[TEST] Synthetic big patent Similarity

Rapport de travail

Stagiaire :
Samuel TAN

Tuteur :
Paul-Alexis DRAY

21 avril 2025

Table des matières

1 Introduction

Dans ce projet, nous explorons l'utilisation de modèles d'embedding dans le thème de la similarité entre documents industriels, en particulier des documents de type brevets. Cette tâche est cruciale dans de nombreux cas d'usage tels que la détection de redondances, l'identification de non-conformités, ou encore la facilitation de la recherche documentaire dans des corpus industriels massifs.
Notre expérimentation s'appuie sur le jeu de données ***dataset_big_patent v3***. L'objectif est d'évaluer dans quelle mesure un modèle d'embedding peut capturer efficacement la similarité sémantique entre les documents et une query, zero shot et après fine-tuning. Pour cela, nous avons sélectionné le modèle `avsolatorio/GIST-small-Embedding-v0` — un modèle léger, adapté aux contraintes de ressources (GPU NVIDIA GeForce RTX 3050). Nous avons mené une première évaluation en zero-shot, suivie d'une phase de fine-tuning, pour comparer les performances selon les cas d'usage.

1.1 Données initiales

Le fichier ***dataset_big_patent_v3.json*** est un jeu de données json (synthétique ou pas). Chaque document contient des sections (anchor, query, positive, negative) et le fichier contient ["anchor" : "extrait de brevet", "query" : "une requête liée au anchor", "positive" : "réponse alignée à la query, proche", "negative" : "réponse floue ou générale à la query, éloignée"].
Analyse exploratoire :

Quantitative :
- Nombre total de documents : 499

Qualitative :
- Vocabulaire technique dense
- Présence de variations lexicales
- Groupes thématiques variés (par exemple, énergie, mécanique, cosmétique)

Cette analyse permet d'anticiper certains défis pour l'entraînement du modèle : bruit lexical par déséquilibre entre classes de brevets, ou encore des particularités sémantiques difficiles à rapprocher ou dissocier sans entrainement.

Refonte du Dataset pour l'entrainement

Code 1 : Code du split dataset python

```python
success = []
failure = []

# Boucle sur les triplets du dataset d'entraînement
for ligne in dataset["train"]:
    # Extraction des trois éléments du triplet : ancre, positif, négatif
    observation = [ligne["anchor"], ligne["positive"], ligne["negative"]]

    # Génération des embeddings via le modèle de référence (non fine-tuné)
    embeddings = model_zero().encode(observation, convert_to_tensor=True)

    # Calcul de la similarité cosinus entre les différentes paires
    similarities = model_zero().similarity(embeddings, embeddings)

    # Comparaison : si l'ancre est plus proche du positif que du négatif,
    ↪  c'est un succès
    if similarities[0][1] > similarities[0][2]:
        success.append(ligne)
    else:
        failure.append(ligne)
```

Pourquoi faire ?
Nous avons jugé que le split trop aléatoire en dataset d'entrainement et de validation pourrait mener à un entrainement inutile. En effet, le modèle décide déjà bien sur une bonne partie de la data considérant la métrique cosine accuracy. Il serait plus intéressant de l'entrainer avec un focus sur ceux fautant.

```python
# Définition des ratios de séparation pour les ensembles d'entraînement et de validation.
# On envoie 50% des succès dans l'entraînement (soit 30% du total initial si succès = 60%),
# et on conserve 85% des échecs pour l'entraînement.
success_ratio = 0.5
failure_ratio = 0.85

# Découpage du dataset des succès en train/valid
success_split_idx = int(success_ratio * len(success_dataset))
success_for_train = success_dataset.select(range(success_split_idx))
success_for_valid = success_dataset.select(range(success_split_idx, len(success_dataset)))

# Découpage du dataset des échecs en train/valid
failure_split_idx = int(failure_ratio * len(failure_dataset))
failure_for_train = failure_dataset.select(range(failure_split_idx))
failure_for_valid = failure_dataset.select(range(failure_split_idx, len(failure_dataset)))

# Fusion des sous-datasets pour créer les ensembles finaux d'entraînement et de validation.
train_dataset = concatenate_datasets([failure_for_train, success_for_train])
valid_dataset = concatenate_datasets([failure_for_valid, success_for_valid])
```

FIGURE 1 – Split dataset d'entrainement.

