Modélisation des sujets avec Apache Spark ML sur un jeu de données StockOverflow

Guy Anthony NAMA NYAM

Mars 2022

1 Introduction

La fouille de textes communément connue sous la terminologie de NLP pour Natural Language Processing permet de traiter et d'exploiter le gisement de textes disponibles sur différentes sources(base de données, page web, etc.). La fouille de textes a une place prépondérante dans le domaine de l'apprentissage automatique de part son vivier d'applications (topic modeling, traduction, l'analyse des sentiments, chatbots, etc...).

1.1 Mise en situation

Le site StackOverflow est une plateforme de questions-réponses liée au développement informatique. Pour poser une question sur ce site, il faut entrer plusieurs étiquettes, mots clés ou sujets liés à la question.

1.2 Problématique

Dans la communauté Stack Overflow, les utilisateurs (généralement les débutants) veulent poser des questions sans toutefois avoir une idée précise du tag (sujet) de la question. Il s'agit d'aider la communauté en développant un système non supervisé qui assigne automatiquement des tags à leur question textuelle.

1.3 Réalisation

Dans le cadre du projet RCP216, j'ai bien voulu partir d'une de mes anciennes réalisations dont voici le lien (https://github.com/AnthonyNama/NLP–Categoriser-automatiquement-des-questions) afin de l'améliorer mais aussi d'avoir une appréciation du framework ou logiciel "SPARK" du point de vue performance en local (temps de calcul et mémoire). De l'existant, voici des possibles contributions :

1. Quelques modifications dans le nettoyage dans la recherche de meilleure performance.

2. Implémentation avec le framework spark : en effet, c'est l'un des apports majeurs dans le sens que je n'avais jamais implémenté une application (distribuée) avec ce framework ; de plus j'ai réalisé l'implémentation en python (pyspark) au détriment de scala du TP.
3. Modélisation : par rapport à l'existant, un nouvel algorithme est implémenté "latent semantics analysis" (LSA) ; De plus la bibliothèque "gensim" utilisée dans l'existant enveloppe l'étape des SVD.

Dans ce travail, nous traiterons des extractions des thèmes (topic modeling)

2 Source de données

Nous Commençons par une description de la source de données et des données.

2.1 Description de la source de données

Les données à usage public ont été importées depuis la plateforme "stackexchange explorer" mise à disposition par StackOverflow. Nous avons requêté la table "Posts" contenant les posts des utilisateurs de ce site. Le choix des données sélectionnées porte sur les posts les plus sollicités. (SELECT * FROM Posts WHERE FavoriteCount $>$ 150). Le choix de '150' précédent est basé sur le nombre de documents raisonnables pour les ressources disponibles. La taille du fichier est d'environ 5,8 Mo.

2.2 Description de la donnée

Les données recueillies comportent 5240 posts soient 5240 documents dans l'approche LSA avec un nombre moyen de 122 mots par document ("Body" + "Title"). Pour les besoins de notre travail, seul quelques colonnes ont été conservées à savoir :

1. "Body" : le corps de la question
2. "Title" : l'entête de la question (autrement dit l'objet)
3. "Tags" : les étiquettes associées aux questions (colonne pas vraiment utilisée dans ce travail)

3 Préparation de la donnée

C'est une étape fondamentale dans la mise œuvre d'un algorithme d'apprentissage automatique car plus la qualité des données est bonne mieux est les performances de régression ou de classification.

```
[ ] file = "data/posts.csv"

[ ] df = pd.read_csv(file)

[ ] df = spark.createDataFrame(df)
```

FIGURE 1 – Création du dataframe spark à partir de la source

3.1 Nettoyage de la donnée

Nous avons commencé par charger les données d'un fichier csv passant par un dataframe Pandas puis vers un dataframe spark :

Affichage des 5 premières lignes :

```
df.show(5)
```

```
+---+--------------------+--------------------+--------------------+
| Id|                Body|               Title|                Tags|
+---+--------------------+--------------------+--------------------+
|  0|<p>What's the cle...|Validate decimal ...|<javascript><vali...|
|  1|<p>I'm designing ...|List of standard ...|<database><standa...|
|  2|<p>What's the eas...|Simplest way to p...|    <php><profiling>|
|  3|<p>Is there any w...|How can I develop...|<ios><iphone><win...|
|  4|<p>I am looking f...|Batch file to del...|<windows><date><b...|
+---+--------------------+--------------------+--------------------+
only showing top 5 rows
```

FIGURE 2 – Affichage des 5 premières lignes du jeu de données

Dimension de la donnée :

```
def shapeDataFrame(d):
    print((d.count(), len(d.columns)))
```

```
shapeDataFrame(df)
```

```
(5240, 4)
```

FIGURE 3 – Les dimensions de notre jeu de donnée initial

3.1.1 Suppression des tags HTML

Avant d'appliquer la première opération de nettoyage, la concaténation des colonnes "Body" et "Title" a été appliquée pour obtenir une colonne "Corpus".

Dans cette colonne, on peut bien constater l'absence des balises "<p>" présentes dans la colonne "Body" de la figure 4.

