de neurones convolutifs permettant de lire les chèques bancaires. S'ensuivra pourtant une période « noire » pour le domaine, une dizaine d'années avec peu d'avancées significatives.

A partir de 2013, des réseaux de neurones comme Word2Vec, Glove puis Vanilla RNNs pour ne citer qu'eux, sont introduits dans la discipline, mais les chercheurs ne sont pas encore satisfaits et commencent à s'intéresser de plus près à ces réseaux de neurones convolutifs (initialement développés pour la reconnaissance d'images) aux résultats époustouflants.



Actuellement, grâce à la puissance de calcul exponentielle des ordinateurs, la disponibilité massive des données en open source de plus en plus importante et le perfectionnement continu des algorithmes de machine learning, le NLP est donc de nouveau en plein essor.

Comment ça marche ?

Les concepts de base

Qu'il s'agisse de traduction automatique ou d'une discussion avec un chatbot, les méthodes de NLP prêtent attention aux hiérarchies afin de mettre en cohérence les mots entre eux

Quelque soit la méthode utilisée, les grandes étapes préalables restent souvent les mêmes, à savoir, de manière simplifiée:

1. **L'analyse lexicale** permet d'extraire les éléments du texte, mots ou groupes de mots, et de les « étiqueter » selon leur catégorie grammaticale (article, nom, verbe, adjectif, etc…). On parle de tokenisation, racinisation et lemmatisation en particulier
2. **L'analyse syntaxique** (ou parsing): il s'agit de comprendre la structure des phrases. Cette étape se base sur un dictionnaire (le vocabulaire) et sur un ensemble de règles syntaxiques (la grammaire) et permet d'identifier les relations syntaxiques entre les mots (sous forme d'*arbres syntaxiques*)
3. **L'analyse sémantique** quant à elle s'attache à la signification même des phrases. Cette étape s'avère souvent très compliquée en raison de l'ambiguïté même de notre langage: par exemple un même mot peut avoir plusieurs sens possibles
4. **L'analyse pragmatique** vient compléter l'analyse sémantique en analysant les mots et phrases proches les uns des autres

Les outils

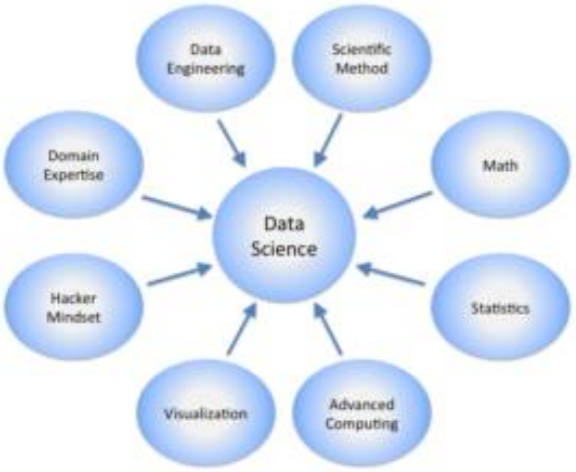
Python est un des langages couramment utilisés pour faire du Traitement Automatique des Langues, et il est très à la mode en ce moment.

Sa particularité est d'être open source et de bénéficier d'une communauté active d'utilisateurs qui collaborent et partagent leurs solutions pour résoudre des problèmes communs.

Ainsi, des « librairies » contiennent différents algorithmes directement utilisables par les data scientists.

D'autres solutions proposent également des algorithmes de NLP, comme R ou SAS par exemple.

Des experts très recherchés



Les compétences d'un data scientist

Pour manipuler ces différents concepts et les outils informatiques dédiés, sans parler des énormes quantités de données, de plus en plus non structurées, les data scientists, data miners et autres ingénieurs Big Data sont particulièrement demandés actuellement sur le marché du travail, et d'autant plus s'ils ont des connaissances dans le domaine du Natural Language Processing.

Les GAFAM ne sont pas en reste et rachètent à coup de millions toutes les startups pour lesquels travaillent les experts en intelligence artificielle ou linguistique (l'exemple le plus criant est le rachat par Google de l'entreprise britannique DeepMind en 2014 pour 628 millions de dollars).

