François Labastie

Projet chef-d'oeuvre E1

Analyse de documents textuels grâce à l'intelligence artificielle

Topic modeling - Latent Dirichlet Allocation

Décembre 2020

Titre professionnel "Développeur en intelligence artificielle"
de niveau 6 enregistré au RNCP sous le n°34757
Passage par la voie de la formation -
parcours de 19 mois achevé le 20 octobre 2020

Introduction

Ce projet a pour objectif de développer une application permettant l'analyse automatique d'articles de presse traitant du Covid-19 dans la version numérique du média Le Monde[1]. Les documents journalistiques sont publiés par sections ou rubriques selon leurs thèmes. Et selon l'évolution de l'actualité, un même sujet peut être publié dans des rubriques différentes. A partir de janvier 2020, les premiers articles sur le Covid-19 ont généralement été diffusés dans les rubriques "international" et "sciences" - pour la majorité des médias - pour ensuite occuper une place importante dans la section "économie", puis "politique" au fur et à mesure de l'évolution de la pandémie. Aussi, des changements d'expressions ou de termes sont observés: Par exemple, en fin d'année 2020, certains articles traitant du "reconfinement" ne mentionnent parfois même plus le terme "Covid".

A partir d'un corpus de plus de 10500 articles du journal Le Monde, ce projet propose d'analyser de manière automatique ces données, en extrayant les thèmes - topics - possibles. Les articles sont caractérisés par leurs titres, section de publication, dates, etc. Mais l'analyse que nous proposons s'intéresse en particulier au contenu pour en extraire les topics afin de remonter les idées principales et étudier leurs fréquences. Nous emploierons ainsi une technique d'intelligence artificielle telle que le "topic modeling" et pourrons également comparer certains éléments statistiques de l'ensemble global des documents.

Les données du projet ont été recueillies de manière illégale, par la technique du scraping. Le Monde n'a pas donné suite à nos requêtes pour ce projet d'étude, ni même d'autres médias tels que l'AFP (qui propose pourtant une API gratuite pour trois mois de tests). Cette démarche illégale n'est donc pas à divulguer mais nous l'assumons dans une démarche non commerciale, comme compétence en data sciences.

[1] https://www.lemonde.fr/

1. Contexte du projet et méthodologie

1.1 Recueil des besoins

L'application s'adresse à toutes personnes intéressées par l'évolution des informations: journalistes, documentalistes, étudiants ou simples lecteurs. Elle permettra d'explorer un ensemble important de données textuelles de manière automatisée, et suggérer des thèmes ou associations de concepts non apparents par une seule approche humaine. Elle donnera également des informations statistiques sur le corpus employé et proposera l'accès en lecture aux contenus utilisés. L'utilisateur pourra ainsi visualiser la liste des topics suggérés par le système et sélectionner une partie du corpus sur lequel effectuer l'analyse. Il pourra également accéder aux documents et faire apparaître des statistiques calculées sur ceux-ci.

Une modélisation de topics réalisée par la revue universitaire américaine *Signs*[2], pourra inspirer notre projet : http://signsat40.signsjournal.org/

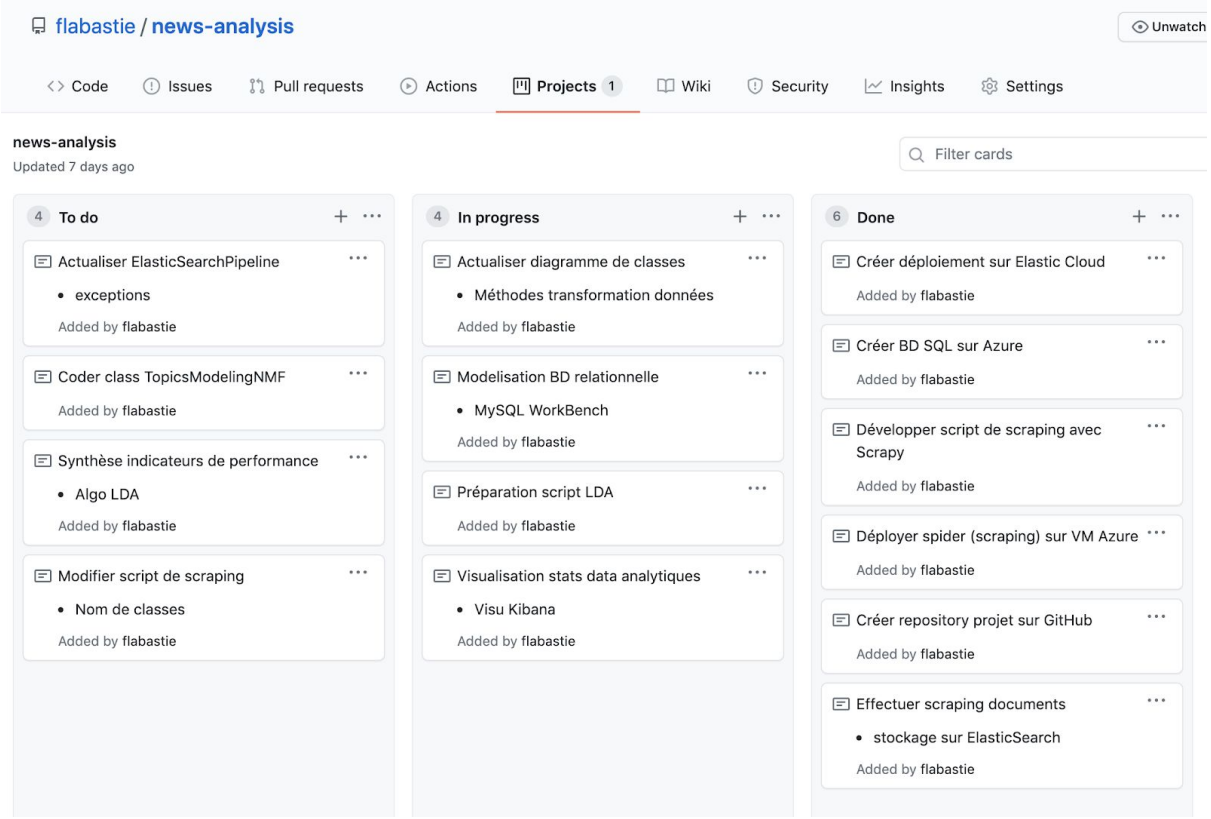
1.2 Méthodologie · Gestion de projet par méthode SCRUM

Le cadre de développement de notre projet se base sur la méthode SCRUM - qui propose un framework de travail holistique itératif - afin de nous concentrer sur nos objectifs de manière la plus productive et créative possible. Nous adopterons ainsi les recommandations AGILE et certains outils spécifiques de la méthode SCRUM.

Suite aux étapes préliminaires de préparation et définition du projet, et de son plan d'exécution, un outil de pilotage et communication tel qu'un tableau kanban est mis en place. Alimenté par une liste "backlog" de tâches et user stories, ce kanban sera l'outil central de notre développement de projet.

[2] http://signsjournal.org/ Revue universitaire féministe américaine

Associé à un wiki d'information, notre outil kanban est disponible en ligne sur le repository GitHub du projet: https://github.com/flabastie/news-analysis



Exemple de Kanban du projet sur GitHub

Le cycle de développement du projet est constitué d'itérations balisées par des "sprints". Nous cherchons alors à produire des livrables le plus rapidement possible que nous améliorons au fur et à mesure des itérations.

La méthode SCRUM requiert en général une équipe avec des rôles précis et distribués : scrum-master, product-owner, etc. Cela n'étant pas le cas dans notre projet, nous conservons cependant l'esprit de SCRUM et les recommandations du manifeste[3] AGILE, dans le cadre des itérations de développement et des échanges avec les parties prenantes.

[3] https://agilemanifesto.org/iso/fr/manifesto.html

2. Modélisation & Infrastructure

2.1 Spécifications fonctionnelles

Afin de définir le périmètre fonctionnel du besoin exprimé, nous modélisons notre application avec UML (Unified Modeling Language), en retenant les diagrammes suivants :

- Diagramme de packages
- Diagramme de cas d'utilisations
- Diagramme de classes

Un diagramme d'infrastructure, proche d'un diagramme de déploiement UML - mais sans en suivre le formalisme - viendra également décrire notre système.

Réalisés avec l'outil en ligne *creately*[4], ces schémas nous permettent de visualiser la conception du système. Dans cette phase nous cherchons principalement à clarifier les choix techniques et définir les composants à développer selon leurs interconnections. Dans une logique de souplesse et d'agilité, notre conception se base sur la simplicité des éléments définis afin de garantir une bonne latitude d'évolution ou de modification.

Notre choix de modélisation avec UML est motivé pour ses avantages en modularité et structuration cohérente des fonctionnalités et des données de notre application. Mais l'argument principal réside dans la communication et la définition du périmètre du projet: Les éléments (diagrammes) présentés sont alors des outils de communication entre toutes les parties prenantes du projet.