L'utilisation de la bibliothèque ***BeautifulSoup*** permet la suppression des balises HTML et XML.

```
from bs4 import BeautifulSoup
```

```
def delete_html_tags(row):
    corpus = BeautifulSoup(str(row['Corpus']), 'html.parser').get_text()
    return (row["Id"], row['Body'], row['Title'], row['Tags'], corpus)
```

```
df = df.rdd.map(delete_html_tags).toDF(["Id", "Body", "Title", "Tags",
                                        "Corpus"])
```

```
df.show(5)
```

```
+---+--------------------+--------------------+--------------------+--------------------+
| Id|                Body|               Title|                Tags|              Corpus|
+---+--------------------+--------------------+--------------------+--------------------+
|  0|<p>What's the cle...|Validate decimal ...|<javascript><vali...|What's the cleane...|
|  1|<p>I'm designing ...|List of standard ...|<database><standa...|I'm designing a d...|
|  2|<p>What's the eas...|Simplest way to p...|    <php><profiling>|What's the easies...|
|  3|<p>Is there any w...|How can I develop...|<ios><iphone><win...|Is there any way ...|
|  4|<p>I am looking f...|Batch file to del...|<windows><date><b...|I am looking for ...|
+---+--------------------+--------------------+--------------------+--------------------+
only showing top 5 rows
```

FIGURE 4 – Suppression des balises html et xml sur le corpus : Body + Title

3.1.2 Mise en minuscule et tokenisation

Une opération séparée permettant de mettre le texte en minuscule a été implémentée, néanmoins le tokenizer de spark réalise d'abord cette opération avec de tokenizer le texte. La colonne "Corpus" présente Voilà l'opération de tokenisation et une sortie.

3.1.3 Identification et remplacement des abréviations

Il s'agit de remplacer les abréviations par leur forme longue. L'algorithme ***Schwartz-Hearst*** a été utilisé pour l'identification des formes courtes. Après l'identification d'abréviations par l'algorithme, un filtrage manuel peut s'avérer pertinent pour la qualité des données après remplacement. Un exemple d'extrait d'abréviations identifiées :

3.1.4 Mappage des formes contractées

Pour cette opération, nous avons utilisé un corpus externe contenant les formes contractées (source : "List of English contractions") et leurs formes

```
tokenizer_corpus = Tokenizer(inputCol="Corpus", outputCol="Corpus_tokenized")
```

```
df = tokenizer_corpus.transform(df)
```

```
df.select("Corpus", "Corpus_tokenized").show(5)
```

```
+--------------------+--------------------+
|              Corpus|    Corpus_tokenized|
+--------------------+--------------------+
|what's the cleane...|[what's, the, cle...|
|i'm designing a d...|[i'm, designing, ...|
|what's the easies...|[what's, the, eas...|
|is there any way ...|[is, there, any, ...|
|i am looking for ...|[i, am, looking, ...|
+--------------------+--------------------+
only showing top 5 rows
```

FIGURE 5 – Tokenisation du corpus et affichage

```
dic_abbreviations
```

```
 'CA': 'Certificate Authority',
 'CDI': 'Contexts and Dependency Injection',
 'CGI': 'Common Gateway Interface',
 'CI': 'Continuous Integration',
 'CLI': 'command line',
 'CORS': 'cross-origin resource sharing',
 'CRTP': 'curiously recurring template pattern',
 'CSP': 'Content Security Policy',
 'CSRF': 'cross-site request forgery',
```

FIGURE 6 – Exemple d'extrait d'abréviations identifié

longues. Une manière alternative d'ajouter une nouvelle colonne autre que la fonction "map" a été utilisée. L'usage des fonctions définies par utilisateurs (udf). Dans la figure 7, on a les formes contractées "what's" et "i'm" de la première colonne qui ont été remplacées respectivement par "what has" et "i am" de la colonne "Corpus_remove_contractions". D'autres formes encores plusieurs intéressantes ont été changées dans le reste du corpus.

3.1.5 Suppression des STOPWORDS

L'implémentation de spark a été utilisée pour cette transformation. Nous pouvons apercevoir la transformation dans la figure 8 sur la deuxième colonne où les mots "what', "has", "the", "i", "am", "is", "there" et "any" ont été supprimés.

```
udfremove_contract_form = F.udf(remove_contract_form, ArrayType(StringType()))
```

```
df = df.withColumn("Corpus_remove_contractions",
                   udfremove_contract_form("Corpus_remove_abbreviations"))
```

```
df.select("Corpus_remove_abbreviations",
          "Corpus_remove_contractions").show(5)
```

```
[Stage 20:>                                                         (0 + 1) / 1]
```

```
+--------------------------+-------------------------+
|Corpus_remove_abbreviations|Corpus_remove_contractions|
+--------------------------+-------------------------+
|       [what's, the, cle...|      [what, has, the, ...|
|       [i'm, designing, ...|      [i, am, designing...|
|       [what's, the, eas...|      [what, has, the, ...|
|       [is, there, any, ...|      [is, there, any, ...|
|       [i, am, looking, ...|      [i, am, looking, ...|
+--------------------------+-------------------------+
only showing top 5 rows
```

