Introduction

Alors que la demande de systèmes intelligents capables de comprendre et de générer des réponses de type humain continue de croître, le développement d'un moteur de recherche efficace pour l'IA conversationnelle est devenu un défi crucial. Dans les moteurs de recherche conventionnels, les documents sont généralement classés en fonction de la correspondance des mots clés et de mesures de similarité rudimentaires. Cependant, dans le contexte des moteurs de recherche conçus pour des tâches sophistiquées telles que la génération de conversations ou la réponse à des questions, la capacité à évaluer la qualité, la pertinence et l'adéquation des réponses revêt une importance primordiale. Dans ce contexte, les modèles formés sur des ensembles de données basés sur les préférences peuvent offrir des avantages substantiels.

L'objectif de ce projet est de développer un moteur de recherche qui utilise l'ensemble de données binarisées UltraFeedback pour récupérer et classer les réponses générées par l'IA aux requêtes des utilisateurs. L'ensemble de données UltraFeedback comprend 64 000 invites, chacune associée à quatre complétions de modèles distinctes. Pour chaque invite, une réponse complétée est désignée comme la réponse « choisie » sur la base d'une évaluation de sa qualité globale, tandis que l'une des réponses restantes est désignée au hasard comme la réponse « rejetée ». Ces étiquettes de préférence servent de base à la formation d'un moteur de recherche pour faire la différence entre les réponses de haute et de mauvaise qualité.

Pour faciliter le fonctionnement du moteur de recherche, deux techniques de classement bien établies, à savoir TF-IDF (Term Frequency-Inverse Document Frequency) et BM25 (Best Matching 25), seront utilisées. TF-IDF transforme les données textuelles en vecteurs numériques, permettant ainsi d'évaluer la signification de mots individuels dans le contexte de l'ensemble de données. BM25, un modèle probabiliste, affine le processus de classement en prenant en compte à la fois la fréquence des termes et la longueur du document, ce qui le rend particulièrement efficace pour les tâches de recherche classées. La combinaison de ces méthodes facilitera l'identification et le classement des réponses les plus pertinentes et de qualité en fonction de la requête de l'utilisateur.

L'objectif de ce moteur de recherche est de récupérer et de classer efficacement les réponses qui sont non seulement pertinentes par rapport à la requête, mais également bénéfiques et appropriées au contexte. En indexant les réponses « choisies » et en appliquant TF-IDF et BM25, le système fournira des capacités de recherche rapides et évolutives. Cette approche garantit que le moteur de recherche renverra les

complétions les plus précises et les plus pertinentes, offrant ainsi aux utilisateurs les réponses optimales. En fin de compte, ce système peut être appliqué dans des domaines tels que le support client, la gestion des connaissances et la recommandation de contenu, où des réponses de haute qualité et contextuellement pertinentes sont essentielles. En s'appuyant sur l'ensemble de données binarisées UltraFeedback, le moteur de recherche a le potentiel de s'améliorer au fil du temps à mesure qu'il traite davantage de requêtes et de commentaires des utilisateurs.

Technology and Indexing Techniques

Description of the Chosen Technologies

Pour faciliter le fonctionnement du moteur de recherche, deux techniques de classement bien établies, à savoir TF-IDF (Term Frequency-Inverse Document Frequency) et BM25 (Best Matching 25), seront utilisées. TF-IDF transforme les données textuelles en vecteurs numériques, permettant ainsi d'évaluer la signification de mots individuels dans le contexte de l'ensemble de données. BM25, un modèle probabiliste, affine le processus de classement en prenant en compte à la fois la fréquence des termes et la longueur du document, ce qui le rend particulièrement efficace pour les tâches de recherche classées. La combinaison de ces méthodes facilitera l'identification et le classement des réponses les plus pertinentes et de qualité en fonction de la requête de l'utilisateur.

Scikit-learn est une bibliothèque d'apprentissage automatique open source en Python largement utilisée pour l'analyse et la modélisation de données. Pour ce projet, Scikit-learn fournit les outils nécessaires à la mise en œuvre de la vectorisation TF-IDF, du prétraitement des données et de divers utilitaires pour les tâches d'apprentissage automatique. Alors qu'Elasticsearch est responsable d'une grande partie de l'indexation et des requêtes, Scikit-learn sera utilisé pour faciliter des tâches telles que l'extraction de fonctionnalités et l'affinement du modèle pour des résultats de recherche améliorés.