Ainsi, après avoir regroupé les succès (success) et les échecs (failure), nous avons constitué les datasets avec les proportions comme le montre la figure 1. Cela pour espérer un entrainement équilibré en évitant de trop apprendre sur le training et oublier les connaissances antérieures.

```
[16]: DatasetDict({
          train: Dataset({
              features: ['anchor', 'positive', 'negative'],
              num_rows: 296
          })
          valid: Dataset({
              features: ['anchor', 'positive', 'negative'],
              num_rows: 203
          })
          test: Dataset({
              features: ['anchor', 'positive', 'negative'],
              num_rows: 499
          })
      })
```

FIGURE 2 – Format du dataset d'entrainement.

1.2 Modèle d'entrainement

Modèle utilisé :

- Nom : avsolatorio/GIST-small-Embedding-v0 (cf [1])
- Taille : 33.4M de paramètres
- Type : modèle Sentence Transformer préentraîné sur des données générales

Description du modèle

Le modèle est ajusté (*fine-tuned*) à partir de **BAAI/bge-small-en-v1.5**, en utilisant le jeu de données **MEDI**, enrichi avec des triplets extraits depuis l'ensemble d'entraînement **MTEB Classification**, en excluant les données provenant de la tâche *Amazon Polarity Classification*. Plus d'info ici).

Paramètres préentraînement

- **Nombre d'époques** : 40
- **Ratio de warmup** : 0,1
- **Taux d'apprentissage** : 5×10^{-6}
- **Taille du batch** : 16
- **Étape de sauvegarde (checkpoint)** : 102 000
- **Température de la loss contrastive** : 0,01

Entrainement :

- Phase 1 : Zero-shot
  Évaluation directe du modèle sans ajustement sur le dataset
- Phase 2 : Fine-tuning
  - Tâche : apprentissage (triplet loss)
  - Données : triplets (anchor/positives/négatives) extraites ou simulées à partir du dataset

- Hyperparamètres :
  - Batch size : 16
  - Epochs : 10
  - Learning rate : 1e-5
  - Optimizer : AdamW (par defaut)
  - Scheduler : Linear Warmup Scheduler (par defaut)

Triplet Loss :

$$\mathcal{L} = \sum_{i=1}^{N} \left[\|f(x_i^a) - f(x_i^p)\|_2^2 - \|f(x_i^a) - f(x_i^n)\|_2^2 + \alpha \right]_+$$

Légende :

- x_i^a : l'**ancre** (anchor)
- x_i^p : l'**exemple positif** (positive)
- x_i^n : l'**exemple négatif** (negative)
- $f(x)$: l'**embedding du texte** x
- α : **marge** (margin)
- $[\cdot]_+$: **fonction ReLU**, i.e., $\max(0, \cdot)$

L'idée est d'apprendre un espace d'embeddings dans lequel les éléments similaires sont rapprochés et les élements différents sont éloignés d'au moins une marge α. Dans notre cas, la distance définie est la cosine similarity.

Pourquoi mener cette expérimentation ?

Dans un contexte industriel, la capacité à mesurer automatiquement la similarité entre documents est essentielle pour des tâches telles que :

- la recherche d'antériorité dans des bases,
- la détection de doublons ou de documents proches,
- l'analyse automatique de non-conformités.
- la recherche et récupération efficace de documents dans les bases

En évaluant un modèle léger comme GIST dans ce cadre, nous visons à démontrer qu'il est possible d'obtenir des performances acceptables sans nécessiter des ressources matérielles importantes, tout en identifiant les limites et pistes d'amélioration pour des cas d'usage concrets.

2 Goals and Scientific Challenges (and SOTA Analysis)

Quels sont les objectifs de l'expérimentation ?

L'objectif principal de cette expérimentation est d'évaluer la capacité d'un modèle d'embedding léger à capturer des similarités sémantiques de documents industriels complexes, en l'occurrence des brevets techniques.
Plus précisément, les buts sont :

- D'évaluer les performances du modèle en zero-shot, c'est-à-dire sans aucun ajustement sur le dataset.
- De mesurer l'impact du fine-tuning sur un corpus spécifique pour améliorer la qualité des embeddings.
- De tester la faisabilité d'un tel système dans un contexte où les ressources de calcul sont limitées.

Comment avons-nous mesuré le succès de l'expérience ?

Le succès sur cette expérience actuelle est défini par :

- L'éventuelle amélioration à la métrique cosine_accuracy observable après fine-tuning par rapport au zero-shot

Quelles hypothèses avons-nous formulées ?

Les hypothèses testées sont les suivantes :

1. Un **modèle léger pré-entraîné** peut capturer des similarités sémantiques pertinentes dans un corpus synthétique industriel.
2. Le **fine-tuning**, même avec peu de données, permet d'améliorer significativement la qualité des embeddings.