FIGURE 7 – Gestion des formes contractées

```
df = remover_corpus.transform(df)
```

```
df.select("Corpus_remove_contractions",
          "Corpus_remove_STOP_WORDS").show(5)
```

```
[Stage 21:>                                                         (0 + 1) / 1]
```

```
+-------------------------+-----------------------+
|Corpus_remove_contractions|Corpus_remove_STOP_WORDS|
+-------------------------+-----------------------+
|      [what, has, the, ...|    [cleanest,, effec...|
|      [i, am, designing...|    [designing, datab...|
|      [what, has, the, ...|    [easiest, way, pr...|
|      [is, there, any, ...|    [way, tinker, iph...|
|      [i, am, looking, ...|    [looking, way, de...|
+-------------------------+-----------------------+
only showing top 5 rows
```

FIGURE 8 – Suppression des mots courants et affichage

3.1.6 Suppression des symboles spéciaux

On y entend par symboles spéciaux tous les caractères qui ne sont pas les lettres de l'alphabet et le caractère espace. Nous avons illustré la suppression des symboles dits spéciaux dans la figure 9. Malgré que le mot "c++" étant bien connu des développeurs, le caractère "+" sera supprimé et donnera le mot "c" qui créera une confusion avec le langage "c" (ce sont les inconvénients de l'automatisation). On a aussi les parenthèses qui seront supprimées, on passe de "(g)vim" à "gvim).

```
df.where(col("Id").between(10, 15)).select("Corpus_remove_STOP_WORDS",
                                           "Corpus_remove_special_characters").show(5)
```

```
+-----------------------+------------------------------+
|Corpus_remove_STOP_WORDS|Corpus_remove_special_characters|
+-----------------------+------------------------------+
|    [want, know, "vir...|            [want, know, virt...|
|    [c, c++, programm...|            [c, c, programmin...|
|    [favorite, (g)vim...|            [favorite, gvim, ...|
|    [long, ago, begin...|            [long, ago, begin...|
|    [difference, unio...|            [difference, unio...|
+-----------------------+------------------------------+
only showing top 5 rows
```

FIGURE 9 – Suppression des symboles et caractères spéciaux puis affichage

3.1.7 Suppression des mots de taille faible

Généralement les mots de taille 1 voir 2 ne sont pas suffisamment porteurs d'informations. Nous avons fait le choix de leur suppression d'autant plus qu'ils sont énormément générés lors des transformations précédentes à l'exemple de "c++" devenu "c", qui est supprimé comme dans la figure 10

```
df.where(col("Id").between(10, 15)).select("Corpus_remove_special_characters",
                                           "Corpus_remove_length_less3").show(5)
```

```
+------------------------------+-------------------------+
|Corpus_remove_special_characters|Corpus_remove_length_less3|
+------------------------------+-------------------------+
|            [want, know, virt...|      [want, know, virt...|
|            [c, c, programmin...|      [programming, lan...|
|            [favorite, gvim, ...|      [favorite, gvim, ...|
|            [long, ago, begin...|      [long, ago, begin...|
|            [difference, unio...|      [difference, unio...|
+------------------------------+-------------------------+
only showing top 5 rows
```

FIGURE 10 – Suppression des mots de taille inférieur à 3

3.1.8 Lemmatisation

La lemmatisation est un traitement lexical d'un mot pour le mettre dans sa forme canonique. Elle n'est pas implémentée dans spark, nous avons utilisé la bibliothèque "spacy" pour cette transformation. Dans la figure 11, les mots "cleanest", "designing", "easiest" et "looking" ont été remplacés par leurs lemmes respectivement "clean", "design", "easy" et "look".

```
nlp = spacy.load('en_core_web_sm', disable=["parser", "ner", "textcat"])
```

```
def lemmatization(values):
    words = []
    for val in values:
        doc = nlp(val)
        words.append(doc[0].lemma_)
    return words
```

```
udflemmatization = F.udf(lemmatization, ArrayType(StringType()))
```

```
df = df.withColumn("Corpus_lemmatized",
                   udflemmatization("Corpus_remove_length_less3"))
```

```
df.select("Corpus_remove_length_less3", "Corpus_lemmatized").show(5)
```

```
[Stage 24:>                                                         (0 + 1) / 1]
+-------------------------+--------------------+
|Corpus_remove_length_less3|   Corpus_lemmatized|
+-------------------------+--------------------+
|      [cleanest, effect...|[clean, effective...|
|      [designing, datab...|[design, database...|
|      [easiest, way, pr...|[easy, way, profi...|
|      [way, tinker, iph...|[way, tinker, iph...|
|      [looking, way, de...|[look, way, delet...|
+-------------------------+--------------------+
only showing top 5 rows
```

FIGURE 11 – Lemmatisation du corpus

3.2 Visualisation

Le corpus final après prétraitement contient 26 484 mots distincts avec une distribution allant de 1 à 5365 occurrences. La figure 12 présente la distribution des mots du corpus sous forme de nuage de mots. Parmi les mots les plus fréquents dans le corpus, nous avons : "use", "file", "like", "code", "get", etc.

4 Modélisation

A ce niveau, nous considérons chaque ligne de notre dataframe comme un document et nous allons construire la matrice documents-termes avec la pondération Tf-ifd.

