

Projet de heuristique et algorithmique du texte






Plan

- Introduction
- Explications détaillées sur l'algorithme naïf de Brute-force et son implémentation
- Présentation du projet
- Instructions pour utiliser l'application.
- Résumé des objectifs atteints.
- Difficultés rencontrées
- Démonstration de l'application
- Perspectives futures
- conclusion



Introduction

L'analyse de similarité entre documents est un domaine important dans le traitement automatique du langage naturel, ayant des applications variées telles que la détection de plagiat, la comparaison de textes juridiques, et l'analyse de documents scientifiques. Ce projet a pour objectif de développer une application capable d'analyser la similarité entre deux documents en utilisant l'algorithme naïf de Brute-force.





Explications détaillées sur l'algorithme naïf de Brute-force et son implémentation

Initialisation

L'algorithme commence par comparer systématiquement toutes les sous-chaînes possibles d'un document avec toutes les sous-chaînes correspondantes de l'autre document. Pour ce faire, il décompose chaque document en sous-chaînes de différentes longueurs, en commençant par les plus courtes (1 caractère) jusqu'à la longueur maximale possible dans les documents. Par exemple, pour un document de 5 caractères, il va comparer les sous-chaînes de 1 à 5 caractères.

Identification des motifs

À chaque comparaison entre deux sous-chaînes de même longueur extraites des deux documents, l'algorithme vérifie s'il y a correspondance. Une correspondance est enregistrée si les sous-chaînes sont identiques. Cela permet d'identifier tous les motifs communs, même ceux qui peuvent être partiellement répétés ou inclus dans d'autres motifs.