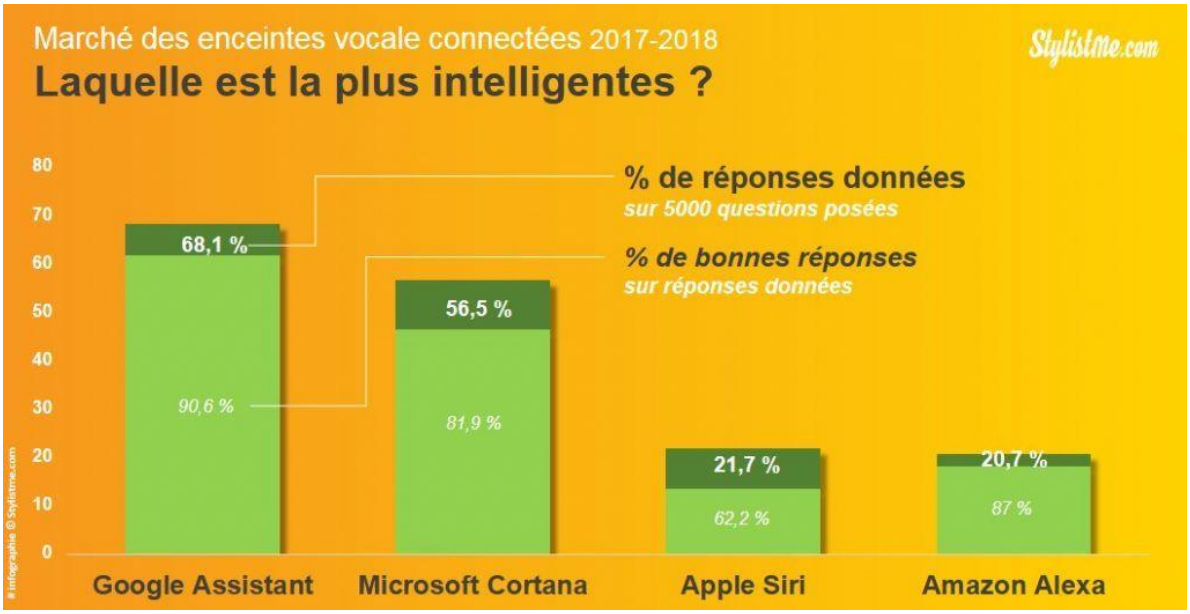
Selon le site internet freelance Upwork, le Traitement Automatique du Langage Naturel se classe premier dans la liste des compétences aux croissances les plus rapides sur le marché du travail mondial.

Quelle sont les applications du NLP ?

Les assistants vocaux

Ces assistants personnels sont des interfaces entre un utilisateur et des fournisseurs de contenus ou de services et sont disponibles sur différents supports: smartphone (système d'exploitation / application), ordinateur et depuis peu **enceinte intelligente**.

Les progrès croissants en Traitement Automatique du Langage de ces dernières années ont vu l'explosion de l'offre de ces assistants et plus particulièrement des enceintes connectées. Bien qu'elles ne tiennent pas encore toutes leurs promesses, les taux d'adoption sont extrêmement élevés (plus élevés que pour les mobiles et les tablettes) et *Gartner* prévoit qu'en 2020 75% des foyers américains en seront équipés.



Test de compréhension des enceintes intelligentes – Crédit: stylistme.com

D'autres applications concrètes

- Les chatbots
- La traduction automatique
- Le résumé automatique de textes
- L'analyse de discours
- L'algorithme RankBrain de Google
- Le filtrage des spams dans nos boîtes mail
- La détection d'évènements
- La traduction des vidéos
- L'analyse de sentiments

TAL en Français

Tokenisation

La tokenisation cherche à transformer un texte en une série de tokens individuels. Dans l'idée, chaque token représente un mot, et identifier des mots semble être une tâche relativement simple. Mais comment gérer en français des exemples tels que: « J'ai froid ». Il faut que le modèle de tokenisation sépare le « J' » comme étant un premier mot.

SpaCy offre une fonctionnalité de tokenisation en utilisant la fonction nlp. Cette fonction est le point d'entrée vers toutes les fonctionnalités de SpaCy. Il sert à représenter le texte sous une forme interprétable par la librairie.

```
def return_token(sentence):
    # Tokeniser la phrase
    doc = nlp(sentence)
    # Retourner le texte de chaque token
    return [X.text for X in doc]
```

Ainsi, en appliquant cette tokenisation à notre phrase, on obtient:

```
return_token(test)
['Bouygues', 'a', 'eu', 'une', 'coupure', 'de', 'réseau', 'à', 'Marseille']
```

Enlever les mots les plus fréquents

Certains mots se retrouvent très fréquemment dans la langue française. En anglais, on les appelle les « stop words ». Ces mots, bien souvent, n'apportent pas d'information dans les tâches suivantes. Lorsque l'on effectue par exemple une classification par la méthode Tf-IdF, on souhaite limiter la quantité de mots dans les données d'entraînement.