[4] https://app.creately.com/

2.1.1 Diagramme de packages

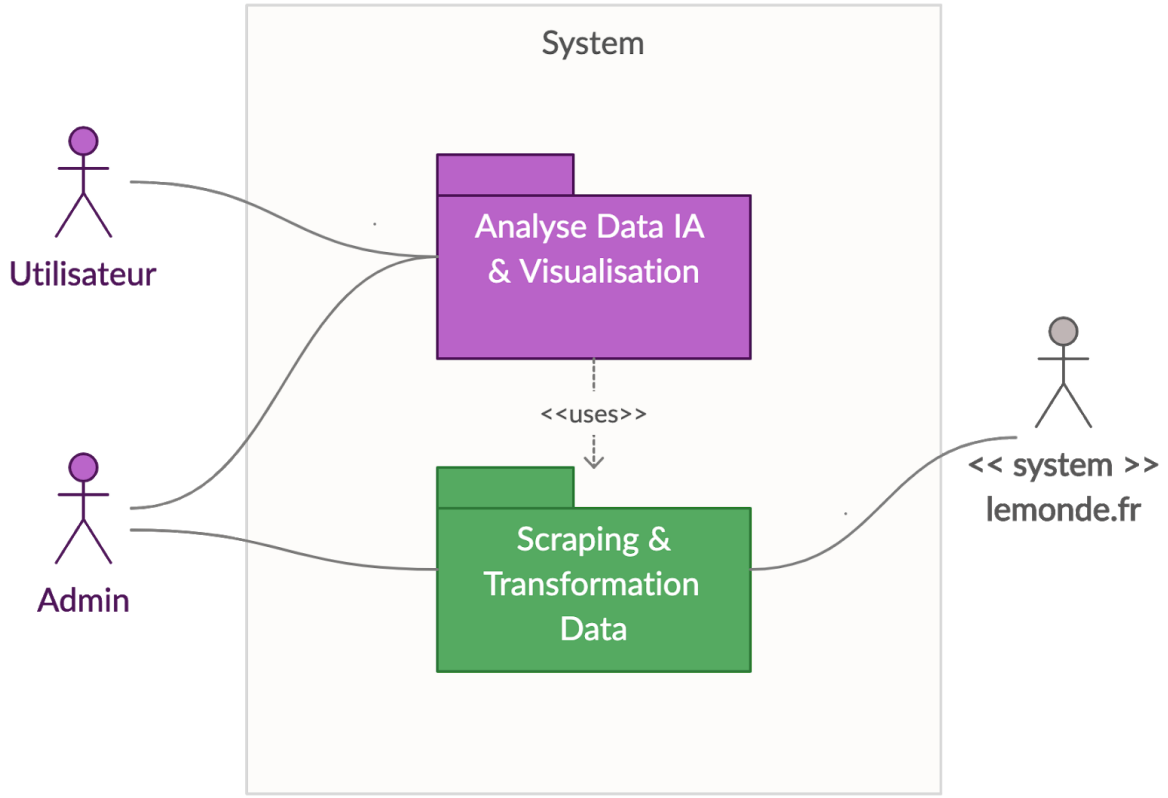


Diagramme de packages du système

Nous divisons notre application en deux grandes parties pour leur autonomie logique, en notant la dépendance de la partie scraping vis à vis du package “Analyse Data IA”, et définissons les acteurs suivants :

- Acteurs principaux
  - Admin → Administrateur du système
  - Utilisateur → Utilisateur du logiciel
- Acteur secondaire
  - lemonde.fr → Système consulté par le logiciel

2.1.2 Diagramme de cas d'utilisations

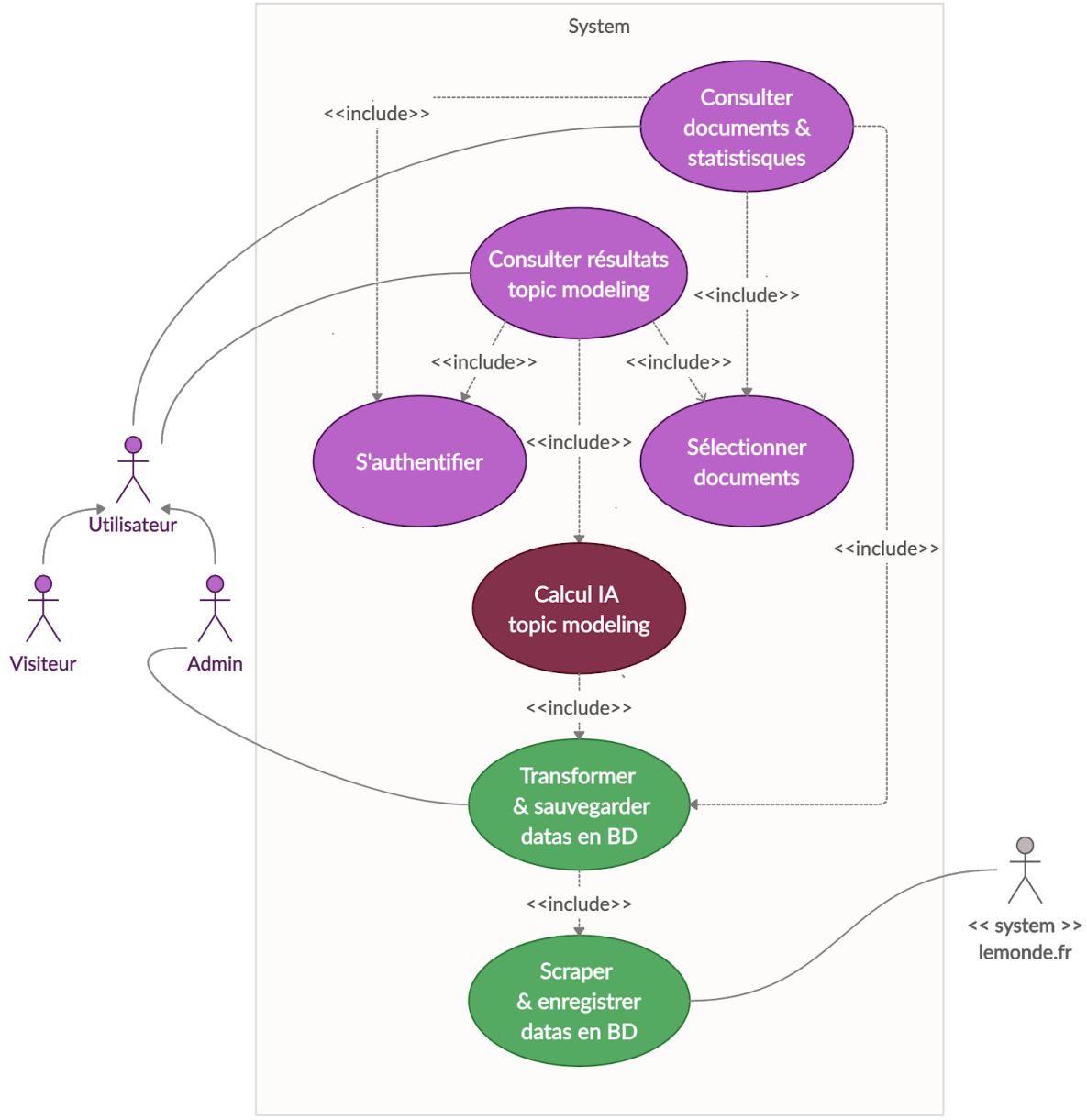


Diagramme de cas d'utilisations du système

Les cas d'utilisations représentent les fonctionnalités ou lots d'actions que vont réaliser nos acteurs. Les deux cas en bas du diagramme correspondent au package "Scraping et transformation data", tandis que les autres se réfèrent au package "Analyse Data IA & visualisation".