L'algorithme naïf de Brute-force

```
File Edit Selection View Go Run Terminal Help
EXPLORER
  algo-naif-brute-force > algo_naif_brute_force_app > utils.py
  OPEN EDITORS
    apps.py algo-naif-brute-...
    tests.py algo-naif-brute-f...
    utils.py algo-naif-brute-f...
    admin.py algo-naif-brut...
  PROJET
    algo-naif-brute-force
    algo_naif_brute_force
    algo_naif_brute_force...
    _pycache_
    migrations
    _init_.py
    admin.py
    apps.py
    forms.py
    models.py
    tests.py
    urls.py
    utils.py
    views.py
    env
    static
    templates
    uploads
    document1.txt
    document2.txt
    .gitignore
    db.sqlite3
    manage.py
    README.md
    requirements.txt
    env

6
7 # Fonction pour charger le contenu d'un document texte
8 def load_document(filepath : str):
9     with open(filepath, 'r', encoding='utf-8') as file:
10         return file.read()
11
12 # Fonction pour trouver les mots ou phrases communs entre deux documents en utilisant l'algorithme naïf brute force
13 def find_common_patterns(doc1, doc2, pattern_length=3):
14     # Diviser les documents en listes de mots
15     words1 = re.findall(r'\w+', doc1.lower())
16     words2 = re.findall(r'\w+', doc2.lower())
17
18     # Utiliser des ensembles pour trouver les motifs communs
19     patterns1 = set()
20     patterns2 = set()
21
22     # Extraire les motifs de longueur spécifiée
23     for i in range(len(words1) - pattern_length + 1):
24         patterns1.add(' '.join(words1[i:i + pattern_length]))
25
26     for i in range(len(words2) - pattern_length + 1):
27         patterns2.add(' '.join(words2[i:i + pattern_length]))
28
29     # Trouver les motifs communs
30     common_patterns = patterns1.intersection(patterns2)
31
32     return common_patterns
33
34 # Fonction pour calculer le pourcentage de mots communs
35 def word_similarity(doc1, doc2):
36     words1 = re.findall(r'\w+', doc1.lower())
37     words2 = re.findall(r'\w+', doc2.lower())
38
39     set_words1 = set(words1)
40     set_words2 = set(words2)
41
42     common_words = set_words1.intersection(set_words2)
43
44     total_words = len(set_words1.union(set_words2))
45
46     if total_words == 0:
47         return 0.0
48
49     return (len(common_words) / total_words) * 100
50
51 # Fonction pour calculer le pourcentage de phrases communes
```

```
EXPLORER
  apps.py
  tests.py
  utils.py
  admin.py
  OPEN EDITORS
    algo-naif-brute-force > algo_naif_brute_force_app > utils.py
    apps.py algo-naif-brute-...
    tests.py algo-naif-brute-f...
    admin.py algo-naif-brut...
  PROJET
    algo-naif-brute-force
    algo_naif_brute_force
    algo_naif_brute_force...
    _pycache_
    migrations
    _init_.py
    admin.py
    apps.py
    forms.py
    models.py
    tests.py
    urls.py
    utils.py
    views.py
    env
    static
    templates
    uploads
    document1.txt
    document2.txt
    .gitignore
    db.sqlite3
    manage.py
    README.md
    requirements.txt
    env

35 def word_similarity(doc1, doc2):
36     if total_words == 0:
37         return 0.0
38
39     return (len(common_words) / total_words) * 100
40
41 # Fonction pour calculer le pourcentage de phrases communes
42 def sentence_similarity(doc1, doc2):
43     sentences1 = re.split(r'[.!?]', doc1.lower())
44     sentences2 = re.split(r'[.!?]', doc2.lower())
45
46     set_sentences1 = set([s.strip() for s in sentences1 if s.strip()])
47     set_sentences2 = set([s.strip() for s in sentences2 if s.strip()])
48
49     common_sentences = set_sentences1.intersection(set_sentences2)
50
51     total_sentences = len(set_sentences1.union(set_sentences2))
52
53     if total_sentences == 0:
54         return 0.0
55
56     return (len(common_sentences) / total_sentences) * 100
57
58 # Fonction pour uploader le premier document
59 def handle_uploaded_file_01(file):
60     with open(settings.BASE_DIR / "uploads/document1.txt", "wb+") as destination:
61         for chunk in file.chunks():
62             destination.write(chunk)
63
64     return load_document(settings.BASE_DIR / "uploads/document1.txt")
65
66 # Fonction pour uploader le deuxième document
67 def handle_uploaded_file_02(file):
68     with open(settings.BASE_DIR / "uploads/document2.txt", "wb+") as destination:
69         for chunk in file.chunks():
70             destination.write(chunk)
71
72     return load_document(settings.BASE_DIR / "uploads/document2.txt")
73
74 # Fonction pour afficher le graphique des statistiques
75 def plot_similarity_statistics(word_sim, sentence_sim, common_patterns_count):
76     labels = ['Similarité des mots', 'Similarité des phrases', 'Motifs communs']
77     values = [word_sim, sentence_sim, common_patterns_count]
78     fig, ax = plt.subplots()
79     ax.bar(labels, values, color=['blue', 'green', 'red'])
80     ax.set_ylabel('Pourcentage / Nombre')
81     ax.set_title('Statistiques de similarité entre les documents')
82     for i, v in enumerate(values):
83         ax.text(i, v + 0.5, f'{v:.2f}%', ha='center', va='bottom')
84
85     # Sauvegarder le graphique dans un fichier
86     img_path = os.path.join(settings.STATICFILES_DIRS[0], 'similarity_plot.png')
87     plt.savefig(img_path)
88     plt.close()
89     return img_path
90
91
92
93
94
95
96
97
98
99
100
101
```



Présentation du projet

Pour la réalisation du projet, nous avons suivi les étapes suivantes :

Implémentation de l'algorithme :

- Nous avons d'abord implémenté l'algorithme de Brute-force naïf en utilisant le langage Python.
- Le choix de Python a été motivé par sa simplicité et son efficacité pour manipuler des textes.

Utilisation de Django :

- Nous avons utilisé le framework Django pour créer une interface utilisateur conviviale.
- Django a été choisi pour ses robustes fonctionnalités de gestion de fichiers et de création d'interfaces web.

Développement de l'interface utilisateur :

- - Une interface accueillante a été développée pour permettre aux utilisateurs de charger deux fichiers texte.
- - L'interface permet de sélectionner et de téléverser facilement les fichiers à comparer.



Présentation du projet

Analyse de similarité :

- Une fois les fichiers chargés, l'application utilise l'algorithme de Brute-force naïf pour analyser la similarité entre les documents.
- L'application identifie les mots et les phrases similaires ainsi que les motifs communs entre les deux fichiers.

Résultats:

- Les résultats de l'analyse sont présentés de manière claire et concise sur l'interface utilisateur.
- Les utilisateurs peuvent visualiser la similarité des mots, des phrases, et des motifs communs entre les deux documents.