Les « stop words » sont établis comme des listes de mots. Ces listes sont généralement disponibles dans une librairie appelée NLTK (Natural Language Tool Kit), et dans beaucoup de langues différentes. On accède aux listes en français de cette manière:

```
from nltk.corpus import stopwords
stopWords = set(stopwords.words('french'))
{'ai',
 'aie',
```

```
 'aient',
 'aies',
 'ait',
 'as',
 'au',
 'aura',
 'aurai',
 'auraient',
 'aurais',
 ...
```

Pour filtrer le contenu de la phrase, on enlève tous les mots présents dans cette liste:

```
clean_words = []
for token in return_token(test):
    if token not in stopWords:
        clean_words.append(token)

clean_words
['Bouygues', 'a', 'coupure', 'réseau', 'Marseille', '.']
```

Tokenisation par phrases

On peut également appliquer une tokenisation par phrase afin d'identifier les différentes phrases d'un texte. Cette étape peut à nouveau sembler facile, puisque a priori, il suffit de couper chaque phrase lorsqu'un point est rencontré (ou un point d'exclamation ou d'interrogation).

Mais que se passerait-t-il dans ce cas-là?

`Bouygues a eu une coupure de réseau à Marseille. La panne a affecté 300.000 utilisateurs.`

Il faut donc une compréhension du contexte afin d'effectuer une bonne tokenisation par phrase.

```
def return_token_sent(sentence):
    # Tokeniser la phrase
    doc = nlp(sentence)
```

```
    # Retourner le texte de chaque phrase
    return [X.text for X in doc.sents]
```

On applique alors la tokenisation à la phrase mentionnée précédemment.

```
return_token_sent("Bouygues a eu une coupure de réseau à Marseille. La
panne a affecté 300.000 utilisateurs.")
```

La tokenisation des phrases retourne alors :

```
['Bouygues a eu une coupure de réseau à Marseille.',
 'La panne a affecté 300.000 utilisateurs.']
```

Stemming

Le stemming consiste à réduire un mot dans sa forme « racine ». Le but du stemming est de regrouper de nombreuses variantes d'un mot comme un seul et même mot. Par exemple, une fois que l'on applique un stemming sur « Chiens » ou « Chien », le mot résultant est le même. Cela permet notamment de réduire la taille du vocabulaire dans les approches de type sac de mots ou Tf-IdF.

Un des stemmers les plus connus est le Snowball Stemmer. Ce stemmer est disponible en français.

```
from nltk.stem.snowball import SnowballStemmer
stemmer = SnowballStemmer(language='french')

def return_stem(sentence):
    doc = nlp(sentence)
    return [stemmer.stem(X.text) for X in doc]
```

Si on applique ce stemmer à notre phrase d'exemple :

```
return_stem(test)
['bouygu', 'a', 'eu', 'une', 'coupur', 'de', 'réseau', 'à', 'marseil', '.']
```

Reconnaissance d'entités nommées (NER)

En traitement automatique du langage, la reconnaissance d'entités nommées cherche à détecter les entités telles que des personnes, des entreprises ou des lieux dans un texte. Cela s'effectue très facilement avec SpaCy.

```
def return_NER(sentence):
    # Tokeniser la phrase
    doc = nlp(sentence)
    # Retourner le texte et le label pour chaque entité
    return [(X.text, X.label_) for X in doc.ents]
```

Puis on peut appliquer la fonction à une phrase d'exemple.

```
return_NER(test)
```

Les entités identifiées sont les suivantes :

```
[('Bouygues', 'ORG'), ('Marseille', 'LOC')]
```

Bouygues est reconnue comme une organisation, et Marseille comme un lieu.

