



UASB03
Projet pour le certificat de spécialisation
Analyste de données massives

RAPPORT

Session 1 - Février 2025

INTRODUCTION.....	4
CONTEXTE.....	4
OBJECTIFS DU PROJET	4
DESCRIPTION DES ÉTAPES DU PROJET	5
SCALABILITÉ.....	5
I. COLLECTE ET STOCKAGE DES DONNÉES.....	7
1. COLLECTE DE DONNÉES D'ARCHIVES D'ARTICLES SCIENTIFIQUES	7
<i>Documents stockés sur la plateforme BioRxiv/MedRxiv</i>	7
<i>Informations complémentaires de publication/citations via OpenCitations.....</i>	7
2. STOCKAGE DES DONNÉES DANS MONGODB.....	7
<i>Stockage des documents collectés</i>	7
<i>Création d'un index et opérations d'agrégations</i>	8
3. RECHERCHES THÉMATIQUES AVEC ELASTICSEARCH	9
<i>Indexation des documents dans ElasticSearch</i>	9
<i>Recherche dans l'index ES et création d'une sous-collection dans MongoDB</i>	9
II. ANALYSE EXPLORATOIRE ET PRÉ-TRAITEMENT	11
1. ENCODAGE DES TEXTES AVEC BIOBERT	11
2. ANALYSE EXPLORATOIRE DES VARIABLES NUMÉRIQUES ET CATÉGORIELLES.....	12
<i>Analyse univariée.....</i>	12
<i>Analyse bivariée et multivariée.....</i>	13
3. PRÉ-TRAITEMENT DES DONNÉES POUR L'APPRENTISSAGE SUPERVISÉ	14
III. APPRENTISSAGE SUPERVISÉ : CLASSIFICATION BINAIRE.....	16
1. RÉGRESSION LOGISTIQUE.....	16
2. SVM LINÉAIRES	17
3. FORÊTS ALÉATOIRES.....	17
4. TEST ET COMPARAISON DES MODÈLES DE CLASSIFICATION	18
IV. APPRENTISSAGE SUPERVISÉ : RÉGRESSION	20
1. MODÈLE LINÉAIRE GÉNÉRALISÉ	20
2. FORÊTS ALÉATOIRES.....	21
3. TEST ET COMPARAISON DES MODÈLES DE RÉGRESSION	21
V. PASSAGE À L'ÉCHELLE	23
1. PROBLÉMATIQUES DU PASSAGE À L'ÉCHELLE.....	23
2. RÉPLICATION ET PARTITIONNEMENT DANS MONGODB	24
3. AGRÉGATIONS DANS MONGODB.....	24
4. ELASTICSEARCH	25
5. SPARK ET MONGODB CONNECTOR	26
CONCLUSION ET PERSPECTIVES.....	27
LIMITATIONS DU PROJET	27
<i>Variables à faible caractère prédictif</i>	27
<i>Représentation sémantique des textes</i>	27
<i>Importance du choix du sujet et objectifs de départ</i>	28
BILAN DU PROJET	28
RÉFÉRENCES	29
ANNEXES.....	30

Introduction

Contexte

La plateforme BioRxiv est une archive de pré-publications dans le domaine des sciences biologiques lancée en 2013 sur le modèle de ArXiv, dédiée aux sciences physiques, mathématiques et informatiques. Elle permet aux chercheurs de déposer et diffuser leurs manuscrits avant leur publication dans des journaux revus par les pairs.

Au cours des dernières années, et plus récemment suite à la pandémie du COVID-19, l'usage de BioRxiv a considérablement augmenté, le nombre de pré-publications déposées sur BioRxiv chaque mois ayant doublé entre 2018 et 2024 [1]. Malgré les préoccupations quant à la qualité du contenu déposé sur ces plateformes [2], leur impact positif sur la publication et la diffusion ultérieure des articles dans des revues scientifiques de qualité est maintenant attestée [3], [4], [5].

Par ailleurs, si la prédiction du nombre de citations et de l'impact des articles publiés est une question déjà largement adressé par des techniques de machine learning [6], [7] ([8] pour une revue sur le sujet), l'usage de méthodes prédictives pour étudier l'avenir des pré-publications dans le domaine biomédical n'a à notre connaissance pas été rapporté. En effet, c'est plutôt l'impact de la pré-publication (ou non) sur le nombre de citations des articles ultérieurs qui a été étudié [4] ou une étude approfondie du devenir des pré-publications BioRxiv [9].

Dans le cadre de ce projet, nous souhaitons donc nous intéresser aux facteurs susceptibles d'impacter la publication de ces pré-publications dans un journal suite à leur dépôt sur la plateforme, et leur futur succès en nombre de citations. Nous aimeraisons pouvoir prédire si une pré-publication sera publiée dans un journal dans l'année suivant son dépôt sur la plateforme, et son nombre de citations lors de la première année de publication dans un journal. Chaque pré-publication sera considérée comme une entité indépendante et nous nous baserons uniquement sur ses caractéristiques intrinsèques. Par caractéristiques intrinsèques, nous entendons l'absence de données concernant le réseau académique des auteurs ainsi que leur précédent succès en termes de nombre de publications, et de citations, ou la réputation du journal dans lequel est publié l'article (impact factor).

Objectifs du projet

Dans le cadre du Certificat d'Analyse de données massives, les objectifs du projet sont multiples :

- Collecte d'une grande quantité de données à partir de plusieurs sources dans une base de données distribuée (NFE204).
- Usage d'un moteur de recherche plein-texte pour sélectionner un sous-ensemble thématique dans le corpus (NFE204).
- Pré-traitement et analyse exploratoire du jeu de données (STA211).
- Encodage des textes avec un modèle adapté (RCP216).
- Construction, test et validation d'un modèle d'apprentissage supervisé pour prédire la publication d'un article dans l'année suivant le dépôt d'une pré-publication (prédicteur binaire, problème de classification) (STA211, RCP216).
- Construction, test et validation d'un modèle d'apprentissage supervisé pour prédire le nombre de citations des articles publiés, durant leur première année de publication (problème de régression) (STA211, RCP216).

Description des étapes du projet

Le schéma suivant (Fig.1) reprend les étapes principales du projet et la table 1 détaille les outils employés et les liens vers les parties correspondantes dans le présent rapport, ainsi que vers les extraits de code présentés en annexe. Les parties I à IV de ce rapport sont dédiées à la description des étapes du projet, tandis que la partie V abordera la problématique du passage à l'échelle.

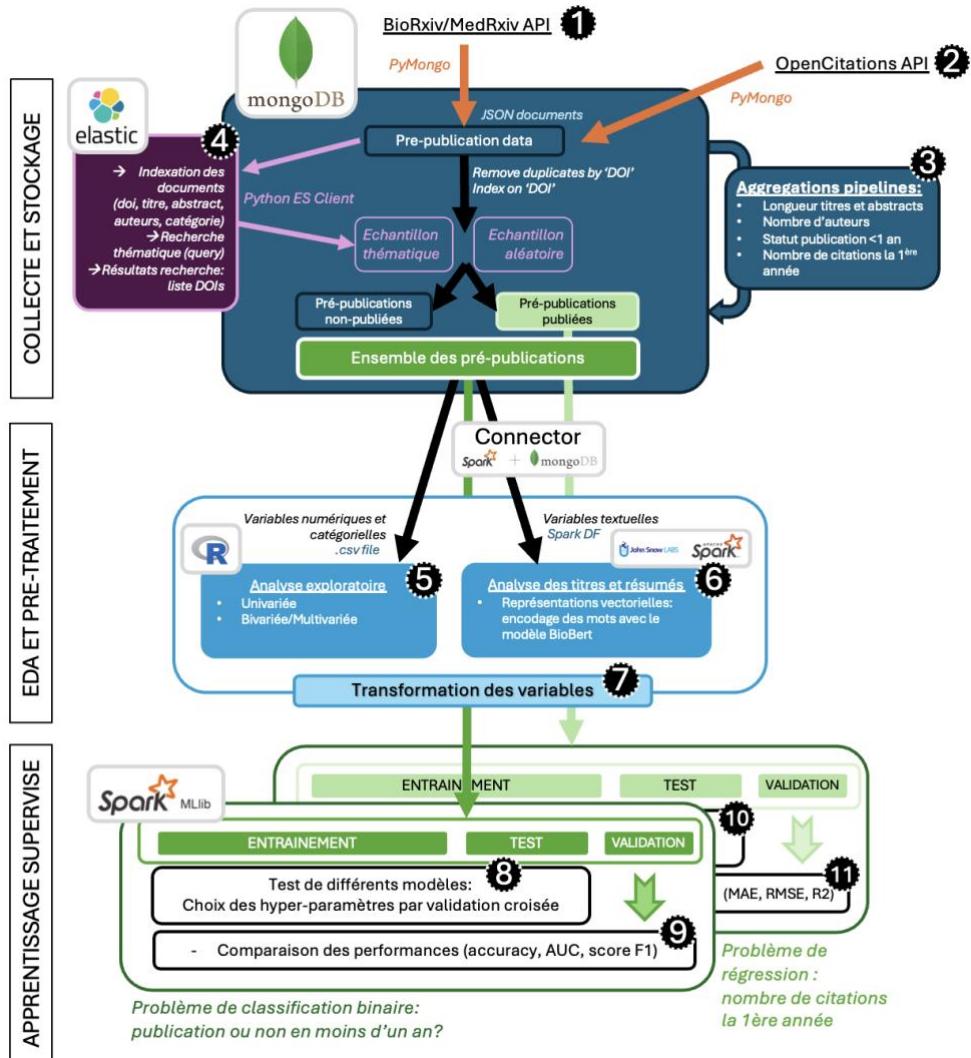
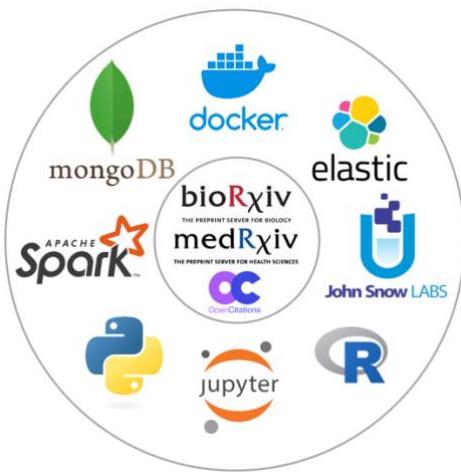


Figure 1: Représentation schématique des différentes étapes du projet.

Scalabilité

Les serveurs de BioRxiv comprennent aujourd'hui près de 500'000 pré-publications, ce qui constitue un ensemble massif de données, qui ne cesse de croître. Malgré la réalisation de ce projet sur une unique machine localement, nous avons intégré des outils et méthodes permettant de réaliser les traitements décrits en prenant en compte les problématiques liées aux caractéristiques des données massives :

- **Volume** : Stockage sur une plateforme distribuée.
- **Variété** : Stockage sous forme de documents JSON puis DataFrames Spark, et traitements spécifiques des variables de types variés (numérique, catégoriel, textuel, vectoriel, etc...).
- **Vélocité** : emploi de méthodes permettant une parallélisation des calculs (Agrégations MongoDB, traitement distribué dans Spark).



Les outils choisis sont présentés dans la figure 2, et peuvent permettre un **passage à l'échelle par distribution** afin de déployer le stockage et traitement sur un cluster de plusieurs machines.

En particulier, les explications liées au **partitionnement** et à la **réPLICATION** des données stockées, l'usage d'algorithmes permettant la **PARALLÉLISATION DES CALCULS**, et l'intégration des différents outils de stockage et de traitement des données, seront abordés dans la dernière partie de ce rapport.

Figure 2: Outils employés pour la réalisation de ce projet.

	Description	Outils employés	Programme / Extrait de code en annexe
1	Collecte et stockage des données des pré-publications via l'API BioRxiv	Python, PyMongo, MongoDB	import_rxiv_program.py Extrait code
2	Collecte des informations de citations avec l'API OpenCitations. Ajout de ces nouvelles variables aux documents ('citations' et 'publiDate')	Python, PyMongo, MongoDB	add_citations.py Extrait code
3	Calcul de nouvelles variables pour chaque document dans MongoDB	Pipelines d'agrégations MongoDB, PyMongo	aggregations_mongodb.py Extrait code
4	Création de collections thématiques par recherche plein-texte	Elastic Search	elastic_search.py Extrait code
5	Export et analyse exploratoire des données (variables numériques et catégorielles)	R	export_collection.py (Extrait) exploratory_data_analysis.R
6	Encodage des titres et résumés des pré-publications avec un modèle Bert	Spark-MongoDB connector, SparkNLP	bioBert_embeddings.py Extrait code
7	Pré-traitement et transformation des variables pour l'apprentissage supervisé	Spark-MongoDB connector	export_collection.py Extrait code
8	Test et ajustement de différents modèles de classification binaire pour déterminer si une pré-publication sera publiée dans un journal dans la première année de son dépôt sur la plateforme d'archive.	Spark MLlib	8_binaryClassif.ipynb Extrait code
9	Comparaison des performances des modèles de classification binaire.	Spark MLlib	9_binaryClassif_ModelComparison.ipynb Extrait code
10	Test et ajustement de différents modèles de régression pour déterminer le nombre de citations dans la première année de publication, pour le sous-ensemble des pré-publications publiée.	Spark MLlib	10_regression.ipynb Extrait code
11	Comparaison des performances des modèles de régression.	Spark MLlib	10_regression.ipynb Extrait code

Table 1: Détail des étapes du projet.

I. Collecte et stockage des données

1. Collecte de données de pré-publications d'articles scientifiques

Documents stockés sur la plateforme BioRxiv/MedRxiv

Un programme a été développé pour récupérer les données sur les pré-publications en utilisant les API mises à disposition par la plateforme BioRxiv [10]. Le programme effectue d'abord une compilation de l'URL avec lequel interroger l'API en fonction de l'intervalle de temps et le serveur (biorxiv ou medriv) spécifié par l'utilisateur. La réponse de l'API permet d'extraire un document JSON pour chaque pré-publication. En raison des restrictions de l'API, la collecte s'effectue par lots de 100 documents. Les données sont donc intégrées au fur et à mesure dans une collection MongoDB à l'aide du package Pymongo (client Python). Un extrait de la fonction utilisée est présenté en [Annexe](#).

Informations complémentaires de publication/citations via OpenCitations

Les documents des pré-publications qui ont ensuite été publiés dans un journal possèdent un champ '*published*' qui contient l'identifiant DOI de l'article publié. Si l'article n'a pas été publié, ce champ est vide. Nous devons collecter des données supplémentaires afin de calculer nos 2 variables cibles :

- Variable '*publiStatus*' : statut de la pré-publication (non publié, publié en 1an, 2an ou plus de 2ans).
- Variable '*cit1year*' : nombre de citations durant la première année de publication.

Les informations de publication nécessaires au calcul de ces variables ne sont pas présentes dans les documents collectés avec l'API BioRxiv, nous devons donc utiliser l'API OpenCitations [11] :

- Date de publication dans le journal
- Citations : Le champ '*timespan*' de chaque citation permet notamment de savoir combien de temps après la publication l'article a été cité.

Pour cela, nous devons effectuer 2 requêtes par document car chacune de ces informations ne peut être accédée que par une URL spécifique de l'API. Ensuite, les informations collectées sont ajoutées aux documents existants dans la collection MongoDB. Un extrait du code utilisé est visible dans [l'annexe 2](#). En pratique, cette étape s'est avérée particulièrement limitante en termes de temps, car il était nécessaire de consulter chaque document de la base itérativement, puis d'effectuer les 2 appels à l'API et la mise à jour dudit document dans la collection.

2. Stockage des données dans MongoDB

Stockage des documents collectés

La base de données NoSQL MongoDB a été choisie pour stocker les documents pour plusieurs raisons :

- **Adapté au format des données** : MongoDB est une base de données documentaire qui stocke les informations en JSON, ce qui correspond parfaitement au format des résumés récupérés via l'API.
- **Scalabilité et résistance aux pannes** : Grâce à son mode distribué, MongoDB permet de gérer efficacement de grands volumes de données, facilitant ainsi le passage à l'échelle. La gestion intégrée de la réplication et du partitionnement garantit la disponibilité et la fiabilité des données, même en cas de panne. Ces aspects seront détaillés en [partie V](#).
- **Interopérabilité** : MongoDB propose un connecteur avec Spark, permettant une intégration fluide avec cet outil de traitement de données distribuées ([Partie V.5](#)).

En prenant un intervalle de temps de 4 ans (2018→2022) nous obtenons un total de 201'259 documents, collectés et stockés dans une collection MongoDB (Table 2). La taille de la collection complète est 529.01MB, avec une taille par document moyenne de 6.29kB.

Au moment de la rédaction de ce rapport, les informations de citation de l'ensemble des prépublications publiées n'ont été collectés que pour environ 102'000 pré-publications (sur les 127'798 publiées).

Source	Intervalle de temps	Nombre de pré-publications collectées	Nombre de pré-publications uniques	Nombre de pré-publications publiées
BioRxiv	01/2018 → 12/2022	221'313	163'180	103'458
MedRxiv	07/2019 → 12/2022	48'693	38'079	24'340
TOTAL		270'006	201'259	127'798

Table 2: Détail du contenu de la collection MongoDB

Création d'un index et opérations d'agrégations

Le DOI (Digital Object Identifier) est un identifiant unique attribué aux publications scientifiques, aux articles et à d'autres ressources numériques. Il fonctionne comme un lien permanent qui garantit l'accès à un document. Dans notre cas précis, chaque pré-publication dispose d'un DOI unique, dans le champ '*doi*', ainsi que d'un autre DOI si l'article a été publié dans un journal (champ '*published*').

Comme nous souhaitons pouvoir identifier les pré-publications sans erreur d'une base de données et d'une collection à l'autre, nous utiliserons ce champ comme identifiant unique des documents. Afin d'accélérer les futures recherches et opérations d'agrégation, **un index unique** doit donc être créé sur ce champ. Pour cela, nous avons d'abord éliminé les duplicates en ne conservant que la version la plus récente (numéro de version le plus élevé) pour chaque DOI.

Nous avons aussi effectué différentes opérations sur les champs des documents de la collection afin de créer de nouvelles variables :

- ‘*titleLength*’ et ‘*abstractLength*’ : longueur du titre et du résumé,
- ‘*numberAuthors*’ : nombre d'auteurs à partir de la liste des auteurs,
- Formattage des dates de pré-publication (‘*date*’) et de publication (‘*publiDate*’),
- ‘*publiStatus*’ : statut de la pré-publication (non publié, publié en 1an, 2 ans, ou plus). Calculé à partir des champs ‘*published*’ et ‘*publiDate*’.
- ‘*citTotal*’, ‘*cit1year*’ et ‘*cit2year*’ : nombre de citations total, la première et la deuxième année de publication, respectivement. Calculé à partir du champ ‘*timespan*’ dans les champs imbriqués de ‘*citations*’. Cela permet d'égaliser entre les articles déposés en 2018 versus ceux déposés en 2022.

Ces opérations pouvant être effectuées localement au niveau de chaque document, cela pourrait correspondre à une tâche de « Map ». Elles ont été réalisées à l'aide de **MongoDB Aggregation Pipeline [12]**, une alternative optimisée au MapReduce (déprécié depuis la version 5.0 de MongoDB).

La fonction pour appliquer ces opérations à tous les documents d'une collection est visible en [Annexe](#). Les détails des pipelines d'agrégation ne sont cependant pas détaillés ici en raison de leur longueur et de leur complexité relative (objets JSON avec de nombreuses indentations), mais elles sont visibles dans le fichier `aggregations_mongodb.py`.

La figure suivante (Fig.3) permet de visualiser un document dans la collection MongoDB, après les opérations d'agrégation permettant d'obtenir les nouveaux champs détaillés plus haut.

