

Tutoriel

Embeddings · Recherche · NLP

Construire une recherche sémantique avec des embeddings

Indexer quelques fiches de cours avec des embeddings et comparer une recherche sémantique à une recherche par mots-clés.

Auteur **Youness Boutyour**

**DURÉE**

55 minutes

NIVEAU

Intermédiaire

FORMAT

Tutoriel pratique

Embeddings · Recherche · NLP · publié le 2026-07-02 · hybrid

Objectifs et livrables

Une personne qui cherche « travailler à distance » peut vouloir retrouver une fiche intitulée « télétravail ». Ce type de décalage de vocabulaire est un bon point de départ pour découvrir la recherche sémantique. Nous allons construire un petit moteur de recherche sur dix fiches et observer son classement.

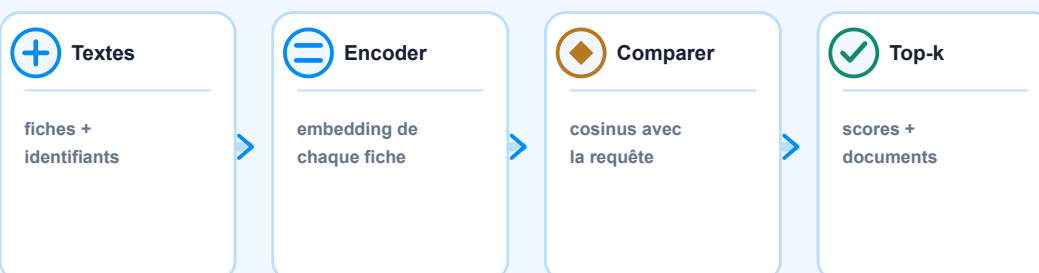
Vous préparerez les textes et leurs identifiants, calculerez leurs embeddings, puis comparerez les vecteurs à celui d'une question avec la similarité cosinus. Les métadonnées permettront de relier chaque score à la fiche correspondante. Un mode de secours est également prévu pour vérifier le chargement et le tri lorsque le modèle local n'est pas disponible.

Nous comparerons les résultats à une recherche exacte afin de repérer les différences entre correspondance de mots et proximité sémantique. Vous examinerez les premiers documents proposés et la pertinence de leurs scores, en gardant à l'esprit que le classement dépend du modèle et du corpus.

Les fichiers `outputs/index.json` et `outputs/search.json` rendront l'index et les recherches consultables. À partir de ces résultats, vous pourrez préparer vos propres requêtes de référence et évaluer la qualité du moteur sur un cas plus proche de vos données.

Comparer des sens dans un espace vectoriel

Schéma de la recherche sémantique : textes, embeddings, similarité cosinus, classement et comparaison aux mots-clés.



● vecteurs · cosinus · top-k · baseline lexicale

recherche-semantique-embeddings

Le score mesure une proximité dans l'espace vectoriel ; il doit être lu avec les documents retournés.

Lire le flux avant de coder

Le schéma transforme les textes en points comparables dans un espace vectoriel. La similarité cosinus indique une proximité entre une question et une fiche, mais elle ne dit pas à elle seule si le document est utile. Il faut toujours lire les premiers résultats et les confronter à une requête de référence.

La comparaison avec la recherche par mots-clés sert de baseline. Si les deux méthodes divergent, observez le vocabulaire, le modèle d'embedding et la taille du corpus avant de conclure. Les scores et les identifiants exportés rendent cette analyse reproductible.

1. Préparer l'environnement

Nous commencerons par vérifier la circulation des données avant d'évaluer la qualité sémantique. Le mode de secours permet de charger les fiches, de calculer des scores et de contrôler le tri sans dépendre d'un modèle. Le mode réel ajoute le service d'embeddings configuré ; son installation et son démarrage restent distincts de ceux des bibliothèques Python.

Exécutez le lecteur de documents une fois les dix fichiers créés à l'étape suivante. Le tri des chemins donne un ordre de chargement stable et `path.stem` fournit l'identifiant dérivé du nom du fichier. Cette correspondance doit rester cohérente entre les textes, les lignes de la matrice et les métadonnées, sinon un bon score pourrait être affiché à côté du mauvais document.