Quels défis scientifiques cette expérience devait-elle relever ?

- **Représenter la sémantique de documents industriels longs**, riches en vocabulaire technique et structurés.
- **Différencier des documents proches lexicalement mais sémantiquement distincts**, et inversement.
- **Gérer les limites des modèles légers** en matière de capacité de représentation.

Quel est l'état de l'art sur ces problématiques ?

La modélisation de la similarité sémantique est une tâche bien connue en NLP, mais elle présente des particularités dans les contextes industriels :

- Le benchmark **MTEB (Massive Text Embedding Benchmark)** est aujourd'hui la référence pour comparer les modèles d'embedding sur des tâches variées (recherche, clustering, classification...).
- Des modèles comme `all-MiniLM-L6-v2`, `mpnet-base-v2` ou encore `Instructor-Large` dominent les benchmarks actuels, bien qu'ils soient plus lourds que GIST.
- L'apparition de modèles spécialisés comme **GIST**, **E5** ou **Cohere Embed** montre un intérêt croissant pour des représentations denses, optimisées pour la recherche sémantique.
- Le fine-tuning par des méthodes de type **Multiple Negatives Ranking Loss (MNRL)**, **Triplet Loss**, ou **Contrastive Learning** est le standard pour adapter les embeddings à des cas d'usage précis.
- L'intégration d'**adapters (AdapterHub)**, permet un fine-tuning léger sur des couches gelées des modèles, offrant un bon compromis entre coût et performance. [2]

3 Contribution

Quelle approche mise en place pour atteindre notre objectif ?

Afin d'évaluer et détecter efficacement la similarité sémantique des documents industriels, notre approche est basée sur le fine tuning du modèle léger `avsolatorio/GIST-small-Embedding-v0`.

- Évaluation **zero-shot** sur le dataset
- Entraînement par **fine tuning**

Pourquoi avons-nous choisi cette approche ?

Cette poche a été retenue pour plusieurs raisons :

- Elle permet d'obtenir une représentation dense et sémantiquement pertinente des documents.
- Elle est cohérente à notre environnement à **ressources limitées**. (modèle léger)
- Le modèle GIST présente de **bonnes performances** dans le benchmark MTEB, tout en restant léger.
- Elle est généralisable à d'autres cas d'usage industriels (non-conformités, rapports qualité...).

Quelles sont les limitations connues de notre contribution ?

Plusieurs limites ont été identifiées :

- Le modèle GIST, bien que léger, présente des **limites en capacité de compression sémantique** pour des documents longs ou complexes.
- Le **fine-tuning** a été réalisé sur un **volume trop restreint** de données.

- La perte éventuelle des capacités antérieures de généralisation dans le domaine due à un dataset pas assez représentatif.

4 Résultats

Les résultats statistiques clés

L'évaluation a été menée en deux phases : une phase **zero-shot** (modèle non ajusté) et une phase **fine-tuning**.

Les résultats observés sont les suivants :

- Le modèle `GIST-small` a démontré une **bonne capacité de représentation en zero-shot**, avec un score moyen de cosinus accuracy de **0,72**

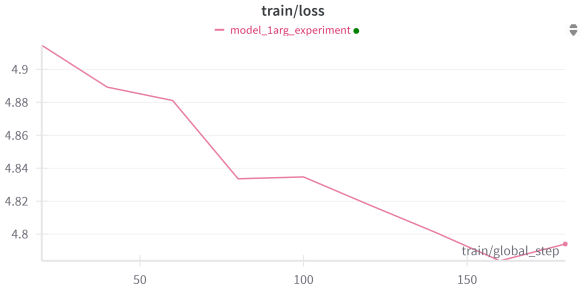

FIGURE 3 – courbe loss du training.

Nous observons que le modèle apprend sur les données train par la baisse de la courbe loss. Il fait une séparation sémantique en voyant le data train.

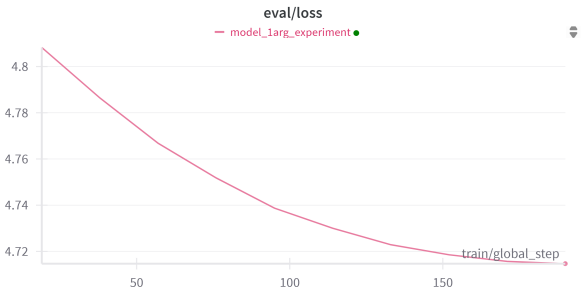

FIGURE 4 – courbe loss de train_test valid.

