Hard-Mining Negatives for Semantic Similarity Model using Sentence Transformers

Jithinanievarghese · Follow

8 min read · Jul 17

A hard-mined negative sample is one which is similar to the anchor but not an exact match with the anchor 😀

Semantic Similarity

Semantic Similarity is the task of evaluating how similar two texts are in terms of meaning. It plays a vital role in an information retrieval pipeline, whether it is product matching in eCommerce or finding a relevant document for a query.

In product matching of eCommerce, matching relevant or exact products from different eCommerce websites will provide valuable insights into pricing data, market dynamics, and competitor practices.

Training a semantic textual similarity model on eCommerce data helps us to retrieve exact or relevant products by extracting the information from the brand, title, specification, and description of the products. However semantic models suffer from the lack of availability of informative negative examples for model training.

What are actually informative negative samples or hard negatives and how do they help in the training of the model?

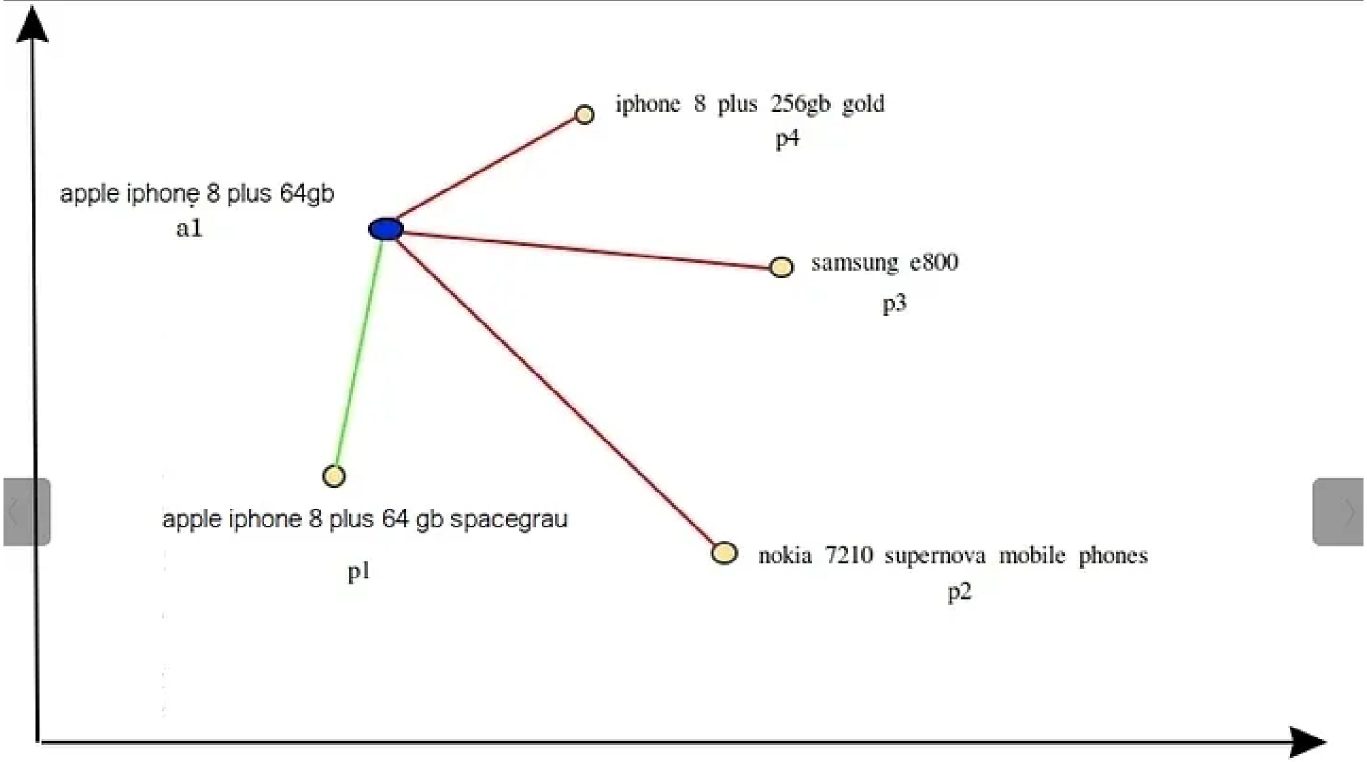
Let's say we have eCommerce product matching task and we have only true labels in our data like the following kaggle dataset which contains only matching product title pairs. The dataset does not provide us with dissimilar pairs because the data was scraped from price comparison websites and they only contain matching pairs. From now on we will be addressing, matching product title pairs as **anchor-positive** pairs. Where '*apple iphone 8 plus 64gb*' is an anchor and '*apple iphone 8 plus 64 gb spacegrau'* will be it's positive.