The screenshot shows the MongoDB Compass interface with the 'Unique' collection selected. The document details are as follows:

```

_id: ObjectId('678b82e6d38f448bcac5e1dd')
doi: "10.1101/001768"
title: "Characterization of cell-to-cell variation in nuclear transport rates ..."
authors: "Durieu, L.; Bush, A.; Grande, A.; Johansson, R.; Janzen, D. L. I.; Go..."
author_corresponding: "Alejandro Colman-Lerner"
author_corresponding_institution: "IFIByNE, DFBMC, FCEN, UBA, Buenos Aires, Argentine"
date: "2022-06-24"
version: "2"
type: "new results"
license: "cc_by_nc_nd"
category: "systems biology"
jatsxml: "https://www.biorxiv.org/content/early/2022/06/24/001768.source.xml"
abstract: "Nuclear transport is an essential part of eukaryotic cell function. Se..."
published: "10.1016/j.isci.2022.105906"
server: "biorxiv"
citations: Array (2)
  0: Object
    journal_sc: "no"
    cited: "omid:br/06290144513 doi:10.1016/j.isci.2022.105906 openalex:W431321932..."
    oci: "062604215254-06290144513"
    timespan: "P1Y2M"
    author_sc: "no"
    citing: "omid:br/062604215254 doi:10.1016/j.xpro.2024.102876"
    creation: "2024-03"
  1: Object
    cit1year: 1
    cit2years: 0
    citTotal: 2
    dateConverted: 2022-06-24T00:00:00.000+00:00
    numberAuthors: 9
    publDateConverted: null
    publStatus: "publi_nodate"
    abstractLength: 1660
    titleLength: 103

```

Figure 3: Capture d'écran de la collection 'Unique' (après élimination des dupliquats) dans MongoDB Compass.

3. Recherches thématiques avec ElasticSearch

Indexation des documents dans ElasticSearch

Afin de pouvoir effectuer des recherches plein texte dans la collection MongoDB, nous l'avons indexée dans Elastic Search (ES). Pour cela, les champs ‘title’, ‘abstract’, ‘category’, ‘authors’, et ‘corresponding_authors_institutions’, ont été extrait de chaque document MongoDB, et indexés au moyen de l'**API Bulk d’Elastic Search** par lots de 10'000 (extrait du code en [annexe](#), Figure 11).

Recherche dans l’index ES et création d’une sous-collection dans MongoDB

Une fois les documents indexés dans le nouvel index ES, il est possible de faire une requête en envoyant une ‘query’ avec le client Python, tel qu’effectué dans la fonction `search_es_index()` en [annexe](#) et dans le script `elastic_search.py`. La fonction retourne la **liste des identifiants DOI** des documents présents dans les résultats de la requête.

Pour ce projet plusieurs requêtes ont été effectuées :

- Pré-publications en lien avec la maladie d'Alzheimer ou la démence ('dementia')
- Pré-publication mentionnant la technique d'édition du génome CRISPR/Cas9 (Prix Nobel 2020) qui est aujourd'hui énormément employée dans de nombreux domaines de la biologie, et optimisée pour de futures applications médicales.

The screenshot shows the ElasticVue interface with a search query in the Kibana console:

```

1. { "query": {
2.   "bool": {
3.     "should": [
4.       {"match_phrase": {"abstract": "CRISPR"}},
5.       {"match_phrase": {"abstract": "CRISPR-Cas9"}},
6.       {"match_phrase": {"abstract": "gene editing"}},
7.       {"match_phrase": {"abstract": "base editing"}},
8.       {"match_phrase": {"abstract": "prime editing"}},
9.       {"match_phrase": {"abstract": "genetic engineering"}}
10.    ]
11.  }
12. }
13. }
14. }

```

The response shows a search result for the term 'CRISPR' across 21 documents, with one hit found:

```

1. {
2.   "took": 21,
3.   "timed_out": false,
4.   "_shards": {
5.     "total": 1,
6.     "successful": 1,
7.     "skipped": 0,
8.     "failed": 0
9.   },
10.  "hits": {
11.    "total": 1,
12.    "value": 4242,
13.    "relation": "eq",
14.    "max_score": 59.5383,
15.    "hits": [
16.      {
17.        "_index": "usb03_unique_full",
18.        "_id": "10.1101/2021.09.27.461852",
19.        "_score": 59.5383,
20.        "_ignored": [
21.          "abstract.keyword"
22.        ],
23.        "_source": {
24.          "title": "Plant-based biosensors for detecting CRISPR-mediated genome",
25.          "abstract": "CRISPR/Cas has recently emerged as the most reliable sys"
26.        }
27.      }
28.    ]
29.  }
30. }

```

Figure 4: Capture d'écran de la recherche CRISPR effectuée via l'interface REST dans ElasticVue.

Les résultats de ces requêtes ont ensuite permis de créer la liste des DOI des documents retournés, utilisée pour créer des sous collections thématiques dans MongoDB (fonction `results_collection_mongoDB()`, en [annexe](#)). Dans cette dernière fonction, l'usage de l'index unique sur le champ DOI est particulièrement utile pour trouver rapidement les documents présents dans la liste retournée par la fonction de recherche précédente. Un schéma intégrant ce processus dans une architecture distribuée avec MongoDB et Elastic Search déployés sur un cluster est présenté en [figure 11](#).

Pour la suite du projet, **3 échantillons** issus de la collection MongoDB seront utilisés : un échantillon issu d'un tirage aléatoire de 10'000 documents dans la collection, et 2 échantillons thématiques issus de la recherche avec ElasticSearch (Table 3).

<i>Échantillon Aléatoire</i>	<i>Échantillon 'CRISPR'</i>	<i>Échantillon 'Alzheimer'</i>
<i>Nombre total articles</i>	10 000	4313

Table 3: Nombre d'articles dans les échantillons utilisés pour le projet.

II. Analyse exploratoire et pré-traitement

1. Encodage des textes avec BioBert

Afin d'effectuer le traitement des champs textuels des documents, nous avons utilisé le **Spark-MongoDB connector** pour traiter les documents de la base sous forme de DataFrames Spark, et le package **SparkNLP** pour le traitement du texte.

Premièrement, nous avons regroupé les titres et résumés des pré-publications dans un champ textuel unique '*merged*' ([annexe 6](#)).

Le traitement appliqué vise à transformer les titres et résumés scientifiques en représentations numériques exploitables pour l'apprentissage supervisé. Tout d'abord, les textes sont assemblés et segmentés en phrases (SentenceDetector), avant d'être tokenisés (Tokenizer) et normalisés (Normalizer) afin d'éliminer les caractères non pertinents. Ensuite, le modèle BioBERT, pré-entraîné sur un corpus d'articles scientifiques dans le domaine biologique et médical ([13], [14]), est utilisé pour générer des encodages des mots du texte (*embeddings*), sous forme de vecteurs de taille 768. Pour obtenir une représentation globale des phrases, une moyenne des vecteurs des mots est calculée (SentenceEmbeddings), permettant ainsi d'extraire des caractéristiques sémantiques moyennes des phrases. Ce pipeline permet donc de capturer des informations sémantiques riches et adaptées au domaine biomédical, avec pour objectif d'optimiser l'analyse et l'exploitation des résumés scientifiques dans la suite du projet.

```
# Assemblage du document
documentAssembler = DocumentAssembler() \
    .setCleanupMode("inplace") \
    .setInputCol("merged") \
    .setOutputCol("document")

# Détection des phrases
sentence = SentenceDetector() \
    .setInputCols(["document"]) \
    .setOutputCol("sentence")

# Tokenisation
tokenizer = Tokenizer() \
    .setInputCols(['document']) \
    .setOutputCol('token')

# Normalisation
normalizer = Normalizer() \
    .setInputCols(["token"]) \
    .setOutputCol("normalized") \
    .setCleanupPatterns(["""[^a-zA-Z0-9]"""])

# Encodage avec un modèle BERT pré-entraîné sur des articles scientifiques
embeddings = BertEmbeddings.pretrained("biomednlp_biomedbert_base_uncased_abstract_fulltext", "en") \
    .setInputCols(["document", "normalized"]) \
    .setOutputCol("biobert_embeddings")

# Encodage des phrases (moyenne des vecteurs de mots)
embeddingsSentence = SentenceEmbeddings() \
    .setInputCols(["sentence", "biobert_embeddings"]) \
    .setOutputCol("sentence_embeddings") \
    .setPoolingStrategy("AVERAGE")

# Récupération des vecteurs de phrases
finisher = EmbeddingsFinisher() \
    .setInputCols("sentence_embeddings") \
    .setOutputCols("output") \
    .setOutputAsVector(True) \
    .setCleanAnnotations(False)

# Pipeline
pipeline_biobert = Pipeline(stages=[\
    documentAssembler, \
    sentence, \
    tokenizer, \
    normalizer, \
    embeddings, \
    embeddingsSentence, \
    finisher \
])

# Application de la pipeline
model_biobert = pipeline_biobert.fit(df_select)
result_biobert = model.biobert.transform(df_select)
```

Table 4: Pipeline de traitement du texte avec SparkNLP.

Pour cette étape d'encodage des textes, nous avons aussi envisagé l'usage de la modélisation des sujets par Latent Dirichlet Allocation (LDA), après un encodage TF (TermFrequency) des mots du corpus. Cependant, il a semblé que la représentation vectorielle des sujets des documents par LDA pouvait être redondante avec le champ '*category*' qui décrit déjà la catégorie à laquelle appartient la publication. Nous avons aussi envisagé de n'utiliser que les 100 premières composantes du vecteur moyen - après réduction par analyse en composante principale (ACP) - pour l'apprentissage supervisé. Une telle projection explique presque 80% de la variance de l'échantillon ([annexe 7](#)), et comme notre cas elle donnait des résultats d'apprentissage encore inférieurs à ceux obtenus avec la représentation originale en 768 dimensions, elle a été écartée.

2. Analyse exploratoire des variables numériques et catégorielles

Avant de mettre au point les modèles d'apprentissage supervisés, il est essentiel d'étudier la distribution des variables originales afin d'ajuster les étapes pré-traitement de ces variables. Ces étapes d'analyse et de pré-traitement ont été réalisées de façon similaire sur les 3 échantillons. Seuls les résultats pour l'échantillon aléatoire sont présentés ici, les observations sur les autres échantillons étant relativement similaires. Seulement la distribution des catégories diffère notablement, à cause de la sélection thématique effectuée. Le recodage a donc été effectué légèrement différemment pour assurer l'absence de catégories avec moins de 3% des documents. Les observations effectuées sur ces échantillons ('CRISPR' et 'Alzheimer') sont détaillées en [annexes 13 et 14](#).

Analyse univariée

Dans un premier temps nous effectuons l'analyse univariée des variables que nous souhaitons utiliser pour la création des deux modèles d'apprentissage supervisé (Table 7). Les graphes supplémentaires sont visibles en [annexe 9](#).

Concernant les variables numériques :

- 'abstractLength' : un document avec un grand outlier pour cette variable a été écarté (q-q plot en annexe).
- Contrairement à 'titleLength' et 'abstractLength' (histogrammes en annexe), la variable 'numberAuthors' nécessite une transformation logarithmique pour normaliser sa distribution (q-q plots en annexe).

La distribution des variables catégorielles permet aussi d'effectuer quelques observations et transformations :

- La variable 'version' possède la valeur 1 dans +99% des documents, elle n'apportera pas beaucoup pour l'apprentissage des modèles, on choisit donc de la supprimer par la suite (diagramme en barre en annexe).
- Le ratio entre les 'servers' biorxiv-medrxiv est d'environ 78/22 % (diagramme en barres en annexe)
- La date de dépôt sert à créer une nouvelle variable 'month_upload' afin d'explorer si le mois de dépôt de la pré-publication peut avoir un impact sur sa publication dans l'année (voir distribution en annexe),
- Les 75 catégories initiales ont été recodées en 18 catégories, en regroupant les catégories les plus proches, afin d'éliminer les catégories aux effectifs trop faibles (détails dans le tableau [en annexe 10](#)).

Enfin, la variable réponse 'publiStatus' montre que la moitié des articles (environ 52%) sont publiés dans la première année suivant leur dépôt (Fig.5, gauche). Cette variable est recodée en binaire pour l'apprentissage supervisé, en regroupant les autres catégories (Fig.5, droite).

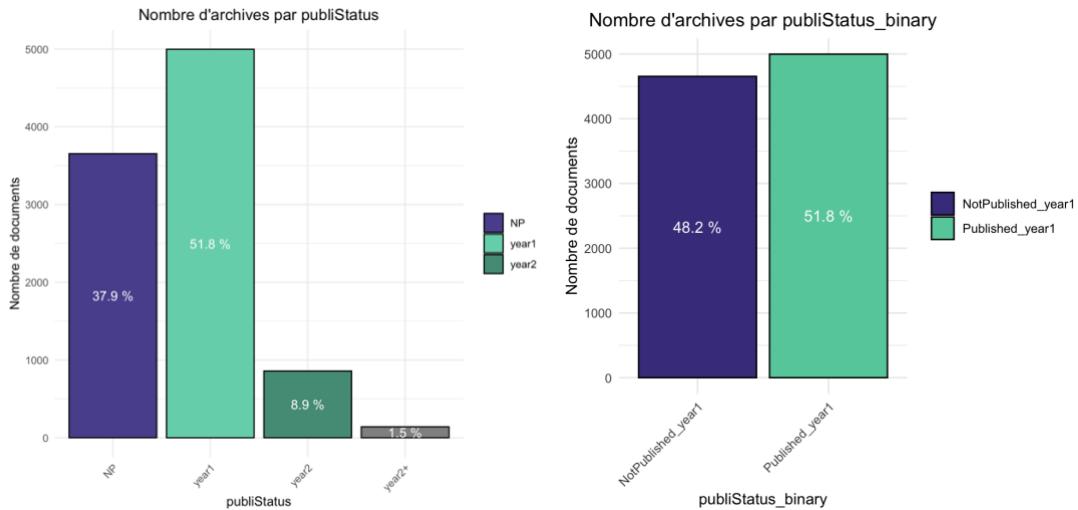


Figure 5: Distribution des documents de l'échantillon aléatoire selon leur statut de publication (gauche) et après recodage en 2 catégories (gauche).

Analyse bivariée et multivariée

La corrélation entre les variables numériques a été étudiée (matrice des corrélations en [annexe 11](#)). Si la légère corrélation entre la longueur du titre et du résumé peut être attendue, on note également une légère corrélation entre le nombre de citations la première année et le nombre d'auteurs. Cette corrélation a d'ailleurs déjà été notée dans d'autres ensembles d'articles scientifiques, et déjà utilisée pour des modèles de prédiction ([8], section 4.1 « Number of authors »).

> chi2_all(df_clean, CatVar)				
	category_grouped	server	publiStatus_binary	month_upload
category_grouped		NA	2.769979e-06	0.100590156
server		NA	1.864732e-03	0.037876266
publiStatus_binary		NA	NA	0.002553456
month_upload		NA	NA	NA

Table 5: Tests d'indépendances entre les variables catégorielles de l'échantillon aléatoire.

Ensuite, l'indépendance entre les différentes variables catégorielles a été étudié à l'aide du test du khi-deux (Table 5). Les p-values obtenues indiquent la significativité des relations observées. Par exemple, des valeurs significatives sont obtenues entre ‘category_grouped’ et ‘publiStatus_binary’ ($p = 2.77e-06$, voir Fig.7) ainsi qu'entre ‘publiStatus_binary’ et ‘month_upload’ ($p = 0.002$, figure en [annexe 11](#)), suggérant une dépendance entre ces variables.

Enfin, concernant l'indépendance des variables catégorielles et numériques, nous avons utilisé le test non-paramétrique de Kruskal-Wallis. Comme pour le nombre de citations la première année, on constate une moyenne légèrement supérieure du nombre d'auteurs parmi les articles publiés la première année (Fig.6, $p=0.002$). Au contraire du statut de publication, il faut noter une absence de différence dans le nombre de citations la première année en fonction de la catégorie de la pré-publication (Table 6 - bas, Fig.7).

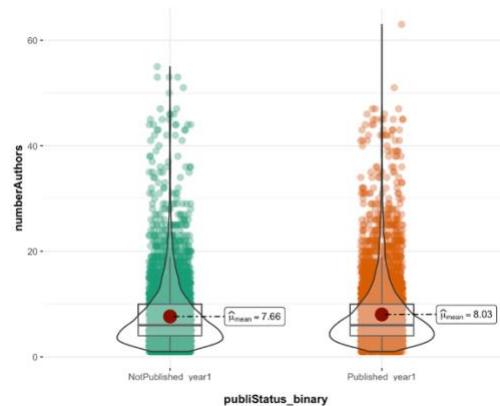


Figure 6: Moyenne du nombre d'auteurs pour les articles publiés ou non dans leur première année suivant leur dépôt sur la plateforme BioRxiv.

```
Kruskal-Wallis rank sum test
data: publiStatus_binary by numberAuthors
Kruskal-Wallis chi-squared = 84.71, df = 51, p-value = 0.002107
Kruskal-Wallis rank sum test
data: category_grouped by cit1year
Kruskal-Wallis chi-squared = 155.06, df = 143, p-value = 0.2318
```

Table 6: Résultats de tests du Kruskal-Wallis pour le status de publication en fonction du nombre d'auteurs (haut) et le nombre de citations la première année en fonction de la catégorie.

catégorie dans la figure 7. Nous avons représenté leur part dans notre échantillon aléatoire (gauche), la proportion publiée lors de la première année de dépôt (centre), et le nombre de citations parmi les pré-publications publiées (droite). On peut noter le nombre relativement plus élevé de citations pour les articles de la catégorie ‘maladies infectieuses’ (*InfectDis*), cela s’explique probablement par l’inclusion dans notre intervalle (2018-2022) d’articles concernant le COVID-19 et pour lesquels la plateforme BioRxiv/MedRxiv a été très employée.

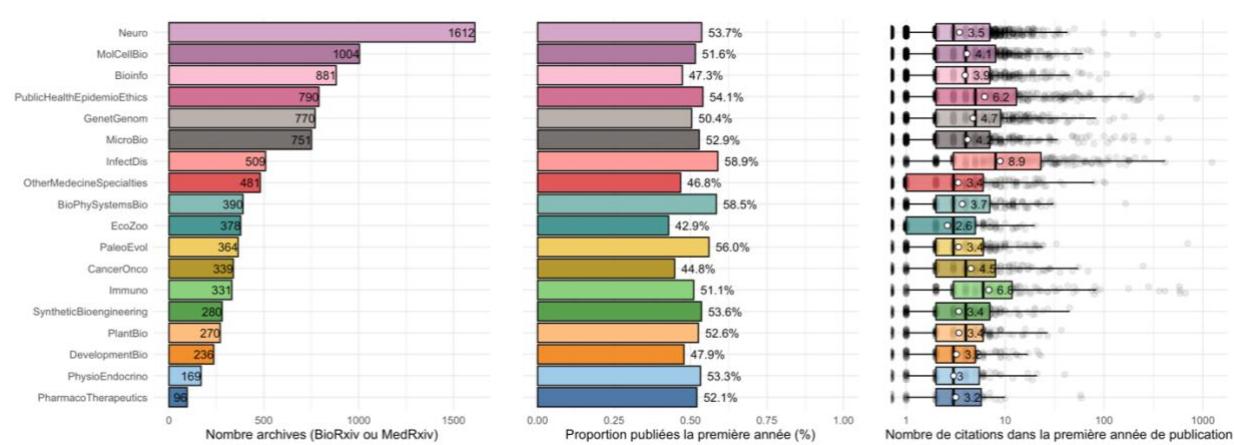


Figure 7: Nombre de pré-publications, proportion publiée la première année et nombre de citations en fonction de la catégorie. Il faut noter l’usage d’une échelle logarithmique sur le graphe de droite.

Une **Analyse Factorielle des Données Mixtes (AFDM)** a été réalisée afin d’explorer les relations entre les variables catégorielles et numériques et de mieux comprendre la structure des données (principaux graphiques en [annexe 12](#)). Les premières dimensions capturent une proportion relativement faible de la variance totale : 6,2% pour la première dimension et 4% pour la deuxième. Une légère colinéarité des variables numériques est observée dans les deux premières dimensions, ce qui traduit une certaine redondance entre certaines variables, notamment entre ‘titleLength’ et ‘abstractLength’. Enfin, la dimension 1 est principalement expliquée par la variable ‘server’, au contraire de la variable cible ‘publiStatus’, qui ne contribue pas à la variance dans les 10 premières dimensions.

3. Pré-traitement des données pour l’apprentissage supervisé

Les observations effectuées au cours de l’analyse exploratoire permettent d’effectuer un prétraitement minutieux dans le but d’optimiser les performances des modèles d’apprentissage supervisés.

Enfin, comme cela a été effectué dans une étude récente [9], on a voulu représenter graphiquement le ‘succès’ des pré-publications en fonction de leur

Nous avons fait le choix d'effectuer le même pré-traitement des données pour pouvoir utiliser et comparer différents types modèles (détails dans la table 7). Le prétraitement comprend notamment :

- Recodage des modalités des catégories en variables numériques (*One-Hot encoding*).
- Normalisation des variables numériques.