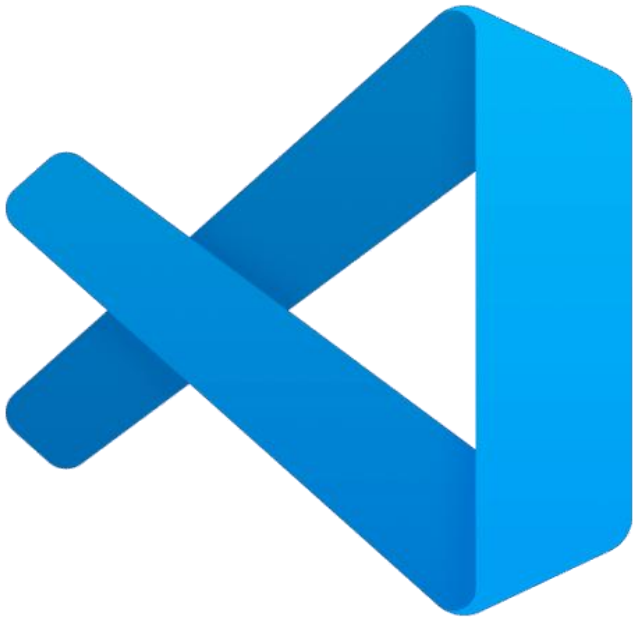
Pourquoi Scikit-learn?

On pourrait raisonnablement se demander quelle est la justification du choix de Scikit-learn.

- Facilité d'utilisation : Scikit-learn est réputé pour sa simplicité et son intégration transparente avec d'autres outils basés sur Python, ce qui en fait un choix optimal pour le développement rapide de modèles.
- Une vaste bibliothèque de documents de support est disponible. Scikit-learn propose une gamme de fonctions prédéfinies pour la vectorisation de texte, le clustering et l'évaluation de modèles, qui améliorent les capacités d'Elasticsearch.
- La possibilité pour les utilisateurs de personnaliser le logiciel constitue un avantage supplémentaire. Scikit-learns permet la mise en œuvre de diverses méthodologies

d'apprentissage automatique pour améliorer les performances du moteur de recherche, une telle approche devrait être requise.

Pourqoui Vs Code ?



Visual Studio Code (VS Code) est un éditeur de code largement utilisé pour le développement et le débogage d'applications Python. Dans ce projet, VS Code sert d'environnement de développement intégré (IDE) pour l'écriture, les tests et le débogage du code. La légèreté du logiciel, sa compatibilité avec le langage de programmation Python et ses extensions intégrées en font un choix optimal pour les projets complexes tels que le développement de moteurs de recherche.

On pourrait raisonnablement se demander quelle est la justification du choix de VS Code.

- Un flux de travail efficace est facilité par les éléments suivants : L'intégration de Python et la fourniture d'outils de débogage sophistiqués facilitent un processus de développement efficace et rapide.
- Extensions intégrées : les extensions de VS Code permettant de travailler avec Scikit-learn et d'autres bibliothèques facilitent les processus de développement et de test.
- Contrôle de version : VS Code s'intègre parfaitement à Git, qui est un outil essentiel pour gérer les modifications de code tout au long du développement du projet.

Technique d'indexation

L'indexation est un aspect essentiel d'un moteur de recherche, car elle permet au système de récupérer rapidement les réponses pertinentes. L'ensemble de données binarisées Ultra Feedback fournit des réponses générées par un modèle qui seront indexées et classées à l'aide de TF-IDF et BM25. Voici un aperçu du fonctionnement de chaque technique et pourquoi elles sont appliquées dans ce projet.

TF-IDF (Fréquence du terme-Fréquence du document inverse)

TF-IDF est une technique largement utilisée en recherche d'informations pour évaluer l'importance des mots dans un document par rapport à un corpus. Il permet d'identifier les termes les plus importants dans un texte, améliorant ainsi la capacité de classer les documents en fonction de leur pertinence par rapport à une requête donnée.