	Product ID	Product Title	Vendor ID	Cluster ID	Cluster Label	Category ID	Category Label
0	2	apple iphone 8 plus 64 gb spacegrau	2	1	Apple iPhone 8 Plus 64GB	2612	Mobile Phones
22	24	apple mnqq2b/a iphone 7 plus 32gb 5.5 12mp sim...	3	2	Apple iPhone 7 Plus 32GB	2612	Mobile Phones
44	46	apple grade b iphone 7 32gb gold handset only	5	3	Apple iPhone 7 32GB	2612	Mobile Phones
66	68	startech.com usb c to hdmi multi monitor adapt...	17	4	Apple iPhone 8 64GB	2612	Mobile Phones
88	90	apple iphone x 64gb space grey	1	5	Apple iPhone X 64GB	2612	Mobile Phones
...	...	...	...	...	...	...	...
4075	4087	samsung e800	48	1814	Samsung SGH-E800	2612	Mobile Phones
4076	4088	nokia 7600	48	1815	Nokia 7600	2612	Mobile Phones
4077	4089	nokia 1100	48	1816	Nokia 1100	2612	Mobile Phones
4078	4090	nokia 6310i silver	48	1817	Nokia 6310i	2612	Mobile Phones
4079	4091	nokia 7210 supernova mobile phones	48	1818	Nokia 7210	2612	Mobile Phones

We can train a semantic similarity model using only anchor-positive pairs with Sentence Transofrmer framework with **MultipleNegatives RankingLoss (MNR)** Loss. Please check out how Sentence Transformer Library can be used for building better semantic models than other techniques like using a BERT encoder. Training or fine-tuning a semantic similarity model using a sentence transformer is pretty simple with few lines of code.

```
# Training or Fine Tuning a sentence transformer model with MNR Loss
from sentence_transformers import SentenceTransformer, losses, InputExample
from torch.utils.data import DataLoader
# where all-MiniLM-L6-v2 is a pre-trained  sentence-transformer model
model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')
train_examples = [InputExample(texts=['Anchor 1', 'Positive 1']),
    InputExample(texts=['Anchor 2', 'Positive 2'])]
train_dataloader = DataLoader(train_examples, shuffle=True, batch_size=32)
train_loss = losses.MultipleNegativesRankingLoss(model=model)
model.fit(train_objectives=[(train_dataloader, train_loss)], epochs=1, warmup_st
```

In MNR Loss, for each anchor, it uses all other positives as a negative sample. Here (a1, p1) will be the positive pairs, and p2, p3, and p4 (positive of other anchors) will be made as a negative sample.



Vector Space embeddings Illustration in which (a1, p1) will be the positive pairs and p1, p2, p3, and p4 will be negative samples.

> *For each a_i, it uses all other p_j as negative samples, i.e., for a_i, we have 1 positive example (p_i) and n-1 negative examples (p_j). It then minimizes the negative log-likehood for softmax normalized scores.*

One of the issues with this random assigning or generation of negatives is that model trained using only random negatives, places two dis-similar queries closer to each other in the embedding space, especially when such queries have shared tokens.

Also, hard negatives samples give better performance than random negatives for semantic similarity as detailed by Nils Reimers in the following video.

> “A hard-mined negative sample is the one which is similar to anchor but not an exact match with anchor”.

