
Sujet :

# Appariement des FAQ: BoW vs sBERT

Projet

présenté par :

**Chengwanli YANG**

**Mustapha LAZREG**

Master 2

**"Langue et Informatique"**

27/02/2022

# Table des matières

# Table des figures

# Liste des tableaux

# Chapitre 1

# Introduction

L'optimisation des parcours clients est devenu une des clés pour offrir une bonne expérience client. Cette optimisation a motivé le développement de nouveaux outils de recherche adaptés, tels que les systèmes de réponse aux questions.

En effet, ce type de système permet à l'utilisateur de poser une question en langage naturel et de retourner la bonne réponse à sa question au lieu d'un ensemble de documents jugés pertinents, comme c'est le cas pour un moteur de recherche.

Cependant, pour les systèmes de réponse aux questions dédiés à la manipulation de textes et de documents Web, la structure des informations requises affecte la précision de ces systèmes. Les FAQ sont plus efficaces pour interagir avec des bases de connaissances structurées.

## 1.1 Corpus

Tout d'abord, nous avons choisi le corpus de FAQ associant des questions d'utilisateurs et des réponses de service client. Le corpus comprend quatre fichiers au format json. Chaque fichier contient un nombre différent de questions et réponses. Une paire de questions et réponses est comme la valeur du dictionnaire et la clé est son index. Il existe au total 109 paires à traiter, ci-dessous un exemple 1.1 du premier sous-corpus.

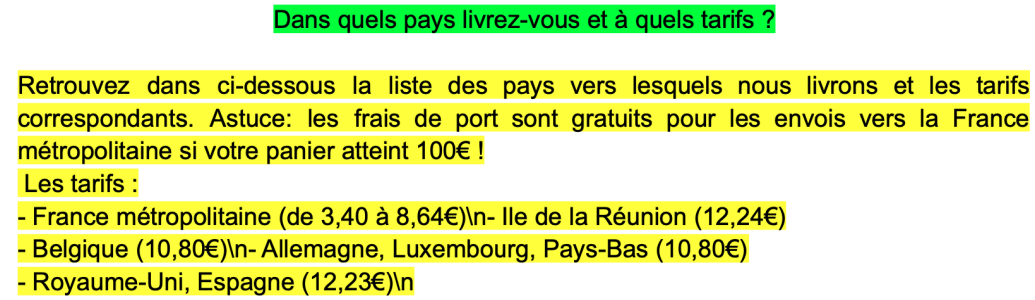


FIGURE 1.1 – Un extrait du sous-corpus "faq_dmc"

## 1.2 Tâche

La tâche principale est l'appariement des questions avec leurs meilleures réponses. C'est-à-dire que nous désirons trouver la meilleure similarité entre la question et la réponse. Dans une conception de moteur de recherche ou d'un chatbot fondé sur une FAQ, le serveur devrait trouver rapidement la bonne réponse, afin de répondre à la question de l'utilisateur. Quelle est la solution la plus pertinente ? Quelle méthode de recherche est-elle plus efficace ? L'objectif de notre travail est de comparer deux méthodes différentes dans le but d'adopter la bonne méthode à utiliser.

# Chapitre 2

# Méthodologie

Nous avons trouvé la bibliothèque SentenceTransformers [Reimers and Gurevych, 2019a], autrement dit sBERT. Elle est basée sur le modèle de langage BERT. La différence est le fait que BERT fait le plongement lexical et sBERT vectorise la phrase. Le SentenceTransformers prend l'entrée des phrases, même si ce sont les phrases courtes ou des textes longs comportant plusieurs phrases. Cependant, le temps d'exécution et le besoin de mémoire sont quadratiques avec la longueur de l'entrée. Cela limite les transformateurs à des entrées d'une certaine longueur. Ainsi, les modèles fournis utilisent une limite de 128 mots par défaut, les phrases plus longues seront tronquées. Il est tout à fait possible de définir la longueur maximale de la séquence, par exemple 200 mots ou 400 mots. Et la sortie sera une matrice Numpy.