Terminal · Bash

```
mkdir -p recherche-semantic-embeddings/{data,src,outputs}
cd recherche-semantic-embeddings
python3 -m venv .venv && source .venv/bin/activate
python -m pip install numpy scikit-learn requests
```

Sortie obtenue

Sortie / exemple

```
Successfully installed numpy scikit-learn requests
```

Commande de vérification

Terminal · Bash

```
python -c "import numpy, sklearn; print('vector stack ok')"
```

Code de l'étape

Python

```
from pathlib import Path

documents = []
for path in sorted(Path("data").glob("*.md")):
    text = path.read_text(encoding="utf-8").strip()
    documents.append({"doc_id": path.stem, "source": path.name, "text": text})
assert len(documents) == 10
```

2. Préparer le corpus et les identifiants

Choisissez des fiches assez différentes pour comprendre les premiers résultats, mais conservez quelques sujets voisins : qualité, ETL et gouvernance rendent le classement moins trivial. Les noms de fichiers constituent ici les identifiants ; renommer une fiche change donc son identité dans les exports. Modifiez plutôt son contenu lorsque vous souhaitez uniquement préciser une description.

Les assertions contrôlent le nombre de fiches et l'unicité des identifiants, pas leur pertinence. Relisez les textes et repérez ceux qui devraient répondre à votre question de test. Cette petite référence manuelle servira à interpréter le top-3. Les identifiants affichés dans les exemples illustrent le format : ceux de votre exécution doivent correspondre aux fichiers réellement présents.

Créez dix fiches métier dans `data/` et attribuez à chacune un `doc_id` stable. Les fiches contiennent assez de contexte pour distinguer des questions proches ; ne mélangez pas le texte avec les métadonnées de provenance.

Code de l'étape

Python

```
from pathlib import Path

documents = {
    "deployment": "Promouvoir un service versionné après les tests et vérifier le retour arrière.",
    "etl": "Contrôler les doublons, les valeurs manquantes et le schéma de chaque lot ETL.",
    "experimentation": "Comparer les modèles avec un protocole reproductible et un jeu de validation.",
    "feature-store": "Garantir la fraîcheur des variables et la même transformation en entraînement et en ligne.",
    "governance": "Définir les responsabilités, les droits d'accès et la conservation des données.",
    "mlops": "Suivre les versions, les métriques, les artefacts et la dérive après déploiement.",
    "model-monitoring": "Surveiller la latence, les erreurs et la distribution des prédictions.",
    "quality": "Contrôler les valeurs manquantes, les doublons et les règles de validation.",
    "rag": "Retrouver des passages pertinents et conserver les sources d'une réponse générée.",
    "security": "Limiter les privilèges, journaliser les accès et protéger les secrets.",
```

```
}
for topic, description in documents.items():
    Path(f"data/{topic}.md").write_text(f"# {topic}\n\n{description}\n", encoding="utf-8")
doc_ids = [path.stem for path in Path("data").glob("*.md")]
assert len(doc_ids) == len(set(doc_ids)) == 10
```

Sortie obtenue

Chaque document possède un identifiant unique et une catégorie.

■ Sortie / exemple

```
documents=10 ids_unique=10 categories=10 average_chars=145
```

Commande de vérification

Terminal · Bash

```
python src/index.py --mock
python -m json.tool outputs/index.json >/dev/null
```

3. Calculer les vecteurs et rechercher

Le code ci-dessous montre le chemin TF-IDF de secours : il utilise des correspondances lexicales pondérées, et non les embeddings d'un modèle de langue. Il valide le calcul de proximité et le classement, mais ne démontre pas qu'une paraphrase est comprise. Pour examiner cet aspect, lancez le mode réel en conservant les mêmes documents et les mêmes requêtes.

La similarité cosinus compare l'orientation des vecteurs et ne s'interprète pas comme une probabilité de pertinence. Dans cet extrait, le vectoriseur est ajusté sur la question et les documents ensemble : les poids peuvent donc changer avec la question. Pour comparer des exécutions, notez le mode utilisé et ne mélangez pas directement les valeurs obtenues par TF-IDF avec celles d'un modèle d'embeddings.