For ***“apple iphone 7 32gb’”*** the hard negative sample will be ***“apple iphone 7 256gb product red’”,*** since both are similar but not an exact match due to different storage sizes. So it is better than a random negative sample like ***“samsung -e800”*** *or* ***“nokia 7210 supernova mobile phones”.*** It will give us a better generalization and performance in identifying the relevant products in the product matching pipeline.

After the generation of hard negatives, we can train our model using **Triplet Loss**

> *Given a triplet of (anchor, positive, negative), the loss minimizes the distance between anchor and positive while it maximizes the distance between anchor*

> *and negative*
>
> *loss = max(||anchor — positive|| — ||anchor — negative|| + margin, 0) , where margin is an important hyperparameter and needs to be tuned respectively.*

```
# Training or Fine Tuning a sentence transformer model with Triplet Loss
from sentence_transformers import SentenceTransformer,  SentencesDataset, Loggin
from sentence_transformers.readers import InputExample

model = SentenceTransformer('sentence-transformers/all-MiniLM-L6-v2')
train_examples = [InputExample(texts=['Anchor 1', 'Positive 1', 'Negative 1']),
    InputExample(texts=['Anchor 2', 'Positive 2', 'Negative 2'])]
train_dataset = SentencesDataset(train_examples, model)
train_dataloader = DataLoader(train_dataset, shuffle=True, batch_size=train_batc
train_loss = losses.TripletLoss(model=model)
model.fit(train_objectives=[(train_dataloader, train_loss)], epochs=1, warmup_st
```

Mining Hard-Negatives using a naive and simple approach

There are several approaches for generating informative negative samples, like the one which is detailed in the latest paper of amazon science Beyond hard negatives in product search: Semantic matching using one-class classification (SMOCC).

But here, we are following a naive and simple approach to generate hard negative samples with the help of sentence transformer model embeddings and intuitive cosine similarity thresholds. Duplicates were removed from the dataset to avoid the chance of duplicated positive pairs.

	cluster_id	anchor	positive	category
0	1	apple iphone 8 plus 64gb	apple iphone 8 plus 64 gb spacegrau	Mobile Phones
1	2	apple iphone 7 plus 32gb	apple mnqq2b/a iphone 7 plus 32gb 5.5 12mp sim...	Mobile Phones
2	3	apple iphone 7 32gb	apple grade b iphone 7 32gb gold handset only	Mobile Phones
3	4	apple iphone 8 64gb	startech.com usb c to hdmi multi monitor adapt...	Mobile Phones
4	5	apple iphone x 64gb	apple iphone x 64gb space grey	Mobile Phones
...	...	...	...	...
1697	1814	samsung sgh-e800	samsung e800	Mobile Phones
1698	1815	nokia 7600	nokia 7600	Mobile Phones
1699	1816	nokia 1100	nokia 1100	Mobile Phones
1700	1817	nokia 6310i	nokia 6310i silver	Mobile Phones
1701	1818	nokia 7210	nokia 7210 supernova mobile phones	Mobile Phones

