



Search Medium

Write



Empowering Natural Language Processing with OpenAI Embeddings: Text Similarity, Semantic Search, and Clustering

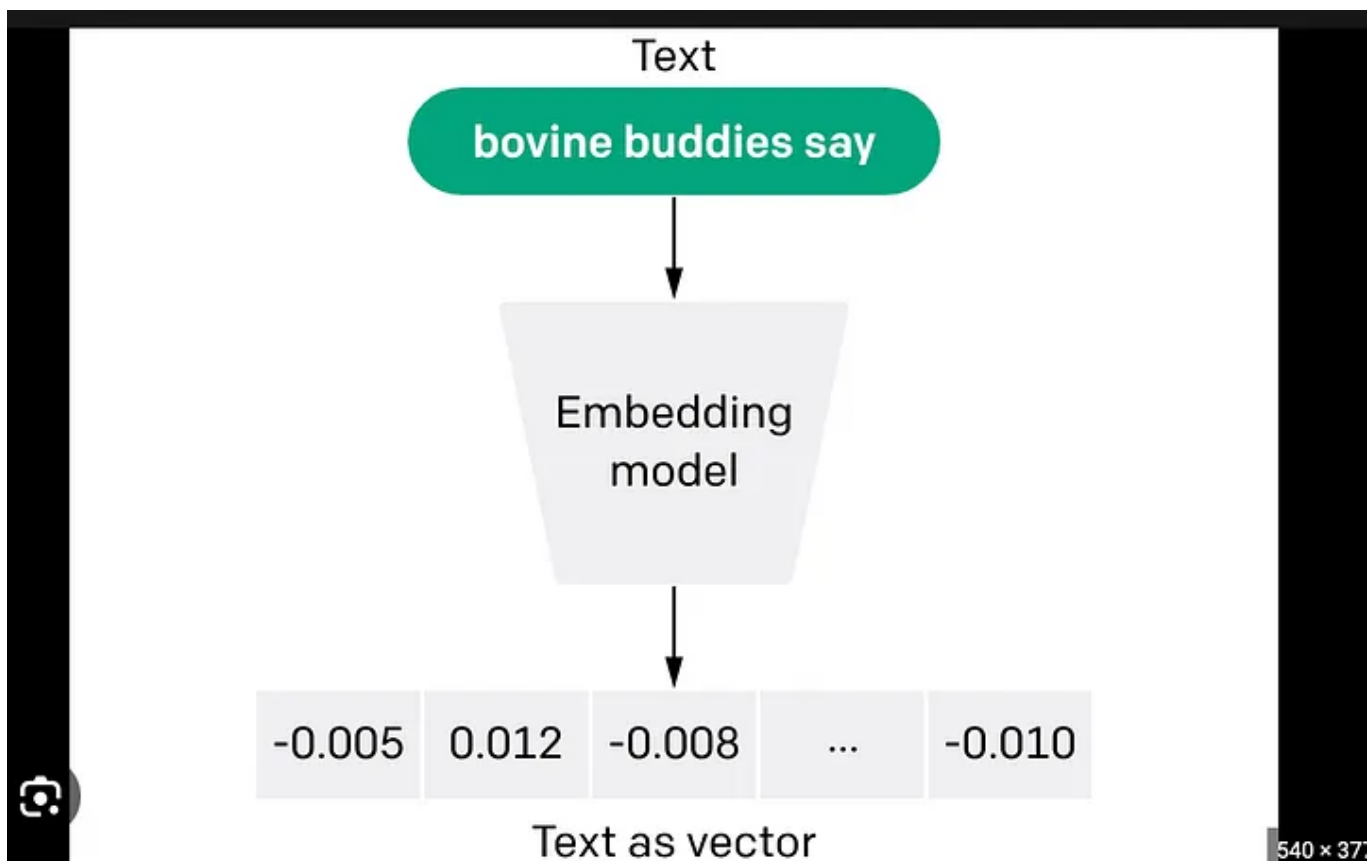


Tejpal Kumawat · Following

6 min read · Jun 11



6



Introduction

OpenAI's groundbreaking embedding and transcription models revolutionize NLP and speech recognition, enhancing accuracy and efficiency. This blog explores OpenAI Embeddings' potential for advanced NLP tasks, while the next focuses on Whisper transcription models. We delve into word embeddings' basics, advantages, and OpenAI's superior performance. Discover applications like text similarity, semantic search, and clustering, as we unveil OpenAI Embeddings' transformative power in NLP.