En effet, différents types de modèles peuvent être sensibles à des biais ou caractéristiques de la distribution des données :

- **Outliers** : les valeurs aberrantes peuvent perturber les performances de certains modèles, comme la **régression logistique** ou les **support vector machines (SVM)**.
- **Catégories peu représentées** : les variables catégorielles avec des modalités rares peuvent introduire du bruit. Les **forêts aléatoires** sont généralement robustes aux déséquilibres de classes, mais les **SVM** peuvent y être sensibles (cela correspond à outliers quand ils sont recodés numériquement).
- **Distribution des variables numériques** : encore une fois, si les forêts aléatoires sont plutôt robustes, les **SVM** linéaires fonctionnent mieux sur des données normalisées dans un intervalle donné ([0;1] par exemple).

De plus, l'étude de la distribution et de la corrélation des variables au cours de l'analyse exploratoire a montré des relations plutôt faibles entre nos variables cibles ('*publiStatus*' et '*citIyear*'), et les différentes variables numériques et catégorielles à notre disposition pour décrire les documents. Comme nous le verrons, cela ne sera pas sans impact sur la qualité des prédictions effectuées par les modèles d'apprentissage supervisé.

VARIABLE	DESCRIPTION	TYPE 1	PRE-TRAITEMENT	TYPE 2
<i>version</i>	Version de l'archive	Numérique	Suppression	/
<i>numberAuthors</i>	Nombre d'auteurs	Numérique	Log, scaling	Numérique
<i>abstractLength</i>	Longueur abstract	Numérique	Suppressionβ outlier, Scaling	Numérique
<i>titleLength</i>	Longueur titre	Numérique	Scaling	Numérique
<i>date</i>	Date de dépôt	Date	Recodage en mois de dépôt, One-hot encoding	Vecteur 12[]
<i>server</i>	Server	Catégoriel (2)	One-hot encoding	Binaire
<i>category</i>	Catégorie de l'article	Catégoriel (75)	Recodage en 18 catégories groupées (annexe 10), One-hot encoding	Vecteur 17[]
<i>author_corresponding_institution</i>	Institution de l'auteur principal (« corresponding author »)	Catégoriel (+500)	Suppression car trop difficile à recoder.	/
<i>BioBertEmbedding</i>	Représentation vectorielle TITRE + ABSTRACT	Array (768)	Transformation en vecteur	Vecteur 768[]
FEATURES				Vecteur 801[]
<i>publiStatus</i>	Statut de la publication	Catégoriel (4)	Recodage en catégorie binaire : publication en moins d'un an ou non	Binaire
<i>citIyear</i>	Nombre de citations la 1 ^{ère} année	Numérique	/	Numérique

Table 7: Variables et transformations.

III. Apprentissage supervisé : classification binaire

Notre premier problème est un problème de classification binaire qui consiste à prédire la valeur de la variable binaire ‘*publiStatus_binary*’ - à savoir si une prépublication est publiée dans la première année suite à son dépôt sur la plateforme BioRxiv/MedRxiv. Pour cela, nous allons utiliser différentes méthodes d'apprentissage supervisé qui peuvent prendre en entrée un grand nombre de variables numériques comme variables explicatives. Nous avons choisi de tester 3 méthodes implémentées dans Spark : la régression logistique, les machines vecteurs de support linéaires (SVM linéaires) et les forêts aléatoires.

Premièrement l'échantillon de données prétraitées est séparé en un **échantillon d'apprentissage** (80%) et dans un **échantillon de test** (20%). L'échantillon d'apprentissage est utilisé pour ajuster les hyperparamètres de chaque méthode tandis que l'échantillon de test sera employé pour comparer les performances des méthodes entre elles. En effet, la performance de chaque modèle dépend du choix des hyperparamètres qui lui sont spécifiques.

Afin d'optimiser ce choix, nous utilisons une recherche en grille (*grid-search*), explorant systématiquement toutes les combinaisons de valeurs. Chaque modèle est ensuite évalué par validation croisée *k-fold*, où l'échantillon d'apprentissage est divisé en k sous-ensembles : le modèle est entraîné sur *k*-1 blocs et testé sur le dernier, l'opération étant répétée *k* fois. L'erreur moyenne obtenue fournit une estimation plus fiable de l'erreur de test. Par souci d'efficacité, nous fixons ici *k*=5 pour limiter le temps de calcul. Pour la classification binaire, nous utiliserons différentes métriques pour comparer les performances, détaillées dans la partie [III.4](#).

Enfin, les résultats ci-après concernent l'échantillon aléatoire, et permettent de mettre au point des prédicteurs généralistes. Des extraits du code sont présentés en [annexe 16](#). Des prédicteurs plus spécialisés entraînés sur les échantillons thématiques ont aussi été testés, et sont mentionnés à la fin de cette partie et en [annexe 18 et 19](#).

1. Régression logistique

La régression logistique est un modèle largement utilisé pour prédire une variable catégorielle. En tant que cas particulier des modèles linéaires généralisés (GLM), elle estime la probabilité d'appartenance à une classe. Ici, il s'agit de l'implémentation ‘binomiale’ pour la prédiction d'une classe binaire. La régression logistique repose sur la fonction sigmoïde, qui transforme une combinaison linéaire des variables explicatives en une probabilité comprise entre 0 et 1 d'appartenir à la classe de sortie (variable cible). C'est parce qu'elle repose sur une combinaison linéaire des variables explicatives qu'elle est sensible aux outliers, car des observations extrêmes peuvent fausser l'apprentissage en donnant trop d'importance à certaines observations.

Dans Spark, cette méthode est régularisée pour éviter le sur-apprentissage et améliorer la généralisation, et cette régularisation est finement ajustée par le réglage des hyperparamètres (Table 8).

RegParam	0.5	Paramètre de régularisation (L2 Ridge car ElasticNet=0.0) .	Réduit le sur-apprentissage en pénalisant les grands coefficients, améliorant ainsi la généralisation.
ElasticNet	0.0	Poids entre régularisation L1 (lasso) et L2 (ridge). Ici, 0 signifie uniquement L2.	Applique une régularisation Ridge, limitant l'influence des valeurs extrêmes sans effectuer de sélection de variables.
MaxIter	25	Nombre maximal d'itérations.	Une valeur trop basse peut empêcher la convergence, une valeur trop élevée peut ralentir l'entraînement.

Table 8: Hyperparamètres utilisés pour le modèle de régression logistique.

2. SVM linéaires

Les Machines Vecteurs de Support (SVM) sont des modèles supervisés qui construisent un hyperplan ou un ensemble d'hyperplans dans un espace de grande dimension afin de réaliser des tâches de classification ou de régression. Un bon hyperplan maximise la marge entre les classes, car une plus grande marge réduit généralement l'erreur de classement. Lorsque les données ne sont pas séparables dans l'espace d'origine, elles peuvent être projetées dans un espace de dimension supérieure pour rendre la séparation possible.

Dans Spark, l'implémentation *LinearSVC* permet d'effectuer une classification binaire avec les SVM linéaires, qui sont parallélisables. Ce modèle optimise la fonction de perte *Hinge Loss* par descente de sous-gradient. Comme le modèle repose sur des distances pour déterminer l'hyperplan optimal, il est essentiel que les variables explicatives soient normalisées afin d'éviter qu'une caractéristique à grande variance ne domine les autres et fausse l'optimisation de la marge. C'est pourquoi nous avons normalisé au préalable les variables numériques telles que la longueur des titres ou des résumés (à l'échelle du millier de caractères).

Les seuls hyperparamètres à ajuster pour les SVM sont le nombre d'itérations et la constante de régularisation (*RegParam*):

RegParam	0.5	Paramètre de régularisation C – inversement proportionnel à l'intensité de régularisation.	Contrôle le compromis entre minimisation de l'erreur d'apprentissage et maximisation de la marge au cours de l'optimisation. Une valeur plus élevée peut augmenter le sur-apprentissage.
MaxIter	15	Nombre maximal d'itérations.	Une valeur trop basse peut empêcher la convergence, une valeur trop élevée peut ralentir l'entraînement.

Table 9: Hyperparamètres utilisés pour le modèle des SVM linéaires.

3. Forêts aléatoires

Les forêts aléatoires (random forests), introduites en 2001, consistent à générer plusieurs arbres de décision différents à partir de sous-ensembles d'observations et de variables. Les prédictions des différents arbres sont ensuite agrégées pour obtenir une prédiction finale. Cette méthode est une variante du *bagging*, et les arbres agrégés ne sont pas élagués.

De plus, contrairement à un simple arbre de décision, les forêts aléatoires introduisent une étape supplémentaire de randomisation dans la sélection des variables explicatives lors de la construction de chaque arbre, ce qui permet de réduire la variance du modèle agrégé. Cette randomisation empêche les variables avec le plus fort pouvoir discriminant de dominer la scission des nœuds, ce qui diminue la corrélation entre les arbres et améliore la capacité de généralisation du modèle. Bien que les forêts aléatoires offrent d'excellentes performances en termes de qualité de prédiction, et une robustesse aux outliers, ils restent cependant plus difficiles à interpréter que des arbres individuels.

Étant donné que des sous-ensembles de variables sont sélectionnés à chaque itération pour construire les arbres, il est possible d'évaluer l'importance des variables explicatives dans la création du modèle avec la fonction `featureImportances`. Cependant, dans notre cas, aucune variable ne se distingue particulièrement, probablement en raison de la faible performance du modèle.

NumTrees	125	Nombre d'arbres à agréger.	Un nombre plus élevé d'arbres tend à améliorer les performances en réduisant la variance et en rendant le modèle plus robuste, bien que cela augmente le temps d'entraînement.
MaxDepth	5	Profondeur maximale de l'arbre. Cela limite la complexité de l'arbre en contrôlant sa hauteur.	Une profondeur modérée évite le sur-ajustement, et qu'une profondeur élevée augmente la complexité du modèle.
MinInstancesPerNode	10	Nombre minimum d'observations nécessaires pour diviser un nœud. Ce paramètre permet de contrôler la taille de l'arbre en limitant sa croissance.	Limiter le nombre d'observations dans un nœud aide à éviter le sur-ajustement en empêchant la création de nœuds trop spécifiques, mais peut également réduire la précision.
FeatureSubsetStrategy	'sqrt'	Stratégie de sélection des prédicteurs utilisés pour la scission de chaque noeud. Nous avons aussi testé 'Log2' comme alternative.	

Table 10: Hyper-paramètres utilisés pour le modèle des forêts aléatoires.

4. Test et comparaison des modèles de classification

Après avoir déterminé les meilleurs paramètres pour chacun des modèles sur l'échantillon d'apprentissage au moyen de la validation croisée (5-fold), nous avons comparé les performances de ces modèles sur l'échantillon de test. Pour cela, les meilleurs modèles ont été entraînés sur l'intégralité des données de l'échantillon d'apprentissage, puis la qualité des prédictions sur l'échantillon d'apprentissage et sur l'échantillon de test, a été estimée à l'aide de 3 mesures (extrait du code en [annexe 17](#)). Ces mesures sont :

- **Accuracy** : Mesure la proportion de prédictions correctes par rapport au nombre total d'observations (taux de vrais positifs+vrais négatifs sur l'ensemble des observations). Elle varie entre 0 et 1, où 1 indique un modèle parfait.
- **Score F1** : C'est la moyenne harmonique entre la précision (la proportion de vrais positifs sur l'ensemble des prédictions positives) et le rappel (la capacité du modèle à détecter les instances positives, le taux de vrais positifs sur la somme des vrais positifs et faux négatifs).
- **AUC (Area Under the Curve)** : Cette métrique évalue la capacité du modèle à distinguer entre les classes positives et négatives. Elle correspond à l'aire sous la courbe **ROC (Receiver Operating Characteristic)** tracée avec le taux de faux positifs en abscisse et le taux de vrais positifs en ordonnée (fig.8). L'AUC varie entre 0 et 1, où 1 représente une discrimination parfaite. Elle est utilisée par défaut dans Spark pour le choix des hyperparamètres dans une classification binaire.

	Ech. Apprentissage			Ech. Test		
	Accuracy	F1	AUC	Accuracy	F1	AUC
LogisticRegression	0.615	0.612	0.658	0.567	0.565	0.590
RandomForest	0.711	0.702	0.810	0.564	0.549	0.583
LinearSVM	0.617	0.615	0.660	0.566	0.563	0.591

Table 11: Comparaison des performances des différents modèles pour le problème de classification binaire.

Les résultats présentés dans le tableau 11 et les courbes ROC (fig. 8) montrent des performances relativement faibles pour les trois modèles testés (Régression Logistique, Forêt Aléatoire, SVM Linéaire), tant sur l'échantillon d'apprentissage que sur l'échantillon de test.

- **Régression logistique** : Le modèle présente des scores d'Accuracy, de F1 et d'AUC assez faibles, avec une légère dégradation sur l'échantillon de test (un prédicteur aléatoire/mauvais prédicteur a une valeur AUC=0.5).
- **Forêts aléatoires** : Bien que ce modèle montre une performance légèrement meilleure sur l'échantillon d'apprentissage avec des scores autour de 0.71 pour l'Accuracy et 0.81 pour l'AUC, la dégradation sur l'échantillon de test (Accuracy de 0.56, F1 de 0.55, AUC de 0.58) indique également un faible pouvoir prédictif sur les données de test. De plus, cela suggère que le modèle souffre de sur-apprentissage malgré l'ajustement des hyperparamètres.
- **SVM linéaires** : Ce modèle a des résultats similaires à ceux de la régression logistique, avec une Accuracy autour de 0.57 et une AUC de 0.59 sur l'échantillon de test, indiquant aussi une capacité de prédiction limitée.

En résumé, les résultats montrent que les modèles, bien que différents dans leurs approches, peinent à prédire correctement et à bien généraliser sur l'échantillon de test. Le faible pouvoir prédictif des modèles suggère que les variables explicatives utilisées dans cette analyse ne sont pas suffisamment discriminantes pour résoudre ce problème de classification binaire de manière efficace, comme suspecté lors de l'analyse exploratoire.

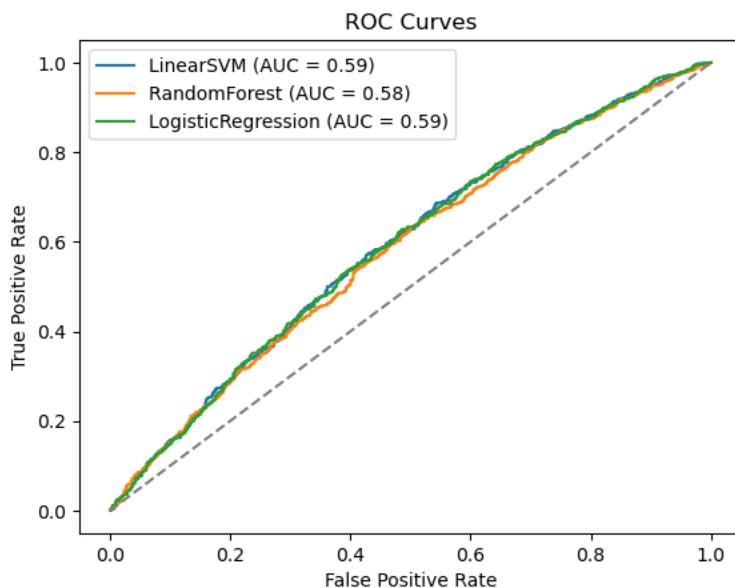


Figure 8: Courbes ROC obtenues pour l'échantillon de test, avec les trois modèles de classification binaire testés.

Enfin, les résultats obtenus avec des modèles ajustés et entraînés sur les échantillons thématiques ‘CRISPR’ et ‘Alzheimer’ montrent des performances similaires (faibles) et souffrent probablement aussi de leur taille encore plus réduite (moins de 5000 observations). Le tableau des performances des modèles testés, et les courbes ROC sont visibles en [annexes 18 et 19](#), respectivement.

IV. Apprentissage supervisé : régression

Notre problème de régression consiste à prédire le nombre de citations d'une pré-publication durant sa première année de publication. Nous devons donc sélectionner le sous-ensemble des pré-publications qui ont été publiés, et nous intéresser à la variable ‘*cit1year*’. Dans notre échantillon aléatoire cela correspond à 5997 documents. Comme visible dans la figure 9, la variable cible possède une distribution très déséquilibrée (dite *skewed*), qui rend l'utilisation de certains modèles, tels que la régression linéaire, inappropriée. En effet, la régression linéaire suppose que la relation entre la variable dépendante et les variables indépendantes suit une distribution normale, ce qui n'est pas le cas ici. De plus, la présence d'un grand nombre de valeurs égales à zéro accentue cette asymétrie et rend les modèles sensibles aux erreurs de prédiction.

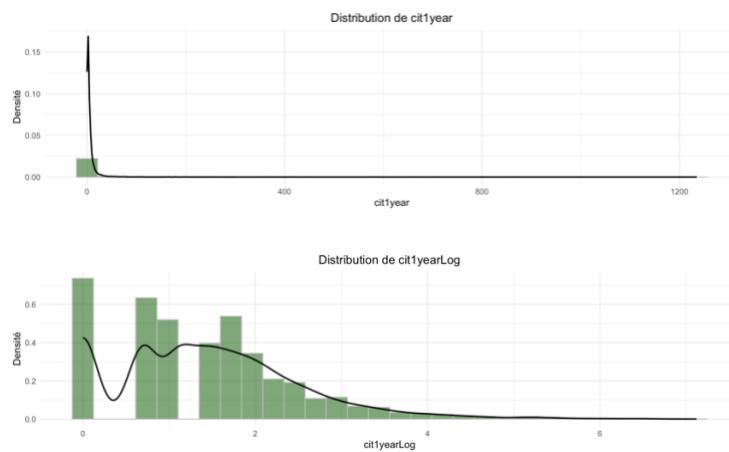


Figure 9: Distribution de la variable *cit1year* originale (haut), et après transformation logarithmique (bas).

Afin de traiter ce déséquilibre, nous avons testé une transformation logarithmique de la variable cible pour rendre sa distribution plus proche d'une distribution normale. Toutefois, même après cette transformation, les performances du modèle sont restées similaires, ce qui suggère que la distribution déséquilibrée persiste.

Nous avons choisi de tester 2 types de modèles implémentés dans Spark, un modèle linéaire généralisé ainsi que les forêts aléatoires.

Pour ajuster les paramètres des modèles et tester leurs performances, nous avons effectué le même traitement que pour la classification binaire : séparation en échantillons d'apprentissage/de test (80/20) et choix des hyperparamètres par validation croisée (*5-fold*). Pour la validation croisée, nous avons choisi d'utiliser l'erreur absolue moyenne (MAE) plutôt que l'erreur quadratique moyenne (RMSE), car la MAE est moins sensible aux valeurs aberrantes.

Enfin, il faut noter que les modèles de régression ont été uniquement testés sur l'échantillon aléatoire, et non sur les échantillons thématiques.

1. Modèle linéaire généralisé

Le modèle linéaire généralisé (GLM) [15] a été choisi en raison de sa flexibilité dans la gestion de différentes distributions de la variable cible, et en particulier pour sa capacité à modéliser des relations non linéaires via des fonctions de lien adaptées. Cependant, cette approche peut encore être sensible à l'asymétrie de la distribution, notamment en présence de valeurs extrêmes. Ici, nous avons opté pour la famille **Tweedie**, car elle est adaptée pour modéliser des données à la fois continues et qui contiennent de nombreux zéros (distributions de données dites *zero-inflated*). Cette famille repose sur la fonction de lien de puissance (*Power link*) [15] qui permet de mieux gérer la relation non linéaire entre la variable indépendante et la variable cible. Cela semble particulièrement pertinent au vu de la distribution des variables explicatives des pré-publications et de la variable ‘*cit1year*’. Les hyperparamètres choisis sont visibles dans la table 12.

Family	'tweedie'	Cette famille est adaptée aux données zero-inflated et aux variables continues. La fonction de lien est PowerLink par défaut pour cette famille de modèles.	
RegParam	0.001	Paramètre de régularisation. Il contrôle la pénalité appliquée aux coefficients du modèle pour éviter le sur-apprentissage.	Une valeur trop faible peut entraîner un sur-apprentissage tandis qu'une valeur trop élevée peut conduire à un modèle trop simple.
MaxIter	50	Nombre maximal d'itérations.	Une valeur trop basse peut empêcher la convergence, une valeur trop élevée peut ralentir l'entraînement.

Table 12: Hyperparamètres sélectionnés pour le modèle linéaire généralisé.