Instructions pour utiliser l'application

- Créez un environnement virtuel: `python -m venv env`
- Activez l'environnement virtuel : `.\env\Scripts\activate`
- Installez les dépendances : `pip install -r requirements.txt`
- Appliquez les migrations de la base de données et lancez le serveur :

`python manage.py migrate`

`python manage.py runserver`

`pip install django`

`python manage.py runserver`



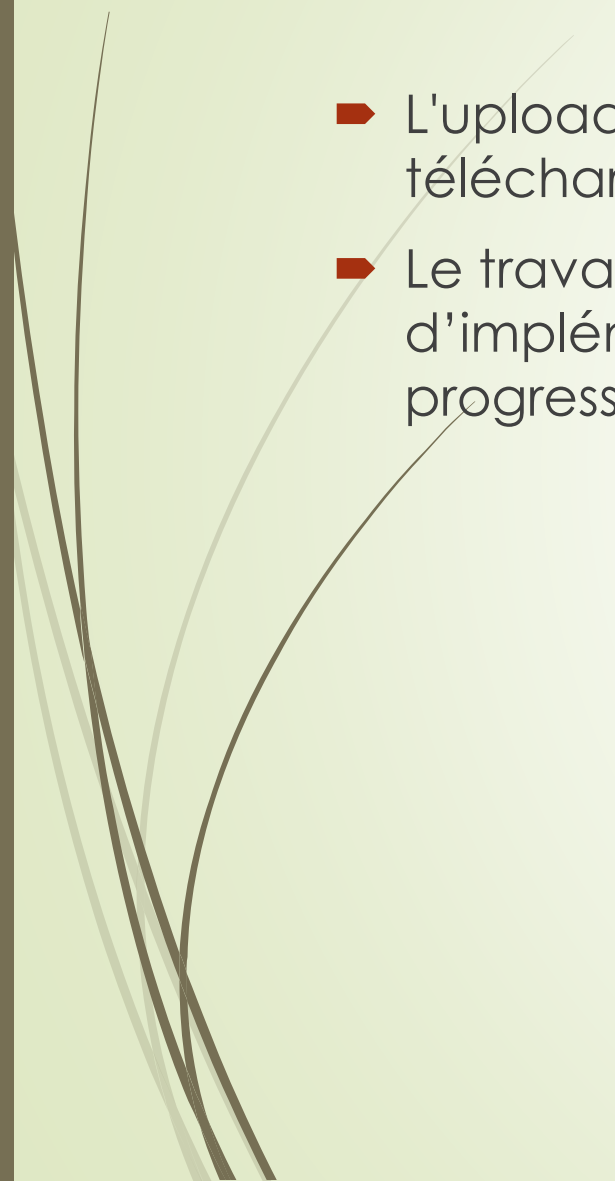
Résumé des objectifs atteints

Nous avons réussi à:

- implémenter l'algorithme avec Django.
- Utiliser un front-end pour l'affichage des données.
- L'application arrive à faire les analyses énumérées dès le départ, à savoir :
 - La similarité des mots
 - La similarité des phrases
 - Les motifs communs