What Are Word Embeddings?

NLP faces a crucial challenge: effectively representing language for machine learning. Embeddings tackle this issue. Embeddings are vector representations in high-dimensional space that capture the meaning and relationships between words or phrases. By mapping words to vectors, we enable machine learning algorithms to easily process linguistic information. These vector-based representations facilitate the understanding of semantic relationships and contextual meaning. Embeddings enhance NLP tasks such as text similarity, sentiment analysis, named entity recognition, and more. They bridge the gap between human language and machine understanding, enabling sophisticated language processing and empowering various NLP applications. With embeddings, NLP algorithms can efficiently interpret and analyze textual data, opening doors to more accurate and powerful language models.

Benefits Of Using Embeddings

The advantage of employing embeddings is that they make it possible to more accurately capture the meaning of words than is possible by merely counting how frequently they appear in a document. Think of the terms "cat" and "dog" as an illustration. As each of these phrases refer to different

kinds of pets, they may show up in comparable settings and hence have comparable embeddings. In contrast, because to their lack of semantic similarity, the terms “computer” and “cat” are likely to have extremely distinct embeddings.

The Benefits of OpenAI Embeddings Over Earlier Embedding Models

Text-embedding-ada-002, a new text-embedding model from OpenAI, performs as well on text classification while outperforming all prior embedding models on text search, code search, and sentence similarity. The /embeddings endpoint's interface has been substantially simplified by combining the features of five separate models. The new model is 90% less expensive than earlier models of comparable size, has a longer context length, and a smaller embedding size, making it a more effective and efficient option for natural language processing jobs.

OpenAI Embeddings Use Cases

OpenAI embedding models and whisper gives us the power to create a diverse set of NLP apps by providing functionalities like text similarity, semantic search, clustering, etc.

Case 1: Text Similarity

OpenAI Similarity Embeddings models are good at capturing semantic similarity between two or more pieces of text.

Let's step up code:

```
!pip install openai  
import pandas as pd
```

```
import openai, numpy as np
from openai.embeddings_utils import get_embedding, cosine_similarity

api_key = 'Your API key here'
openai.api_key = api_key
```

Initialize your API key. You can get your own API key from —
<https://platform.openai.com/account/api-keys>

```
texts = ["eating food", "I am hungry", "I am traveling" , "exploring new places"]
resp = openai.Embedding.create(
    input= texts,
    engine="text-similarity-davinci-001")

embedding_a = resp['data'][0]['embedding']
embedding_b = resp['data'][1]['embedding']
embedding_c = resp['data'][2]['embedding']
embedding_d = resp['data'][3]['embedding']

li = []
for ele in resp['data']:
    li.append(ele["embedding"])

## Finding text similarity percentages
for i in range(len(texts) - 1):
    for j in range(i + 1, len(resp["data"])):
        print("text similarity percentage between",texts[i], "and", texts[j],"is")
```

Create the text-similarity-davinci-001 model from scratch. Then we give it a list of text to process as input. The embeddings of the sentences are then kept in the appropriate variables. Finally, we use the NumPy package to find the degree of similarity between the sentences using vector dot product.

Case 2: Semantic Search

```
import openai
api_key = 'Your API key here'
openai.api_key = api_key

datafile_path = "https://cdn.openai.com/API/examples/data/fine_food_reviews_with"
df = pd.read_csv(datafile_path)
df.head()
```

Using the pandas module, we loaded the data frame and stored it in the variable df. Let's examine the data frame now.