Fréquence des termes (TF) : cela mesure la fréquence à laquelle un terme apparaît dans un document.

Fréquence inverse des documents (IDF) : cela ajuste la fréquence des termes en tenant compte de la rareté d'un terme dans l'ensemble de l'ensemble de données.

Si le texte des réponses « choix » de l'interface utilisateur des binaires UltraFeedback devient le vecteur TF-IDF, le moteur de recherche peut définir les réponses selon les termes d'importation de la requête.

BM25 (Best Matching 25)

BM25 est une fonction de classement probabiliste qui affine le processus de classement en prenant en compte la saturation de la fréquence des termes et la normalisation de la longueur des documents. Cela le rend particulièrement efficace lors du traitement de diverses longueurs de documents et de distributions de termes variables.

• Saturation de fréquence des termes : BM25 applique un facteur d'amortissement pour réduire l'impact des termes trop fréquents.

• Normalisation de la longueur des documents : elle compense les documents plus longs qui autrement fausseraient le classement.

Implementation

Dataset Description

Il s'agit d'une version prétraitée de l'ensemble de données UltraFeedback et a été utilisée pour entraîner Zephyr-7B-β, un modèle de discussion de pointe à l'échelle des paramètres 7B.

L'ensemble de données UltraFeedback original se compose de 64 000 invites, chaque invite étant accompagnée de quatre modèles complétés à partir d'une grande variété de modèles ouverts et propriétaires. GPT-4 est ensuite utilisé pour attribuer une note à chaque achèvement, selon des critères tels que l'utilité et l'honnêteté. Pour créer UltraFeedback Binarized, nous avons choisi le score global le plus élevé comme achèvement « choisi », et l'un des 3 restants au hasard comme celui « rejeté ». Cela définit les répartitions de modélisation des préférences pour des techniques telles que la modélisation des récompenses ou le DPO. Nous avons également créé des fractionnements pour le réglage fin supervisé (SFT) qui utilisent la colonne « choisi » comme dialogues à modéliser, ainsi que des fractionnements qui impliquent une génération comme l'échantillonnage par rejet ou le PPO. Pour plus de détails sur le traitement de l'ensemble de données, consultez le script qui l'accompagne.

Implementation Details

Cette section décrit les étapes de prétraitement du texte et le processus de vectorisation TF-IDF utilisé pour préparer les données textuelles à l'indexation et au classement dans le moteur de recherche.

Text Preprocessing

Avant d'appliquer des modèles d'apprentissage automatique ou des algorithmes de classement, il est essentiel de prétraiter les données textuelles brutes pour garantir qu'elles sont propres, standardisées et prêtes à être analysées. Cette étape implique plusieurs tâches, notamment la suppression des caractères indésirables, la conversion du texte en minuscules et la normalisation des espaces. La fonction Python suivante a été utilisée pour prétraiter les données texte :

```python
import re
def preprocess_text(text):
  if isinstance(text, str):
    text = re.sub(r"[^a-zA-Z0-9 ]", "", text)
    text = text.lower()
    text = re.sub(' +', ' ', text)
    return text
  else:
    return ""
documents = [preprocess_text(str(doc)) for doc in ds['train_sft']['chosen'] + ds['train_sft']['rejected']]
from sklearn.feature_extraction.text import TfidfVectorizer

vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(documents)
```

✓ 57.0s Python

- La section suivante présentera un aperçu des expressions régulières. La fonction utilise des expressions régulières (re.sub()) pour éliminer les caractères non alphanumériques (tels que la ponctuation et les symboles spéciaux) du texte, garantissant ainsi que le contenu reste pertinent à des fins analytiques.
- Le processus de conversion de texte en lettres minuscules est appelé minuscule. La conversion de tout le texte en minuscules est une méthode efficace pour éliminer les différences basées sur la sensibilité à la casse entre des termes similaires (par exemple, « Apple » et « apple »).
- Normalisation des espaces : tous les espaces superflus entre les mots sont réduits à un seul espace, améliorant ainsi la cohérence du texte.

