Member-only story

From Synonyms to GPT-3: The Ultimate Guide to Text Augmentation for Improving Minority Class Labels in NLP

Harshmeet Singh Chandhok · Following

Published in Towards AI · 8 min read · May 10

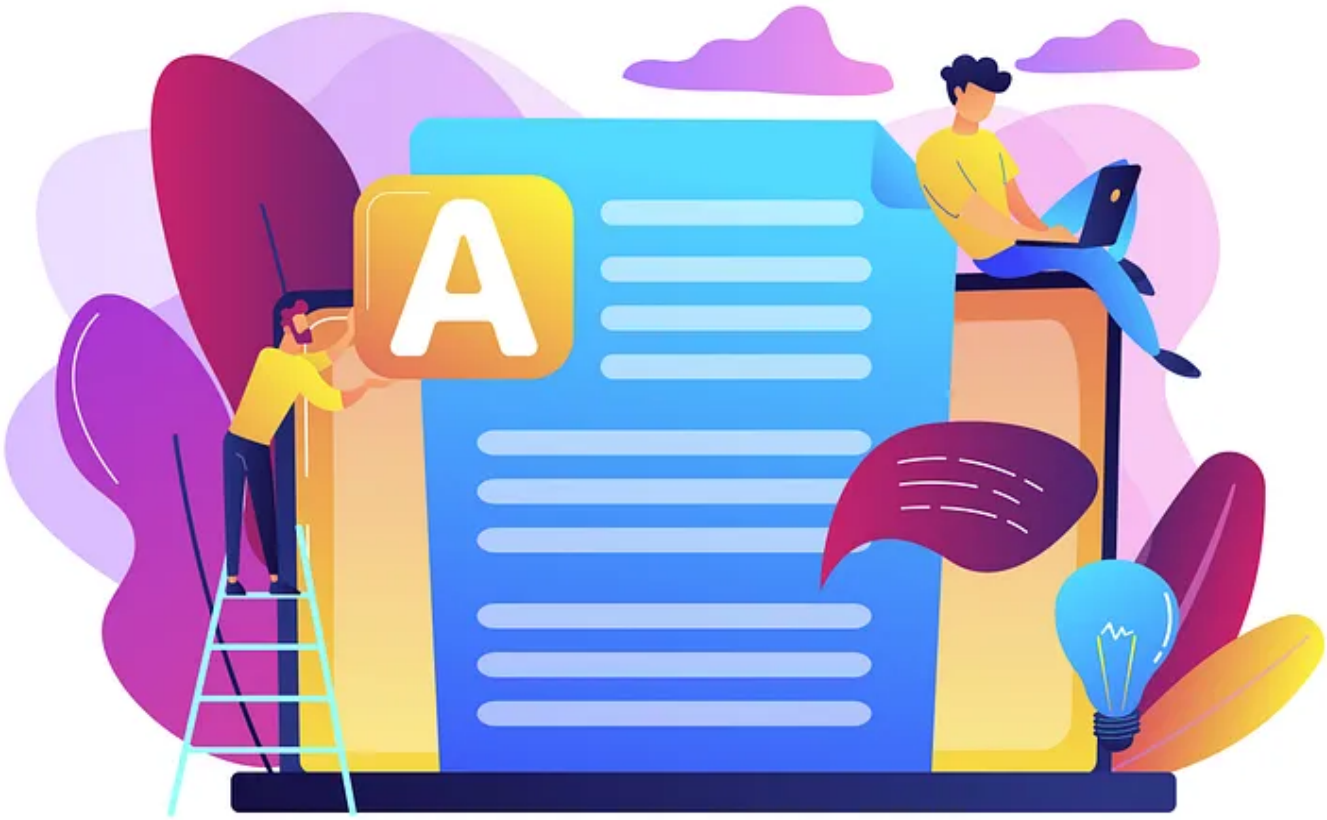


Image by vectorjuice on Freepik

> *"In NLP, we often encounter problems related to class imbalance, where minority classes are underrepresented in the training data. Text augmentation techniques can help address this issue, improving the performance of our models for all classes."*
>
> — *Rachel Thomas, Co-Founder of* ***fast.ai***

Text Augmentation uses transformations to create new data from existing text. Effective learning and accuracy enhancement are essential for NLP models.