	ProductId	UserId	Score	Summary	Text	combined	n_tokens	babbage_similarity	babbage_search
0	B003XPF9BO	A3R7JR3FME8XQB	5	where does one start...and stop... with a tre...	Wanted to save some to bring to my Chicago fam...	Title: where does one start...and stop... wit...	51	[-0.01274053193628788, 0.010849879123270512, ...	[-0.01880764216184616, 0.019457539543509483, ...
1	B003JK537S	A3JBPC3WFUT5ZP	1	Arrived in pieces	Not pleased at all. When I opened the box, mos...	Title: Arrived in pieces; Content: Not pleased...	35	[-0.024154752492904663, 0.0024838377721607685, ...	[-0.03571609780192375, 0.010356518439948559, ...
2	B000JMBE7M	AQX1N6A51QOKG	4	It isn't blanc mange, but isn't bad ...	I'm not sure that custard is really custard wi...	Title: It isn't blanc mange, but isn't bad ...	277	[0.0032693513203412294, 0.017815979197621346, ...	[-0.010433986783027649, 0.024620095267891884, ...
3	B004AHGBX4	A2UY46X0OSNVUQ	3	These also have SALT and it's not sea salt.	I like the fact that you can see what you're g...	Title: These also have SALT and it's not sea s...	246	[-0.03584608808159828, 0.03424076735973358, -0...	[-0.040209852159023285, 0.0380499609687805, ...
4	B001BORBHO	A1AFOYZ9HSM2CZ	5	Happy with the product	My dog was suffering with itchy skin. He had ...	Title: Happy with the product; Content: My dog...	87	[0.005218076519668102, 0.018165964633226395, ...	[0.010450801812112331, 0.022801749408245087, ...

The data frame comprises nine columns, but we'll concentrate on two of them, babbage_similarity and babbage_search, together. The combined column basically combines the content of the text column and the summary column, which each contain the review's title and summary. While looking for similarities between two sentences, we will use the babbage_similarity function, and while looking for information within a text, we will use the babbage_search function.

```
df["babbage_search"] = df.babbage_search.apply(eval).apply(np.array)
```

```
df["babbage_similarity"] = df.babbage_similarity.apply(eval).apply(np.array)
```

Convert the contents of columns — babbage_search and babbage_similarity from string to a NumPy ndarray.

```
# search through the reviews for a specific product
def search_reviews(df, search_query, n=3):
    embedding = get_embedding(
        search_query,
        engine="text-search-babbage-query-001"
    )
    df["similarities"] = df.babbage_search.apply(lambda x: cosine_similarity(x,
    embedding))

    top_n = df.sort_values("similarities", ascending=False).head(n)
    # res = top_n.combined.str.replace("Title: ", "").str.replace("; Content:",
    return top_n

res = search_reviews(df, "delicious beans", n=3)
res['combined'].to_list()
```

We have developed a special search_reviews method that will look for the user's request in the reviews and provide the top 3 reviews that are most similar to it. Data frame, user question, and a number of reviews are the three inputs for our technique.

The semantic search embedding model is then set up inside the method. The similarity score between each review and the user query is then stored in a new column that we call similarities in our data frame.

The top three reviews with the highest similarity score are then returned after sorting our data frame in descending order according to the similarities column.

Case 3: Clustering

In the clustering process, groupings of documents that are more similar to one another than to documents in other clusters are called clusters. These clusters can be utilised for many NLP applications, including text classification, sentiment analysis, and recommendation systems, to find themes, subjects, or patterns within a dataset.

```
# source: https://stackoverflow.com/questions/55619176/how-to-cluster-similar-se
from sklearn.cluster import KMeans
# Corpus with example sentences
corpus = ['A man is eating food.',
          'A man is eating a piece of bread.',
          'Horse is eating grass.',
          'A man is eating pasta.',
          'A Woman is eating Biryani.',
          'The girl is carrying a baby.',
          'The baby is carried by the woman',
          'A man is riding a horse.',
          'A man is riding a white horse on an enclosed ground.',
          'A monkey is playing drums.',
          'Someone in a gorilla costume is playing a set of drums.',
          'A cheetah is running behind its prey.',
          'A cheetah chases prey on across a field.',
          'The cheetah is chasing a man who is riding the horse.',
          'man and women with their baby are watching cheetah in zoo'
        ]
```

We import the KMeans clustering algorithm from sklearn module and then we create our own data corpus on which we will be performing clustering.

```
response = openai.Embedding.create(
    input=corpus,
    model="text-similarity-babbage-001"
)
```

```
corpus_embeddings = [ d['embedding'] for d in response['data']]  
# Normalize the embeddings to unit length  
corpus_embeddings = corpus_embeddings / np.linalg.norm(corpus_embeddings, axis=
```

In this step, we create embeddings for our data corpus using openai embedding model — “text-similarity-babbage-001”.