2.1.3 Diagramme de classes

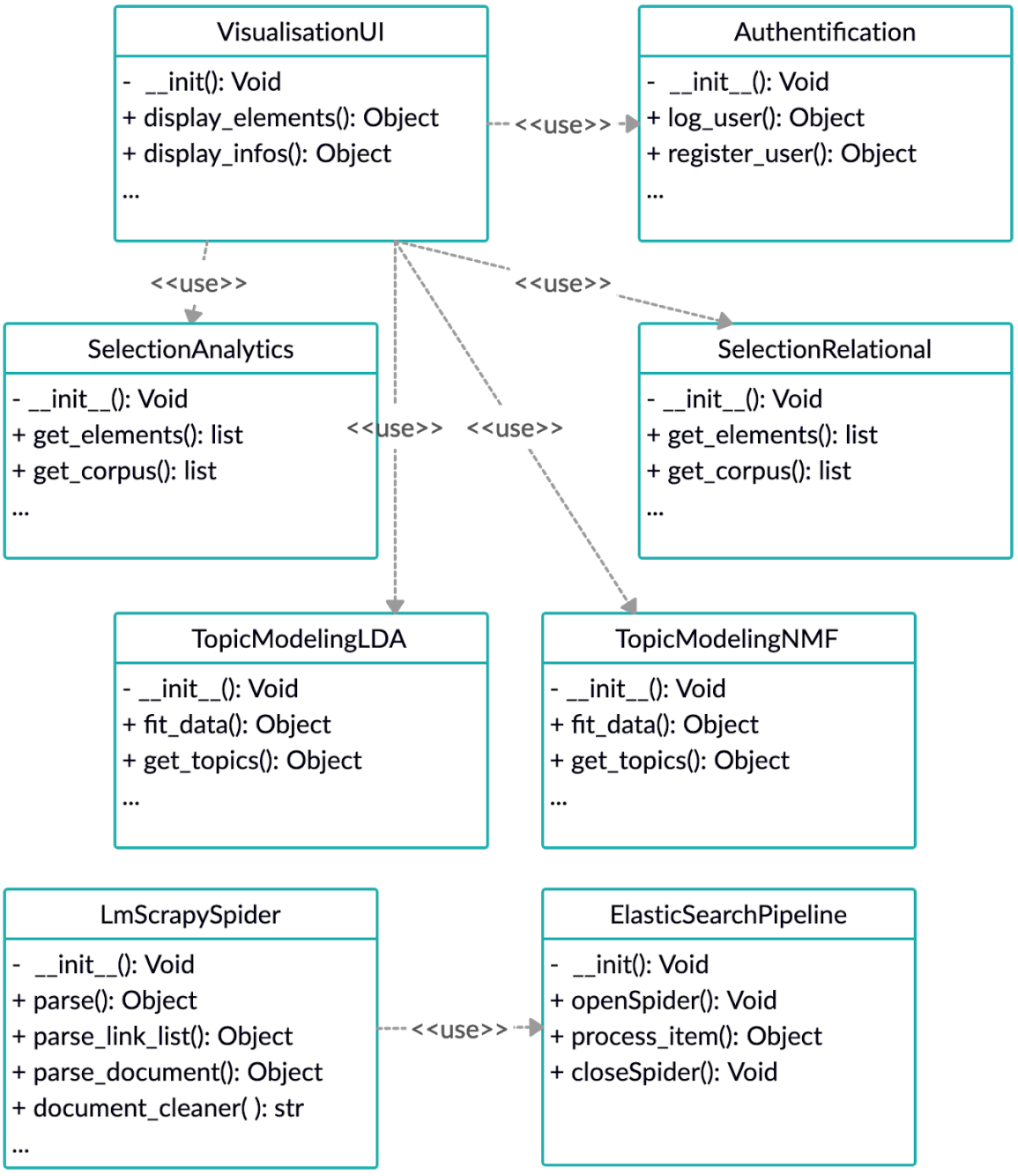


Diagramme de classes du système

Cette vue statique nous permet de modéliser les concepts du domaine d'application en décrivant la structure interne. Le package “Scraping” est illustré par les deux classes en bas de diagramme, tandis que les autres classes correspondent au package “Analyse Data IA & visualisation”.

2.2 Infrastructure du système

2.2.1 Eléments d'infrastructure

Nous présentons ci-après les justifications de choix des outils nécessaires à la réalisation de l'application, visibles également sur le diagramme en page suivante.

ElasticSearch

Une solution NoSQL pour héberger les données non structurées justifiée ici par le type de nos données analytiques. Le choix d'un déploiement ElasticSearch Cloud[5] est motivé par les possibilités offertes en NLP ainsi que par les outils tels que Kibana de la suite Elastic.

Azure SQL database

Le service de base de données relationnelle *Azure SQL Database* nous permet de stocker les données issues de nos transformations, ainsi qu'effectuer certains calculs - dont statistiques - grâce à des requêtes SQL.

VM sur Azure ML Studio

Les outils tels qu'Azure ML Studio nous assistent dans les opérations de Data Science. C'est à partir de machines virtuelles Azure que nous effectuons une partie des processus, du scraping au calcul IA.

Azure Web Apps

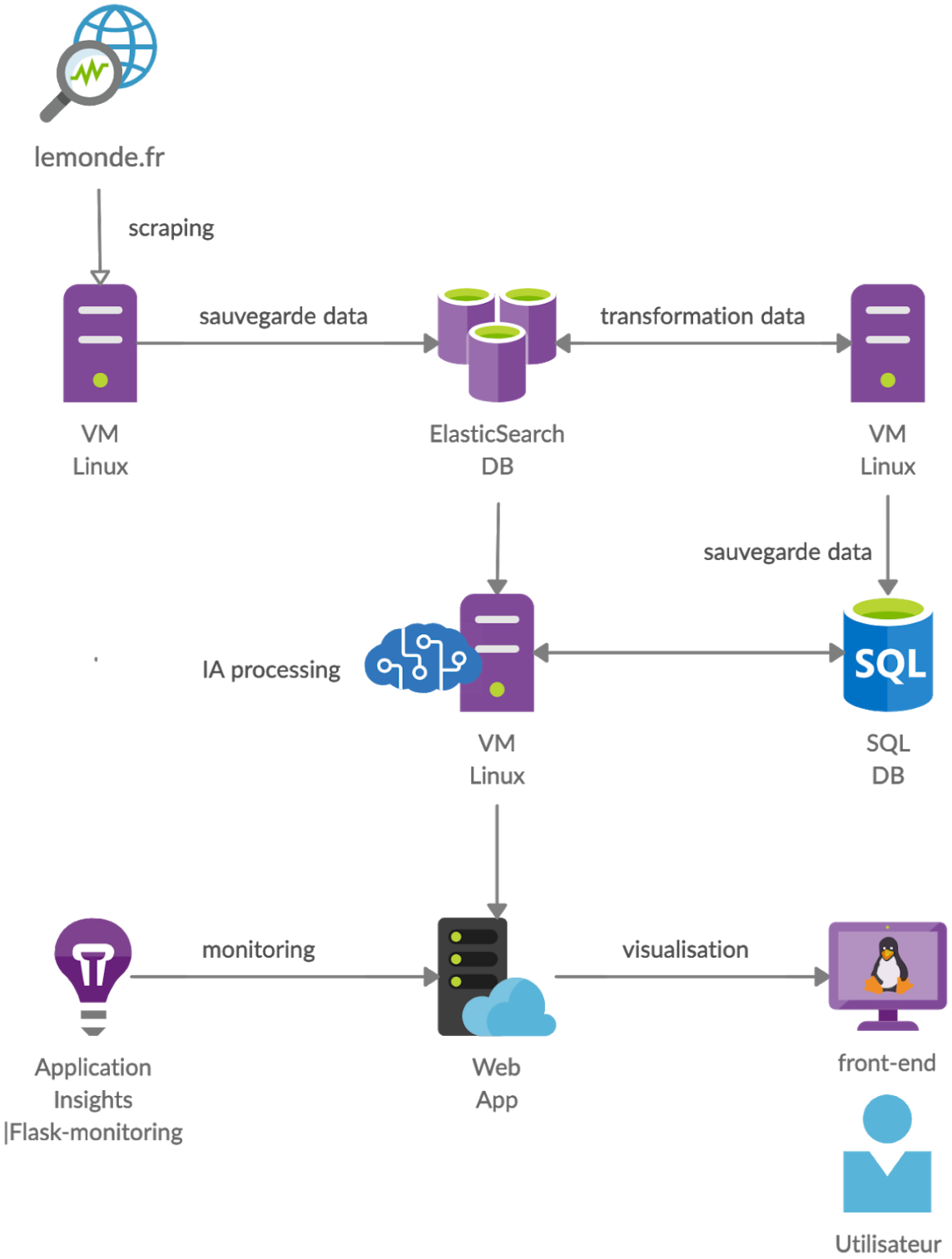
La plate-forme de Microsoft basée sur le cloud computing héberge le site web de notre application via son service *Azure Web Apps*.

Azure Application Insights

Notre application est monitorée par *Azure Application Insights*, un service extensible de gestion des performances des applications (APM).

[5] https://cloud.elastic.co/

2.2.2 Diagramme de l'infrastructure



Infrastructure générale du système

3. Recueil des données & traitements

3.1 Recueil et qualification des données

Les données du projet sont constituées de documents journalistiques textuels (articles, chroniques, etc.) issus de la version en ligne du journal Le Monde[6]. Ces données sont obtenues par scraping[7] à l'aide du framework "scrapy"[8] .

Sur *lemonde.fr* un moteur de recherche[9] permet de requêter des documents par mots-clés et par dates. La liste de liens retournés nous sert de point de départ pour récupérer les documents correspondants. En construisant un script python avec le framework scrapy, nous mettons en place une stratégie d'acquisition des données qui sont directement injectées - par mapping automatique - dans une base ElasticSearch, avec les champs suivants:

- link → url du document
- date → Date de publication du document
- section → Section (ex: sciences, politique …)
- type → Type (ex: article, chronique ...)
- title → Titre du document
- teaser → Résumé textuel du document
- author → Nom et prénom de l'auteur
- content_html → Contenu du document en html
- content_text → Contenu du document en format texte
- content_all → Concaténation titre + teaser + texte
- doc_token → content_all tokenisé
- doc_stem → content_all avec stemming

Le corpus total se compose de plus de 10500 documents - publiés entre le 1er janvier et le 15 décembre 2020 - soit plus d'un million de mots.