Spacy offre des « Visualizers », des outils graphiques qui permettent d'afficher les résultats de reconnaissances d'entités nommées ou d'étiquetage par exemple.

```
from spacy import displacy

doc = nlp(test)
displacy.render(doc, style="ent", jupyter=True)
```

On peut également préciser des options à DisplaCy pour n'afficher que les organisations, d'une certaine couleur:

```
doc = nlp(test)
colors = {"ORG": "linear-gradient(90deg, #aa9cfc, #fc9ce7)"}
options = {"ents": ["ORG"], "colors": colors}

displacy.render(doc, style="ent", jupyter=True, options=options)
```

L’étiquetage morpho-syntaxique

L’étiquetage morpho-syntaxique ou Part-of-Speech (POS) Tagging en anglais essaye d’attribuer une étiquette à chaque mot d’une phrase mentionnant la fonctionnalité grammaticale d’un mot (Nom propre, adjectif, déterminant…).

```
def return_POS(sentence):
    # Tokeniser la phrase
    doc = nlp(sentence)
    # Retourner les étiquettes de chaque token
    return [(X, X.pos_) for X in doc]
```

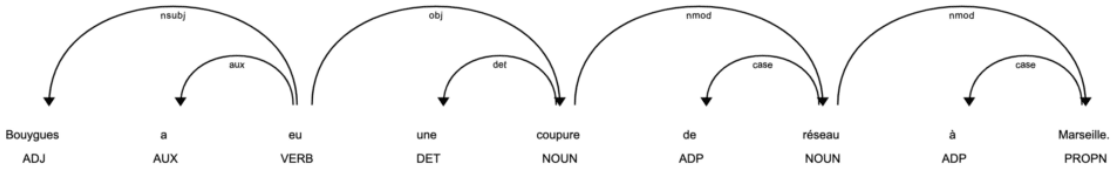
Les étiquettes identifiées sont les suivants :

```
return_POS(test)
[(Bouygues, 'ADJ'),
 (a, 'AUX'),
 (eu, 'VERB'),
 (une, 'DET'),
 (coupure, 'NOUN'),
 (de, 'ADP'),
 (réseau, 'NOUN'),
 (à, 'ADP'),
 (Marseille, 'PROPN')]
```

On remarque que Bouygues n’est pas identifié comme un nom propre, mais comme un Adjectif. Pour le reste, les tags sont corrects.

Spacy dispose également d’une option de visualisation qui nous permet simplement d’afficher les étiquettes identifiées ainsi que les dépendances entre ces étiquettes.

```
doc = nlp(test)
displacy.serve(doc, style="dep")
```



Embedding par mot

Avec SpaCy, on peut facilement récupérer le vecteur correspondant à chaque mot une fois passé dans le modèle pré-entraîné en français.

Cela nous sert à représenter chaque mot comme étant un vecteur de taille 96.

```
import numpy as np

def return_word_embedding(sentence):
    # Tokeniser la phrase
    doc = nlp(sentence)
    # Retourner le vecteur lié à chaque token
    return [(X.vector) for X in doc]
```

Ainsi, lorsqu'appliqué à la phrase test, on obtient :

```
return_word_embedding(test)
[array([ -1.8685186 ,   1.3645297 ,  -2.3505871 ,  -1.233012  ,
         -3.702136  ,   1.3316352 ,  -1.3532144 ,  -3.879726  ,
         -7.051861  ,  -2.8570302 ,  -2.409908  ,   3.3500502 ,
          3.8512042 ,  -0.5462021 ,  -1.7187259 ,  -5.341373  ,
          ...
```

Cette information nous sert notamment lorsque l'on cherche à caractériser la similarité entre deux mots ou deux phrases.

Similarité entre phrases

Afin de déterminer la similarité entre deux phrases, nous allons opter pour une méthode très simple :

- déterminer l'embedding moyen d'une phrase en moyennant l'embedding de tous les mots de la phrase
- calculer la distance entre deux phrases par simple distance euclidienne

Cela s'effectue très facilement avec SpaCy !

```
def return_mean_embedding(sentence):
    # Tokeniser la phrase
    doc = nlp(sentence)
    # Retourner la moyenne des vecteurs pour chaque phrase
    return np.mean([(X.vector) for X in doc], axis=0)
```

On peut alors tester notre fonction avec plusieurs phrases :

```
test_2 = "Le réseau sera bientot rétabli à Marseille"
test_3 = "La panne réseau affecte plusieurs utilisateurs de l'opérateur"
test_4 = "Il fait 18 degrés ici"
```

Ici, on s'attend à ce que la phrase 2 soit la plus proche de la phrase test, puis la phrase 3 et 4.