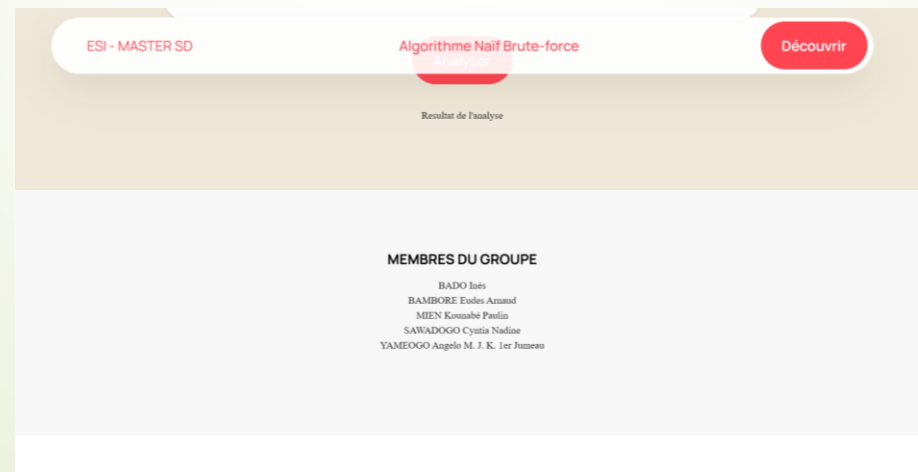
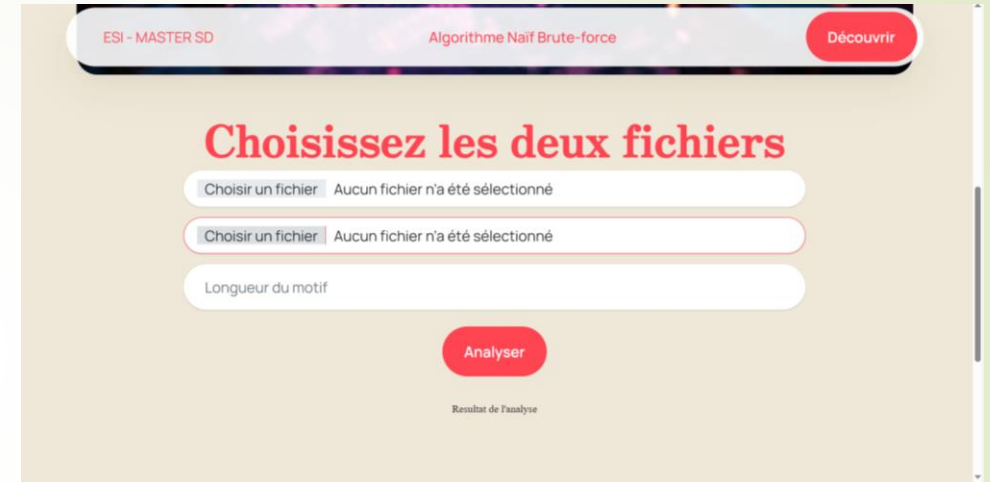


Difficultés rencontrées

- L'upload des documents avec Django : La mise en place du système de téléchargement de fichiers s'est avérée complexe.
 - Le travail en groupe : Nous avons eu des divergences sur la façon d'implémenter et d'optimiser l'algorithme, ce qui a parfois ralenti notre progression.
- 