Nous observons une baisse de la loss dans le train_test valid qui nous indiquerait que le modèle sépare mieux les termes que sa version zero shot au cours de son entrainement

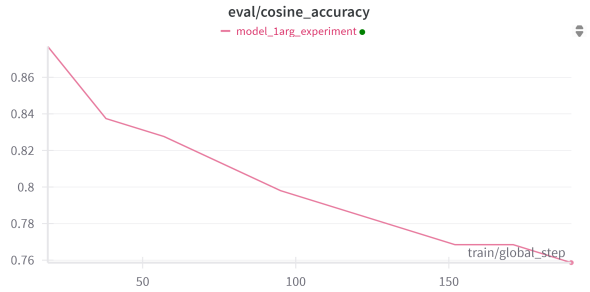

FIGURE 5 – courbe cosine_accuracy de train_test vali.

Nous observons que le cosine_accuracy du train_test valid baisse tout aussi ce quui parait contre intuitif vu la baisse de sa loss.

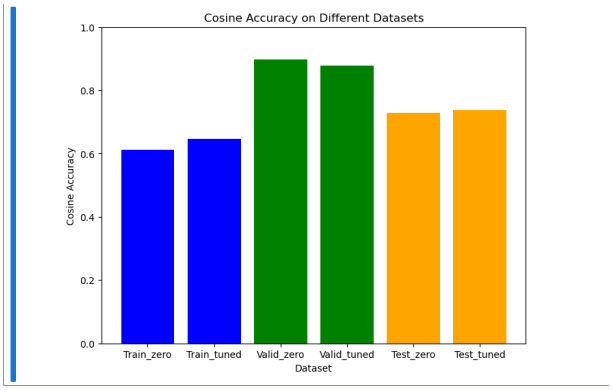

FIGURE 6 – Comparaison de performances entre le zero shot et le fine tuned

Sur le dataset d'entrainement, nous observons une amélioration du score comparé au zero shot. En ce qui concerne le valid, le modèle performe moins et on constate une légère amélioration sur l'ensemble.

Quels sont les aspects critiques de ces résultats par rapport à nos attentes ?

- Le modèle va s'améliorer évidemment sur le dataset d'entrainement car il a déjà vu et s'ajuste selon ces données. Il est nécessaire de tester sur un ensemble de documents de la même classe pour juger plus rigoureusement de son efficacité.
- Les données sont bruitées lexicalement.
- Le fine-tuning corrige partiellement ces biais, mais reste dépendant de la qualité des exemples d'entraînement et de leur quantité.

5 Connaissances acquises

Qu'avons-nous appris de cette expérimentation ?

Que s'est-il passé par rapport à ce que nous attendions ?

- Le modèle zero-shot a eu de meilleures performances que prévu.
- Le fine tuned se trompait sur des données que le zero shot réussit
- Certains cas ambigus restent problématiques, même après adaptation.

Qu'avons-nous bien fait ? Et mal fait ?

Points positifs :

- Choix judicieux du modèle.
- Pipeline complet et reproductible.
- Des hyperparamètres standards

Points à améliorer :

- Augmenter le volume et la diversité. Avoir un dataset représentatif
- Intégrer d'autres métriques d'évaluation(ex : Cosine_spearman) pour être plus objectif dans l'évaluation des relations sémantiques
- Poursuivre l'optimisation du fine-tuning en variant les hyperparamètres.

6 Propositions pour la suite

Quelle devrait être notre prochaine expérimentation ?

Nous proposons les pistes suivantes :

- Étendre le fine-tuning à un corpus plus large de brevets nombreux par classe.
- Tester d'autres modèles légers du MTEB leaderboard (ex. `all-MiniLM`, `e5-small-v2`) (En faisant attention aux ressources) .

Que s'attend-on à apprendre ? Quels défis scientifiques seraient abordés ?

Cette suite expérimentale permettrait de :

- Approfondir la compréhension des relations sémantiques dans un contexte technique.
- Évaluer la généralisation des modèles à des domaines spécifiques.

Les défis abordés restent :

- La détection fine de similarité sur des formulations variées.
- La robustesse des représentations face à la diversité industrielle.
- L'adaptation légère mais efficace de modèles à faibles ressources.

7 Bibliographie

[1] *MTEB leaderboard*. URL : https://huggingface.co/spaces/mteb/leaderboard. (Date de consultation : 16/04/2025).

[2] *adapter documentation*. URL : https://docs.adapterhub.ml/overview.html. (Date de consultation : 16/04/2025).