[6] https://www.lemonde.fr/
[7] https://fr.wikipedia.org/wiki/Web_scraping
[8] https://scrapy.org/
[9] https://www.lemonde.fr/recherche/

3.2 Import des données non structurées

La persistance des données brutes et non structurées est effectuée sur une base NoSQL ElasticSearch déployée sur le cloud Elastic[10]. L'insertion - par mapping automatique - est effectuée via le module ElasticSearch pour python selon la classe *ElasticSearchPipeline* suivante:

```
import logging
from elasticsearch import Elasticsearch, exceptions
import json
# useful for handling different item types with a single interface
from itemadapter import ItemAdapter

class ElasticSearchPipeline(object):
    '''
        Import data to ElasticSerach db
    '''
    index_name = 'news_analysis'
    es = None

    def open_spider(self, spider):
        '''
            Connection to ElasticSerach db
            :param spider: spider
            :type spider: object
            :return: None
        '''
        logging.warning("SPIDER OPENED FROM PIPELINE")

        # load credentials
        with open('../../config.json') as fconfig:
            credentials = json.load(fconfig)
            domain = credentials['elk']['DOMAIN']
            user = credentials['elk']['USER']
            password = credentials['elk']['PASSWORD']
            port = credentials['elk']['PORT']

        self.es = Elasticsearch(
            [domain],
            http_auth=(user, password),
            scheme="https",
            port=port,)
```

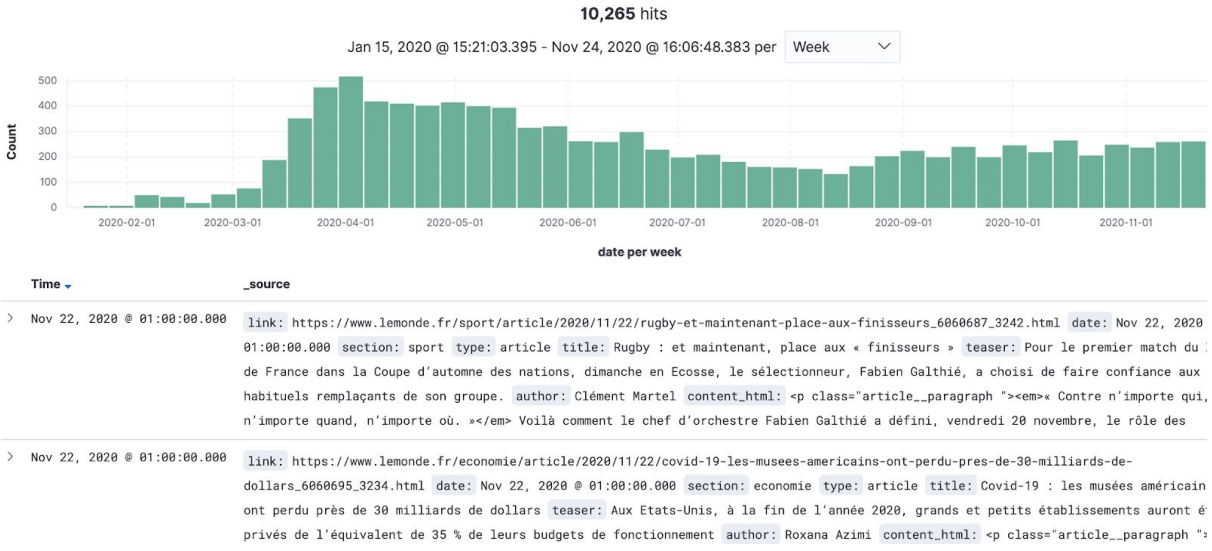
[10] https://cloud.elastic.co/

```
        # Confirming there is a valid connection to Elasticsearch
        try:
            # use the JSON library's dump() method for indentation
            info = json.dumps(self.es.info(), indent=4)
            # pass client object to info() method
            print ("Elasticsearch client info():", info)

        except exceptions.ConnectionError as err:
            # print ConnectionError for Elasticsearch
            print ("\nElasticsearch info() ERROR:", err)
            print ("\nClient host is invalid or cluster is not running")
            # change the client's value to 'None' if ConnectionError
            self.es = None

    def close_spider(self, spider):
        '''
            Close connection to ElasticSerach db
            :param spider: spider
            :type spider: object
            :return: None
        '''
        logging.warning("SPIDER CLOSED FROM PIPELINE")

    def process_item(self, item, spider):
        '''
            Insert data to ElasticSerach db
            :param item: data document
            :type item: object
            :param spider: spider
            :type spider: object
            :return: None
        '''
        self.es.index(index=self.index_name, body=item)
        return item
```

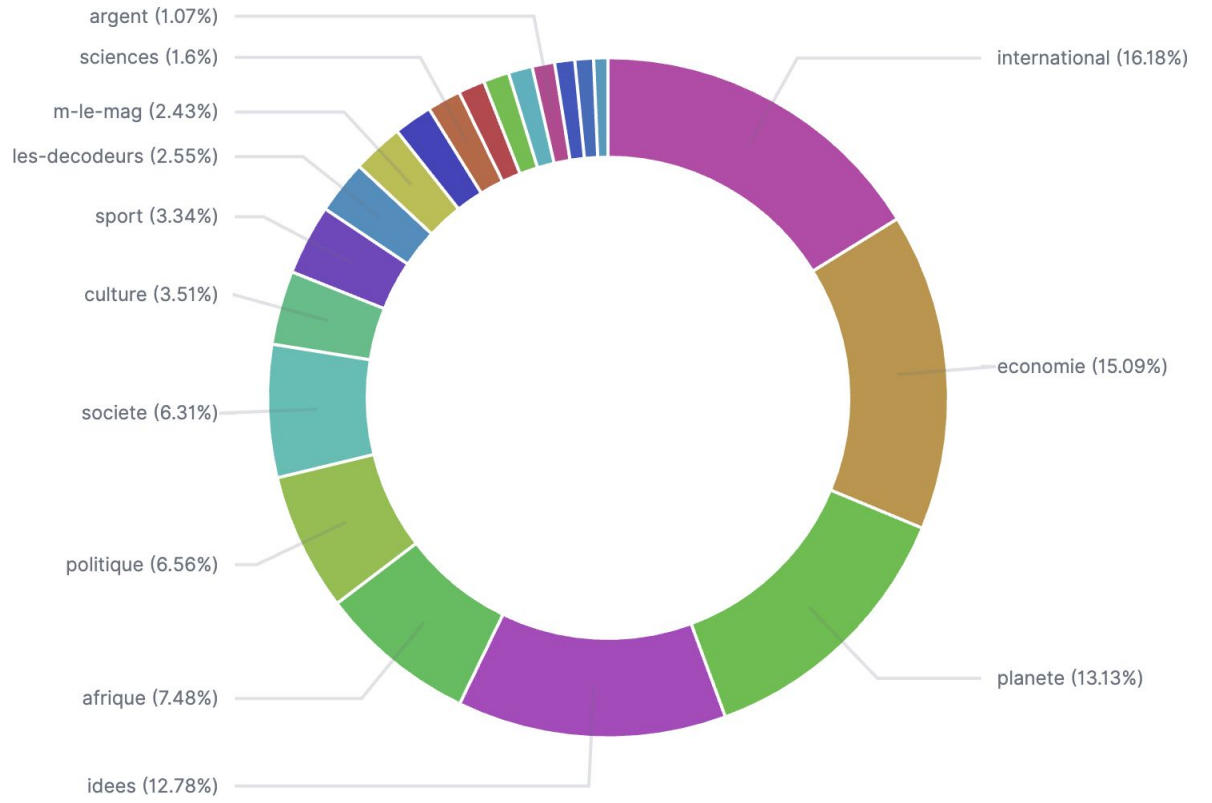
Import des données brutes dans ElasticSearch db

3.3 Exploration et transformation des données analytiques

Une première étude exploratoire des données non structurées est effectuée avec l'outil Kibana - de la suite Elastic - en produisant les visualisations suivantes:



Visualisation de documents/semaine avec Kibana



Visualisation de documents par sections avec Kibana

3.3.1 Tokenisation, stopwords et stemming

Lors du scrapping, le champ *content_all* est nettoyé de toute ponctuation et tokenisé grâce à la librairie nltk, pour alimenter le champ *doc_token*. Le processus se poursuit avec une fonction de stemming (racinisation) pour compléter le champ *doc_stem*. Le choix d'utiliser nltk dans cette phase de suppression des stopwords est justifié par le souhait de customiser la liste des mots effacés du corpus, afin de ne pas supprimer automatiquement certains mots fréquents, tels que "Covid, France, président, etc." Nous obtenons alors notre propre liste de stopwords, nommée "custom_sw" et constituée en amont des traitements dont nous présentons quelques fonctions ci-dessous:

```
def tokenizer_punctuation(self, sample_text):
    '''
        tokenizer_punctuation : removes punctuation
        :param sample_text: text content
        :type sample_text: str
        :return tokenizer: tokenized words
        :rtype: list
    '''
    # tokenizer definition
    tokenizer = nltk.RegexpTokenizer(r'\w+')
    # return text without punctuation
    return tokenizer.tokenize(sample_text)

def remove_stopwords(self, custom_sw, content):
    '''
        remove_stopwords : removes stopwords
        :param custom_sw: stopwords
        :type custom_sw: list
        :param content: content
        :type content: list
        :return doc_token: words without stopwords
        :rtype: list
    '''
    temp_tokens = []
    doc_token = []
    # tokenize content
    temp_tokens = self.tokenizer_punctuation(content.lower())
    doc_token += [w for w in temp_tokens if not w in self.custom_sw]
    return doc_token
```

```
def french_stemmer(self, doc_token):
    '''
        french_stemmer : stemes stopwords
        :param doc_token: tokenized words
        :type doc_token: list
        :return doc_stem: stemmed words
        :rtype: list
    '''
    # french stemmer
    stemmer = FrenchStemmer()
    doc_stem = []
    # stemming
    doc_stem += [stemmer.stem(w) for w in doc_token]
    # return stemmed tokens
    return doc_stem
```

3.3.2 Eléments de recherche

En NLP, les requêtes booléennes de type tout ou rien ne sont pas suffisantes,, aussi il est courant de raisonner en termes de pertinence pour valider un document comme satisfaisant à une recherche.[11] Certains outils de métrique, distance et similarité répondent à ce besoin, ainsi que les caractéristiques telles que "*poids des mots*", et "*fréquences des termes*".

Les poids de mots sont valorisés selon deux principes[12] :

1. Plus un terme est présent dans un document, plus il est représentatif du contenu de ce document.
2. Moins un terme est présent dans un corpus, et plus une occurrence de ce terme est significative.

La fréquence d'un terme t dans un document d correspond au nombre d'occurrences de t dans d, avec $n_{t,d}$ étant le nombre d'occurrences de t dans d. Ce que nous illustrons par la formule et l'exemple suivant:

$$\mathrm{tf}(t, d) = n_{t,d}$$

[11] http://b3d.bdpedia.fr/ri-ranking.html
[12] http://b3d.bdpedia.fr/ri-ranking.html#s2-recherche-plein-texte

terme	d1	d2	d3
pandémie	30	10	12
covid	0	5	40
confinement	8	5	14

Exemple de fréquences de termes

Pour éviter l'effet induit par la taille - qui amènerait à classer les longs documents systématiquement en tête -, ces valeurs de tf peuvent être normalisées : chaque tf étant divisé par le nombre total de termes dans le document.

Un autre élément à prendre en compte est la fréquence inverse d'un terme dans les documents (inverse document frequency, ou **idf**), qui mesure l'importance d'un terme par rapport à une collection D de documents. Un terme qui apparaît rarement peut être considéré comme plus caractéristique d'un document qu'un autre, très commun.

L'idf d'un terme t correspond alors à la division du nombre total de documents par le nombre de documents contenant au moins une occurrence de t. Et pour conserver cette valeur idf dans un intervalle comparable à celui du tf[13], on prend le logarithme de cette fraction :

$$\mathrm{idf}(t) = \log \frac{|D|}{\left|\{d' \in D \mid n_{t,d'} > 0\}\right|}$$

3.3.3 TF-IDF

tf (normalisé ou non) et **idf** peuvent être combinés pour obtenir le poids **tf.idf** d'un terme **t** dans un document. C'est ainsi le produit des deux valeurs précédentes. Le TF-IDF (de l'anglais term frequency-inverse document frequency) est donc une méthode de pondération souvent utilisée en recherche d'information et en particulier dans la fouille de textes.

[13] http://b3d.bdpedia.fr/ri-ranking.html#s2-recherche-plein-texte

Pour les calculs de tf.idf de notre projet, scikit-learn propose une classe *TfidfVectorizer*[14] permettant la conversion de documents bruts en matrice de caractéristiques TF-IDF. Nous emploierons également la classe *CountVectorizer*[15] nous permettant de convertir une collection de documents texte en une matrice de nombres de tokens. Nous disposons ainsi d'outils accessibles pour notre étude, avec d'une part les champs de données analytiques brutes, les champs traités par tokenizer, stopwords, puis stemming, et d'autre part des outils de calcul tf.idf de scikit-learn.

3.3.4 TF-IDF dans ElasticSearch[16]

Par défaut, ElasticSearch gère la similarité selon un algorithme nommé BM25. Un paramétrage est cependant possible, selon que l'on souhaite employer le TF/IDF ou booléen, avec les choix suivants :

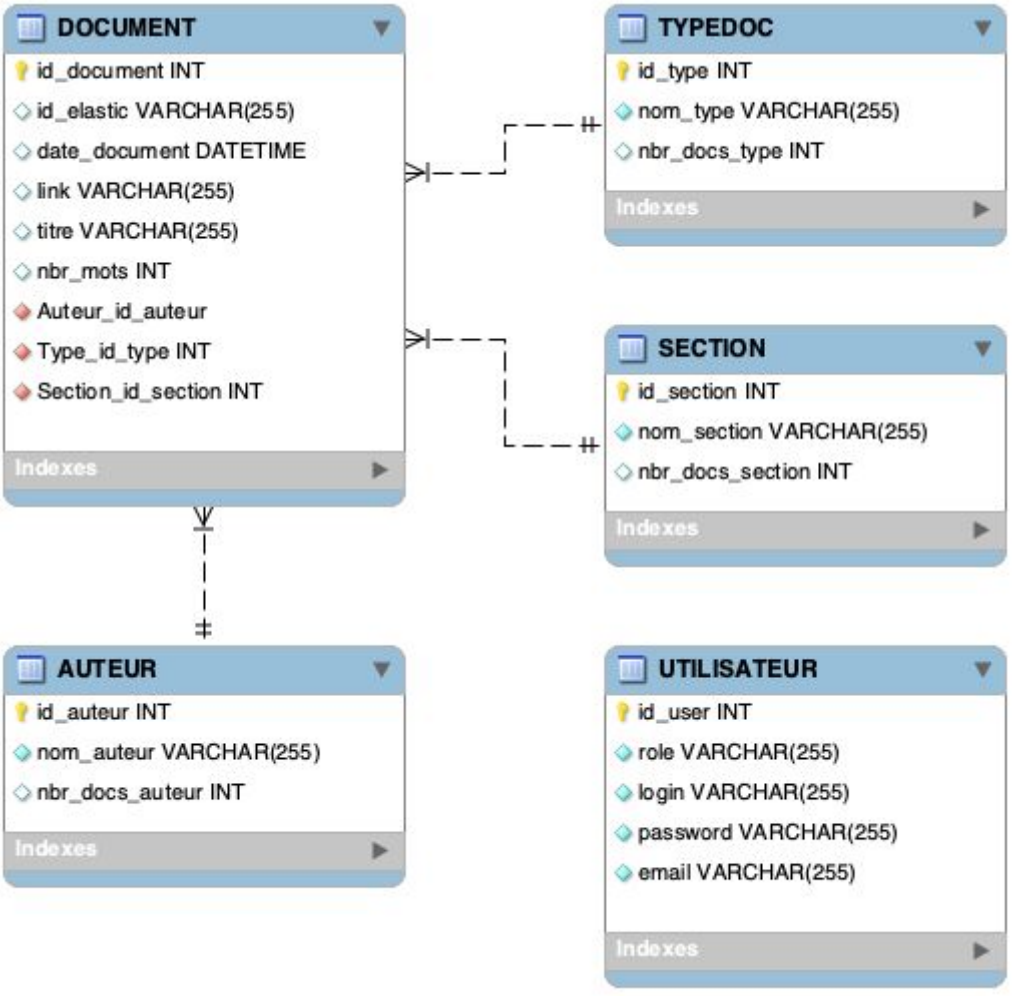
- BM25
  L'algorithme Okapi BM25, utilisé par défaut dans Elasticsearch et Lucene.
- classic
  L'algorithme TF / IDF, l'ancien par défaut dans Elasticsearch et Lucene. (Déconseillé dans la version 7.0.0.)
- booléen
  Une simple similitude booléenne, utilisée lorsque le classement en texte intégral n'est pas nécessaire et que le score ne doit être basé que sur le fait que les termes de la requête correspondent ou non.

Okapi BM25[17] est une méthode de pondération utilisée en recherche d'information. en application d'un modèle probabiliste de pertinence. C'est l'algorithme que nous emploierons dans ce projet, au niveau des requêtes effectuées par ElasticSearch.