2. Forêts aléatoires

Comme pour la classification binaire, les forêts aléatoires font preuve de robustesse face à la non-linéarité et aux distributions déséquilibrées, et elles sont peu sensibles aux valeurs aberrantes, ce qui est crucial dans notre cas. De plus, ces modèles sont capables de capturer des interactions complexes entre les variables explicatives, ce qui peut améliorer la performance globale du modèle. La description des hyperparamètres n'est pas détaillée ici, elle est déjà présentée dans la table 10.

NumTrees	50
MaxDepth	10
MinInstancesPerNode	2
FeatureSubsetStrategy	'sqrt'

Table 13: Hyperparamètres sélectionnés pour le modèle de forêts aléatoires pour la régression

Par ailleurs, il est intéressant de noter que la variable avec l'importance la plus élevée pour la prédiction de ce modèle de régression est la variable 'numberAuthorsLog' (v=0.022, importance relative v de la variable, obtenue avec la fonction featureImportances). Elle correspond au nombre d'auteurs de la pré-publication, et nous avons effectivement noté une corrélation entre cette variable et la variable cible 'cityyear' au cours de l'analyse exploratoire (graphiques en [annexe 11](#), CoefPearson=0.15).

3. Test et comparaison des modèles de régression

Nous avons utilisé trois métriques pour évaluer les performances de nos modèles de régression:

- **Erreur Absolue Moyenne (MAE)** : mesure de l'écart moyen entre les prédictions du modèle et les valeurs réelles, sans tenir compte de la direction de l'erreur (positive ou négative). Elle est calculée comme la moyenne des valeurs absolues des erreurs de prédiction. Une valeur plus basse indique un modèle plus précis, car elle reflète la moyenne des erreurs sur l'ensemble des prédictions.
- **Erreur Quadratique Moyenne (RMSE)** : mesure de la racine carrée de la moyenne des carrés des erreurs. Elle pénalise plus sévèrement les grandes erreurs, car les erreurs sont élevées au carré. Cela permet de mettre en évidence les prédictions très éloignées des valeurs réelles. Elle est donc plus sensible aux grandes erreurs que la MAE
- **Score R² (Coefficient de détermination)** : mesure de la proportion de la variance de la variable cible qui est expliquée par le modèle. Elle prend une valeur entre 0 et 1, où 1 signifie que le modèle explique parfaitement la variance des données, et 0 signifie qu'il ne l'explique pas du tout. Un R² négatif indique que le modèle est moins performant qu'un simple modèle basé sur la moyenne des valeurs cibles.

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i| \quad RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2} \quad R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{y}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2}$$

avec y_i la valeur réelle, et \hat{y}_i la prédiction du modèle

Les résultats des deux modèles, à savoir le modèle de régression linéaire généralisée (LinearGM) et les forêts aléatoires (RandomForest), sont présentés dans la table 14 et la figure 10. Les performances sont globalement décevantes, en particulier sur les données de test. Le score R² indique que les deux modèles expliquent moins de 10% de la variance dans le nombre de citations sur les données de test. Les erreurs dans la prédiction sont importantes, comme en témoignent les valeurs élevées de la MAE et RMSE, et sont clairement dues aux valeurs extrêmes rencontrées dans les données, que le modèle peine à expliquer avec l'ensemble des variables explicatives à notre disposition (Fig.10, bas).

	Ech. Apprentissage		
	MAE	RMSE	R2
LinearGM	11.27	27.45	0.24
RandomForest	7.83	26.08	0.31
Ech. Test			
	MAE	RMSE	R2
LinearGM	13.39	41.99	0.06
RandomForest	8.85	41.78	0.07

Table 14: Performances des 2 modèles sur le problème régression.

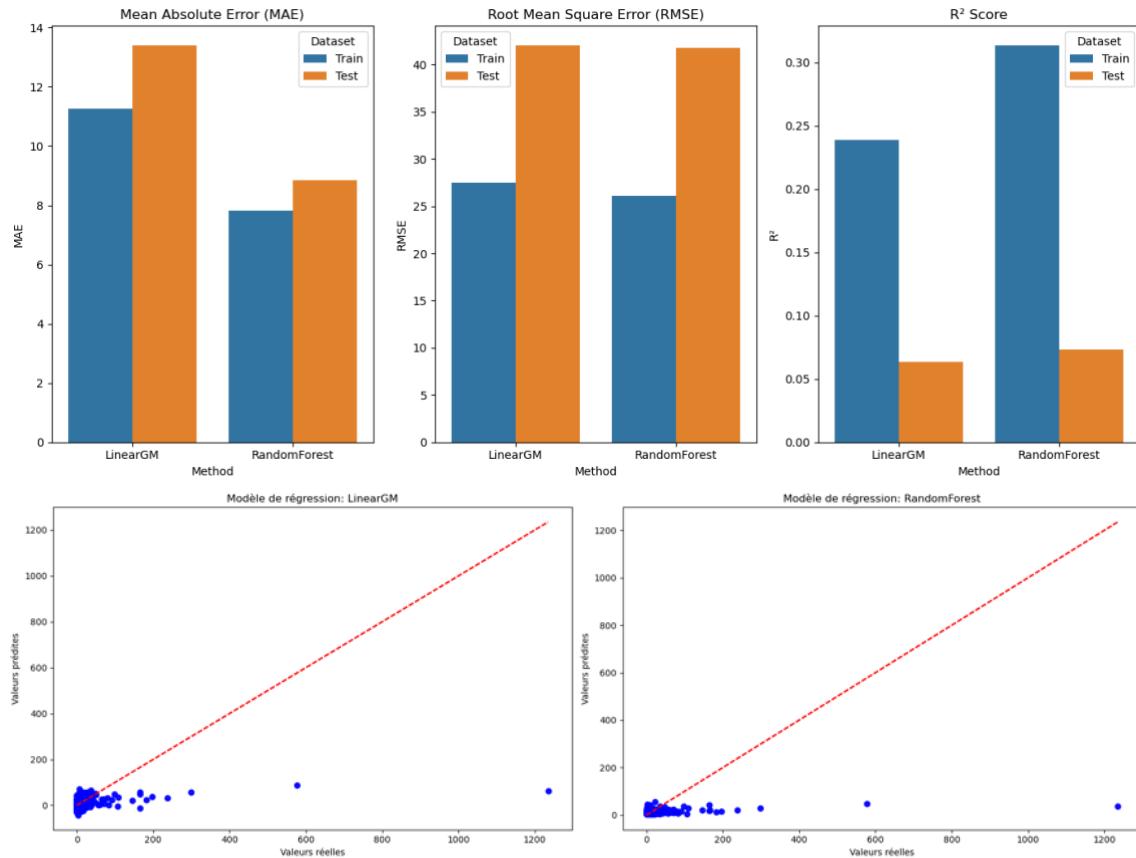


Figure 10: Visualisation graphique des performances de 2 modèles sur les 3 métriques utilisées (haut) et nuages de points des observations réelles versus prédictes pour chaque modèle (bas).

V. Passage à l'échelle

1. Problématiques du passage à l'échelle

Nous avons intégré des outils et méthodes permettant de réaliser les traitements décrits sur un volume de données massives. Pour donner une idée de l'ordre de grandeur des données traitées ici, les volumes des collections utilisées sont reportés dans la table 15.

Afin de gérer de potentiels grands volumes de données, il est nécessaire de « passer à l'échelle » par distribution, c'est-à-dire être capable de déployer efficacement le stockage et le traitement des données sur plusieurs machines afin de gérer une grande quantité de données. Pour cela, il est nécessaire d'assurer à la fois un stockage distribué dans une base de données adaptée, ici MongoDB, et d'utiliser des méthodes permettant une parallélisation des calculs. Les traitements parallélisables sont détaillés dans la table 16.

	Volume total	Nombre de documents
MongoDB – Collection ‘Unique’	529.01MB (dont index : 6.6MB)	201k
Elastic Search – Index ‘Unique’	457MB	201k
MongoDB – Collection ‘Sample’	114.32 MB (dont index : 389kB)	10k
MongoDB – Collection ‘CRISPR’	56.07MB (dont index : 208kB)	4.3k
MongoDB – Collection ‘Alzheimer’	33.64MB (dont index : 155kB)	2.9k

Table 15: Taille des différentes collections créées pour le projet. Il est important de noter que les collections ‘Sample’, ‘CRISPR’ et ‘Alzheimer’ ont une taille beaucoup plus importante par document à cause de l’ajout de nouvelles variables notamment l’encodage BioBert (vecteur de 768 dimensions).

D’une part nous allons évoquer le passage à l’échelle du stockage dans MongoDB ([partie V.2](#)) et dans Elastic Search ([partie V.4](#)). D’autre part, nous détaillerons les aspects du traitement des données dans MongoDB ([partie V.3](#)), et avec Spark ([partie V.4](#)).

Étape	Description	Parallélisable	Outils
1-2	Collecte des données des pré-publications et citations via les API BioRxiv et OpenCitations.	Non	
Stockage documents JSON dans MongoDB			
3	Calcul de nouvelles variables pour chaque document dans MongoDB	Oui	Agrégations MongoDB
4.1	Indexation des documents dans Elastic Search	Non	/
4.2	Recherche plein-texte dans l’index Elastic Search	Oui	Elastic Search
5	Export et analyse exploratoire des données (variables numériques et catégorielles)	Non	/
Intégration des documents de MongoDB dans des DataFrames Spark avec le MongoDB-Spark connector			
6	Encodage des titres et résumés des pré-publications avec un modèle Bert	Oui	Spark NLP
7	Pré-traitement et transformation des variables pour l’apprentissage supervisé	Oui	Spark, Spark MLlib
8-11	Apprentissage supervisé (classification et régression)	Oui	Spark MLlib

Table 16: Détail des étapes parallélisables ou non du projet.

2. RéPLICATION ET PARTITIONNEMENT DANS MONGODB

Les principaux composants de MongoDB sont :

- **mongod** : Le processus principal de MongoDB, responsable du stockage et de la gestion des données.
- **mongo** : le shell en ligne de commande permettant d'interagir avec la base.
- **mongos** : un routeur permettant de gérer les requêtes dans un cluster distribué.
- **Config Servers** : ils gèrent la configuration du cluster et la répartition des données.
- **Shards** : ils contiennent les données réparties sous forme de '*chunks*'.
- **Replica Sets** : Assurent la réPLICATION et garantissent la haute disponibilité des données

MongoDB permet un passage à l'échelle ‘horizontal’ (*horizontal scaling*) grâce à l’usage de son mécanisme de ‘sharding’. La répartition des données sur plusieurs nœuds physiques dans des ‘*chunks*’ (des portions de données) est effectuée à l'aide d'une clé de hachage (*hashing key*). Cette clé de hachage doit être unique et bien choisie, afin d'assurer une répartition équitable des données. Grâce à l'utilisation d'un index unique sur la clé de hachage, MongoDB impose l'unicité sur l'ensemble de la combinaison de clés et non sur les composants individuels de la clé de hachage. Cet index peut par exemple être basée sur un index composé d'un index unique avec la clé de hachage en préfixe.

Enfin, ce sont les routeur *mongos* qui effectuent l'acheminement des requêtes depuis le client vers les *shards* [16] (Fig.11).

Concernant la réPLICATION des données, chaque *shard* est composé d'un *Replica Set*, qui assure la redondance des données et la tolérance aux pannes [17]. Un *Replica Set* est constitué de :

- un **nœud primaire** qui gère les écritures et synchronise les répliques.
- un ou plusieurs **nœuds secondaires** qui reçoivent une copie des données en temps réel.
- éventuellement un **nœud arbitre** qui participe aux élections mais ne stocke pas de données. En effet, en cas de panne du nœud primaire, une élection automatique est déclenchée pour élire un nouveau nœud primaire parmi les nœuds secondaires, garantissant ainsi la continuité du service.

3. Agrégations dans MongoDB

Les **Aggregation Pipelines** de MongoDB permettent de traiter et d'analyser des données efficacement en enchaînant plusieurs étapes de transformation [12]. Elles sont particulièrement adaptées aux bases distribuées, car elles permettent d'exécuter des opérations directement sur les *shards*, sans devoir rapatrier toutes les données sur un seul nœud.

Chaque pipeline est une séquence d'étapes (*stages*) qui appliquent des transformations aux documents. Pour ce projet, nous avons par exemple utilisé les opérateurs *\$match*, *\$merge* ou *\$group*. Comme pour les traitements MapReduce, les premières étapes des pipelines sont exécutées localement sur chaque *shard* avant d'envoyer les résultats au routeur *mongos* pour être agrégés. Cela permet de paralléliser les calculs et réduire les transferts de données entre nœuds. D'après la documentation MongoDB, l'usage des Aggregation Pipelines est favorisé car il offre « de meilleures performances et une plus grande facilité d'utilisation que MapReduce » [18].

4. ElasticSearch

Elastic Search permet aussi une scalabilité horizontale, et les index sont également divisés en plusieurs partitions appelées ‘shards’(Fig.11).

Un *shard* primaire contient une fraction des données de l’index, et Elastic Search répartit automatiquement les *shards* sur les différents nœuds du cluster. De plus, Elastic Search réplique ces *shards* primaires dans des *replica shards* pour assurer la redondance des données. Ces réplicas sont stockés sur des nœuds distincts des *shards* primaires, garantissant ainsi une tolérance aux pannes. Ainsi, comme visible sur la figure 11, un même noeud (data node) ne contient pas forcément tous les *shards* primaires, ils sont répartis entre les noeuds du cluster [19].

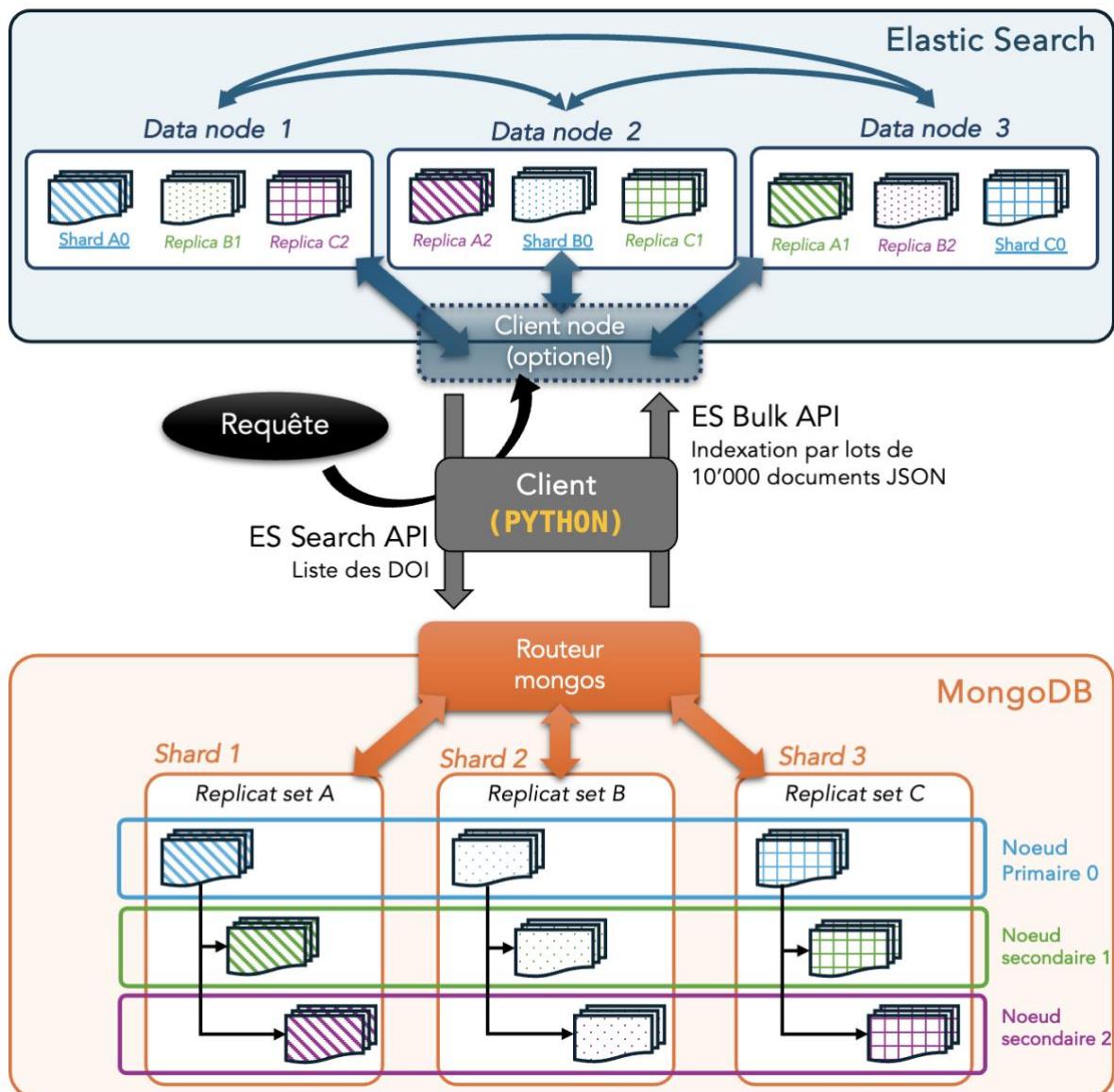


Figure 11: Représentation schématique d'un cluster Elastic Search et d'un cluster MongoDB. Les formes avec un remplissage différent représentent des ‘chunks’ de données (A, B et C), la couleur indiquant le version du réplicat. Au centre, les échanges de données effectués via le client Python lors de l'indexation d'une collection dans Elastic Search. Lors du retour de la liste de DOI, l'index unique créé sur ce champ dans MongoDB permet de trouver rapidement les résultats de la recherche Elastic Search et de les grouper dans une nouvelle sous-collection thématique (extrait de code [en annexe 4 et 5](#)).

5. Spark et MongoDB connector

Apache Spark est un *framework* open source de calcul distribué conçu pour l'analyse de données à grande échelle. Il se distingue par sa vitesse d'exécution, grâce à son traitement et stockage des données intermédiaires en mémoire vive. Cette spécificité s'avère particulièrement efficace pour des algorithmes itératifs comme les algorithmes de machine learning. De plus, même en cas de panne, les données perdues sont rapidement recalculées grâce à une historisation des opérations ('grains traçables'). Depuis la version 1.6, Spark utilise des DataFrames en remplacement des RDD (Resilient Distributed Datasets). Les DataFrames comprennent des données distribuées rangées en colonnes nommées, et utilisent un stockage binaire beaucoup moins volumineux que celui des RDD (JavaObjects), favorisant encore la rapidité des calculs. Pour ce projet, nous avons utilisé la librairie Spark-MLib, une librairie de machine learning intégrée à Spark, optimisée pour le calcul distribué. Pour le traitement du texte, le package de traitement du langage naturel Spark NLP, qui s'appuie sur Spark MLlib et TensorFlow pour offrir des modèles de NLP distribués, a été particulièrement utile.

Spark fonctionne en mode maître-esclave, avec un nœud pilote (*driver*) qui coordonne l'exécution des tâches (objet *SparkContext*) et alloue des ressources entre les applications. Un *Cluster Manager* répartit ensuite les tâches entre les *executors* dans les nœuds de travail (*workers*), qui réalisent les calculs et stockent des données pour les applications.

Le MongoDB-Spark Connector permet d'intégrer MongoDB avec Apache Spark pour traiter efficacement de grandes quantités de données stockées dans la base de données distribuée [20]. Ce connecteur utilise l'API Spark DataFrame pour charger, transformer et analyser les données stockées dans une collection MongoDB. Il permet notamment de lire et écrire des collections MongoDB sous forme de DataFrames. On peut voir des exemples d'utilisation de ce connecteur dans les extraits de code des [annexes 6](#), avec SparkNLP, et [15](#), pour le pré-traitement de des données avant l'apprentissage supervisé.

Ainsi, MongoDB expose les données opérationnelles à la couche de traitement distribué de Spark afin de fournir un traitement rapide et en temps réel [21]. La combinaison des requêtes Spark avec les index MongoDB permet de filtrer les données, d'éviter les balayages de collections complètes et d'offrir une réactivité à faible latence. En effet, l'un des principaux avantages du connecteur est son respect

du principe de **Data Locality**. Ce concept vise à minimiser les déplacements de données en exécutant les traitements directement sur les nœuds où les données sont stockées (Fig.12). Dans le cas de MongoDB, cela signifie que les *shards* de la base sont répartis sur plusieurs serveurs, et Spark peut tirer parti de cette distribution en lançant des tâches de calcul sur les mêmes nœuds, réduisant ainsi la latence et les transferts de données.