Next, we create a list of `corpus_embeddings` where we store all the embeddings.

Finally, we Normalized the embeddings to unit length. We need to normalize the vectors in order to ensure that all dimensions are treated equally.

```
clustering_model = KMeans(n_clusters=3)  
clustering_model.fit(corpus_embeddings)  
cluster_assignment = clustering_model.labels_  
print(cluster_assignment)  
  
clustered_sentences = {}  
for sentence_id, cluster_id in enumerate(cluster_assignment):  
    if cluster_id not in clustered_sentences:  
        clustered_sentences[cluster_id] = []  
  
    clustered_sentences[cluster_id].append(corpus[sentence_id])  
clustered_sentences
```

In the last step, we build a `KMeans` instance and provide `n_clusters = 3` in order to get three clusters. The corpus of embeddings we got in the previous phase were then used to fit the model. Then, after initialising a dictionary,

we pass the cluster id as the dictionary's key and the clustered sentences as their value.

output:

```
{0: ['A monkey is playing drums.',  
    'Someone in a gorilla costume is playing a set of drums.',  
    'A cheetah is running behind its prey.',  
    'A cheetah chases prey on across a field.',  
    'The cheetah is chasing a man who is riding the horse.',  
    'man and women with their baby are watching cheetah in zoo'],  
1: ['A man is eating food.',  
    'A man is eating a piece of bread.',  
    'Horse is eating grass.',  
    'A man is eating pasta.',  
    'A man is riding a horse.',  
    'A man is riding a white horse on an enclosed ground.'],  
2: ['A Woman is eating Biryani.',  
    'The girl is carrying a baby.',  
    'The baby is carried by the woman']}
```

Conclusion

In this blog, we've learnt about the robust capabilities of OpenAi and how to use them to create cutting-edge applications. We began by studying embeddings and its different applications. By utilising the information from this blog, we will create an app in the following blog.

Reference

<https://platform.openai.com/docs/guides/embeddings>

OpenAI

Embedding

Text Similarity

Clustering

Semantic Search

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 Word2vec

★ · 15 min read



[View list](#)



Written by Tejpal Kumawat

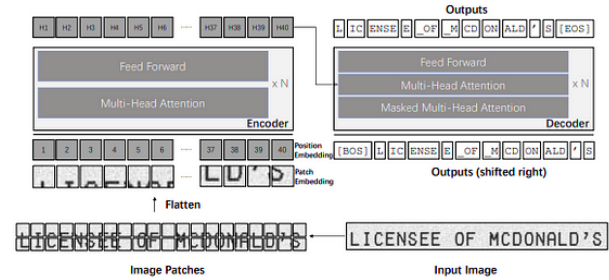
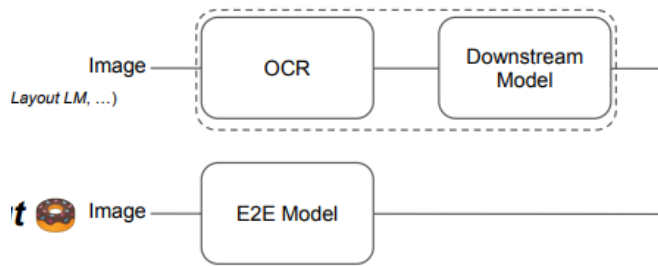
199 Followers

Following



Artificial Intelligence enthusiast that is constantly looking for new challenges and researching cutting-edge technology to improve the world !!!!!!!!!!!!!!!

More from Tejpal Kumawat



Tejpal Kumawat

Donut: OCR-Free Document Understanding with Donut

Introduction

9 min read · Mar 6



46



2



...



Tejpal Kumawat

TrOCR—Transformer-based Optical Recognition Model

Introduction

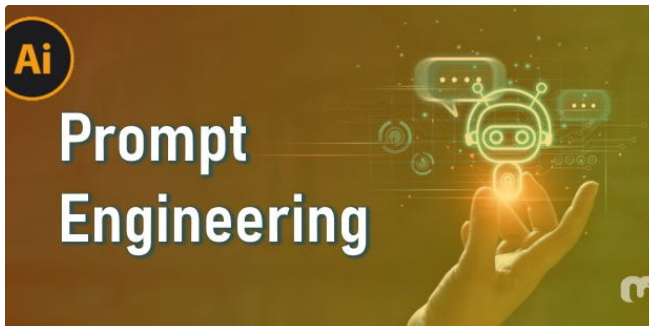
5 min read · Mar 5



36



...