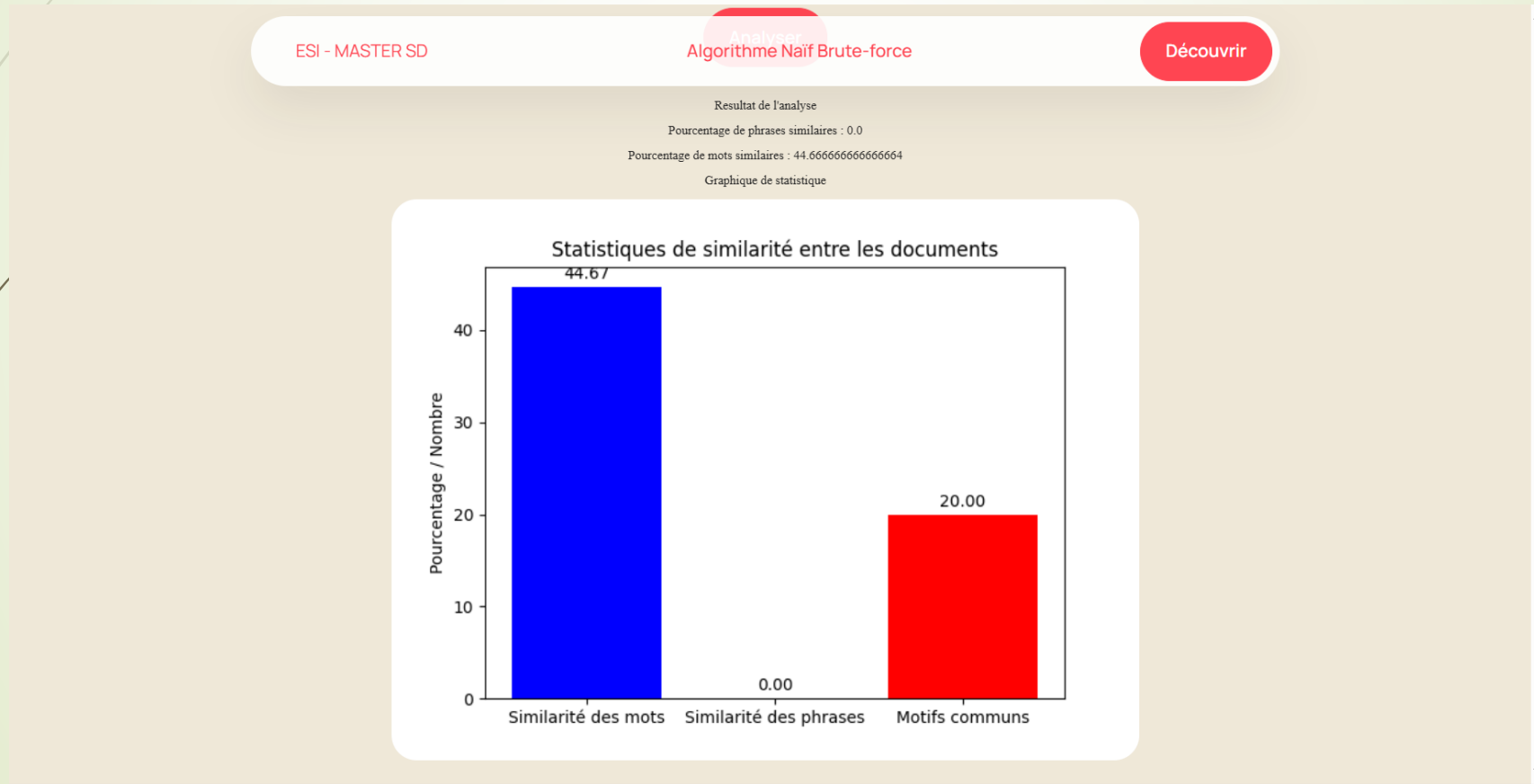
Démonstration de l'application

➤ Page d'accueil



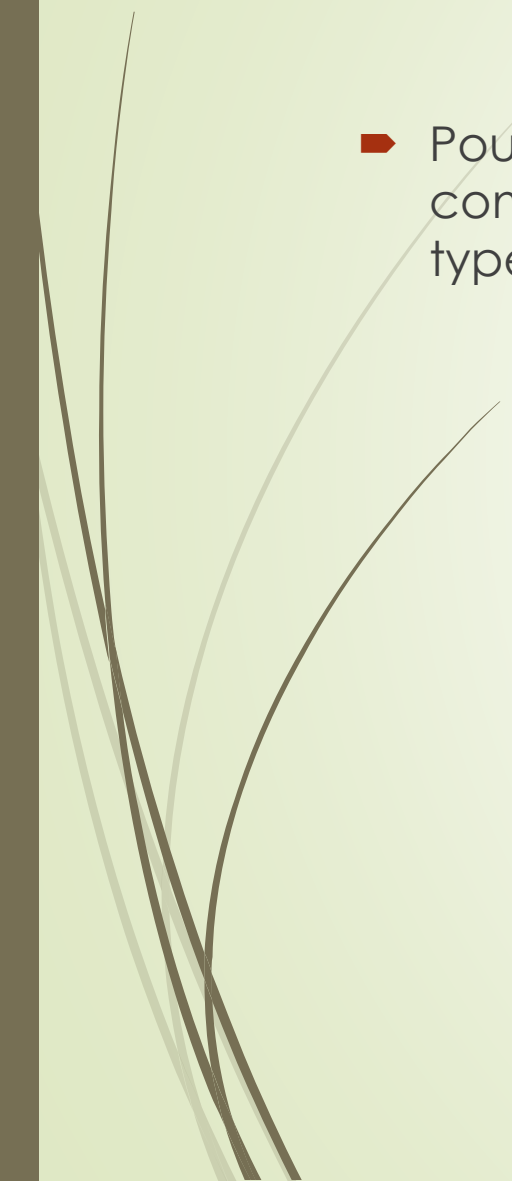
Démonstration de l'application

- Résultats après l'analyse de deux documents





Perspectives futures

- Pour l'instant notre application ne prend en compte que des fichiers txt nous comptons l'améliorer de telle sorte a ce qu'elle puisse prendre en compte plusieurs type de fichiers (pdf, word, powerpoint ...)
- 



Conclusion

Ce projet nous a permis de plonger profondément dans le domaine du traitement automatique du langage naturel, en particulier dans l'analyse de similarité entre documents. Grâce à l'implémentation de l'algorithme naïf de Brute-force, nous avons pu développer une application capable de comparer efficacement des textes. Malgré les défis rencontrés, tels que la gestion de l'upload des documents et les divergences dans l'optimisation de l'algorithme, nous avons acquis une compréhension solide des concepts et techniques nécessaires. Ce projet a non seulement enrichi nos connaissances techniques, mais a également renforcé notre capacité à travailler en équipe et à résoudre des problèmes complexes de manière collaborative.