4.1 Tf-idf

Nous commençons par construire la matrice de documents-termes pour les fréquences des termes Tf. Nous avons défini à 2000 (environ 1/13 des mots du

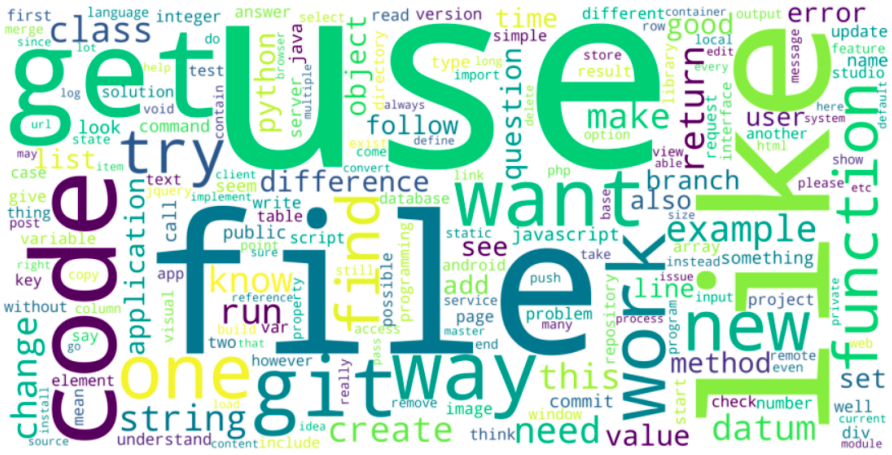


FIGURE 12 – Nuage de mots pour la distribution du corpus

corpus) la taille du vocabulaire (mots plus fréquents). La taille du vocabulaire est assez faible comparée au nombre de mots du corpus mais contraint par les ressources disponibles. La sélection de la taille du vocabulaire correspond à une sorte de suppression des mots de fréquences inférieur à un seuil que nous avons réalisé plus haut. Un extrait du code est présenté ci-dessous :

```
vocab_size = 2000
```

```
cv = CountVectorizer(inputCol="Corpus_lemmatized", outputCol="rawFeatures",
                     vocabSize=vocab_size)
```

```
cv_model = cv.fit(data)
```

```
featurizedData = cv_model.transform(data)
```

```
featurizedData.where(col("Id").between(35, 40)).select("rawFeatures").show(4, truncate=False)
```

```
[Stage 34:>                                                         (0 + 1) / 1]
+----------------------------------------------------------------------------------------------------------+
|rawFeatures                                                                                               |
+----------------------------------------------------------------------------------------------------------+
|(2000,[4,50,128,270,699,897,1552],[2.0,2.0,1.0,2.0,2.0,2.0,4.0])                                          |
|(2000,[0,18,57,108,109,683,711],[2.0,2.0,1.0,2.0,2.0,1.0,2.0])                                            |
|(2000,[0,9,26,280,374],[1.0,1.0,2.0,1.0,2.0])                                                             |
|(2000,[0,9,31,202,250,275,280,328,411,463,886,901],[1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,1.0,2.0])|
+----------------------------------------------------------------------------------------------------------+
only showing top 4 rows
```

FIGURE 13 – Matrice de comptage des termes par document

Nous allons partir de la matrice documents-termes précédente pour obtenir les pondérations idf. Nous pouvons voir dans l'exemple de la figure 14 que les pondérations tf-idf implémentées peuvent dépasser 1.

```
idf = IDF(inputCol="rawFeatures", outputCol="features")
idfModel = idf.fit(featurizedData)
```

```
documentTermMatrix = idfModel.transform(featurizedData)
```

```
documentTermMatrix.where(col("Id").between(86, 86)).select("features").show(1,truncate=False)
```

```
[Stage 306:==============================================>          (4 + 1) / 5]
+-----------------------------------------------------------------------------+
|features                                                                     |
+-----------------------------------------------------------------------------+
|(2000,[9,12,26],[1.5888536713462145,2.1773882794395214,4.237095558833176])|
+-----------------------------------------------------------------------------+
```

FIGURE 14 – Matrice document-term contenant les pondérations tfidf

4.2 Choix du nombre de sujets

Nous cherchons le nombre de concept pertinent pour notre matrice document-terme. Pour celà, nous calculons la décomposition correspondant au nombre de colonnes(2000) de la matrice de documents-termes, puis nous choisissons le nombre de concept de variance cumulée explicative supérieur ou égal au seuil choisi. Nous avons fixé le seuil à 0.95. Nous avons obtenu comme mentionné sur la figure 15 un nombre de concept égal à 1100. Le temps d'exécution est d'environ 278s sans calculer la matrice gauche(U) de la décomposition en valeur singulière, soit moins de 5 min. Ce temps reste très raisonable au vu du corpus et des ressources.