[14] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html
[15] https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html
[16] https://www.elastic.co/guide/en/elasticsearch/reference/current/similarity.html
[17] https://fr.wikipedia.org/wiki/Okapi_BM25

3.4 Import des données structurées

Pour effectuer certains calculs statistiques, nous souhaitons récupérer une partie des données analytiques et les structurer. Pour cela nous modélisons une base de données relationnelle déployée sur Azure SQL. Elle accueillera les données de nos documents sans les parties textuelles “longues” (contenus d’articles). En parallèle, cette base accueillera également des données d’authentification pour la partie web de l’application (Table UTILISATEUR).



MCD - Schéma Entités-Relations de la base SQL

3.4.1 Création base Azure SQL

```
CREATE TABLE AUTEUR (
  id_auteur INT NOT NULL IDENTITY PRIMARY KEY,
  nom_auteur VARCHAR(255) NOT NULL,
  nbr_docs_auteur INT NULL
)
CREATE TABLE TYPEDOC (
  id_type INT NOT NULL IDENTITY PRIMARY KEY,
  nom_type VARCHAR(255) NOT NULL,
  nbr_docs_type INT NULL,
)
CREATE TABLE SECTION (
  id_section INT NOT NULL IDENTITY PRIMARY KEY ,
  nom_section VARCHAR(255) NOT NULL,
  nbr_docs_section INT NULL,
)
CREATE TABLE DOCUMENT (
  id_document INT NOT NULL IDENTITY PRIMARY KEY,
  id_elastic VARCHAR(255) NULL,
  date_document DATETIME NULL,
  link VARCHAR(255) NULL,
  titre VARCHAR(255) NULL,
  nbr_mots INT NULL,
  Auteur_id_auteur INT NOT NULL,
  Type_id_type INT NOT NULL,
  Section_id_section INT NOT NULL,
  INDEX fk_Document_Auteur_idx (Auteur_id_auteur ASC),
  INDEX fk_Document_Type1_idx (Type_id_type ASC),
  INDEX fk_Document_Section1_idx (Section_id_section ASC),
  CONSTRAINT fk_Document_Auteur
    FOREIGN KEY (Auteur_id_auteur)
    REFERENCES AUTEUR (id_auteur),
  CONSTRAINT fk_Document_Type1
    FOREIGN KEY (Type_id_type)
    REFERENCES TYPEDOC (id_type),
  CONSTRAINT fk_Document_Section1
    FOREIGN KEY (Section_id_section)
    REFERENCES SECTION (id_section),
)

CREATE TABLE UTILISATEUR (
  id_user INT NOT NULL IDENTITY PRIMARY KEY,
  role VARCHAR(255) NOT NULL,
  login VARCHAR(255) NOT NULL,
  password VARCHAR(255) NOT NULL,
  email VARCHAR(255) NOT NULL,
)
```

Code de création de la base Azure SQL

3.4.2 Insertion de données en base Azure SQL

Les tables de notre base SQL utiles aux informations statistiques du corpus sont alimentées par un script spécifique, dont nous présentons deux fonctions principales ci-après. Les données sont sélectionnées depuis la base ElasticSearch, et insérées dans la base relationnelle Azure SQL.

```
def insert_elements(self, table_name, datalist):
'''
    Get types list
    :param table_name: table_name
    :type table_name: str
    :param datalist: datalist
    :type datalist: list
    :return: None
'''
cursor = self.conn.cursor()
if table_name == 'TYPEDOC':
    # Insert one by one.
    for item in datalist:
        cursor.execute("INSERT INTO TYPEDOC (nom_type, nbr_docs_type)
VALUES (?,?)", item['key'], item['doc_count'])
        cursor.commit()
        print(item)
if table_name == 'AUTEUR':
    # Insert one by one.
    for item in datalist:
        cursor.execute("INSERT INTO AUTEUR (nom_auteur, nbr_docs_auteur)
VALUES (?,?)", item['key'], item['doc_count'])
        cursor.commit()
        print(item)
if table_name == 'SECTION':
    # Insert one by one.
    for item in datalist:
        cursor.execute("INSERT INTO SECTION (nom_section,
nbr_docs_section) VALUES (?,?)", item['key'], item['doc_count'])
        cursor.commit()
        print(item)
```

```
def get_elements_list(self, element_name):

'''
    get_elements_list
    :param self: self
    :type self: None
    :param element_name: element_name
    :type element_name: str
    :return: elements_list
    :rtype: list of dics (doc_count & key)
'''
res = self.es.search(index=self.index_name, body={
        "size": 0,
        "aggs": {
            "Articles": {
                "filter": {
                    "range": {
                        "date": {
                            "gte": "2020-01-01T00:00:00.00"
                        }
                    }
                },
                "aggs": {
                    "GroupBy": {
                        "terms": { "field": element_name + ".keyword",
"size": 10000 }
                    }
                }
            }
        }
    }
)
elements_docs = res['aggregations']['Articles']['GroupBy']['buckets']
# sorting by doc_count desc
elements_list  = [item for item in sorted(elements_docs, key = lambda i:
i['doc_count'], reverse=True)]
# list of dics (doc_count & key)
return elements_list
```

Code d'insertion de données dans la base Azure SQL

4. Analyse grâce à l'intelligence artificielle

4.1 Topic modeling · Présentation · Etat de l'art

En présence de grands volumes de données de type texte non structuré, nous souhaitons aborder l'information de manière automatique. Certains outils utiles pour structurer les données, permettent d'atteindre rapidement les thèmes d'intérêts, filtrer le bruit et détecter de nouvelles thématiques.

Par exemple, il peut être intéressant de :

- Détecter des "trending topics" de Twitter
- Trouver de nouveaux sujets d'information abordés par les médias
- Organiser un corpus de textes scientifiques autour de thématiques

C'est dans ce cadre qu'intervient la modélisation de sujets qui représente les différentes approches permettant cette détection. Plusieurs algorithmes répondent à ce besoin, dont le Latent Dirichlet Allocation (LDA) que nous adoptons dans ce projet. Ces algorithmes ne sont cependant pas applicables directement à toutes les situations, à moins de choisir certaines variantes spécifiques à des problématiques précises. Il s'agit d'une famille de méthodes utilisées essentiellement en exploration, permettant de détecter si effectivement de grandes catégories sont abordées, et ensuite les affiner lors du passage en production.

Latent Dirichlet Allocation (LDA) est un modèle de Machine Learning non supervisé spécifique au NLP qui vise à relier les fréquences de mots dans les documents avec la présence ou l'absence de certains thèmes. Le nombre de topics est défini à l'avance mais les topics eux-mêmes sont découverts automatiquement par cette méthode.[18]

[18] CHAUMARTIN François-Régis & LEMBERGER Pirmin,
Le traitement automatique des langues, Ed. Dunod, 2020

4.2 Présentation de Latent Dirichlet Allocation[19]

Prenons par exemple l'ensemble de phrases suivant :

- *J'aime manger du brocoli et des bananes.*
- *J'ai mangé un smoothie aux bananes et aux épinards pour le petit déjeuner.*
- *Les chinchillas et les chatons sont mignons.*
- *Ma sœur a adopté un chaton hier.*
- *Regardez ce mignon hamster grignoter un morceau de brocoli.*

"Latent Dirichlet allocation" est un moyen de découvrir automatiquement les sujets contenus dans ces phrases. Par exemple, en supposant que nous ayons défini un nombre de sujets (topics) à 2, LDA pourrait produire quelque chose comme :

➔ **Phrases 1 & 2**: 100% Topic A
➔ **Phrases 3 & 4**: 100% Topic B
➔ **Phrase 5**: 60% Topic A, 40% Topic B

➔ **Topic A**: 30% brocoli, 15% bananes, 10% petit-déjeuner, 10% grignotage, …
Nous pouvons interpréter le sujet A comme traitant de la nourriture.
➔ **Topic B**: 20% chinchillas, 20% chatons, 20% mignon, 15% hamster, …
Nous pouvons interpréter le sujet B comme traitant des animaux mignons

Le modèle Latent Dirichlet Allocation - LDA - est ainsi un modèle probabiliste génératif qui permet de décrire des collections de documents de texte ou d'autres types de données discrètes. LDA fait partie d'une catégorie de modèles appelés "topic models", qui cherchent à découvrir des structures thématiques cachées dans de vastes archives de documents.[20]

[19] http://blog.echen.me/2011/08/22/introduction-to-latent-dirichlet-allocation/
[20] http://alberto.bietti.me/files/rapport-lda.pdf

4.3 Description de l'algorithme LDA[21]

Le fonctionnement repose sur un nombre K de thèmes fixé, et l'on cherche à apprendre les thèmes représentés dans chaque document ainsi que les mots associés à ces thèmes, ou topics.

A l'initialisation, un topic est attribué à chaque mot de chaque document, selon une distribution de Dirichlet[22] sur un ensemble de K thèmes. Un premier "modèle de sujet" est généré, avec des topics présents dans les documents ainsi que les mots de définition de ces thèmes.