Ainsi, l'intégration du stockage dans MongoDB et du traitement de données avec Spark via le connecteur permet d'obtenir un **système distribué complet scalable**, tant au niveau des ressources de stockage que des ressources de calcul.

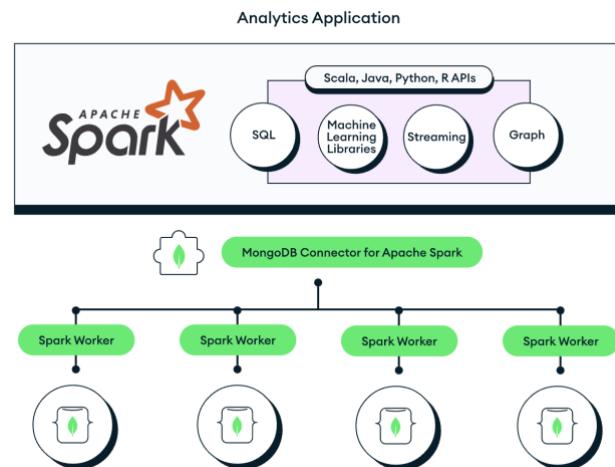


Figure 12: Le connecteur MongoDB pour Apache Spark. Il permet d'utiliser toutes les API de Spark, y compris Scala, Java, Python et R. Les données MongoDB sont matérialisées sous forme de DataFrames et de Datasets pour l'analyse avec les librairies spécialisées de Spark (SQL, Mlib, Streaming, ...). Source de l'image: [20]

Conclusion et perspectives

Au cours de ce projet, nous avons voulu nous intéresser au ‘succès’ de publication des pré-publications déposées sur la plateforme BioRxiv/MedRxiv et aux variables qui peuvent permettre de le prédire. Premièrement, nous avons collecté un grand jeu de données de plus de 200k documents comprenant des variables telles que le titre, le résumé, les auteurs, ou encore la catégorie de chaque pré-publication déposée sur la période 2018-2022. Ensuite, nous avons indexé cette collection dans le moteur de recherche Elastic Search afin de pouvoir sélectionner des sous-ensembles thématiques. Nous avons aussi calculé de nouvelles variables et collecté des données supplémentaires concernant la publication subséquente et les citations de ces pré-publications. En particulier, nous souhaitions nous intéresser au fait qu'une prépublication ait été publiée ou non durant la première année suivant son dépôt (variable ‘*publStatus*’), et si oui, au nombre de ses citations durant la première année de publication (variable ‘*cit1year*’). Nous avons d'abord réalisé une analyse exploratoire des données d'un échantillon aléatoire et d'échantillons thématiques, et en particulier les liens entre les différentes variables. Cette analyse a déjà montré une corrélation relativement faible entre les variables explicatives collectées et les variables réponses que nous souhaitions pouvoir prédire.

Ensuite, dans le but d'entraîner des algorithmes d'apprentissage supervisé à prédire nos 2 variables cibles, nous avons encodé les variables textuelles (le titre et le résumé) à l'aide d'un modèle spécialisé pour les textes scientifiques biomédicaux, appelé BioBert. Enfin, nous avons entraîné des modèles de classification binaire et de régression dans le but de prédire la variable ‘*publStatus*’ et ‘*cit1year*’ respectivement. Les prédictions obtenues ont offert des performances particulièrement faibles, et nous allons discuter ci-après les différentes limitations du projet qui peuvent expliquer cette observation.

Limitations du projet

Variables à faible caractère prédictif

Comme nous l'avons déjà mentionné, il est apparu au cours de l'analyse exploratoire que les variables explicatives à notre disposition n'avaient probablement pas un lien assez important pour prédire efficacement nos variables cibles. De nombreux articles présentent des modèles de prédiction pour le nombre de citations d'articles scientifiques [6], [22], [23] et d'après une revue sur le sujet [8], nombre d'entre eux se basent sur des variables en lien avec les auteurs (leur productivité, leur réseau, le succès de leurs précédentes publications) ou avec le journal dans lequel est publié l'article (impact factor, nombre de citations). Or, ici, nous avons voulu utiliser uniquement des variables intrinsèques des pré-publications.

Dans le but d'améliorer nos prédictions, il pourrait donc être intéressant de collecter d'autres données, afin d'enrichir l'ensemble des variables explicatives de nos documents.

Représentation sémantique des textes

Par ailleurs, il est possible que notre représentation sémantique des textes ait été insuffisante pour en capturer le contenu, et être à même d'en prédire le futur publication. Par exemple, il aurait pu être intéressant de ne pas prendre uniquement en compte la représentation des mots, mais aussi celle des 2- ou tri-grammes afin de saisir des expressions plus complexes. Il pourrait aussi être intéressant d'essayer des modèles de modélisation de sujet ou thème (BertTopic, LDA) plutôt que du contenu littéral des textes.

Importance du choix du sujet et objectifs de départ

Enfin, il est important de revenir sur le fait que les questions initiales posées au début du projet sont loin d'être des questions triviales d'un point de vue conceptuel :

- La qualité sémantique ou même le sujet d'une publication scientifique ne sont pas forcément liées à la rapidité de leur publication, et au nombre de citations dans les premières années de publication - variables qui ne sont elles-mêmes pas forcément synonyme de 'succès' de la publication.
- Le choix de la variable binaire '*publiStatus_binary*' qui indique si une prépublication a été publiée dans un journal durant la première année de dépôt est peut-être un choix maladroit. Elle a été choisie pour pouvoir comparer des prépublications de 2018 et de 2022 en s'affranchissant de variabilité de la date de dépôt. Cependant, elle a le désavantage de regrouper dans le label 'négatif' à la fois des pré-publications qui ne seront jamais publiées, et des pré-publications qui ont pris un temps plus long pour être publiés, par exemple à cause d'un temps de révision plus long dans une revue prestigieuse. Cet effet est documenté dans cette étude complète sur le devenir des pré-publications [9].

Bilan du projet

Malgré des résultats d'apprentissage supervisé relativement décevants, ce projet a constitué une opportunité unique pour mettre en pratique les concepts et outils étudiés au cours du certificat, tout en intégrant les défis aux données massives:

- **NFE204** : Gestion des bases de données distribuées (MongoDB, moteur de recherche ElasticSearch), API REST, MapReduce.
- **STA211** : Analyse exploratoire des données, prétraitements, algorithmes d'apprentissage supervisé.
- **RCP216** : Utilisation de Spark pour le calcul distribué, Spark MLlib pour l'apprentissage supervisé à grande échelle, Spark NLP pour le traitement des textes, visualisation de données.

Au-delà des aspects techniques, ce projet a aussi permis de mettre en évidence l'importance du **choix de la problématique**, des **sources de données**, et des **variables analysées**. Personnellement, il est maintenant plus clair que la compréhension approfondie des données et de leur structure est au cœur de la pertinence des modèles et de l'optimisation de leur performance.

Références

- [1] “bioRxiv Reporting.” Accessed: Jan. 06, 2025. [Online]. Available: <https://api.biorxiv.org/reports/>
- [2] C. F. D. Carneiro *et al.*, “Comparing quality of reporting between preprints and peer-reviewed articles in the biomedical literature,” *Res. Integr. Peer Rev.*, vol. 5, no. 1, p. 16, Dec. 2020, doi: 10.1186/s41073-020-00101-3.
- [3] D. Y. Fu and J. J. Hughey, “Releasing a preprint is associated with more attention and citations for the peer-reviewed article,” *eLife*, vol. 8, p. e52646, Dec. 2019, doi: 10.7554/eLife.52646.
- [4] N. Fraser, F. Momeni, P. Mayr, and I. Peters, “The relationship between bioRxiv preprints, citations and altmetrics,” *Quant. Sci. Stud.*, vol. 1, no. 2, pp. 618–638, Jun. 2020, doi: 10.1162/qss_a_00043.
- [5] H. Liu, G. Hu, and Y. Li, “The enhanced research impact of self-archiving platforms: Evidence from bioRxiv,” *J. Assoc. Inf. Sci. Technol.*, vol. 75, no. 8, pp. 883–897, 2024, doi: 10.1002/asi.24932.
- [6] L. D. Fu and C. Aliferis, “Models for Predicting and Explaining Citation Count of Biomedical Articles,” *AMIA. Annu. Symp. Proc.*, vol. 2008, pp. 222–226, 2008.
- [7] M. Wang, S. Jiao, J. Zhang, X. Zhang, and N. Zhu, “Identification High Influential Articles by Considering the Topic Characteristics of Articles,” *IEEE Access*, vol. 8, pp. 107887–107899, 2020, doi: 10.1109/ACCESS.2020.3001190.
- [8] K. Kousha and M. Thelwall, “Factors associating with or predicting more cited or higher quality journal articles: An Annual Review of Information Science and Technology (ARIST) paper,” *J. Assoc. Inf. Sci. Technol.*, vol. 75, no. 3, pp. 215–244, 2024, doi: 10.1002/asi.24810.
- [9] R. J. Abdill and R. Blekhman, “Tracking the popularity and outcomes of all bioRxiv preprints,” *eLife*, vol. 8, p. e45133, Apr. 2019, doi: 10.7554/eLife.45133.
- [10] “Access and Search MedRxiv and BioRxiv Preprint Data.” Accessed: Jan. 06, 2025. [Online]. Available: <https://docs.ropensci.org/medrxivr/>
- [11] A.-M. Badolato, “OpenCitations | Des références bibliographiques ouvertes,” Ouvrir la Science. Accessed: Feb. 11, 2025. [Online]. Available: <https://www.ouvrirlascience.fr/opencitations-des-references-bibliographiques-ouvertes>
- [12] “Aggregation Operations - MongoDB Manual v8.0.” Accessed: Feb. 11, 2025. [Online]. Available: <https://www.mongodb.com/docs/manual/aggregation/>
- [13] Y. Gu *et al.*, “Domain-Specific Language Model Pretraining for Biomedical Natural Language Processing,” *ACM Trans. Comput. Healthc.*, vol. 3, no. 1, pp. 1–23, Jan. 2022, doi: 10.1145/3458754.
- [14] “microsoft/BiomedNLP-BiomedBERT-base-uncased-abstract-fulltext · Hugging Face.” Accessed: Feb. 10, 2025. [Online]. Available: <https://huggingface.co/microsoft/BiomedNLP-BiomedBERT-base-uncased-abstract-fulltext>
- [15] “Generalized Linear Regression.” Accessed: Feb. 11, 2025. [Online]. Available: <https://spark.apache.org/docs/latest/ml-classification-regression.html#generalized-linear-regression>
- [16] “Sharding - MongoDB Manual v8.0.” Accessed: Feb. 11, 2025. [Online]. Available: <https://www.mongodb.com/docs/manual/sharding/>
- [17] “Replication - MongoDB Manual v8.0.” Accessed: Feb. 11, 2025. [Online]. Available: <https://www.mongodb.com/docs/manual/replication/>
- [18] “Map-Reduce - MongoDB Manual v8.0.” Accessed: Feb. 11, 2025. [Online]. Available: <https://www.mongodb.com/docs/manual/core/map-reduce/>
- [19] “Nodes and shards | Elasticsearch Guide [8.17] | Elastic.” Accessed: Feb. 11, 2025. [Online]. Available: <https://www.elastic.co/guide/en/elasticsearch/reference/current/nodes-shards.html>
- [20] “Connector For Apache Spark,” MongoDB. Accessed: Feb. 11, 2025. [Online]. Available: <https://www.mongodb.com/products/integrations/spark-connector>
- [21] “Apache Spark And MongoDB – Turning Analytics Into Real-Time Action,” MongoDB. Accessed: Feb. 11, 2025. [Online]. Available: <https://www.mongodb.com/resources/products/integrations/apache-spark-and-mongodb-turning-analytics-into-real-time-action>
- [22] L. D. Fu and C. F. Aliferis, “Using content-based and bibliometric features for machine learning models to predict citation counts in the biomedical literature,” *Scientometrics*, vol. 85, no. 1, pp. 257–270, Oct. 2010, doi: 10.1007/s11192-010-0160-5.
- [23] M. Thelwall, “Can the quality of published academic journal articles be assessed with machine learning?,” *Quant. Sci. Stud.*, vol. 3, no. 1, pp. 208–226, Apr. 2022, doi: 10.1162/qss_a_00185.

Annexes

Annexe 1 : Import Des Documents Depuis L'api Biorxiv Dans La Collection Mongodb	31
Annexe 2 : Ajout Des Citations Aux Documents De La Collection Avec L'api Opencitations	32
Annexe 3 : Calcul Des Variables Bibliométriques Par Pipeline D'agrégation Mongodb.....	33
Annexe 4: Import Des Documents Dans Un Index Elasticsearch	34
Annexe 5 : Recherche Dans Elasticsearch Et Création De Sous-Collection Mongodb	35
Annexe 6 : Encodage Des Textes Avec Biobert Dans Sparknlp.....	36
Annexe 7 : Pca Sur Les 768 « Features » De L'encodage Des Mots Avec Biobert	38
Annexe 8 : Export Des Données Pour Analyse Exploratoire.....	39
Annexe 9 : Plots Issus De L'analyse Univariée De L'échantillon Aléatoire.....	40
Annexe 10: Recodage Des 75 Catégories Originales En 18 Catégories, Par Thématique	41
Annexe 11 : Plots Issus De L'analyse Bivariée De L'échantillon Aléatoire	42
Annexe 12 : Plots Issus De L'analyse Multivariée (Afdm) Sur L'échantillon Aléatoire	43
Annexe 13 : Analyse Exploratoire Sur L'échantillon 'Crispr'	44
Annexe 14: Analyse Exploratoire Sur L'échantillon 'Alzheimer'	45
Annexe 15 : Prétraitment Des Données Pour L'apprentissage.....	46
Annexe 16 : Sélection Des Meilleurs Modèles De Classification Binaire Par Validation Croisée	48
Annexe 17: Comparaison Des Performances De Différents Modèles Pour La Classification Binaire	49
Annexe18 : Performances De Modèles De Classification Binaires Entrainés Sur L'échantillon Thématique 'Crispr'.50	
Annexe19 : Performances De Modèles De Classification Binaires Entrainés Sur L'échantillon Thématique 'Alzheimer'	51
Annexe 20: Sélection Des Meilleurs Modèles De Régression Par Validation Croisée	52
Annexe 21 : Comparaison Des Performances De Différents Modèles Pour La Régression	53

Annexe 1 : Import des documents depuis l'API BioRxiv dans la collection MongoDB

```

class rxiv_collector() :

    def __init__(self, mongo_db_name):
        self.uri = "mongodb://localhost:27018/" # Info de connection a la base de donnée
        self.client = MongoClient(self.uri)
        self.database = self.client["UASB03"]
        self.collection_name = mongo_db_name
        print('Instance created')

    def biorxiv_upload(self, cursor, server, abstract_url):

        print(f'URL {server}:{abstract_url}')
        biorxiv_response = urllib.request.urlopen(abstract_url).read()

        # Parse the JSON and collect the list of articles "collection"
        response_json = json.loads(biorxiv_response)
        collect_preprints = response_json.get("collection", [])

        try:
            collection = self.database[self.collection_name] # Upload dans la base
            result = collection.insert_many(collect_preprints)
            print(result.acknowledged)

        except Exception as e:
            print(f"Cursor error: {cursor}") # Imprission du curseur pour pouvoir reprendre la
collecte
            raise Exception(
                "The following error occurred: ", e)

    def biorxiv_collect(self, entry_dict):
        server = entry_dict.get('server')
        date_interval = entry_dict.get('date_interval')
        counter = entry_dict.get('counter')

        url=f'https://api.biorxiv.org/details/{server}' # Creation de l'URL pour questionner
l'API
        abstract_url = f'{url}/{date_interval}/{counter}'
        print(f'URL {server}:{abstract_url}')
        biorxiv_response = urllib.request.urlopen(abstract_url).read()

        # Informations sur le nombre total d'articles obtenus avec la requete
        response_json = json.loads(biorxiv_response)
        messages = response_json.get("messages", [])
        count_new_papers = int(messages[0].get('count_new_papers', 0))
        total = int(messages[0].get('total', 0))
        print("Messages:", messages)
        print(f'Count of new papers: {count_new_papers}, Total: {total}')

        while counter <= (total): # Boucle pour collecter les articles par 100 (max autorisé
par l'API)
            print("Cursor:", counter)
            self.biorxiv_upload(counter, server, abstract_url)
            counter=counter+100
        print("Finished")

```

Annexe 2 : Ajout des citations aux documents de la collection avec l'API OpenCitations

```

class DataCollector():

    def __init__(self, mongo_db_collection_name):
        self.base_url = 'https://opencitations.net'
        self.http_headers = {"authorization": "587ee329-c5d0-4512-8272-1742619d35e6"}
        self.mongodatabase= mongo_db_collection_name
        print('DataCollector initialized - DOI list is empty')

    def collect_cit(self, doi): # API Call OpenCitations pour collecter les citations
        url = f'{self.base_url}/index/api/v2/citations'
        sorting = 'sort=desc(timexpan)'
        cit_url = f'{url}/{doi}:{doi}?{sorting}'
        print(f'URL OpenCit:{cit_url}')

        try:
            request = urllib.request.Request(cit_url, headers=self.http_headers)
            opencit_response = urllib.request.urlopen(request, timeout=60).read()
            response_json = json.loads(opencit_response)
            print(f'Response: {response_json}')
            return response_json
        except Exception as e:
            print(f'Error collecting publication date for DOI {doi}: {e}')
            return ''

    def collect_pubdate(self, doi): # API Call OpenCitations pour collecter la date de publication
        url = f'{self.base_url}/meta/api/v1/metadata'
        pubdate_url = f'{url}/{doi}:{doi}'
        print(f'URL OpenCit:{pubdate_url}')

        try:
            request = urllib.request.Request(pubdate_url, headers=self.http_headers)
            opencit_response = urllib.request.urlopen(request, timeout=60).read()
            pubdate = json.loads(opencit_response)[0].get("pub_date")
            print(f'Publication Date: {pubdate}')
            return pubdate
        except Exception as e:
            print(f'Error collecting citations for DOI {doi}: {e}')
            return ''

    def collect_and_upload_CitData(self):
        # Connection à la collection MongoDB
        uri = "mongodb://localhost:27018/"
        client = MongoClient(uri)
        database = client["UASB03"]
        collection = database[self.mongodatabase]
        query = { # Requête pour les articles sans date de publication et sans citations
            "published": {"$ne": "NA"},
            "publiDate": {"$exists": False},
            "citations": {"$exists": False}
        }
        print(f'Number of results: {collection.count_documents(query)}')
        cursor = collection.find(query)

        for document in cursor:
            doi = document.get("published")
            print(doi)

            try: # Collecte de la date de publication
                pubdate = self.collect_pubdate(doi)
            except Exception as e:
                print(f'Error collecting pubdate for DOI {doi}: {e}')
                pubdate = ''
                continue

            try: # Collecte des citations
                citations = self.collect_cit(doi)
            except Exception as e:
                print(f'Error collecting cit for DOI {doi}: {e}')
                citations = ''
                continue

            if pubdate != '' and citations != '':
                result = collection.update_one(
                    {"published": doi},
                    {"$set": {"publiDate": pubdate, "citations": citations}}
                )
                print(result.matched_count)
                print(result.modified_count)
            else: # Pas d'upload si les 2 informations ne sont pas disponibles
                print('No data available')

        client.close()
        print('Finished')

```

Annexe 3 : Calcul des variables bibliométriques par pipeline d'agrégation MongoDB

```

class newFields_mongo():
    def __init__(self, mongo_db_collection_name):
        self.client = MongoClient('mongodb://localhost:27018/')
        self.db = self.client['UASB03']
        self.collection= mongo_db_collection_name
        self.connect=self.db[self.collection]
        print(f'Ready to add fields to {self.collection}')

    def create_index_doi(self):
        collection= self.db[self.collection]
        pipeline_duplicates = [
            {
                '$group': {
                    '_id': '$doi',
                    'maxVersion': {
                        '$max': '$version'
                    },
                    'doc': {
                        '$first': '$$ROOT'
                    }
                }
            },
            {
                '$replaceRoot': {
                    'newRoot': '$doc'
                }
            },
            {
                '$out': 'temp_collection'
            }
        ]
        try:
            collection.aggregate(pipeline_duplicates)
        except Exception as e:
            print(e)
        collection.drop()
        self.db['temp_collection'].rename(self.collection)

    #-Création d'un index unique sur le champ 'doi'
    indexes = self.connect.index_information()
    if 'doi_1' not in indexes:
        self.connect.create_index([('doi', 1)], unique=True)

    print(f'Ready to add new fields to the collection {self.collection} - Unique index on doi created')

    def add_new_fields(self):
        collection= self.db[self.collection]
        pipelines = [{}]
        for pipeline in pipelines:
            try:
                collection.aggregate(pipeline)
            except Exception as e:
                print(e)

        self.client.close()
        print(f'New fields added to the collection {self.collection}')