Le sBERT apporte 3 avantages principaux. Premièrement, la performance est plus puissante que celle de BERT dans la similarité cosinus des phrases. Deuxièmement, la vitesse de calcul est plus rapide que celle de BERT. Pour clustering hiérarchique de 10 000 phrases, 5 secondes pour sBERT, par contre, BERT a besoin de 65 heures. Troisièmement, sBERT fournit les modèles pré-entraînés multilingues, le plus grand modèle comprend 50 langues [Reimers and Gurevych, 2019b].

Nous avons choisi 3 modèles à tester. Le premier modèle all-mpnet-base-v2 dispose de la meilleure qualité, car il contient 1 billion de paires entraînées. Le deuxième modèle all-MiniLM-L6-v2 est 5 fois plus rapide que le premier modèle, puisqu'il y a moins de dimensions dans des vecteurs. Mais il a aussi la bonne qualité. Le troisième modèle distiluse-base-multilingual-cased-v1 comprend 15 langues, y compris le français, il s'adapte bien à notre corpus.

Dans la partie script Python, nous avons d'abord parcouru chaque fichier de corpus, et récupéré les questions et les réponses dans les listes, puis nous

avons fait en sorte que nos données soient stockées de la manière suivante : une question suivie par toutes les réponses. Nous avons vectorisé chaque liste afin de comparer chaque question avec toutes les réponses, avec la similarité cosinus, nous avons obtenu la similarité entre chaque pairs (question/réponse) sous une liste. Nous avons récupéré la valeur maximale de la liste qui correspond à la distance de similarité la plus proche.

Dans la partie d'évaluation, nous avons ajouté la méthode CountVectorizer comme le groupe de contrôle.

# Chapitre 3

# Interprétation et discussion des résultats

Nous avons créé une boucle afin de visualiser parfaitement les résultats de tous les sous-corpus avec tous les modèles utilisés. La figure 5.1 présente un exemple de résultat obtenu.

Nous constatons d'abord que le fichier testé et le modèle utilisé sont bien indiqués. De plus, les résultats sont au format : telle réponse est la plus proche de telle question, son score de similarité et la petite conclusion : la réponse est bonne (couleur verte) ou mauvaise (couleur rouge). En outre, nous avons fait le calcul du temps d'exécution pour chaque modèle, il est à la fin du résultat.

Nous avons travaillé sur plusieurs méthodes de vectorisation, soit sBERT ainsi que BoW. La méthode sBERT comporte quatorze modèles, dont trois que nous allons utilisés : all-mpnet-base-v2, all-MiniLM-L6-v2 et distiluse-base-multilingual-cased-v1 [Reimers and Gurevych, 2020]. Tandis que la méthode BoW, nous avons décidé d'utiliser CountVectorizer, nous l'avons choisi pour mieux connaître notre point de vue par rapport à sBERT.

Dans le tableau3.1 ci-dessous, nous remarquons que le premier modèle de sBERT est le plus performant. Le second est beaucoup plus rapide tout en restant performant. Cependant, le dernier modèle reste le plus adapté pour notre travail. Après avoir testé les quatorze modèles de sBERT, c'est le plus adapté à notre corpus à en juger par à la moyenne de précision qui est égal à 44%. D'un point de vue global, nous pouvons voir que sBERT est bien plus performant que BoW, cependant la méthode BoW est bien plus rapide.

| | Modèle\Corpus | 3_faq_syp | 3_syp | 2_questions_sorbonne | 1_faq_dmc | | |
|---|---|---|---|---|---|---|---|
| sBERT | all-mpnet-base-v2 | 0.27 | 0.11 | 0.26 | 0.6 | 0.31 | 6900.68 sec |
| | all-MiniLM-L6-v2 | 0.25 | 0.15 | 0.26 | 0.4 | 0.265 | 2243.56 sec |
| | distiluse-base-multilingual-cased-v1 | 0.54 | 0.13 | 0.28 | 0.8 | 0.4375 | 1671.73 sec |
| BoW | CountVectorizer | 0.25 | 0.07 | 0.26 | 0.4 | 0.245 | 0.69 sec |