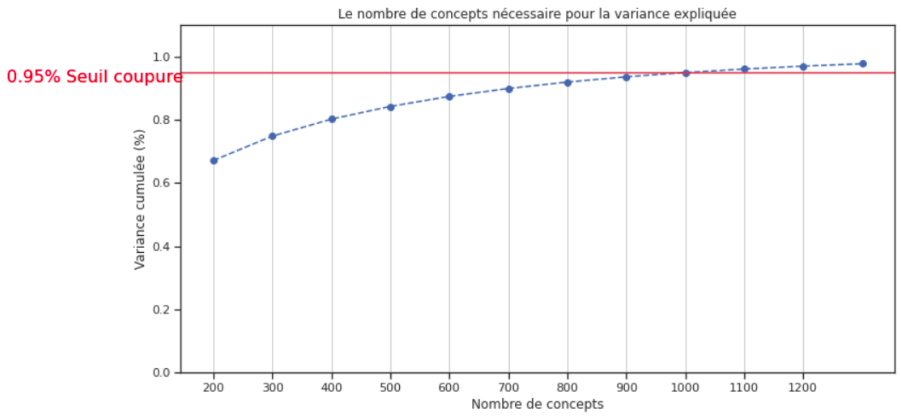


FIGURE 15 – Evolution de la variance cumulée en fonction du nombre de concept

4.3 Latent semantics analysis

L'analyse sémantique latente est une application de la décomposition en valeurs singulières (SVD). Cette décomposition est implémentée dans spark. En partant de la matrice des pondérations Tf-idf, on calcule les matrices U, S et V en précisant le nombre de composantes voulu comme mentionné dans la figure 16. Le temps d'exécution est 291s avec calcul de la matrice U, soit toujours moins de 5 minutes sur une matrice de dimension 5240 * 2000 pour 1100 sujets. L'expérimentation a été réalisée sur une machine de 8 CPU de 1.6 Ghz et 8Go de RAM.

```
n_topics
```

```
1100
```

```
mat = RowMatrix(documentTermMatrix.select("features").rdd. \
                map(lambda v: Vectors.dense(v.features.toArray())).cache())
```

```
start = time.time()
svd = mat.computeSVD(n_topics, computeU=True)
print(str(time.time() - start) + 's')
```

```
291.4759421348572s
```

FIGURE 16 – Api pyspark pour calculer la décomposition en valeur singulière

4.4 Résultats

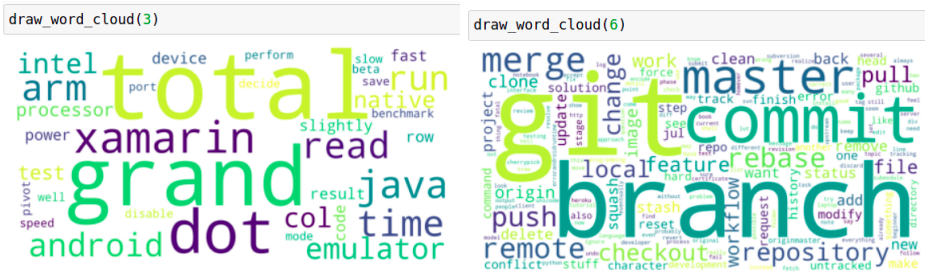
Dans cette partie, nous allons nous intéresser aux 6 premiers sujets (sur les 1100 déterminés) par leurs 10 premiers termes qui nous semblent interprétables mais aussi pour une simplification de l'analyse comme présenté sur la figure 17.

```
display_top_terms(topics_terms, 10, 6)
```

```
Topic 1: hover errorwindowmanager href clipboard checkbox onclick dropdown uitextview dosomethe blah

Topic 2: total grand loadmodule file dot run xamarin java time read

Topic 3: total grand dot xamarin java arm time read run android

Topic 4: null public php git operator branch insert strtok integer image

Topic 5: null strtok insert serrmsg sqliteexecdb sqlite second branch git transaction

Topic 6: git branch commit master merge remote push rebase repository checkout
```

FIGURE 17 – Les termes les plus pertinents des sujets identifiés du corpus

Les sujets identifiés peuvent aussi être représentés sous forme de nuage de mots représentatif de la distribution de pertinente du mot dans le sujet. La figure 18 donne l'exemple deux exemples du nuage des mots pour les sujets 5 et 8.



(a) word cloud du sujet 3 (b) word cloud du sujet 6

FIGURE 18 – Nuage de mots de deux exemples de sujets

Nous avons dans la figure 19 suivante la représentation graphique des 10 premiers termes de chaque sujet pour visualiser l'homogénéité des sujets. Nous avons utilisé l'analyse en composantes principales pour la réduction de dimension qui est implémentée dans pyspark et en annexe la visualisation des mêmes termes en utilisant les techniques de réductions que sont "T-SNE"(T-distributed Stochastic Neighbor Embedding)25 et "UMAP"(Uniform Manifold Approximation and Projection)26 qui ne sont pas implémentées dans pyspark au moment de cette rédaction. Les données d'entrée ont d'abord été normalisées (centré et réduit) avant d'appliquer la technique de réduction de dimension.