```

Annexe 4 : Import des documents dans un index Elastic Search

```

class Index_ES():

    def __init__(self, db, mongodb_source_collection, ES_username, ES_password):
        # Connexion à la collection MongoDB
        self.client = MongoClient('mongodb://localhost:27018/')
        self.db = self.client[db]
        self.collection = self.db[mongodb_source_collection]
        try:
            document = self.collection.find_one()
            if document:
                print(f"Connection to MongoDB collection {mongodb_source_collection} successfully!")
            else:
                print("Connection to MongoDB collection is successful, but the collection is empty.")
        except Exception as e:
            print("Failed to connect to MongoDB collection:", e)

        # Connexion à Elasticsearch
        urllib3.disable_warnings()
        self.es = Elasticsearch(hosts="https://localhost:9200",
                               basic_auth=(ES_username, ES_password),
                               verify_certs=False)
        if self.es.ping():
            print("Connected to Elasticsearch successfully!")
        else:
            print("Failed to connect to Elasticsearch.")

        print(self.es.info())

    def export_collection_toES(self, new_index_name):
        # Création d'un index dans ES
        if not self.es.indices.exists(index=new_index_name):
            self.es.indices.create(index=new_index_name)
            print(f"Index {new_index_name} created successfully.")
        else:
            print(f"Index {new_index_name} already exists.")

        # Collecte seulement les champs textuels des documents dans MongoDB
        pipeline_export = [
            {'$project': {
                '$doi': 1,
                'title': 1,
                'abstract': 1,
                'authors': 1,
                'author_corresponding_institution': 1,
                'category': 1
            }}]
        results = self.collection.aggregate(pipeline_export)
        results_list = list(results)
        print(f"Number of documents retrieved from MongoDB {self.collection}: {len(results_list)}")

        # Transfert des documents dans l'index ES par batch de 10'000
        output_iterate = []
        i=0

        for document in results_list:
            output_iterate.append({"index": {"_index": new_index_name, "_id": document.get('doi')}})
            output_iterate.append({
                "title": document.get('title'),
                "abstract": document.get('abstract'),
                "doi": document.get('doi'),
                "authors": document.get('authors'),
                "author_corresponding_institution": document.get('author_corresponding_institution'),
                "category": document.get('category')})

            if i % 10000 == 0 and i != 0:
                self.es.bulk(operations=output_iterate, index=new_index_name)

            count_index = self.es.count(index=new_index_name)
            print(f"Number of documents in '{new_index_name}':", count_index["count"])

            output_iterate = []
            i += 1

        # Transfert des derniers documents après la boucle
        if output_iterate:
            self.es.bulk(operations=output_iterate, index=new_index_name)
            count_index = self.es.count(index=new_index_name)
            print(f"Number of documents in '{new_index_name}':", count_index["count"])

```

Annexe 5 : Recherche dans Elastic Search et création de sous-collection MongoDB

```

query_alzheimer = {
    "query": {
        "query_string": {
            "query": "(alzheimer* OR dementia*)",
            "default_operator": "OR"
        }
    }
}

query_crispr = {
    "query": {
        "bool": {
            "should": [
                {"match_phrase": {"abstract": "CRISPR"}},
                {"match_phrase": {"abstract": "CRISPR-Cas9"}},
                {"match_phrase": {"abstract": "gene editing"}},
                {"match_phrase": {"abstract": "base editing"}},
                {"match_phrase": {"abstract": "prime editing"}},
                {"match_phrase": {"abstract": "genetic engineering"}}
            ]
        }
    }
}

query_longCovid = {
    "query": {
        "bool": {
            "should": [
                {"match_phrase": {"title": "long COVID"}},
                {"match_phrase": {"title": "post-acute sequelae of SARS-CoV-2"}},
                {"match_phrase": {"title": "post-COVID syndrome"}},
                {"match_phrase": {"title": "persistent COVID symptoms"}},
                {"match_phrase": {"title": "COVID brain fog"}}
            ]
        }
    }
}

```

```

class index_ES():
    def __init__(self, db, mongodb_source_collection, ES_username, ES_password):
        # Connection à MongoDb et ElasticSearch

    def search_ES_index(self, es_index_name, query):
        # Exécution de la recherche
        response = self.es.search(index=es_index_name, body=query, size=10000)
        print(response)
        total_hits = response["hits"]["total"]["value"]
        doi_list = [hit["_source"].get("doi", "N/A") for hit in response["hits"]["hits"]]

        # Output results
        print(f"Number of results: {total_hits}")
        print(f"length of doi list: {len(doi_list)}")
        return doi_list

    def results_collection_mongoDB(self, new_collection_mongodb, doi_list):

        if new_collection_mongodb in self.db.list_collection_names():
            print(f"Collection {new_collection_mongodb} already exists.")
        else:
            output_collection = self.db[new_collection_mongodb]
            output_collection.create_index([('doi', 1)], unique=True)
            print(f"Collection {new_collection_mongodb} created successfully with index on 'doi'.")

        pipeline_query_output = [
            {'$match': {'doi': {'$in': doi_list}}},
            {'$merge': {'into': f'{new_collection_mongodb}' ,
                       'on': 'doi',
                       'whenMatched': 'merge', 'whenNotMatched': 'insert'}}
        ]

        try:
            self.collection.aggregate(pipeline_query_output)
            print('Search results integrated successfully to a new MongoDB collection')
        except Exception as e:
            print(e)

```

Annexe 6 : Encodage des textes avec BioBert dans SparkNLP

```

class BioBertEmbeddings():

    def __init__(self, mondodb_base, mongo_db_collection):
        os.environ['PYSPARK_PYTHON'] = sys.executable
        os.environ['PYSPARK_DRIVER_PYTHON'] = sys.executable
        print(f"Pyspark version {pyspark.__version__}")
        self.db=mondodb_base
        self.collection=mongo_db_collection

        # create a spark session
        self.spark = SparkSession \
            .builder \
            .master("local") \
            .appName("MongoDB") \
            .config("spark.mongodb.read.connection.uri", "mongodb://localhost:27018/{self.db}") \
            .config("spark.mongodb.write.connection.uri", "mongodb://localhost:27018/{self.db}") \
            .config('spark.jars.packages', 'org.mongodb.spark: mongo-spark-
connector_2.12:10.4.0,com.johnsnowlabs.nlp:spark-nlp_2.12:5.5.1') \
            .getOrCreate()

    def WordEmbedding(self, embedding_model):
        df = self.spark.read \
            .format("mongodb") \
            .option("uri", "mongodb://localhost:27018/UASB03") \
            .option("database", "UASB03") \
            .option("collection", self.collection) \
            .load() \
            .select("doi", "title", "titleLength", "abstract", "abstractLength")

        df= df.select("doi", "title", "titleLength", "abstract", "abstractLength")\
            .withColumn("merged_text", concat("title", lit(" "), "abstract"))

        df.printSchema()
        print(f'Number of documents in the df: {df.count()}')
        print("Top of the dataframe:")
        df.show(10, truncate=100)
        # Pipeline Spark NLP
        # Assemblage du document
        documentAssembler = DocumentAssembler() \
            .setCleanupMode("inplace") \
            .setInputCol("merged_text") \
            .setOutputCol("document")
        # Detection des phrases
        sentence = SentenceDetector() \
            .setInputCols(["document"]) \
            .setOutputCol("sentence")
        # Tokenisation
        tokenizer = Tokenizer() \
            .setInputCols(['document']) \
            .setOutputCol('token')
        # Normalisation
        normalizer = Normalizer() \
            .setInputCols(["token"]) \
            .setOutputCol("normalized") \
            .setCleanupPatterns(["""[^a-zA-Z0-9]"""])
        # Encodage avec un modèle BERT pré-entraîné sur des articles scientifiques
        embeddings = BertEmbeddings.pretrained(embedding_model, "en") \
            .setInputCols(["document", "normalized"]) \
            .setOutputCol("biobert_embeddings")
        # Encodage des phrases (moyenne des vecteurs de mots)
        embeddingsSentence = SentenceEmbeddings() \
            .setInputCols(["sentence", "biobert_embeddings"]) \
            .setOutputCol("sentence_embeddings") \
            .setPoolingStrategy("AVERAGE")
        # Récupération des vecteurs de phrases
        finisher = EmbeddingsFinisher() \
            .setInputCols("sentence_embeddings") \
            .setOutputCols("output") \
            .setOutputAsVector(True) \
            .setCleanAnnotations(False)

        # Pipeline
        pipeline_biobert = Pipeline(stages=[

            documentAssembler,
            sentence,
            tokenizer,
            normalizer,
            embeddings,
            embeddingsSentence,
            finisher])

```

```

# Application de la pipeline
model_biobert = pipeline_biobert.fit(df)
result_biobert = model_biobert.transform(df)
result_biobert = result_biobert.withColumn("features", col("output") [0])
print("Schema after embedding:")
result_biobert.printSchema()
vector_size = result_biobert.select("features").first()["features"].size
print(f"Size of the vectors in the features column: {vector_size}")
result_biobert.select("title", "features").show(10, 300)

df.unpersist(blocking = True)
result_biobert.persist(StorageLevel.MEMORY_AND_DISK)

# Define a UDF to convert Vector to list of floats
vector_to_array = udf(lambda vector: vector.toArray().tolist(), ArrayType(FloatType()))
output_mongo = result_biobert.select("title", "abstract", "doi", "features") \
    .withColumn("features", vector_to_array("features"))
output_mongo.printSchema()

# Ecriture dans une nouvelle collection temporaire MongoDB
new_collection_name=f"temp_BioBert_{self.collection}"
output_mongo.write.format("mongodb") \
    .mode("append") \
    .option("database", "UASB03") \
    .option("collection", new_collection_name) \
    .save()

# Ajout du champ dans la collection originale
client = MongoClient('mongodb://localhost:27018/')
db = client[self.db]
new_collection = db[new_collection_name]
# Create index on doi in the new collection
indexes= new_collection.index_information()
if 'doi_1' not in indexes:
    new_collection.create_index([('doi', 1)], unique=True)

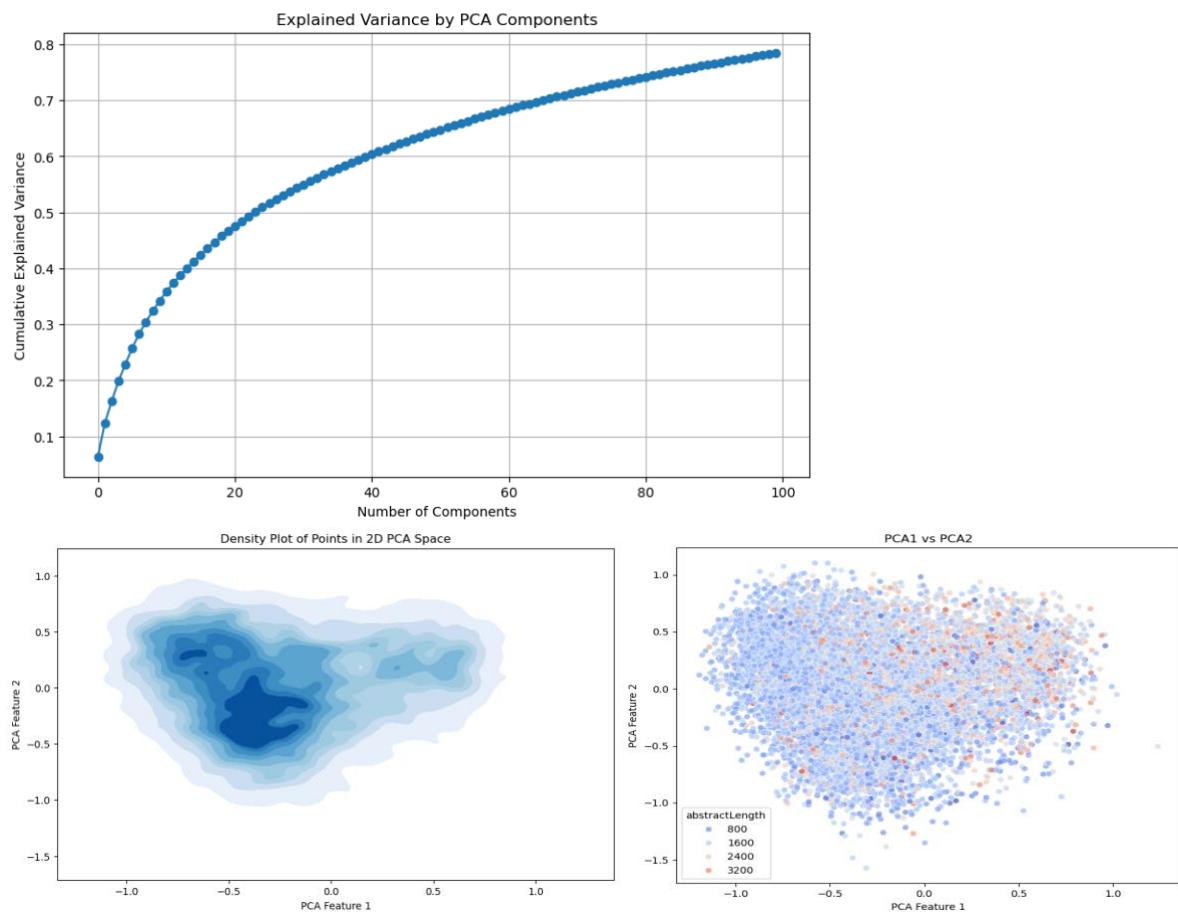
pipeline_add_embeddings = [
    {'$addFields': {'BioBertEmbedding_str': '$features'}},
    {'$unset': ['_id', 'features']},
    {'$merge': {'into': self.collection, 'on': 'doi',
               'whenMatched': 'merge', 'whenNotMatched': 'insert'}}
]

try:
    new_collection.aggregate(pipeline_add_embeddings)
except Exception as e:
    print(e)

new_collection.drop() # Suppression de la collection temporaire
document = db[self.collection].find_one()
print(document)
print("Upload to MongoDB finished")

```

Annexe 7 : PCA sur les 768 « features » de l'encodage des mots avec BioBert



Annexe 8 : Export des données pour analyse exploratoire

```

pipeline_export = [
    {"$match": {"$or": [
        { "$and": [
            { "published": { "$ne": 'NA'}},
            { "publiStatus": { "$ne": "publi_nodate"}}
        ]},
        { "published": 'NA'}
    ]}}
]

def export_csv(mongodb_name, collection_name, output_path, output_name):
    client = MongoClient('mongodb://localhost:27018/')
    db = client[mongodb_name]
    collection = db[collection_name]

    results = collection.aggregate(pipeline_export)
    results_list = list(results)
    print(f"Number of documents retrieved in {collection_name}: {len(results_list)}")

    list_fields=['doi', 'server', 'version', 'category', 'publiStatus', 'date',
    'author_corresponding_institution',
    'titleLength', 'abstractLength', 'numberAuthors', 'citlyear',
    'BioBertEmbedding_str']

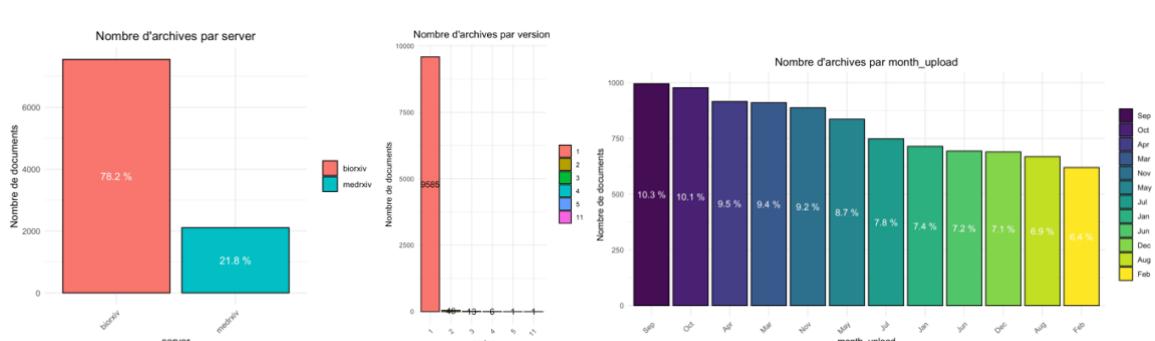
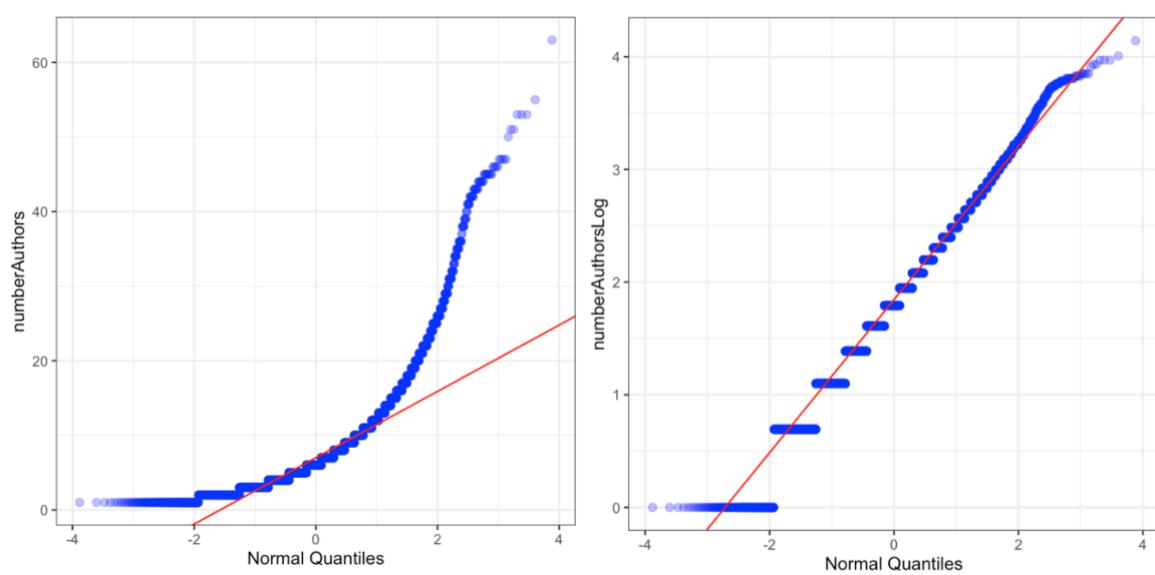
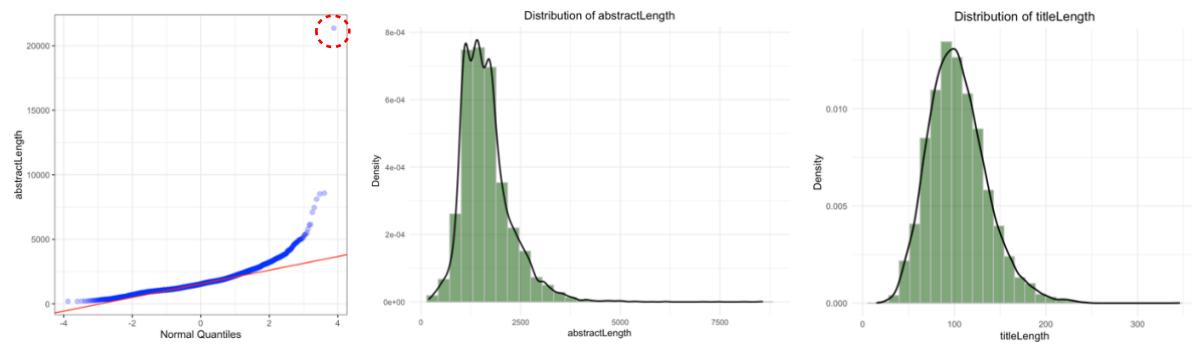
    parsed_results = []
    for result in results_list:
        result_iter = {}
        for field in list_fields:
            # Ajout de chaque champs dans un dictionnaire
            result_iter[field] = result.get(field, None)
        parsed_results.append(result_iter)

    df = pd.DataFrame(parsed_results, columns=list_fields)
    # Change type de version and renommage de la colonne BioBert
    df['version'] = df['version'].astype('int64', errors='ignore')
    df.rename(columns={'BioBertEmbedding_str': 'BioBert'}, inplace=True)

    print(df.info())
    # Export the dataframe to a .csv file
    df.to_csv(f'{output_path}/{output_name}.csv', index=False)