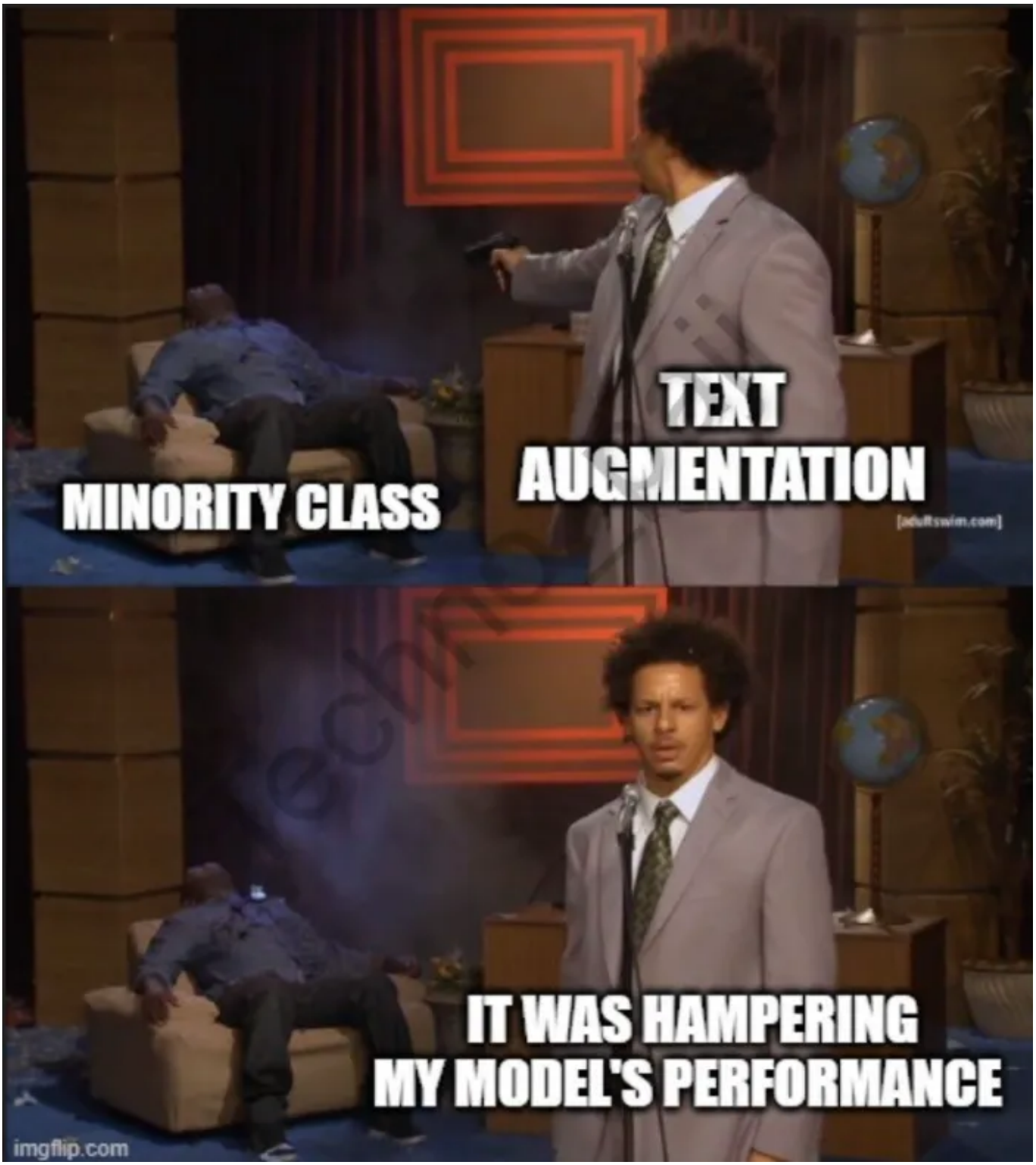
However, it can be challenging to gather diverse data, which can result in biased models that ⚠ **perform poorly for minority classes** ⚠

So how exactly Text Augmentation can solve the problem 🤔❓

By creating fresh instances for uncommon occurrences or underrepresented classes, text augmentation can assist in resolving this issue by boosting the diversity and balance of the training data. Text augmentation makes NLP models more accurate and robust, especially for underrepresented groups. By including variations in the training data, text augmentation also helps minimize overfitting.