1699 rows × 4 columns

Our primary motive is to find the most similar positive sample (from other anchors), which is not an exact match by comparing the cosine similarity of sentence embeddings. We will be using the pre-trained sentence transformer model all-MiniLM-L6-v2 for generating the embeddings for our sentences. Let's dive into code.

```
from sentence_transformers import SentenceTransformer, util
import numpy as np


class HardMineNegatives():
    """
    Hard-mining Negatives for training a semantic similairty task with Triplet L
    Here we find the nearest negatives of a query in a search pool
    by using sentence transformer model embeddings and cosine similarity ratio.
    param: model_path: path of sentence transformer model
    param: search_max_threshold:  maximimum cosine similarity ratio
    param: search_min_threshold: minimum cosine similarity ratio
    param: search_limit: total length of data in which we want to search, only i
    param: top_n_results: number of top nearest negative  to be returned, defaul
    """
    def __init__(self, model_path: str, **kwargs):
```

```
        self.model = SentenceTransformer(model_path)
        self.search_max_threshold = kwargs['search_max_threshold'],
        self.search_min_threshold = kwargs['search_min_threshold']
        self.search_limit = kwargs.get('search_limit')
        self.top_n_results = kwargs.get('top_n_results') if kwargs.get('top_n_re

    def get_hard_mined_negatives(self, anchor: str,  search_pool:np.ndarray):
        """
        to retrieve embeddings from sentence transformer model for anchor and se
        find the cosine similairty ratio between the  anchor and search pool sen
        apply search thresholds and return the top nearest negatives based on th
        cosine similarity scores.
        if no data is found in between the self.search_max_threshold and self.se
        we will take the results between 0 and less than self.search_min_thresho

        param: anchor: source text to which we need to find the nearest negative
        param: search_pool: numpy array of sentences from which
               we need to find the cosine similarity ratios with the anchor.
               any meta value for sentences can be given after next index of
               sentence, in the form
               search_pool = array([
                    ['apple iphone 8 256 gb gold', "mobile", "1001"],
                    ['apple iphone 7 plus 32gb silver', "mobile", "1002"]])
               where "mobile", "1001" are meta values,
               the returned results will contain the respective cosine similarit
               ratio at the last index of each sentence array
               result = array([
                    ['apple iphone 8 256 gb gold', "mobile", "1001", 69.5],
                    ['apple iphone 7 plus 32gb silver', "mobile", "1002", 70.5]]
               where 69.5 and 70.5 are cosine similarity ratios.
        """
        self.search_limit = self.search_limit if self.search_limit else search_p
        search_pool = search_pool[: self.search_limit]
        # shuffle data to search in random pool of data, in case of search limit
        np.random.shuffle(search_pool)
        sentences = [anchor] + [row[0] for row in search_pool]
        embeddings = self.model.encode(sentences, convert_to_tensor=False)
        source_vector = embeddings[0]
        # calculate the cosine similairty with the other sentences in search poo
        similarity = [round(util.cos_sim(source_vector, embed).numpy()[0][0]*100
        similarity = np.array(similarity)
        negative_indices = np.where((similarity <= self.search_max_threshold) &
        if not negative_indices[0].shape[0]:
            negative_indices = np.where((similarity < self.search_min_threshold)
        negative_indices = negative_indices[0]
        # take respective selected indices
        search_pool = np.take(search_pool, negative_indices, axis=0)
        similarity = np.take(similarity, negative_indices, axis=0)
        # reshape to concatenate with meta values of search pool
        similarity = similarity.reshape(-1, 1)
```

```
        # concat the ratio to the meta values of search pool
        search_pool = np.concatenate((search_pool, similarity), axis=1)
        # sort the data in descending order
        search_pool = search_pool[search_pool[:, -1].argsort()][::-1]
        return search_pool[:self.top_n_results]
```

```
anchor = 'apple iphone 7 32gb'
search_pool = df[df.anchor != anchor]
search_pool.reset_index(drop=True, inplace=True)
search_pool = search_pool.drop_duplicates()
search_pool = search_pool.loc[:, ['positive', 'cluster_id']]
search_pool = search_pool.to_numpy()
```

```
model_path = 'sentence-transformers/all-MiniLM-L6-v2'
```

```
print(f'Mining negatives for "{anchor}"')
obj = HardMineNegatives(
    model_path=model_path,
    search_max_threshold=65,
    search_min_threshold=50,
    search_limit=None,
    top_n_results=3)
```

```
Mining negatives for "apple iphone 7 32gb"
```

```
%%time
top_results = obj.get_hard_mined_negatives(anchor, search_pool)
top_results = pd.DataFrame(top_results, columns=['negative', 'cluster_id', 'cosine_similairty_ratio'])
top_results
```

```
CPU times: user 26.2 s, sys: 52.1 ms, total: 26.3 s
Wall time: 4.47 s
```

	negative	cluster_id	cosine_similairty_ratio
0	apple iphone 7 256gb product red	470	64.37
1	iphone xr 64gb red	807	63.3
2	apple iphone 8 256 gb red	65	62.81

Top 3 Hard Negatives for `apple iPhone 7 32gb`

For anchor ‘*apple iphone 7 32gb’,* we search the negatives in other positive data as in the above code. We got great results like *‘apple iphone 7 256gb product red’* and ‘*apple iphone 8 256 gb red’* which are similar and not an exact match (like Virat Kohli doppelgangers 😛). So here we searched for the top 3 similar sentences that have a cosine similarity ratio between 65 and 50.