```

Annexe 9 : Plots issus de l'analyse univariée de l'échantillon aléatoire

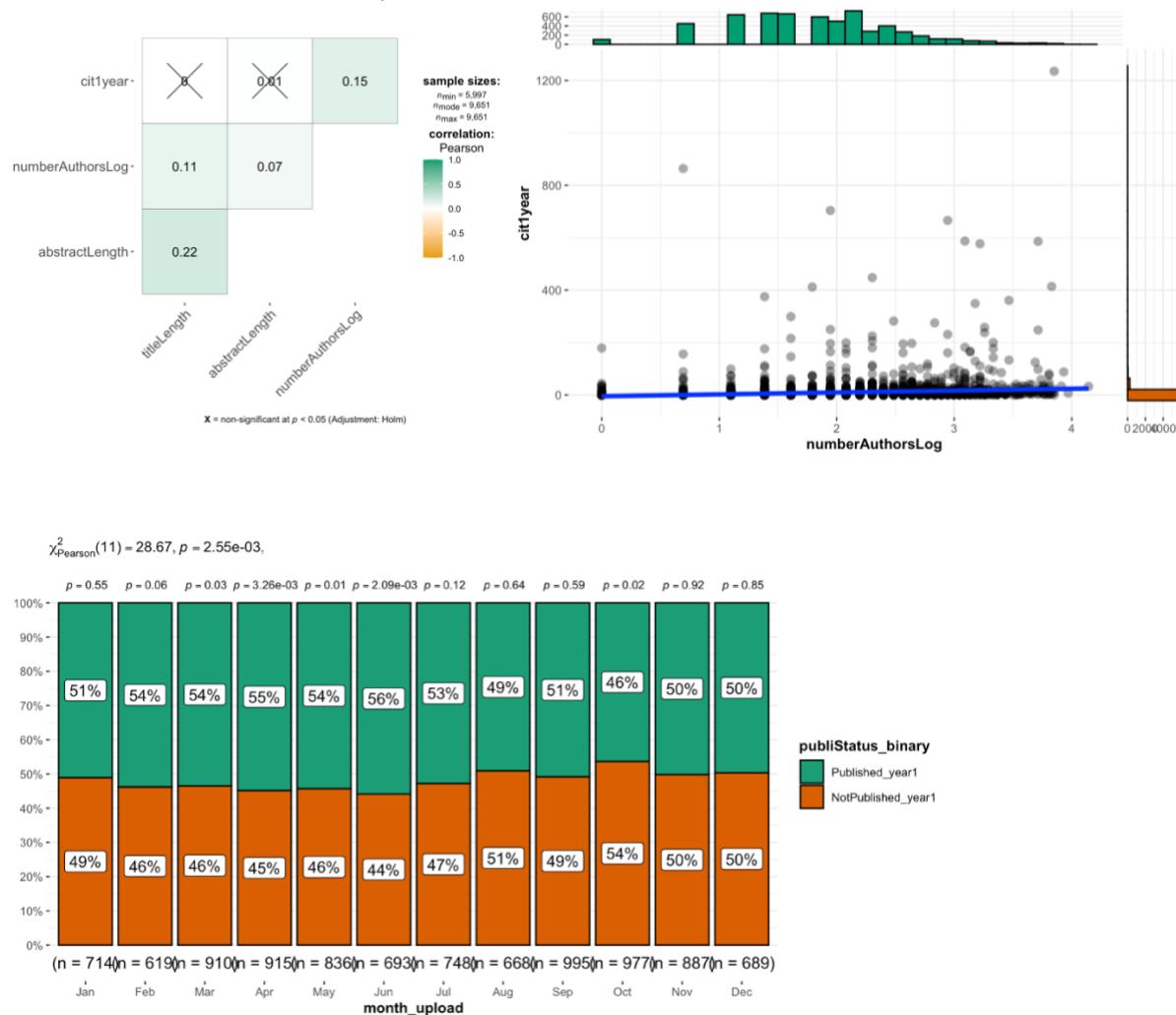


Distribution des archives en fonction du server de dépôt (gauche), version maximum déposée (centre) et mois de dépôt (droite).

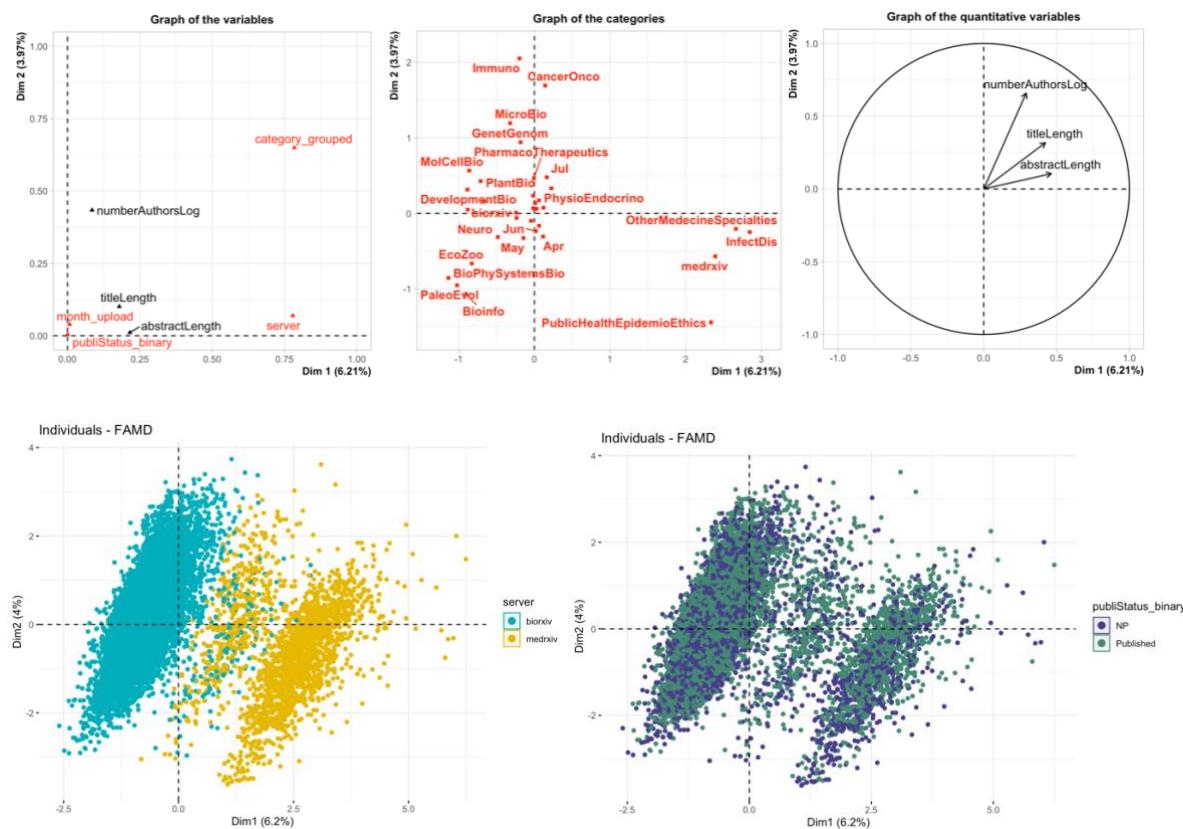
Annexe 10: Recodage des 75 catégories originales en 18 catégories, par thématique

	category	n	category_grouped	n	
1	neuroscience	1308	1	Neuro	1612
2	microbiology	751	2	MolCellBio	1004
3	bioinformatics	650	3	Bioinfo	881
4	infectious diseases	479	4	PublicHealthEpidemicEthics	790
5	cell biology	445	5	GenetGenom	770
6	epidemiology	417	6	MicroBio	751
7	biophysics	390	7	InfectDis	509
8	genomics	379	8	OtherMedecineSpecialties	481
9	evolutionary biology	346	9	BioPhySystemsBio	390
10	ecology	335	10	EcoZoo	378
11	immunology	310	11	PaleoEvol	364
12	genetics	299	12	CancerOnco	339
13	biochemistry	289	13	Immuno	331
14	cancer biology	279	14	SyntheticBioengineering	280
15	molecular biology	270	15	PlantBio	270
16	plant biology	270	16	DevelopmentBio	236
17	developmental biology	236	17	PhysioEndocrin	169
18	bioengineering	226	18	PharmacoTherapeutics	96
19	public and global health	219			
20	systems biology	174			
21	physiology	132			
22	animal behavior and cognition	121			
23	genetic and genomic medicine	92			
24	psychiatry and clinical psychology	92			
25	neurology	91			
26	pharmacology and toxicology	78			
27	pathology	64			
28	oncology	60			
29	health informatics	57			
30	synthetic biology	54			
31	cardiovascular medicine	46			
32	scientific communication and education	44			
33	zoology	43			
34	health policy	36			
35	radiology and imaging	33			
36	pediatrics	31			
37	hiv aids	30			
38	gastroenterology	28			
39	endocrinology	25			
40	health systems and quality improvement	24			
41	intensive care and critical care medicine	24			
42	health economics	22			
43	respiratory medicine	22			
44	allergy and immunology	21			
45	obstetrics and gynecology	21			
46	ophthalmology	20			
47	nephrology	19			
48	paleontology	18			
49	occupational and environmental health	15			
50	geriatric medicine	14			
51	rehabilitation medicine and physical therapy	14			
52	otolaryngology	13			
53	rheumatology	13			
54	nutrition	12			
55	medical education	11			
56	pharmacology and therapeutics	11			
57	primary care research	11			
58	emergency medicine	10			
59	hematology	10			
60	surgery	10			
61	pain medicine	9			
62	palliative medicine	9			
63	sports medicine	9			
64	addiction medicine	8			
65	sexual and reproductive health	8			
66	dermatology	7			
67	transplantation	7			
68	nursing	6			
69	dentistry and oral medicine	5			
70	orthopedics	5			
71	anesthesia	4			
72	clinical trials	4			
73	toxicology	3			
74	medical ethics	2			
75	urology	1			

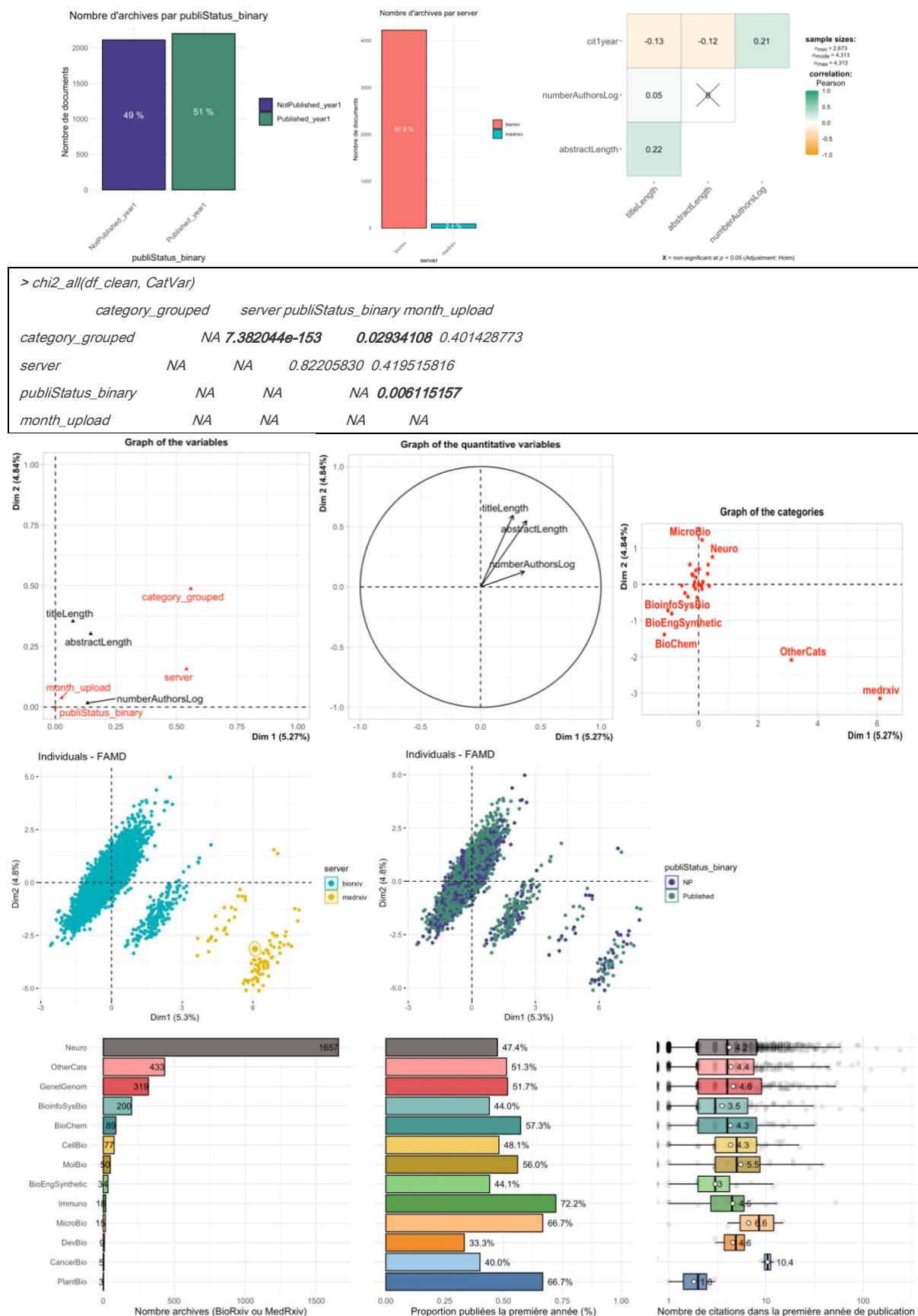
Annexe 11 : Plots issus de l'analyse bivariée de l'échantillon aléatoire



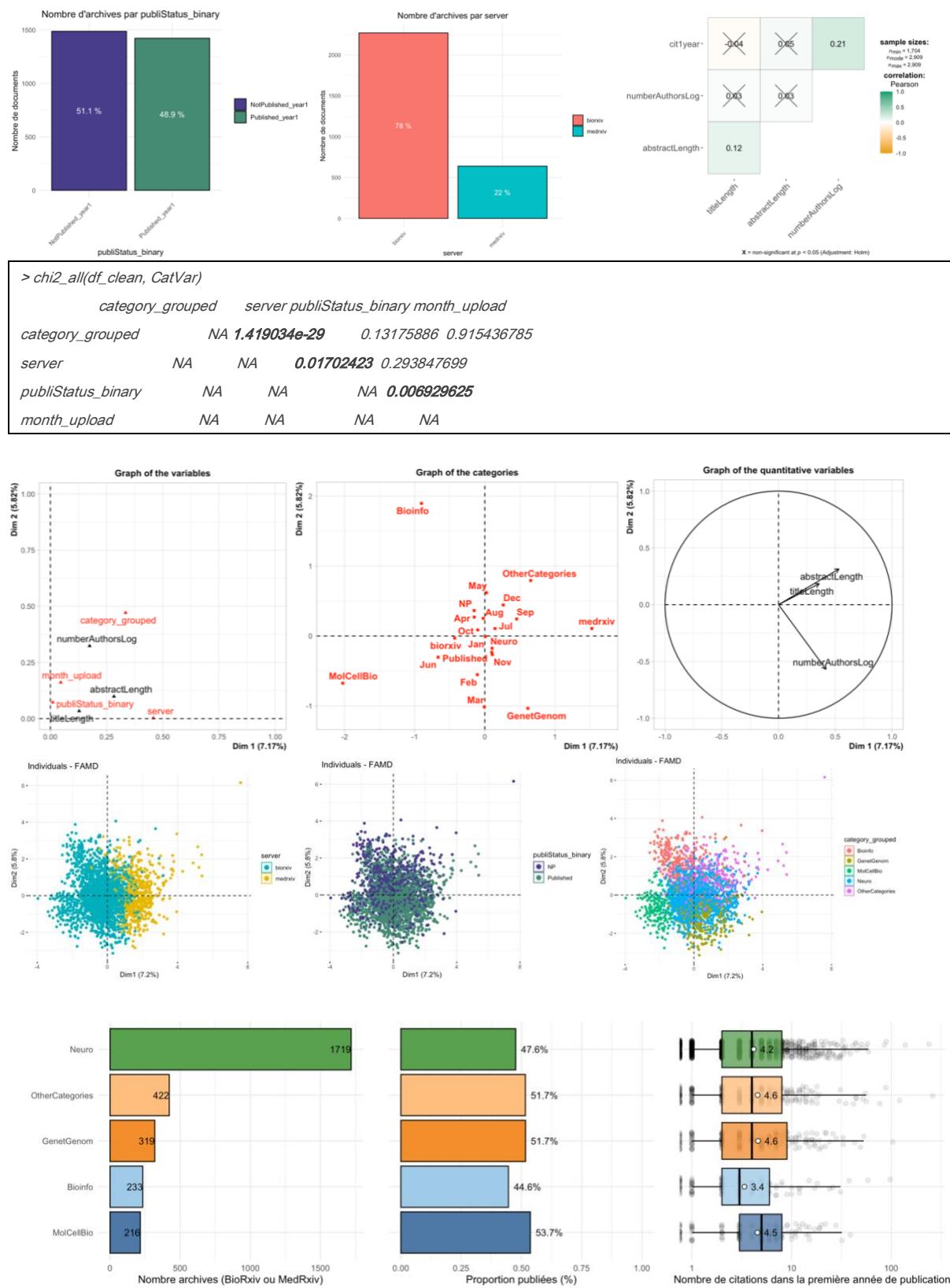
Annexe 12 : Plots issus de l'analyse multivariée (AFDM) sur l'échantillon aléatoire



Annexe 13 : Analyse exploratoire sur l'échantillon 'CRISPR'



Annexe 14: Analyse exploratoire sur l'échantillon 'Alzheimer'



Annexe 15 : Pré-traitement des données pour l'apprentissage

```

pipeline_export = [
    {"$match": {"$or": [
        { "$and": [
            { "published": { "$ne": 'NA'}},
            { "publisStatus": { "$ne": "publi_nodate"}}
        ]},
        { "published": 'NA'}
    ]}}
]

array_to_vector = udf(lambda array: Vectors.dense(array), VectorUDT())

```

```

class pretreatment_spark():

    def __init__(self, mongodb_base, mongodb_collection):
        os.environ['PYSPARK_PYTHON'] = sys.executable
        os.environ['PYSPARK_DRIVER_PYTHON'] = sys.executable
        print(f"Pyspark version {pyspark.__version__}")
        self.db=mongodb_base
        self.collection=mongodb_collection

        # create a spark session
        self.spark = SparkSession \
            .builder \
            .master("local") \
            .appName("MongoDB") \
            .config("spark.mongodb.read.connection.uri", f"mongodb://localhost:27018/{self.db}") \
            .config("spark.mongodb.write.connection.uri", f"mongodb://localhost:27018/{self.db}") \
            .config('spark.jars.packages', 'org.mongodb.spark:mongo-spark-connector_2.12:10.4.0') \
            .getOrCreate()
        # Lecture contenu collection MongoDB
        df = self.spark.read \
            .format("mongodb") \
            .option("uri", f"mongodb://localhost:27018/{self.db}") \
            .option("database", "UASB03") \
            .option("collection", mongodb_collection) \
            .option("aggregationPipeline", str(pipeline_export)) \
            .load()

        df = df.select('doi', 'server', 'category', 'publisStatus', 'date', '#version',
'author_corresponding_institution',
          'titleLength', 'abstractLength', 'numberAuthors', 'cityyear',
          'BioBertEmbedding_str')

        print(df.count())

        #Transformations des variables
        df = df.withColumn("BioBert_vec", array_to_vector(col("BioBertEmbedding_str")))\ \
            .drop('BioBertEmbedding_str')\ \
            .withColumn("numberAuthorsLog", log(col("numberAuthors")))\ \
            .withColumn("month_upload", month(to_date(col("date"))))

        # Recodage catégories
        df = df.withColumn("category_grouped",
                           when(col("category").isin(["neuroscience", "neurology", "psychiatry and clinical psychology",
                                         "animal behavior and cognition"])), "Neuro")
                           .when(col("category").isNotNull(), "Other"))

        df_output = df.select('server', 'category_grouped', 'month_upload', 'publisStatus',
          'titleLength', 'abstractLength', 'numberAuthorsLog', 'cityyear',
          'BioBert_vec').cache()

        df.unpersist()
        df_output.printSchema()
        df_output.show(10)
        ## Vérification contenu et répartition des données de la collection
        df_output.groupby("publisStatus").count().orderBy(desc('count')).show()
        df_output.groupby("server").count().orderBy(desc('count')).show()
        df_output.groupby("month_upload").count().orderBy(desc('count')).show()
        df_output.groupby("category_grouped").count().orderBy(desc('count')).show()
        self.df_output=df_output