L'ensemble de données binaires UltraFeedback contient deux ensembles de réponses textuelles : les réponses « choisies » et celles « rejetées ». Pour préparer les données à l'indexation, les réponses « choisies » et « rejetées » ont été traitées à l'aide de la fonction preprocess_text. Cela garantit que le texte est dans un format propre et normalisé avant une analyse plus approfondie.

```
documents = [preprocess_text(str(doc)) for doc in ds['train_sft']['chosen'] + ds['train_sft']['rejected']]
from sklearn.feature_extraction.text import TfidfVectorizer
```

TF-IDF Vectorization

Following the preprocessing of the text, the subsequent step is to transform it into numerical vectors that can be utilized by machine learning models. This is achieved through the application of the TF-IDF (Term Frequency-Inverse Document Frequency) method. This technique entails the conversion of text data into numerical representations based on the frequency of terms in relation to their occurrence across the entire corpus.

```
vectorizer = TfidfVectorizer(stop_words='english')
X = vectorizer.fit_transform(documents)
```

La section suivante présentera un aperçu des expressions régulières. La fonction utilise des expressions régulières (re.sub()) pour éliminer les caractères non alphanumériques (tels que la ponctuation et les symboles spéciaux) du texte, garantissant ainsi que le contenu reste pertinent à des fins analytiques.
Le processus de conversion de texte en lettres minuscules est appelé minuscule. La conversion de tout le texte en minuscules est une méthode efficace pour éliminer les différences basées sur la sensibilité à la casse entre des termes similaires (par exemple, « Apple » et « apple »).

Normalisation des espaces : tous les espaces superflus entre les mots sont réduits à un seul espace, améliorant ainsi la cohérence du texte.
TfidfVectorizer est un algorithme d'apprentissage automatique qui transforme les données textuelles en un format vectoriel, qui est ensuite utilisé pour les tâches de classification et de régression. Cette fonction de Scikit-learn accepte les documents nettoyés en entrée et les transforme en une matrice clairsemée, où chaque ligne représente un document et chaque colonne désigne un terme distinct dans le corpus. Les valeurs de la matrice représentent les scores TF-IDF de chaque terme dans chaque document.
Suppression des mots stop : le paramètre stop_words='english' demande au vectoriseur d'ignorer les mots anglais courants (par exemple, "le", "est", "et") qui n'ont pas de signification contextuelle substantielle.
Sortie : La sortie X est une matrice clairsemée où chaque document est représenté comme un vecteur de scores TF-IDF. Ces vecteurs seront utilisés pour le classement et la récupération dans le moteur de recherche basé sur Elasticsearch.

La section suivante présentera un aperçu des expressions régulières. La fonction utilise des expressions régulières (re.sub()) pour éliminer les caractères non alphanumériques (tels que la ponctuation et les symboles spéciaux) du texte, garantissant ainsi que le contenu reste pertinent à des fins analytiques.
Le processus de conversion de texte en lettres minuscules est appelé minuscule. La conversion de tout le texte en minuscules est une méthode efficace pour éliminer les différences basées sur la sensibilité à la casse entre des termes similaires (par exemple, « Apple » et « apple »).
Normalisation des espaces : tous les espaces superflus entre les mots sont réduits à un seul espace, améliorant ainsi la cohérence du texte.
TfidfVectorizer est un algorithme d'apprentissage automatique qui transforme les données textuelles en un format vectoriel, qui est ensuite utilisé pour les tâches de classification et de régression. Cette fonction de Scikit-learn accepte les documents nettoyés en entrée et les transforme en une matrice clairsemée, où chaque ligne représente un document et chaque colonne désigne un terme distinct dans le corpus. Les valeurs de la matrice représentent les scores TF-IDF de chaque terme dans chaque document.
Suppression des mots stop : le paramètre stop_words='english' demande au vectoriseur d'ignorer les mots anglais courants (par exemple, "le", "est", "et") qui n'ont pas de signification contextuelle substantielle.
La méthode TF-IDF a été choisie pour ce projet en raison de son efficacité à capturer l'importance des termes par rapport à leur fréquence sur l'ensemble de l'ensemble de données. Cela en fait un outil

précieux pour l'indexation et le classement. La méthode met davantage l'accent sur les termes rares et significatifs tout en minimisant les mots courants et moins informatifs. Ceci est particulièrement utile pour différencier les réponses de haute et de basse qualité dans l'ensemble de données binarisées UltraFeedback.