Other Examples

```
print(f'Mining negatives for "{anchor}"')
obj = HardMineNegatives(
    model_path=model_path,
    search_max_threshold=65,
    search_min_threshold=50,
    search_limit=None,
    top_n_results=3)
```

```
Mining negatives for "samsung sgh-e800"
```

```
%%time
top_results = obj.get_hard_mined_negatives(anchor, search_pool)
top_results = pd.DataFrame(top_results, columns=['negative', 'cluster_id', 'cosine_similairty_ratio'])
top_results
```

```
CPU times: user 26.4 s, sys: 116 ms, total: 26.5 s
Wall time: 4.52 s
```

	negative	cluster_id	cosine_similairty_ratio
0	samsung e1100 mobile phone	1595	64.65
1	samsung m8800 pixon mobile phone	1659	64.47
2	samsung i780 mobile phone	1724	63.71

```
print(f'Mining negatives for "{anchor}"')
obj = HardMineNegatives(
    model_path=model_path,
    search_max_threshold=65,
    search_min_threshold=50,
    search_limit=None,
    top_n_results=3)
```

```
Mining negatives for "google pixel 2 64gb"
```

```
%%time
top_results = obj.get_hard_mined_negatives(anchor, search_pool)
top_results = pd.DataFrame(top_results, columns=['negative', 'cluster_id', 'cosine_similairty_ratio'])
top_results
```

```
CPU times: user 26.2 s, sys: 84.2 ms, total: 26.3 s
Wall time: 4.48 s
```

	negative	cluster_id	cosine_similairty_ratio
0	google pixel 128 gb 5 quite black android smar...	587	60.81
1	motorola g6 play 32 gb blue	78	59.06
2	motorola moto g6 play 32gb deep indigo	47	57.39

Top 3 Hard Negatives for 'samsung sgh-e800' and 'google pixel 2 64gb'

Why we chose a cosine similarity ratio between 65 and 50?

Choosing 65 as a maximum and 50 as a minimum threshold is a rule-based decision or task-specific. Since the sentences that are above 70 or 80, or 90 may contain samples that are an exact match (outliers in data), even if we have unique anchor pairs like the following case.

```
print(f'Mining negatives for "{anchor}"')
obj = HardMineNegatives(
    model_path=model_path,
    search_max_threshold=95,
    search_min_threshold=75,
    search_limit=None,
    top_n_results=5)
```

```
Mining negatives for "apple iphone 8 plus 64gb"
```

```
%%time
top_results = obj.get_hard_mined_negatives(anchor, search_pool)
top_results = pd.DataFrame(top_results, columns=['negative', 'cluster_id', 'cosine_similairty_ratio'])
top_results
```

```
CPU times: user 30.7 s, sys: 128 ms, total: 30.9 s
Wall time: 5.24 s
```

	negative	cluster_id	cosine_similairty_ratio
0	apple iphone 8 plus 64 gb rt	77	90.83
1	apple iphone 8 plus 256gb gold	9	75.36

```
df[df.positive == 'apple iphone 8 plus 64 gb rt']
```

	cluster_id	anchor	positive	category
76	77	apple iphone 8 plus (product) red special edit...	apple iphone 8 plus 64 gb rt	Mobile Phones

Outlier in data even after maintaining unique anchor positive pairs.

When we set 95 as a maximum threshold and searched for top similar negative samples we got `*apple iphone 8 plus 64 gb rt*`, a relevant product to anchor '*apple iphone 8 plus 64gb*'. These negative samples can hinder the model performance if their count is high in the training data.