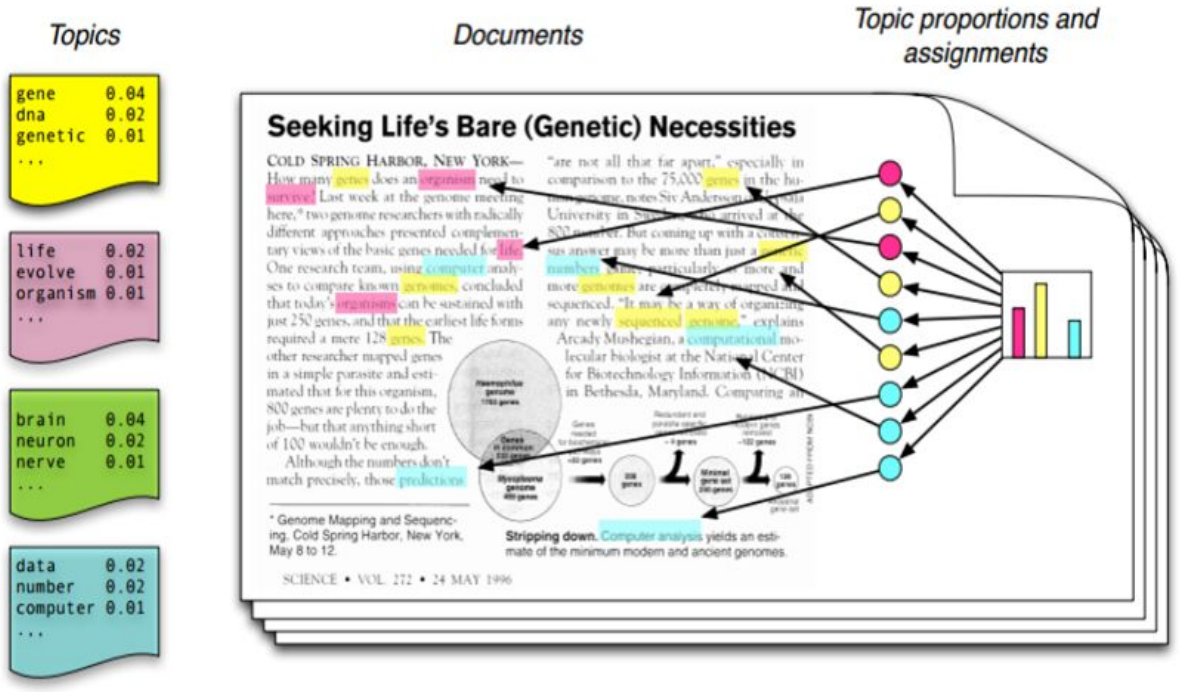


Schéma descriptif pour LDA[23]

La phase d'apprentissage consiste à améliorer ce modèle de sujet défini aléatoirement à l'initialisation: Dans chaque document, chaque mot est sélectionné pour actualiser son thème de correspondance. Ce nouveau thème est celui qui aurait la probabilité la plus élevée de le générer dans ce document.

[21] https://fr.wikipedia.org/wiki/Allocation_de_Dirichlet_latente
[22] https://fr.wikipedia.org/wiki/Loi_de_Dirichlet
[23] https://www.researchgate.net/publication/334132970_Deep_LDA_A_new_way_to_topic_model

L'hypothèse repose sur l'idée que tous les thèmes sont corrects, sauf pour le mot en question. Des probabilités sont calculées pour ces associations, puis l'étape est relancée. En répétant ce processus un grand nombre de fois, les assignations se stabilisent. Un mélange de thèmes présents dans chaque document est obtenu en comptabilisant chaque représentation de ces topics. Enfin, les mots associés à chaque thème sont obtenus en calculant leurs occurrences dans le corpus.

Nous proposons d'appliquer LDA à notre corpus, en l'occurrence un ensemble de plusieurs milliers d'articles du Monde traitant du Covid-19, car nous souhaitons explorer les structures thématiques cachées dans ces documents. Ceux-ci étant caractérisés par leurs titres, auteurs, dates et sections, nous pouvons également comparer de manière statistique certaines occurrences de thématiques et leurs évolutions.

4.4 Autres algorithmes de topic modeling

L'algorithme NMF[24] pour Negative Matrix Factorisation (factorisation matricielle non négative), est une alternative à LDA. Il fait partie d'un groupe d'algorithmes en analyse multivariée dans lesquels, contrairement à l'ACP, les facteurs ne sont pas orthogonaux et ne permettent pas de représentation, mais en réduisant la dimension ils permettent des classifications non supervisées et des modèles prévisionnels. Ces algorithmes sont principalement conçus pour l'analyse de très grandes matrices dans les domaines du e-commerce, text mining, etc.

Le modèle BERT[25] (Bidirectional Encoder Representations from Transformers), qui est un modèle NLP développé par Google en 2018 propose également une solution de topic modeling. Dans le projet qui nous concerne, nous implémenterons uniquement LDA et NMF, en réservant BERT pour une évolution future de notre application.

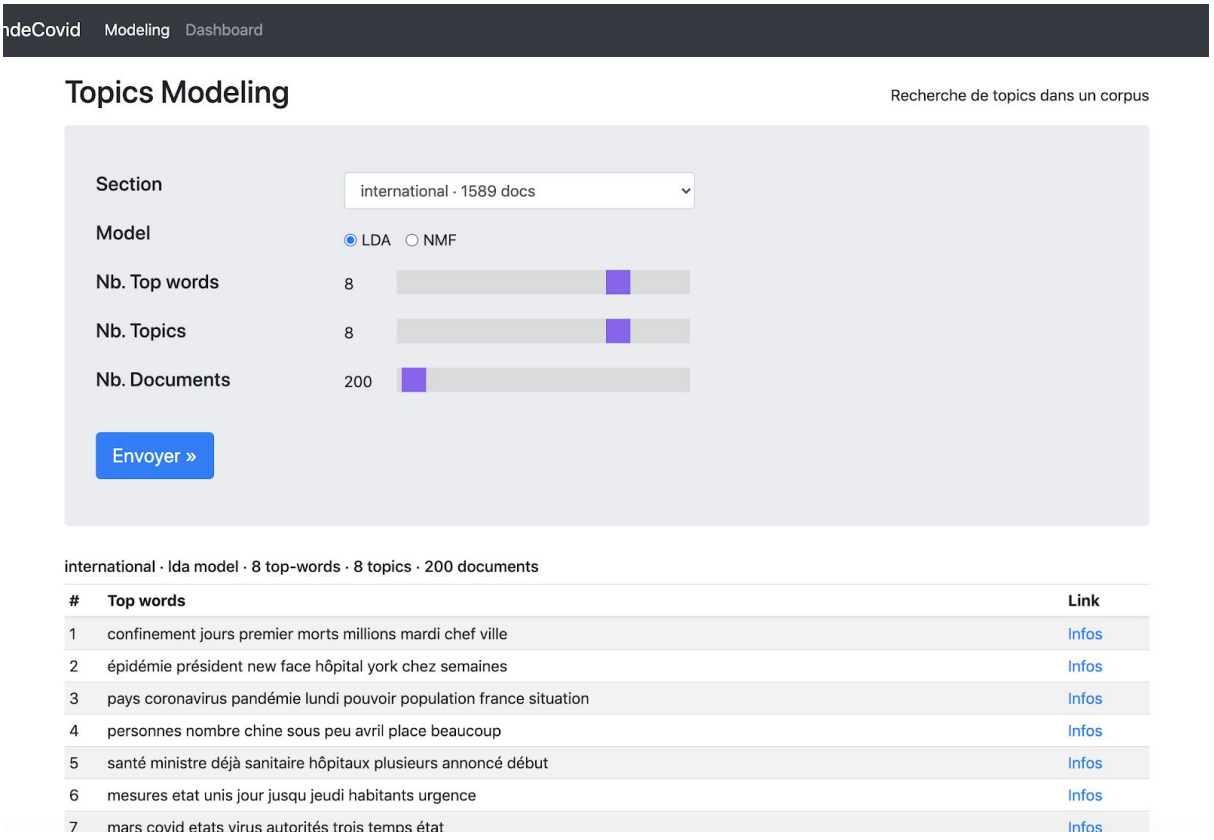
[24] https://www.math.univ-toulouse.fr/~besse/Wikistat/pdf/st-m-explo-nmf.pdf
[25] https://towardsdatascience.com/topic-modeling-with-bert-779f7db187e6

5. Développement & tests

5.1 Création du site web

La partie web de l'application est développée en python/htlm/css/js, avec les frameworks python Flask[26] et Bootstrap[27] pour la partie front-end. Hébergée sur le service Azure Web App, nous en présentons une vue ci-dessous :

https://lemondecovid.azurewebsites.net/



Vue de l'application (page Topics Modeling)

L'utilisateur sélectionne une section de recherche pour le corpus ainsi que les paramètres de création de topics. L'application retourne la liste des topics trouvés, sous forme de séries de mots, et un lien permet ensuite d'accéder aux documents concernés.