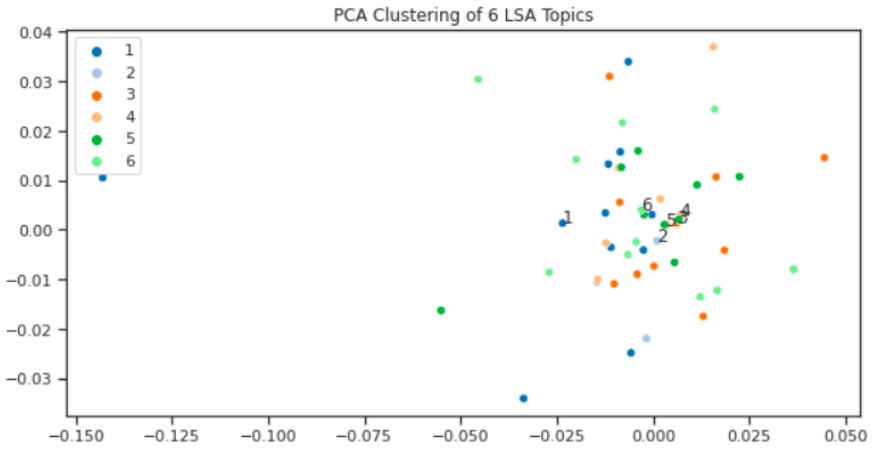


FIGURE 19 – Sujets identifiés du corpus d'entrée

Comme autre résultat direct, nous pouvons aussi présenter les documents les plus pertinents pour les différents sujets. Nous avons représenté les 3 premiers documents(représentés par leur titre) les plus pertinents dans les 6 premiers sujets. Le séparateur entre documents est l'espace multiple.

```
display_top_docs(topics_docs, 3, 6)
```

```
Topic 1: What is Mocking?   Favorite (G)Vim plugins/scripts?   Capturing mobile phone traffic on Wireshark


Topic 2: How to compare the performance of Android Apps written in Java and Xamarin C#? Anyway to check quantitati
ve data (code & results)   Error message "Forbidden You don't have permission to access / on this server"   Twitte
r image encoding challenge


Topic 3: How to compare the performance of Android Apps written in Java and Xamarin C#? Anyway to check quantitati
ve data (code & results)   How can I pivot a dataframe?   In PHP, what is a closure and why does it use the "use"
identifier?


Topic 4: Improve INSERT-per-second performance of SQLite   Reference — What does this symbol mean in PHP?   Twitte
r image encoding challenge


Topic 5: Improve INSERT-per-second performance of SQLite   How do I achieve the theoretical maximum of 4 FLOPs per
cycle?   Git workflow and rebase vs merge questions


Topic 6: Git workflow and rebase vs merge questions   Various ways to remove local Git changes   Git: Create a bra
nch from unstaged/uncommitted changes on master
```

FIGURE 20 – Les documents les plus pertinents des sujets identifiés du corpus

4.5 Interprétation

Les résultats des sujets sont décrits par les premiers termes affectés au sujet. Les termes sont classés par ordre croissant de pertinence. Par exemple, dans le sujet 4, le terme "null" décrit au mieux le sujet que le terme "public" et ainsi de suite. Basé sur les termes décrivant les sujets et de part nos connaissances, nous pouvons déduire les valeurs suivantes de thèmes pour les 6 premiers sujets :

Sujet 1 : user interface
Sujet 2 : programmation java
Sujet 3 : programmation mobile (android java)
Sujet 4 : programmation php
Sujet 5 : base de données
Sujet 6 : gestion de version

Comme on peut le constater, on a les mêmes termes qui figurent dans différents sujets simultanément (par exemple, "null", "insert" et "git" dans les topic 4 et 5). Celà se confirme dans la représentation graphique de la figure 19 où nous avons des termes dans l'espace assez proche pour différents sujets. Nous rappelons que nous avons utilisé l'analyse en composantes principales pour passer de l'espace de dimension 1100 à 2. Ceci peut s'expliquer par le fait que les différents documents tournent autour d'un même "root" sujet à savoir l'informatique car la méthode LSA engendre des biais de modélisation lorsqu'il y'a des dépendances entre les documents.

5 Analyse des similarités

Nous avons étudié quatre cas d'analyse de similarité à partir de la décomposition en valeurs singulières (U, S, V) à savoir : la similarité terme à terme, la similarité document à terme, la similarité document à document et enfin la similarité document à un groupe de termes.

5.1 Termes similaires à un mot/terme

La tâche de trouver l'ensemble des termes les plus similaires à un terme particulier revient à trouver la similarité cosinus entre un terme et tous les autres termes. On commence par normaliser les lignes de la matrice réduite $V_k S_k$ à norme 1, puis à multiplier la ligne correspondant à ce terme par les autres lignes de la matrice réduite précédente. Nous avons illustré cette tâche avec l'exemple suivant ; nous avons considéré le terme "remote" et nous avons obtenu les 8 premiers termes similaires suivants : "push", "git", "branch", "local", "origin", "master", "ref", et "repository ce qui correspond bien à un jargon proche du terme de départ renvoyant tous à la thématique de gestion de version.

```
top_term_scores = top_terms_for_term("remote", VS_normalized, 10)
```

```
22/03/26 10:44:52 WARN TaskSetManager: Stage 71 contains a task of
nded task size is 1000 KiB.
```

```
display_top_terms_term(top_term_scores, 8)
```

```
('push', 0.5783838429181447)
('git', 0.4855094328255168)
('branch', 0.4805367499749322)
('local', 0.4415034715702866)
('origin', 0.42098966160111445)
('master', 0.3767379929752571)
('ref', 0.33315826391965864)
('repository', 0.3307213424416697)
```

FIGURE 21 – Les 8 premiers termes les plus similaires au terme 'remote'