Running whole rows in the CPU had latency issues so I ran the code in the following Kaggle notebook with GPU. So the final results we got are

```
final_results
```

	cluster_id	anchor	positive	negative	anchor_negative_cosine_similairty_ratio	category
0	1	apple iphone 8 plus 64gb	apple iphone 8 plus 64 gb spacegrau	apple iphone 7 256 gb matt schwarz	64.87	Mobile Phones
1	2	apple iphone 7 plus 32gb	apple mnqq2b/a iphone 7 plus 32gb 5.5 12mp sim...	apple iphone 5s 64gb gold	63.86	Mobile Phones
2	3	apple iphone 7 32gb	apple grade b iphone 7 32gb gold handset only	apple iphone 7 256gb product red	64.37	Mobile Phones
3	4	apple iphone 8 64gb	startech.com usb c to hdmi multi monitor adapt...	iphone xr 64gb white	64.76	Mobile Phones
4	5	apple iphone x 64gb	apple iphone x 64gb space grey	apple iphone 8 plus 256gb gold	64.84	Mobile Phones
...	...	...	...	...	...	...
1696	1814	samsung sgh-e800	samsung e800	samsung e1100 mobile phone	64.65	Mobile Phones
1697	1815	nokia 7600	nokia 7600	nokia 6500 slide mobile phone	64.81	Mobile Phones
1698	1816	nokia 1100	nokia 1100	nokia 5530 illuvial mobile phone	64.91	Mobile Phones
1699	1817	nokia 6310i	nokia 6310i silver	nokia 6212 nfc mobile phone	65.00	Mobile Phones
1700	1818	nokia 7210	nokia 7210 supernova mobile phones	nokia 2600 classic mobile phone	64.90	Mobile Phones

1701 rows × 6 columns

Final results

Conclusion

After mining the negatives we can merge the data to the kaggle dataset and train our model using Sentence Transformer with Triplet Loss as Loss function. All training steps are detailed in the official documentation of Sentence Transformers.

In real-world product matching problems we won't be using title alone for mining hard negatives, we will be using brand, title, specification, and description for the generation of better informative samples. Also, here we took only one category for mining ie **`Mobile Phones`**. When we have more category data it is preferred to search for negatives in the respective categories since they have the highest probability of having better hard negatives. ie category **`Cameras`** may not generate better negatives for **`Mobile Phones`**. It also saves the latency in searching the hard negatives in the search pool.

Please let me know in the comments if you find this article useful and feel free to mention any corrections which I need to make in the future. You can reach out to me on my Linkedin profile

Thanks for reading 😊

NLP Semantic Similarity Sentence Transformers Information Retrieval Product Matching

More from the list: "NLP"

Curated by Himanshu Birla

Jon Gi... in Towards Data ...

Characteristics of Word Embeddings

11 min read · Sep 4, 2021

Jon Gi... in Towards Data ...

The Word2vec Hyperparameters

6 min read · Sep 3, 2021

Jon Gi... in

The Word2ve

15 min rea

View list

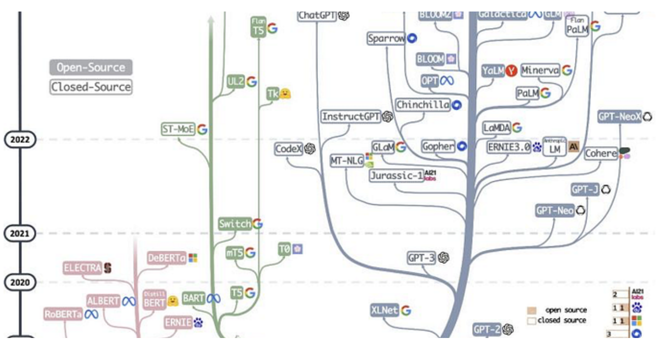
Written by Jithinanievarghese

Follow

3 Followers

Python Developer working in eCommerce domain , developing product matching algorithms using NLP, ML and Deep Learning

Recommended from Medium



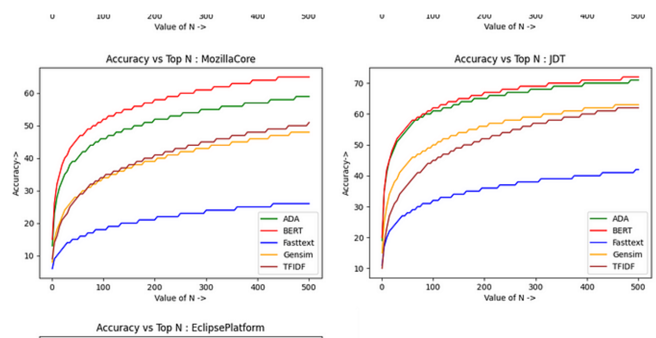
Haifeng Li

A Tutorial on LLM

Generative artificial intelligence (GenAI), especially ChatGPT, captures everyone's...