[26] https://flask.palletsprojects.com/en/1.1.x/
[27] https://getbootstrap.com/

5.2 Tests unitaires · pytest

Nous appliquons des tests unitaires sur nos deux packages (Analyse Data IA & visualisation, et Scraping & transformation data), afin de garantir le fonctionnement correct de notre programme. L'objectif étant d'isoler le comportement de chaque partie de code à tester et de vérifier que le code est conforme à ce qui est attendu. Nous utilisons l'outil pytest pour effectuer les tests unitaires, dont nous présentons un exemple ci-après.

```
import pytest
from queries.selection import SelectionAnalytics

def test_get_elements_list():
    '''
        test get_elements_list
        :param element_name: element_name
        :type element_name: str
        :return: elements_list
        :rtype: list of dics (doc_count & key)
    '''
    # param get_elements_list()
    param = 'section'
    # SelectionAnalytics instance
    process_doc = SelectionAnalytics()
    # use function get_elements_list()
    result_get_elements_list = process_doc.get_elements_list(param)
    # test result of function get_elements_list is list
    assert isinstance(result_get_elements_list, list) == True
    # test elements result are dict
    assert isinstance(result_get_elements_list[0], dict) == True
    # test 1st elements of dict is int
    assert isinstance(result_get_elements_list[0]['doc_count'], int) ==
True
    # test 2nd elements of dict is string
    assert isinstance(result_get_elements_list[0]['key'], str) == True
```

```python
def test_get_custom_corpus():
    '''
        test get_custom_corpus
        :param section_name: section_name
        :type section_name: str
        :param query_size: query_size
        :type query_size: int
        :return: (custom_corpus, total_hits)
        :rtype: tuple (custom_corpus, total_hits)
    '''
    # params get_custom_corpus()
    param_section_name = 'emploi'
    param_query_size = '10'
    # SelectionAnalytics instance
    process_doc = SelectionAnalytics()
    # use function get_custom_corpus()
    corpus = process_doc.get_custom_corpus(param_section_name,
param_query_size)
    # test result of function get_custom_corpus is tuple
    assert isinstance(corpus, tuple) == True
    # test corpus[0] is list
    assert isinstance(corpus[0], list) == True
    # test corpus[1] is int
    assert isinstance(corpus[1], int) == True
```

```
(scrapy-env) pc-54:lemondecovid francois$ pytest
============================= test session starts ==============================
platform darwin -- Python 3.7.7, pytest-5.4.1, py-1.8.1, pluggy-0.13.1
rootdir: /Users/francois/development/projects/00_news-analysis/lemondecovid
collected 2 items

tests/test_queries_elastic.py .F                                         [100%]

=================================== FAILURES ===================================
_____________________________ test_get_custom_corpus _____________________________

    def test_get_custom_corpus():
        '''
            test get_custom_corpus
            :param section_name: section_name
            :type section_name: str
            :param query_size: query_size
            :type query_size: int
            :return: (custom_corpus, total_hits)
            :rtype: tuple (custom_corpus, total_hits)
        '''
        # params get_custom_corpus()
        param_section_name = 'emploi'
        param_query_size = '10'
        # SelectionAnalytics instance
        process_doc = SelectionAnalytics()
        # use function get_custom_corpus()
        corpus = process_doc.get_custom_corpus(param_section_name, param_query_size)
        # test result of function get_custom_corpus is tuple
>       assert isinstance(corpus, list) == True
E       AssertionError: assert False == True
E        +  where False = isinstance(['droit', 'retrait', 'applique', 'salariés', 'estimant', 'insuffisamment', ...], 115), list)

tests/test_queries_elastic.py:47: AssertionError
---------------------------- Captured stdout call -----------------------------
```

Exemple de code de tests unitaires avec pytest[28]

[28] https://docs.pytest.org/en/latest/

5.3 Documentation · pydoc

Au cours du projet, le code informatique est documenté grâce à l'outil pydoc[29], intégré à python, qui nous permet de générer une documentation en pages html comme l'exemple ci-dessous. Pour cela, des docstrings sont écrites au coeur de chaque classe et fonctions de notre application.

Exemple de documentation avec l'outil pydoc

[29] https://docs.python.org/fr/3/library/pydoc.html

5.4 Monitoring

Nous souhaitons maintenir notre application en détectant et corrigeant les éventuels dysfonctionnements[30]. Nous appliquons en premier lieu une méthode d'identification des éléments et parties critiques à l'aide du tableau suivant, qui nous permet d'évaluer les niveaux de criticité.

Par exemple, le risque "*Le Monde modifie le code html de ses articles*" nous donne une probabilité de risque "Probable" avec une gravité de risque "Très grave", qui nous amène à un facteur de risque de 6.

Probabilité du risque	Peu grave	Grave	Très grave	Fatal
Certaine	4	8	12	16
Fort probable	3	6	9	12
Probable	2	2	6	8
Peu probable	1	2	3	4
				Gravité du risque

Base de calcul des facteurs de risques

Nous établissons ainsi une liste de risques à évaluer pour notre projet, que nous associerons à un outil de monitoring pour surveiller nos données instantanées, et conserver un historique des données.

[30] https://plume-next.mathrice.fr/fr/ressource/applications-et-methodes-de-monitoring-java

Risque	Criticité	Indicateur de monitorage	Action corrective
R1: *Le Monde modifie le code html de ses articles*	6	Scraping de nouvelles données impossible	Modification du code spider scraping
R2: *Le Monde met en place une défense anti scraping*	8	Correctif de scraping difficile à mettre en oeuvre	Modification d'outils et infrastructure de scraping
R3: Un utilisateur partage les accès du site web	1	Nombre d'accès élevés au cours des 5 dernières minutes.	Restriction d'accès ou bien scaling de l'application
R4: Effacement des données analytiques (ou hacking bd)	2	Utilisation des données impossible	Récupération d'une sauvegarde sur Elastic cloud
R5: Résultats attendus insatisfaisants en topics modeling	6	Comparaisons et évaluations de résultats peu concordants	Vérification et amélioration du code de topics modeling
R6: Effacement des données structurées (bd SQL)	1	Utilisation des données statistiques impossible	Récupération des données depuis selection data analytics
R7: Intrusion site web de l'application et hacking	6	Application web défaillante et intrusion détectée par monitoring	Réinstallation d'une sauvegarde sur le site web

Tableau indicateur de monitorage

Conclusion

Nous souhaitons - par ce projet - faciliter l'exploration de documents textuels dans un corpus de grande taille. Le choix du média Le Monde fut motivé par la richesse et la qualité de ses informations. La présence d'un moteur de recherche sur le site lemonde.fr[31] nous a également permis de sélectionner au préalable les documents souhaités, correspondant au mot-clé "covid". Mais la valeur ajoutée de notre application repose sur l'intelligence artificielle, par les algorithmes de topics modeling employés. Il s'agit cependant d'un prototype, qui nous permet de tester diverses méthodes exploratoires en NLP.

Les pistes d'améliorations de notre application reposent sur trois bases: le perfectionnement du code, l'optimisation des outils et la précision dans l'emploi des algorithmes d'IA, en est la première. La seconde piste nous guide vers une conception logicielle davantage orientée par les besoins utilisateurs, dans le domaine linguistique en particulier. En effet, les résultats de nos topics peuvent apparaître comme de simples messages cryptiques à un utilisateur néophyte, car dépourvus d'explications ou de données justifiant leurs productions. La solution à ce problème nous mène vers la troisième orientation d'amélioration : la statistique. Des données chiffrées sur les résultats produits, ainsi que des paramètres quantifiants et qualifiants les requêtes de recherche, permettraient de guider l'utilisateur dans son parcours.

L'évolution de ce projet repose également sur une meilleure compréhension du domaine linguistique, par une montée en compétence des acteurs, soit par l'entrée d'un spécialiste - linguiste - dans l'équipe projet. L'intelligence artificielle étant finalement un outil, celui-ci se doit de répondre aux des besoins utilisateurs pour faciliter leurs requêtes de manière fluide, comme un outil de recherche naturel et invisible.

[31] https://www.lemonde.fr/recherche/

Annexes · Outils

➔ GitHub
Repository du projet
https://github.com/flabastie/news-analysis

➔ Scrapy
Framework open source d'extraction de données
à partir de sites Web.
https://scrapy.org/

➔ Visual Studio Code
Éditeur de code extensible développé par Microsoft.
https://code.visualstudio.com/

➔ Anaconda
Distribution libre et open source pour langages de programmation Python et R appliquée au développement d'applications dédiées à la science des données et à l'apprentissage automatique.
https://www.anaconda.com/products/individual

➔ NLTK Natural Language Toolkit
Plate-forme pour création de programmes Python pour travailler avec des données en langage humain.
http://www.nltk.org/

➔ MySQL Workbench
Logiciel de gestion et d'administration de bases de données MySQ.
https://www.mysql.com/fr/products/workbench/

➔ Microsoft Azure
Plate-forme applicative cloud de Microsoft.
https://azure.microsoft.com/fr-fr/

➔ Flask
Framework open-source de développement web en Python.
https://flask.palletsprojects.com/en/1.1.x/

➔ Bootstrap
Framework CSS pour la création du design de sites et d'applications web.
https://getbootstrap.com/