Appelez l'endpoint d'embeddings local si disponible ; sinon, activez `--mock` pour produire une représentation déterministe. Normalisez les vecteurs avant le produit scalaire.

Sortie obtenue

Une question renvoie trois résultats triés par score décroissant, avec `doc_id` et catégorie.

Sortie / exemple

```
query="qualité des données" top_k=3
1 doc_id=doc-07 score=0.82 category=data-quality
2 doc_id=doc-02 score=0.74 category=governance
3 doc_id=doc-09 score=0.61 category=monitoring
```

Le mode réel utilise `/v1/embeddings` avec `GENAI_EMBED_MODEL` ; lancez `python src/search.py --real "comment vérifier la qualité des données ?" --top-k 3` après avoir configuré l'endpoint.

Code de l'étape

Python

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics.pairwise import cosine_similarity

vectorizer = TfidfVectorizer(strip_accents="unicode")
matrix = vectorizer.fit_transform([question] + [doc["text"] for doc in documents])
scores = cosine_similarity(matrix[0:1], matrix[1:])[0]
results = sorted(
    ({doc_id: doc["doc_id"], "score": float(score)}
     for doc, score in zip(documents, scores)),
    key=lambda item: item["score"], reverse=True,
)[:3]
```

Commande de vérification

Terminal · Bash

```
python src/search.py --mock "comment vérifier la qualité des données ?" --top-k 3
```

4. Comparer avec une recherche exacte

Cette comparaison répond à une question précise : quels documents chaque méthode fait-elle apparaître ? Les ensembles d'identifiants permettent de séparer les résultats communs des résultats propres à une méthode. Ils ne disent pas encore lesquels sont utiles ; ouvrez les fiches concernées et justifiez leur présence par rapport à la question, sans vous limiter à leur score.

L'extrait de recherche exacte teste la présence du mot `qualité` . Ce choix rend le fonctionnement visible, mais ne constitue pas un moteur lexical complet. Un document retrouvé uniquement par le mode sémantique peut être pertinent ou être un faux positif. Testez plusieurs formulations et conservez les jugements associés ; un top-3 différent ne prouve pas, à lui seul, une meilleure recherche.

Écrivez une question dont les mots diffèrent du corpus. Comparez les trois premiers résultats du mode sémantique et du mode mots-clés dans `outputs/comparison.json`.

Sortie obtenue

La comparaison indique les documents communs et les documents retrouvés par un seul mode.

■ Sortie / exemple

```
semantic_hits=3 exact_hits=1 overlap=1 semantic_only=2
```

Commande de vérification

Terminal · Bash

```
python src/compare.py --mock
test -s outputs/comparison.json
```

Code de l'étape

Python

```
semantic_ids = {item["doc_id"] for item in results}
keyword_ids = {doc["doc_id"] for doc in documents
               if "qualité" in doc["text"].casefold()}
comparison = {
    "semantic_only": sorted(semantic_ids - keyword_ids),
    "keyword_only": sorted(keyword_ids - semantic_ids),
}
Path("outputs/comparison.json").write_text(
    json.dumps(comparison, indent=2, encoding="utf-8")
)
```

Exemples de sorties

L'index contient dix documents, la recherche retourne trois résultats triés et la comparaison est enregistrée. Les scores exacts changent avec le modèle d'embeddings.

Dépannage et diagnostics

Un score ne suffit pas à établir la pertinence. Inspectez le texte des trois résultats et conservez la version du modèle, la dimension des vecteurs et la question originale.

Vérification finale

- ☐ identifiants stables ;
- ☐ vecteurs normalisés ;
- ☐ top-3 trié ;
- ☐ comparaison sémantique/mots-clés enregistrée.

Limites et prolongements

Un corpus plus grand nécessite une base vectorielle et une stratégie de mise à jour. La qualité doit être évaluée sur des questions annotées.

Références

- [Ollama — embeddings](#)
- [scikit-learn — cosine similarity](#)
- [Wikimedia Commons — HNSW \(CC BY 4.0\)](#)