15 min read · Sep 14

372



Avinash Patil

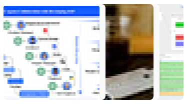
Embeddings: BERT better than ChatGPT4?

In this study, we compared the effectiveness of semantic textual similarity methods for...

4 min read · Sep 19

3 1

Lists

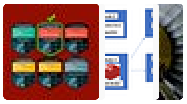


Natural Language Processing

669 stories · 283 saves

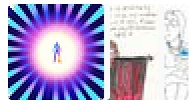
The New Chatbots: ChatGPT, Bard, and Beyond

13 stories · 133 saves



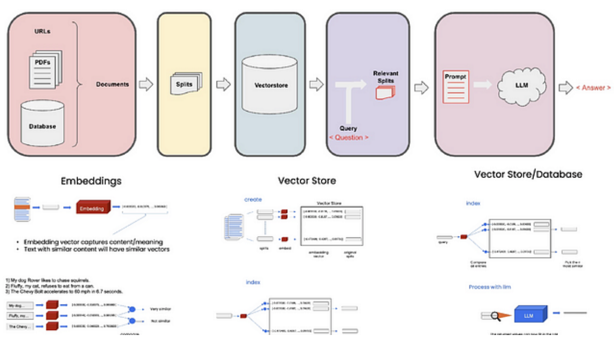
New_Reading_List

174 stories · 133 saves



Staff Picks

465 stories · 317 saves



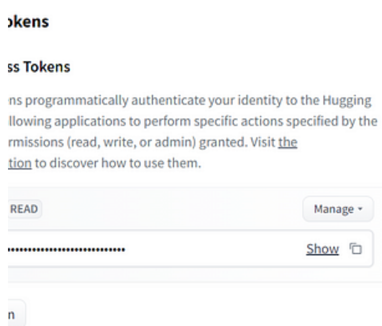
TeeTracker

Chat with your PDF (Streamlit Demo)

Conversation with specific files

4 min read · Sep 15

56



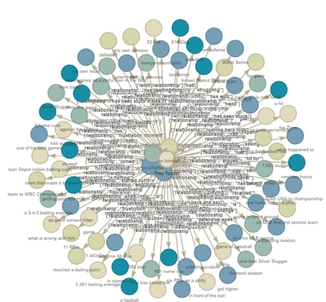
Ankit

Generating Summaries for Large Documents with Llama2 using...

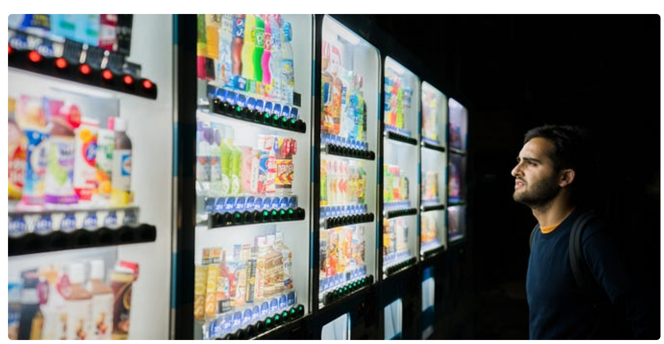
Introduction

11 min read · Aug 28

103 3



Wenqi Glantz in Better Programming



Ovbude Ehi

7 Query Strategies for Navigating Knowledge Graphs With...

Exploring NebulaGraph RAG Pipeline with the Philadelphia Phillies

· 17 min read · 4 days ago

501 4

Recommendation Engines

Practical Techniques: Content-Based Filtering, Collaborative Filtering and...

11 min read · May 8

5

See more recommendations