5.2 Documents similaires à un mot/terme

Pour déterminer la similarité des documents à un terme, nous devons multiplier la matrice réduire $U_k S_k$ correspondant à l'ensemble de documents au vecteur v_k correspondant au terme correspondant dans la matrice réduite V_k puis extraire les documents de scores les plus élevés. Dans l'illustration suivante, nous avons juste rapporté les titres correspondants aux questions et non tout le document (corps + titre). Ainsi pour le terme "Python", nous avons une

très bonne correspondance des questions qui en parlent, celà déjà visible sur le titre de la question.

```
top_doc_scores = top_docs_for_term("python", US_normalized,  svd, 10)
```

```
22/03/26 10:45:39 WARN TaskSetManager: Stage 111 contains a task of very large size (6344 KiB).
ended task size is 1000 KiB.
```

```
display_top_docs_term(top_doc_scores, 10)
```

```
(4786, 'What is the Python 3 equivalent of "python -m SimpleHTTPServer"', 0.8279069339605447)
(911, 'What are metaclasses in Python?', 0.8182019561618477)
(2857, 'Why do people write #!/usr/bin/env python on the first line of a Python script?', 0.813
(2472, 'Python vs Cpython', 0.6745112577144737)
(547, 'What is memoization and how can I use it in Python?', 0.6233820647321762)
(2289, 'Python coding standards/best practices', 0.6202038071948015)
(5036, 'Using both Python 2.x and Python 3.x in IPython Notebook', 0.6067655036872226)
(4990, 'How to install python3 version of package via pip on Ubuntu?', 0.5941687979665358)
(3135, 'If Python is interpreted, what are .pyc files?', 0.5868801368927662)
(3651, "Does Python have a string 'contains' substring method?", 0.5565902657784093)
```

FIGURE 22 – Documents les plus similaires au terme 'python'

5.3 Documents similaires à un autre document

Pour trouver les documents similaires à un autre, on s'intéresse à la matrice réduite U_k de la décomposition en valeurs singulières. On pondère les colonnes de U_k avec les valeurs singulières correspondantes S_k par un produit matriciel $(U_k S_k)$ puis on normalise chaque ligne à la norme 1. La similarité entre deux documents comme le cosinus des deux lignes correspondantes de la matrice normalisée. Dans l'exemple illustré ci-dessous, la première ligne représente le titre du document à chercher les documents similaires et le reste des lignes correspondent aux documents similaires représentés par leur titre. Nous avons pris un document dans la plage de 5240 documents de notre corpus, ici 89 et nous avons les 10 premiers résultats (document pivot inclus).

```
top_doc_scores = top_docs_for_doc(89, US_normalized, 10)
```

```

```

```
display_top_docs_doc(top_doc_scores, 10)
```

```
(89, 'Display number with leading zeros', 1.0)
(1123, 'How can I pad an integer with zeros on the left?', 0.49736048650522696)
(2430, 'Renaming files in a folder to sequential numbers', 0.4711960098708759)
(2339, 'Given a number, find the next higher number which has the exact same set of digits
r', 0.43914410937477333)
(1081, 'How to pad zeroes to a string?', 0.3726326024483606)
(1103, 'Extract substring in Bash', 0.34863378910674714)
(4534, 'Format number to always show 2 decimal places', 0.3372481487573161)
(1441, 'How do I replace NA values with zeros in an R dataframe?', 0.3333795171433901)
(1276, 'Best practice for using assert?', 0.3262286747880354)
(1340, 'Most efficient way to create a zero filled JavaScript array?', 0.3110292398851079)
```

FIGURE 23 – Documents les plus similaires au document 'Display number ...'

5.4 Documents similaires à un groupe de plusieurs mots

L'opération consiste à les colonnes de la matrice transposée de V_k par un vecteur colonne donc la valeur est nulle si le terme n'est pas présent dans le groupe de termes sinon par la fréquence inverse du document (idf). Puis on multiplie le résultat obtenu par la matrice des documents, plus précisément par $U_k S_k$ pour obtenir un vecteur des scores des documents. Nous avons utilisé la paire de terme suivante ('java', 'version') et nous avons obtenu les résultats documents suivants représentés par leur titre ci-dessous. Nous avons même au niveau des intitulés les documents qui évoquent à la fois des concepts des "java" et "version".

```
display_top_docs_terms(top_doc_scores, 10)
```

```
(1981, 'How to set or change the default Java (JDK) version on macOS?', 2.4168693948722586)
(4768, 'Mac OS X and multiple Java versions', 2.2034537016106066)
(3408, 'How to install Java 8 on Mac', 2.0717842555492845)
(2714, "What's the difference between getPath(), getAbsolutePath(), and getCanonicalPath() in Java?", 2.0690091150
374093)
(3624, 'How can I avoid Java code in JSP files, using JSP 2?', 2.066351633388028)
(1431, 'How to change to an older version of Node.js', 1.8934678642317915)
(4650, 'How to fix java.lang.UnsupportedClassVersionError: Unsupported major.minor version', 1.8860189508077791)
(1449, 'Why am I getting a NoClassDefFoundError in Java?', 1.8492388082045517)
(2074, 'Is Java "pass-by-reference" or "pass-by-value"?', 1.8023896245100193)
(2437, 'How do I install Java on Mac OSX allowing version switching?', 1.8018669445054547)
```

FIGURE 24 – Documents les plus similaires à la paire de termes ('java', 'version)

6 Conclusion

La fouille de textes nécessite énormément de ressources. Le nettoyage de textes est une opération critique pour les modèles s'appuyant sur les sacs de mots. Ce travail regorge d'une part les opérations de nettoyage qui constituent un bon tremplin pour bon nombre d'applications puis la modélisation de l'application réalisée.