Tejpal Kumawat

Basics of Prompt Engineering

What is LLMs ?

30 min read · Jun 4



12



...



Tejpal Kumawat

Is CNN Extinct? Transformer-Based Vision Models Explained →...

Introduction

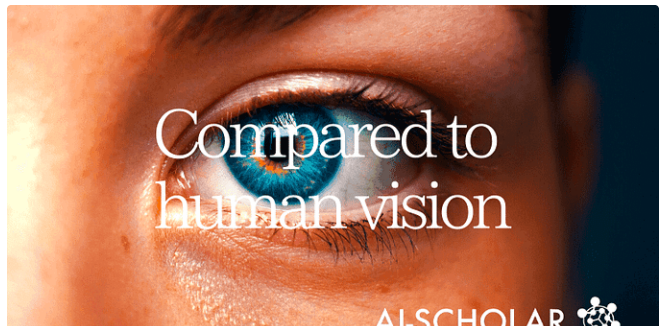
11 min read · Apr 5



36



...



See all from Tejpal Kumawat

Recommended from Medium



Alyx

Semantic Search with FAISS

HuggingFace get_nearest_example and Cosine Similarity Search

9 min read · Jul 15



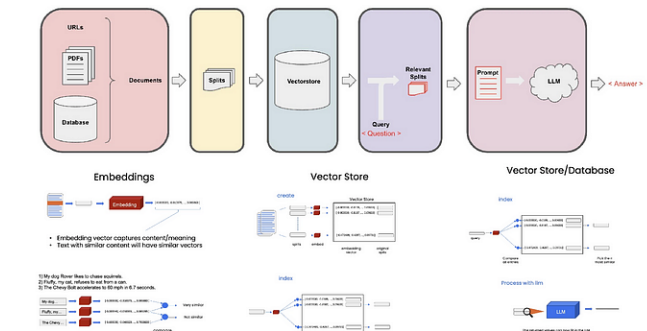
71



1



56



TeeTracker

Chat with your PDF (Streamlit Demo)

Conversation with specific files

4 min read · Sep 15

Lists



AI Regulation

6 stories · 138 saves



Generative AI Recommended Reading

52 stories · 274 saves



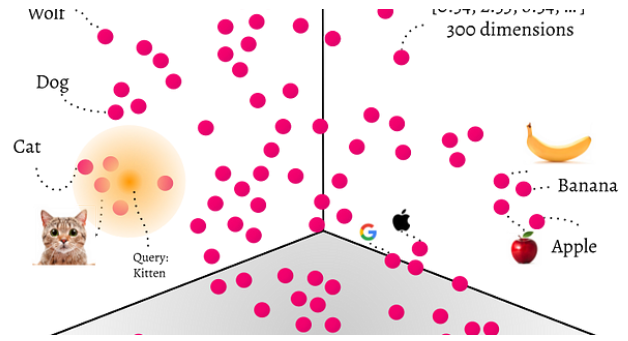
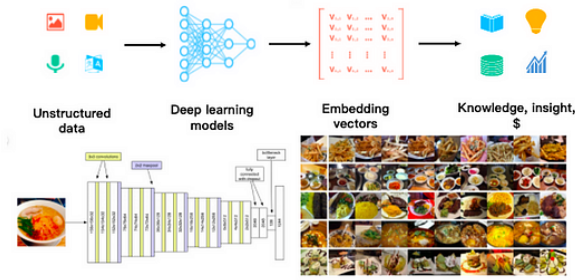
Natural Language Processing


669 stories · 283 saves



Coding & Development

11 stories · 200 saves



 Jayita Bhattacharyya in GoPenAI

Primer on Vector Databases and Retrieval-Augmented Generation...

Vector Databases Generation (RAG)
Langchain Pinecone HuggingFace Large...

9 min read · Aug 16



228



1



9



 CodeGPT

Improve LLMs Responses with Vector Databases

Optimizing language models using vector data provides a powerful approach to...

4 min read · Sep 22

See more recommendations