```

```

def classification(self, output_path, output_name):
    df_classif=self.df_output.withColumn("label", when(col("publisStatus").isin(["year1"]), 1)
                                         .otherwise(0)) \
        .drop('cityYear') \
        .cache()

    indexer_classif = StringIndexer(inputCols=["server", "category_grouped", ],
                                     outputCols=["server_index", "category_grouped_index"])

    encoder_classif = OneHotEncoder(inputCols=["server_index", 'month_upload', "category_grouped_index"],
                                    outputCols=["server_encoded", 'month_upload_encoded',
                                    "category_grouped_encoded"])

    # Assemble numerical features to be scaled into a single vector
    assembler_subset = VectorAssembler( inputCols=['titleLength', 'abstractLength', 'numberAuthorsLog'],
                                         outputCol="subsetFeatures")

    # scaler for assembled numerical features
    scaler_subset = StandardScaler(inputCol='subsetFeatures', outputCol='scaled_subsetfeatures') \
        .setWithStd(True) \
        .setWithMean(True)

    assembler_all = VectorAssembler( inputCols=['server_encoded', 'month_upload_encoded',
                                                'category_grouped_encoded',
                                                'scaled_subsetfeatures',
                                                'BioBert_vec'], outputCol="features")

    pipeline_pretreat_classification=Pipeline(stages=[

        indexer_classif,
        encoder_classif,
        assembler_subset,
        scaler_subset,
        assembler_all])

    df_classif = pipeline_pretreat_classification.fit(df_classif).transform(df_classif)
    df_classif.printSchema()
    df_classif.show(10, truncate=100)
    print("Features vector size:", df_classif.select("features").head()[0].size)

    df_classif.write.mode("overwrite").parquet(f"{output_path}/{output_name}")
    return df_classif

def regression(self, output_path, output_name):
    df_reg=self.df_output.filter(col('cityYear').isNotNull()) \
        .cache()

    indexer_reg = StringIndexer(inputCols=["server", "category_grouped", "publisStatus"],
                                 outputCols=["server_index", "category_grouped_index", "publisStatus_index"])

    encoder_reg = OneHotEncoder(inputCols=["server_index", 'month_upload', "category_grouped_index",
                                         "publisStatus_index"],
                                outputCols=["server_encoded", 'month_upload_encoded',
                                "category_grouped_encoded", "publisStatus_encoded"])

    # Assemble numerical features to be scaled into a single vector
    assembler_subset = VectorAssembler( inputCols=['titleLength', 'abstractLength', 'numberAuthorsLog'],
                                         outputCol="subsetFeatures")

    # scaler for assembled numerical features
    scaler_subset = StandardScaler(inputCol='subsetFeatures', outputCol='scaled_subsetfeatures') \
        .setWithStd(True) \
        .setWithMean(True)

    assembler_all = VectorAssembler( inputCols=['server_encoded', 'month_upload_encoded',
                                                'category_grouped_encoded', 'publisStatus_encoded',
                                                'scaled_subsetfeatures',
                                                'BioBert_vec'], outputCol="features")

    pipeline_pretreat_regression=Pipeline(stages=[

        indexer_reg,
        encoder_reg,
        assembler_subset,
        scaler_subset,
        assembler_all])

    df_reg = pipeline_pretreat_regression.fit(df_reg).transform(df_reg)
    df_reg.printSchema()
    df_reg.show(10, truncate=100)
    print("Features vector size:", df_reg.select("features").head()[0].size)
    df_reg.write.mode("overwrite").parquet(f"{output_path}/{output_name}")
    return df_reg

```

Annexe 16 : Sélection des meilleurs modèles de classification binaire par validation croisée

```
# Partitionner les données en apprentissage (80%) et test (20%)
partitions = df_classif.randomSplit([0.8, 0.2], seed=100)
train = partitions[0].cache() # conservé en mémoire
test = partitions[1]
```

```
lr = LogisticRegression(featuresCol = 'features', labelCol = 'label')
pipelineLR = Pipeline().setStages([lr])

# Définition de la grille des hyperparamètres
paramGrid_lr = (ParamGridBuilder()
    .addGrid(lr.regParam, [0.1, 0.2, 0.3, 0.4, 0.5])
    .addGrid(lr.elasticNetParam, [0, 0.01, 0.1, 0.2])
    .addGrid(lr.maxIter, [5, 10, 25])
    .build())

cv_lr = CrossValidator().setEstimator(pipelineLR) \
    .setEstimatorParamMaps(paramGrid_lr) \
    .setNumFolds(5) \
    .setEvaluator(BinaryClassificationEvaluator())

cvLRmodel = cv_lr.fit(train)
```

```
linSVC = LinearSVC().setFeaturesCol("features").setLabelCol("label")\
pipeline_svc = Pipeline().setStages([linSVC])

# Définition de la grille des hyperparamètres
paramGrid_svc = ParamGridBuilder().addGrid(linSVC.regParam, [0.1, 0.3, 0.5, 0.6, 0.8]) \
    .addGrid(linSVC.maxIter, [5, 10, 15, 20]) \
    .build()

cv_svc = CrossValidator().setEstimator(pipeline_svc) \
    .setEstimatorParamMaps(paramGrid_svc) \
    .setNumFolds(5) \
    .setEvaluator(BinaryClassificationEvaluator())
cvSVCmodel = cv_svc.fit(train)
```

```
rf = RandomForestClassifier().setFeaturesCol("features").setLabelCol("label").setSeed(100)
pipelineRF = Pipeline().setStages([rf])

# Définition de la grille des hyperparamètres
paramGrid_rf = ParamGridBuilder() \
    .addGrid(rf.maxDepth, [3, 5, 8]) \
    .addGrid(rf.numTrees, [100, 125, 150]) \
    .addGrid(rf.minInstancesPerNode, [2, 5, 10]) \
    .addGrid(rf.featureSubsetStrategy, ["sqrt", "log2"]) \
    .build()

cv_rf = CrossValidator().setEstimator(pipelineRF) \
    .setEstimatorParamMaps(paramGrid_rf) \
    .setNumFolds(5) \
    .setEvaluator(BinaryClassificationEvaluator())
cvRFmodel = cv_rf.fit(train)

# Importances des variables pour le meilleur modèle
best_RFmodel = cvRFmodel.bestModel.stages[-1]
feature_importances = best_RFmodel.featureImportances
print(feature_importances)
```

Annexe 17 : Comparaison des performances de différents modèles pour la classification binaire

```

models = {
    'LinearSVC': cvSVCmodel,
    'RandomForest': cvRFmodel,
    'LogisticRegression': cvLRmodel
}

evaluators = {
    'Accuracy': MulticlassClassificationEvaluator(metricName="accuracy"),
    'F1': MulticlassClassificationEvaluator(metricName="f1"),
    'ROC': BinaryClassificationEvaluator(metricName="areaUnderROC")
}

def evaluate_models(train, test, models, evaluators):
    results = []

    for model_name, model in models.items():
        # Collection des meilleurs modèles pour chaque type
        best_model = model.getEstimator().copy(model.bestModel.extractParamMap())
        best_model = best_model.fit(train)

        # Prédictions sur les données de test
        predictions_train = model.transform(train)
        predictions_test = model.transform(test)

        # Evaluation des prédictions
        for metric_name, evaluator in evaluators.items():
            metrics_train = {
                'Model': model_name,
                'Dataset': 'Train',
                'Metric': metric_name,
                'Value': evaluator.evaluate(predictions_train)
            }
            metrics_test = {
                'Model': model_name,
                'Dataset': 'Test',
                'Metric': metric_name,
                'Value': evaluator.evaluate(predictions_test)
            }

            # Store the results
            results.append(metrics_train)
            results.append(metrics_test)

        # Plot courbe ROC
        roc_data = predictions_test.select("label", "rawPrediction").rdd.map(lambda row:
(float(row["rawPrediction"])[1]), float(row["label"]))).collect()
        roc_df = pd.DataFrame(roc_data, columns=["Probability", "Label"])
        roc_df = roc_df.sort_values(by="Probability", ascending=False)
        roc_df["TPR"] = roc_df["Label"].cumsum() / roc_df["Label"].sum()
        roc_df["FPR"] = (1 - roc_df["Label"]).cumsum() / (1 - roc_df["Label"]).sum()
        roc_auc = evaluators['ROC'].evaluate(predictions_test)
        plt.plot(roc_df['FPR'], roc_df['TPR'], label=f'{model_name} (AUC = {roc_auc:.2f})')

        plt.ylabel('True Positive Rate')
        plt.xlabel('False Positive Rate')
        plt.title('ROC Curves')
        plt.legend(loc='best')
        plt.plot([0, 1], [0, 1], 'k--', color='grey', label='Random Classifier (AUC = 0.5)')
        plt.show()

    # Convert results to a DataFrame
    results_df = pd.DataFrame(results)
    return results_df

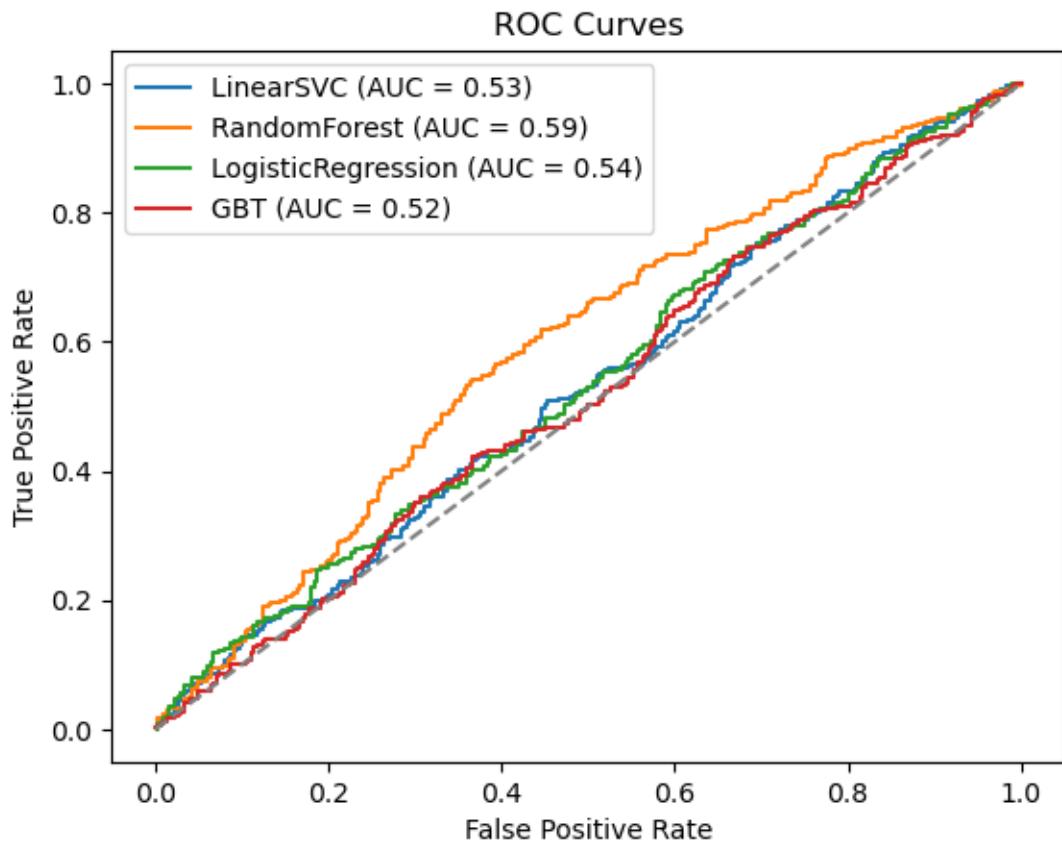
results_df = evaluate_models(train, test, models, evaluators)

```

Annexe18 : Performances de modèles de classification binaires entraînés sur l'échantillon thématique 'CRISPR'

	Ech. Apprentissage			Ech. Test		
	Accuracy	F1	AUC	Accuracy	F1	AUC
LinearSVM	0.641	0.640	0.658	0.603	0.604	0.594
RandomForest	0.646	0.515	0.923	0.639	0.499	0.575
LogisticRegression	0.696	0.666	0.731	0.630	0.594	0.577
GBT	0.991	0.991	1.000	0.584	0.558	0.538

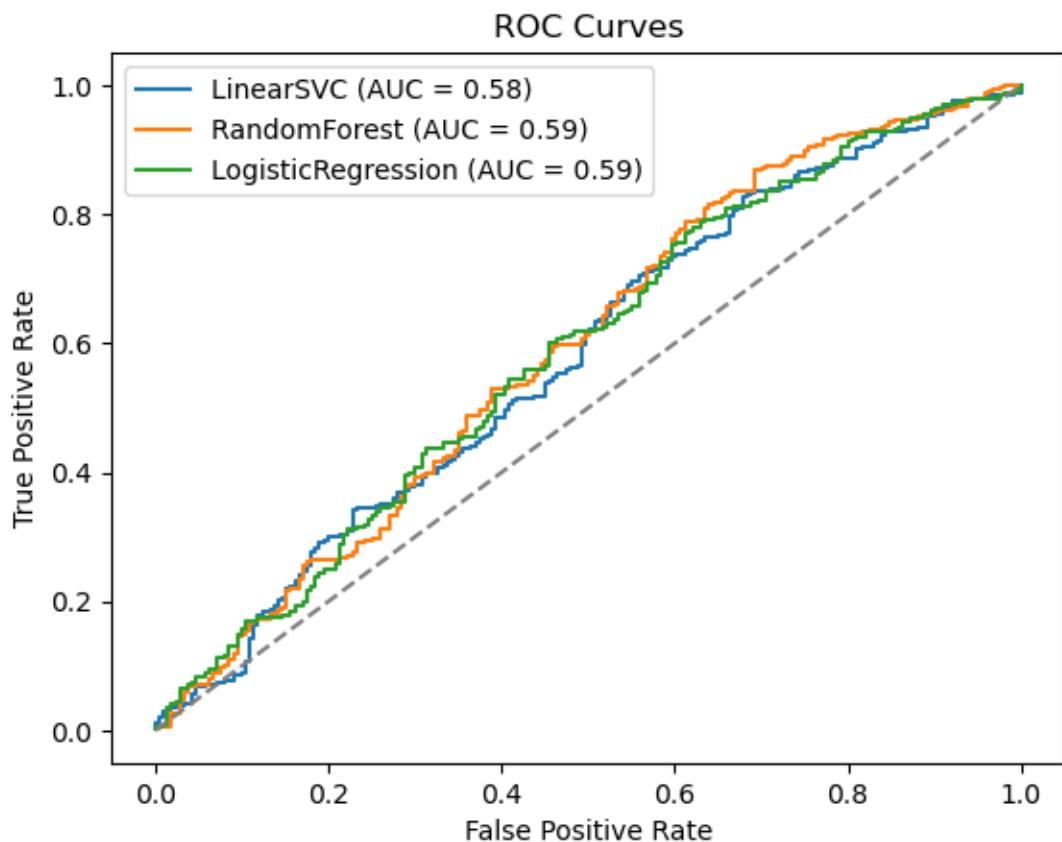
Résultats pour 4 modèles différents.



Annexe19 : Performances de modèles de classification binaires entraînés sur l'échantillon thématique 'Alzheimer'

	Ech. Apprentissage			Ech. Test		
	Accuracy	F1	AUC	Accuracy	F1	AUC
LinearSVM	0.633	0.634	0.659	0.592	0.592	0.579
RandomForest	0.647	0.544	0.948	0.634	0.497	0.591
LogisticRegression	0.630	0.550	0.669	0.641	0.564	0.586

Résultats pour 3 modèles différents.



Annexe 20 : Sélection des meilleurs modèles de régression par validation croisée

```
# Partitionner les données en apprentissage (80%) et test (20%)
partitions = df_reg.randomSplit([0.8, 0.2], seed=100)
train = partitions[0].cache() # conservé en mémoire
test = partitions[1]
```

```
glr = GeneralizedLinearRegression(family="tweedie").setFeaturesCol("features").setLabelCol("citlyear")
pipeline_glr = Pipeline().setStages([glr])

# Définition de la grille des hyperparamètres
paramGrid_glr = (ParamGridBuilder()
    .addGrid(glr.regParam, [0.001, 0.01, 0.1])
    .addGrid(glr.maxIter, [25, 50, 100])
    .build())

cv_glr = CrossValidator().setEstimator(pipeline_glr) \
    .setEstimatorParamMaps(paramGrid_glr) \
    .setNumFolds(5) \
    .setEvaluator(RegressionEvaluator(metricName="mae", labelCol="citlyear"))

cvGLRmodel = cv_glr.fit(train)
```

```
rf = RandomForestRegressor().setFeaturesCol("features").setLabelCol("citlyear").setSeed(100)
pipelineRF = Pipeline().setStages([rf])

paramGrid_rf = ParamGridBuilder() \
    .addGrid(rf.numTrees, [25, 50, 100]) \
    .addGrid(rf.maxDepth, [2, 5, 10]) \
    .addGrid(rf.minInstancesPerNode, [2, 5, 10, 15]) \
    .addGrid(rf.featureSubsetStrategy, ["sqrt", "log2"]) \
    .build()

# Définition de la grille des hyperparamètres
cv_rf = CrossValidator().setEstimator(pipelineRF) \
    .setEstimatorParamMaps(paramGrid_rf) \
    .setNumFolds(5) \
    .setEvaluator(RegressionEvaluator(metricName="mae", labelCol="citlyear"))

cvRFmodel = cv_rf.fit(train)
```

Annexe 21 : Comparaison des performances de différents modèles pour la régression

```

models = {
    'LinearGM': cvGLRmodel,
    'RandomForest': cvRFmodel
}

evaluators = {
    'MAE': RegressionEvaluator(metricName="mae", labelCol="cit1year"),
    'RMSE': RegressionEvaluator(metricName="rmse", labelCol="cit1year"),
    'R2': RegressionEvaluator(metricName="r2", labelCol="cit1year")
}

def evaluate_models(train, test, models, evaluators):
    results = []

    # Creation figure 'scatter plot'
    n=len(models)
    fig, axes = plt.subplots(1, n, figsize=(18, 6))

    for model_name, model in models.items():
        # Collection des meilleurs modèles pour chaque type
        best_model = model.getEstimator().copy(model.bestModel.extractParamMap())
        best_model = best_model.fit(train)

        # Prédictions sur les données de d'apprentissage et de test
        predictions_train = model.transform(train)
        predictions_test = model.transform(test)

        # Evaluation des prédictions
        for metric_name, evaluator in evaluators.items():
            metrics_train = {
                'Model': model_name,
                'Dataset': 'Train',
                'Metric': metric_name,
                'Value': evaluator.evaluate(predictions_train)
            }
            metrics_test = {
                'Model': model_name,
                'Dataset': 'Test',
                'Metric': metric_name,
                'Value': evaluator.evaluate(predictions_test)
            }
            results.append(metrics_train)
            results.append(metrics_test)

        # Scatter plot Réel VS Prediction
        predictions_scatter = predictions_test.select("cit1year", "prediction").toPandas()
        index = list(models.keys()).index(model_name)
        axes[index].scatter(predictions_scatter["cit1year"],
                            predictions_scatter["prediction"], alpha=0.6, color='blue')
        axes[index].plot([min(predictions_scatter["cit1year"]),
                          max(predictions_scatter["cit1year"])],
                        [min(predictions_scatter["cit1year"]), max(predictions_scatter["cit1year"])],
                        color='red', linestyle='--')
        axes[index].set_title(f"Modèle de régression: {model_name}")
        axes[index].set_xlabel("Valeurs réelles")
        axes[index].set_ylabel("Valeurs prédictées")

    plt.tight_layout()
    plt.show()

    results_df = pd.DataFrame(results)
    return results_df

results_df = evaluate_models(train, test, models, evaluators)
print(results_df)

```