Avec Numpy, la distance euclidienne se calcule simplement :

```
np.linalg.norm(return_tensor(test)-return_tensor(test_2))
np.linalg.norm(return_tensor(test)-return_tensor(test_3))
np.linalg.norm(return_tensor(test)-return_tensor(test_4))
```

On obtient alors les résultats suivants :

```
16.104986
17.035103
22.039303
```

La phrase 2 est bien identifiée comme la plus proche, puis la phrase 3 et 4. On peut également utiliser cette approche pour classifier si une phrase appartient à une classe ou à une autre.

Transformers

Si vous avez suivi l'actualité du traitement automatique du langage (TAL) ces derniers mois, vous avez surement entendu parlé des Transformers, des modèles état-de-l'art qui apprennent des représentations vectorielles à partir d'un texte d'entrée et qui dépassent les capacités humaines sur certains points. Ces modèles peuvent être pré-entraînés et sont mis à disposition par Hugging Face, une startup spécialisée dans le traitement de langage naturel.

Les modèles les plus récents mis à disposition par Hugging Face spécifiques au français sont entraînés avec XLM, une architecture qui a depuis été battue par de nombreux autres modèles, dont BERT.

BERT est également disponible dans une version multi-linguage, entraîné sur le Wikipedia de plus de 104 langues, sous le nom de : bert-base-multilingual-cased.

Les transformers sont particulièrement bons pour:

- la génération de texte
- apprendre une représentation vectorielle
- prédire si une phrase est la suite d'une autre
- les tâches de questions/réponses

Nous allons ici donner un exemple de comment utiliser les transformers pour prédire si une phrase est la suite d'une autre. On trouve des applications de ceci lorsque l'on cherche par exemple à segmenter des fils de discussions en plusieurs blocs distincts.

On suppose ici que PyTorch et Transformers sont installés (`pip install transformers`).

```
import torch
from transformers import *
```

Ensuite, on doit charger un tokenizer et un modèle.

```
# Tokeniser, Model
tokenizer = BertTokenizer.from_pretrained('bert-base-multilingual-cased')
model = BertForNextSentencePrediction.from_pretrained('bert-base-
multilingual-cased')
model.eval()
```

La ligne `model.eval()` permet de passer le modèle en mode évaluation.

Par la suite, on passe au modèle une séquence de texte convertie en tokens, et des délimitations des phrases:

```
text = "Comment ça va ? Bien merci, un peu stressé avant l'examen"
# Texte tokenisé
tokenized_text = tokenizer.tokenize(text)
# Convertir le texte en indexs
indexed_tokens = tokenizer.convert_tokens_to_ids(tokenized_text)
segments_ids = [0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
```

On transforme le tout en tenseurs interprétables par le modèle :

```
# Transformer en tenseurs
tokens_tensor = torch.tensor([indexed_tokens])
segments_tensors = torch.tensor([segments_ids])
```

Finalement, on prédit si la deuxième phrase est la suite de la première :

```
predictions = model(tokens_tensor, segments_tensors)
```

Si l'élément en position 0 est plus grand que celui en position 1, la phrase est la suite de la première. Autrement, c'est une nouvelle discussion qui commence.

```
if np.argmax(predictions) == 0:
    print("Suite")
else:
    print("Pas la suite")
```

Conclusion

Finalement, même si on constate que les progrès en traitement automatique du langage ont été variés et nombreux depuis la deuxième moitié du XXème siècle, et qu'on utilise désormais certaines applications concrètes de ces travaux tous les jours sans même nous en rendre compte, nous ne sommes pas encore arrivés à percer le mystère de notre langage « naturel ». Toutefois, l'utilisation plus récente des réseaux de neurones et les espoirs qu'ils suscitent nous laissent entrevoir un futur peut être pas si lointain où les machines seront enfin capables de nous comprendre parfaitement.

Sources:

- https://fr.wikipedia.org/wiki/Traitement_automatique_du_langage_naturel
- https://fr.ryte.com/wiki/Natural_language_processing
- https://www.cairn.info/revue-langages-2008-3-page-95.htm
- https://jolicode.com/blog/natural-language-processing-avec-des-petits-morceaux-de-nodejs-dedans
- https://www.1and1.fr/digitalguide/web-marketing/vendre-sur-internet/le-traitement-automatique-du-langage-naturel-taln/
- https://stylistme.com/marche-des-enceintes-intelligentes-vocales-google-home/
- https://mbamci.com/experience-client-ia-revolution/
- https://mbamci.com/alpha-zero-ai-autodidacte/