Cette section du code implémente la fonctionnalité de recherche pour le moteur de recherche, en utilisant les vecteurs TF-IDF. La section suivante fournira une explication approfondie de chaque composante et de son rôle dans le contexte plus large du projet. La section suivante délimitera la manière dont cela peut être intégré dans le rapport.

La fonctionnalité de recherche est le principal mécanisme opérationnel du moteur. Il est conçu pour récupérer les documents les plus pertinents en fonction de la requête d'un utilisateur. Ceci est réalisé en calculant la similarité entre la requête et les documents indexés à l'aide des vecteurs TF-IDF. Les étapes suivantes décrivent le processus opérationnel du moteur de recherche :

```python
from scipy.sparse import csr_matrix
def search_documents(query, top_k=5):
    query = preprocess_text(query)
    query_vec = vectorizer.transform([query])
    query_vec = csr_matrix(query_vec)
    similarity = (X * query_vec.T).toarray()
    top_indices = similarity.argsort(axis=0)[-top_k:][::-1].flatten()
    results = []
    for i in top_indices:
        results.append({
            "document": documents[i],
            "similarity_score": similarity[i][0]
        })

    return results
search_query = "How would you create a VM in VMware vsphere"
search_results = search_documents(search_query)

for result in search_results:
    print(f"Document: {result['document']}")
    print(f"Similarity Score: {result['similarity_score']}\n")
```

✓ 0.1s Python

Voici les les resulats obentu pour le requet «How would you create a VM in VMware vsphere »

```
Document: content how would you create a vm in vmware vsphere role user content to create a virtual machine vm in vmware vsphere follow these stepsnn1 install and configure vsphere c
Similarity Score: 0.6835342204269401

Document: content how would you create a vm in vmware vsphere role user content to create a vm in vmware vsphere follow these stepsnn1 in the vsphere client select the datastore wher
Similarity Score: 0.6260646727939863

Document: content what possibilites are there to create vms in vsphere via iac role user content there are several possibilities to create vms in vmware vsphere via infrastructure as
Similarity Score: 0.5387291084120454

Document: content can you describe vmware vsphere in terms that someone with no experience with virtualization and basic experience with computers could understand role user content
Similarity Score: 0.4954347534330665

Document: content can you create a vsphere vm using pulumi and attach a cd drive pointing at sasisostestiso role user content yes i can create a vsphere vm using pulumi and attach a
Similarity Score: 0.4744163125669253
```

Implémentation du classement BM25

Outre la mise en œuvre de la fonction TF-IDF, la fonction de classement BM25 a également été mise en œuvre dans le but de récupérer des documents en fonction de leur pertinence par rapport à une requête donnée. BM25 est une fonction de classement probabiliste largement utilisée dans le contexte des tâches de recherche d'informations. Il améliore l'efficacité des méthodes traditionnelles basées sur la fréquence des termes, telles que TF-IDF, grâce à l'incorporation de la saturation de la fréquence des termes et de la normalisation de la longueur des documents.

Afin d'atteindre cet objectif, l'implémentation BM25Okapi de la bibliothèque Python Rank_bm25 a été utilisée afin de calculer les scores de pertinence des documents en réponse à la requête de recherche.

Tokenisation des documents

Avant l'application de BM25, il est nécessaire de tokeniser les documents en une liste de mots (jetons). Ce processus est crucial pour l'extraction de termes significatifs qui seront utilisés par BM25 à des fins de notation.