De nombreuses difficultés ont été rencontrées pendant ce projet, la première et la plus indéniable est le manque de documentation python de spark en général mais plus encore sur la modélisation "latent semantic analysis".
L'autre difficulté surtout quand on est débutant avec le framework est la gestion des ressources. Bien que le framework exploite effectivement l'ensemble des CPU disponibles sur la machine ce qui est une bonne chose, par contre certaines tâches sont gourmandes en ressources mémoires, ce qui a entraîné pas mal de crash faute de mémoire malgré un corpus de taille de 5,8 Mo et les 8 Go de RAM à disposition.
L'implémentation spark de python est assez limitée ce qui pose parfois un problème de frontière entre une application "pure" spark et python. De ce qui a été réalisé, nous avons quasiment utilisé spark dans l'ensemble de l'implémentation excepté la visualisation car n'ayant pas vraiment identifié dans la documentation une bibliothèque dédiée à cette tâche. De notre point de vue, nous pouvons dire qu'une application distribuée est une application qui contient des parties d'implémentation distribuée.

L'autre difficulté parfois est l'interprétation des résultats de la décomposition (LSA) surtout quand on retrouve des mêmes mots dans différents sujets, encore pire quand ils ont le même ordre dans les différents sujets. Plusieurs critères existent dans la sélection du nombre de concept à déduire, nous avons choisi ce qui semble le plus naturel à savoir un regroupement des plus homogènes. Néanmoins, malgré une homogénéité de 95% l'analyse des composants partagent beaucoup de termes identiques avec le même ordre de pertinence pour plusieurs sujets. Ce qui nous amène à dire la méthode LSA n'est pas adaptée à ce jeu de données dues à la dépendance entre les documents. D'autres méthodes pourraient être explorées pour améliorer les résultats telles que PLSA, ou LDA voir ESA.

Nous avons pendant l'implémentation vu que l'analyse sémantique latente dépend énormément des pondérations. Il serait donc intéressant d'utiliser une autre pondération pour évaluer la pertinence de la pondération "inverse document frequency" (idf) implémenté dans spark. Durant ce projet, nous avons aussi étudié l'analyse des similarités de plusieurs cas à partir de la décomposition en valeur singulière ; C'est dire à quel point cette décomposition à une importance dans beaucoup d'applications. La mise en œuvre des cas de similarité est assez facile lorsqu'on a une bonne compréhension matricielles et de leurs manipulations.

Le mot de fin porte sur le déploiement de l'implémentation réalisée, il serait intéressant de la déployer sur un cluster de machine et non plus localement pour apprécier les avantages du framework spark sur des données massives.

A T-SNE

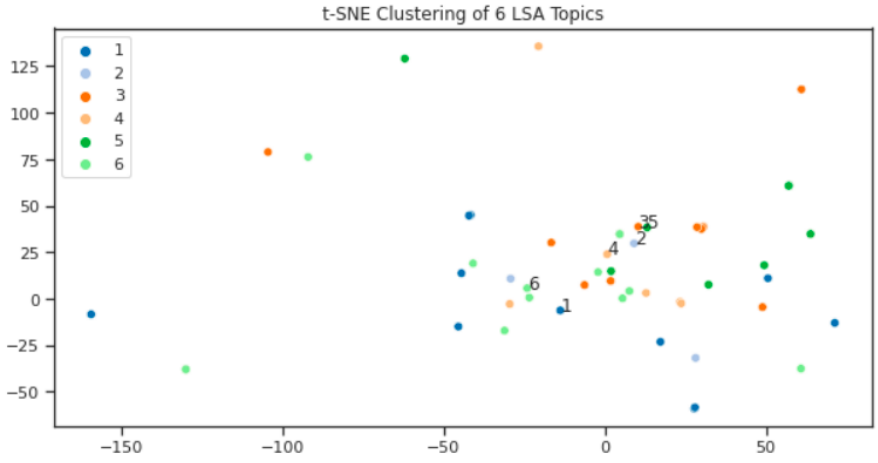


FIGURE 25 – T-SNE pour les 10 premiers termes des 6 premiers sujets

B UMAP

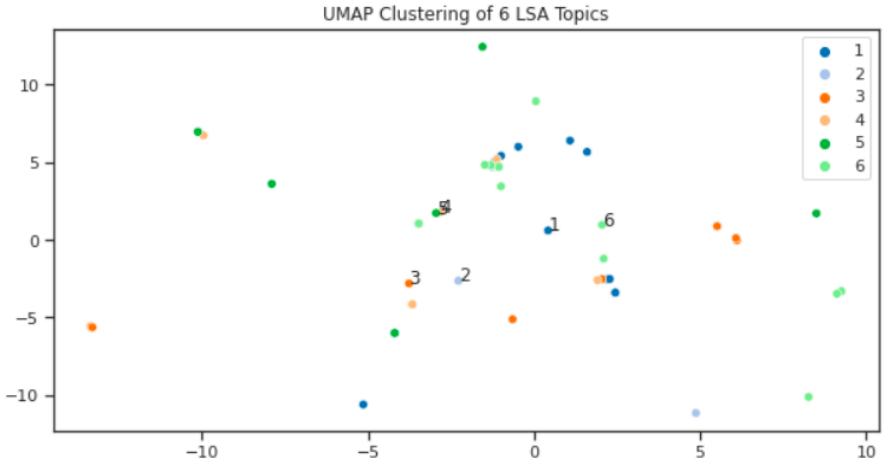


FIGURE 26 – UMAP pour les 10 premiers termes des 6 premiers sujets