TABLE 3.1 – Mesures de performance : précision

La deuxième méthode d'évaluation que nous avons utilisée est le calcul des positions des bonnes réponses. Cette fois, nous nous sommes focalisés sur de mauvaises réponses. Si la réponse est mauvaise, c'est-à-dire que la bonne réponse de la question correspondante ne se situe pas au premier rang des résultats. Autrement dit elle n'est pas la plus proche réponse. Par conséquent, nous voulions savoir où la bonne question se trouve, elle est plus loin du premier rang ou plus proche.

| | Modèle\Corpus | 3_faq_syp | 3_syp | 2_questions_sorbonne | 1_faq_dmc | Moyenne |
|---|---|---|---|---|---|---|
| 1 | Sentence Transformer (modèle multilingue) | 2 | 28.25 | 17.83 | 1.2 | 12.320 |
| 2 | CountVectorizer | 3.79 | 26.7 | 19.35 | 1.8 | 12.910 |

TABLE 3.2 – Rangs des bonnes réponses

Nous avons fait l'addition de toutes les positions des bonnes réponses, ensuite calculé la moyenne. Dans le tableau 3.2 ci-dessus, nous montrons le résultat obtenu. La moyenne des rangs des bonnes réponses de SENTENCE-TRANSFORMERS est de 12.32 et celle de COUNTVECTORIZER est de 12.91. Puisqu'elles n'ont pas la grande différence, nous pouvons conclure que ces 2 méthodes ont fait le travail.

# Chapitre 4

# Conclusion

Après la réalisation de ce projet, nous avons appris une nouvelle méthode de vectorisation, et nous nous sommes rendu compte qu'une méthode qui vise non seulement à trouver des mots-clés, mais aussi à déterminer l'intention et la signification contextuelle des mots est plus efficace qu'une méthode de vectorisation classique.

Enfin, durant ce travail nous avons constaté que certaines valeurs dans le sous-corpus SYP correspondent à un échange de mails de plusieurs question/-réponse, ce qui a influencé notre traitement et a baissé la performance de nos modèles.

Il serait intéressant d'augmenter la taille de notre jeu de données, afin de voir le résultat obtenu. Mais également de tester d'autres méthodes de vectorisation.

# Chapitre 5

# Annexe

Les scripts Python que nous avons développés sont regroupés sur github :

— Le répertoire GitHub

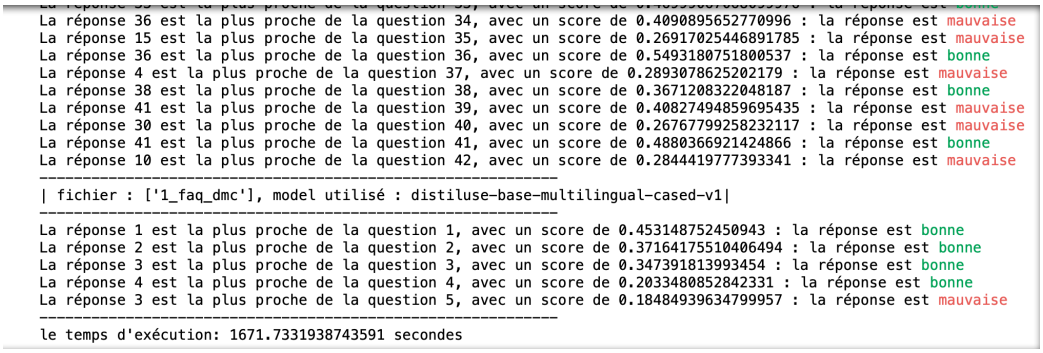


FIGURE 5.1 – Exemple resultat obtenu

# Bibliographie

[Reimers and Gurevych, 2019a] Reimers, N. and Gurevych, I. (2019a). Sentence-bert : Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

[Reimers and Gurevych, 2019b] Reimers, N. and Gurevych, I. (2019b). Sentence-bert : Sentence embeddings using siamese bert-networks.

[Reimers and Gurevych, 2020] Reimers, N. and Gurevych, I. (2020). Making monolingual sentence embeddings multilingual using knowledge distillation. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.