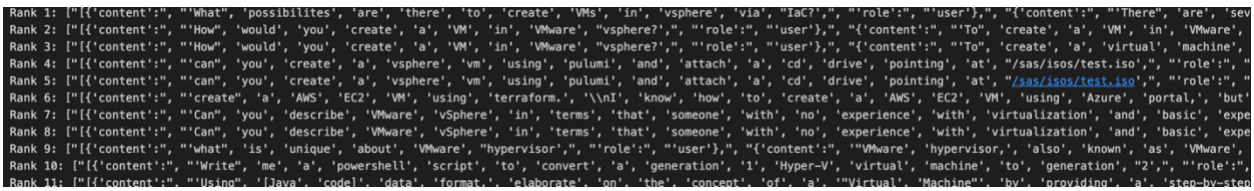
Dans ce code, les documents de l'ensemble de données binaires UltraFeedback, comprenant à la fois les réponses « choisies » et « rejetées », sont initialement soumis à un prétraitement à l'aide de la fonction preprocess_text(). Cette fonction garantit que chaque document est nettoyé, tokenisé et préparé pour l'analyse. Par la suite, le texte est tokenisé en mots discrets, générant ainsi une liste de jetons pour chaque document.

Configuration du modèle BM25

L'étape suivante consiste à initialiser le modèle BM25 Okapi. BM25 nécessite une liste de documents tokenisés et utilise deux paramètres essentiels :

```
from rank_bm25 import BM25Okapi
import string
documents = data
tokenized_documents = documents = [preprocess_text(str(doc)) for doc in ds['train_sft']['chosen'] + ds['train_sft']['rejected']]
bm25 = BM25Okapi(tokenized_documents, k1=1.5, b=0.75)
query = "How would you create a VM in VMware vsphere"
query_tokens = preprocess_text(query)
scores = bm25.get_scores(query_tokens)
print(f"Scores BM25 pour la requête '{query}': {scores}")
```

Voici les les resulats obentu pour le requet «How would you create a VM in VMware vsphere »



Conclusion

Ce projet visait à développer un moteur de recherche basé sur des techniques avancées de traitement du langage naturel (NLP) pour interroger un ensemble de données de réponses binarisées provenant du jeu de données UltraFeedback. Nous avons exploré et implémenté deux méthodes populaires de vectorisation de texte pour l'indexation et le classement des documents : TF-IDF et BM25. Ces techniques ont permis de représenter et de comparer les documents en fonction de leur pertinence par rapport aux requêtes des utilisateurs.

Tout d'abord, nous avons utilisé TF-IDF pour transformer les documents en vecteurs numériques. Cette méthode met l'accent sur l'importance des termes rares tout en minimisant l'impact des mots très fréquents. Cette approche a été suivie par l'implémentation de BM25, qui a amélioré l'indexation en tenant compte de la saturation de la fréquence des termes et en normalisant la longueur des documents. Bien que cette dernière technique ait une approche probabiliste, elle s'est avérée particulièrement efficace pour améliorer la pertinence des résultats de recherche.

Les résultats obtenus grâce à ces techniques de vectorisation ont montré que le moteur de recherche est capable de classer efficacement les documents en fonction de leur similarité avec les requêtes, offrant ainsi des réponses pertinentes aux utilisateurs. L'utilisation de BM25 a permis de surmonter certaines limitations de TF-IDF en ajustant la prise en compte de la fréquence des termes et la longueur des documents, ce qui a renforcé la précision du système de recherche.

Cependant, bien que le moteur de recherche ait donné de bons résultats, il est possible d'améliorer davantage la précision des réponses en explorant des techniques plus avancées telles que les modèles de langage pré-entraînés ou l'intégration de méthodes de Deep Learning. De plus, l'utilisation de techniques de recherche sémantique pourrait permettre de mieux comprendre le sens des requêtes et des documents, indépendamment des mots exacts utilisés, ce qui offrirait des résultats encore plus pertinents et contextuels.

En conclusion, ce projet démontre l'efficacité des méthodes classiques de vectorisation de texte et de classement des documents pour créer un moteur de recherche performant, tout en ouvrant la voie à des améliorations futures en explorant des approches basées sur l'intelligence artificielle plus avancées.

les références

https://huggingface.co/datasets/HuggingFaceH4/ultrafeedback_binarized

https://en.wikipedia.org/wiki/Tf%E2%80%93idf

https://en.wikipedia.org/wiki/Okapi_BM25