As a result, the underlying patterns in the data are represented in more **reliable** and **generalizable** ways.

> *Stats of* how effective this method *is is shown at the end of this blog* 💯



Source: Image by Author

Text Augmentation Techniques

To expand the amount of samples for minority class labels, a number of text augmentation techniques can be applied.

There's a "**BONUS**" method or you can say a "**HACK**" in the last method

which I have tried many times and many people don't know about this method. **(method no 7)**

YOU WILL BE AMAZED BY THAT METHOD FOR SURE 🤯🤯!!

Below, we'll go over a few of the most popular methods.

1. Synonym Substitution

Replacing text-au terms in the text with their synonyms is one of the easiest text enhancement strategies. The Hugging Face library, which offers access to a variety of pre-trained language models that may be used for text augmentation, can be used to accomplish this. Here is an illustration of how to substitute synonyms in a text using the Hugging Face library:

```
from transformers import pipeline

augmentor = pipeline("text2text-generation", model="facebook/bart-large-cnn")
text = "The village of Konoha was attacked by Uchiha Madara."
synonym = augmentor(text, max_length=57, num_return_sequences=1)[0]['generated_t

print(synonym)
```

In this instance, we are creating a synonym for the text using the BART model of Meta. The ***num_return_sequences*** argument defines how many generated texts should be returned, while the ***max_length*** parameter specifies the maximum length of the generated text. By substituting synonyms for existing examples, this technique can be used to generate a huge number of new examples of text data.

2. Random Insertion

Another text-augmentation method involves sprinkling words throughout the text at random. The ChatGPT's Openai library, which is based on the GPT-3 architecture, can be used to accomplish this. Here is an illustration of how to add words at random to a text using ChatGPT:

```
import openai
openai.api_key = "YOUR_API_KEY"

def insert_word(text):
    prompt = f"Insert a word in the following sentence: '{text}'\nNew sentence:"
    response = openai.Completion.create(
        engine="davinci",
        prompt=prompt,
        temperature=0.7,
        max_tokens=50,
        n=1,
        stop=None,
        )
    return response.choices[0].text.strip()

text ="The nine tails fox jumps over the Uchihas"
inserted_text = insert_word(text)
print(inserted_text)
```

In this illustration, we are generating a new sentence by adding a word to the old sentence using the OpenAI API. The ***temperature, max_tokens,*** and ***n*** parameters regulate the randomness and length of the created text, respectively. The ***prompt*** option specifies the prompt that will be used for text generation. We can generate a huge number of new examples of text data that are identical to the original text by randomly adding words into the text.

3. Back Translation:

Text that has been translated into another language and then back into the original language is another method of text augmentation. The Google Translate API can be used for this, giving users access to a variety of language models that have already been trained and can be applied to text translation. Here is an illustration for the back translation purpose:

```
#!pip install --upgrade googletrans==4.0.0-rc1

from googletrans import Translator

def back_translate(text, target_language="fr", source_language="en"):
    translator = Translator(service_urls=['translate.google.com'])
    translation = translator.translate(text, dest=target_language).text
    back_translation = translator.translate(translation, dest=source_language).t
    return back_translation

text = "The quick brown fox swims and goes over the lazy man"
translator = Translator(service_urls=['translate.google.com'])
back_translated_text = back_translate(text, target_language="fr", source_languag
print(back_translated_text)
```

In this example, we will randomly remove words from the text using a straightforward Python technique. The likelihood of each word being eliminated is determined by the *p* parameter. We can generate a huge number of new samples of text data that are identical to the original text but have different word selections and sentence structures by randomly deleting words from the text.

4. Random Deletion

Randomly eliminating words from the text is another method for text enhancement. A straightforward Python program that chooses words to remove from the text at random can be used to do this. Here is an illustration of how to use Python's random deletion feature:

```
import random

def random_deletion(text, p=0.1):
  words = text.split()
  if len(words) == 1:
    return words[0]
  remaining_words = [word for word in words if random.uniform(0, 1) > p]
  if len(remaining_words) == 0:
    return random.choice(words)
  return " ".join(remaining_words)

text = "The quick brown fox jumps over the lazy dog"
deleted_text = random_deletion(text)
print(deleted_text)
```

In this case, we will randomly remove words from the text using a straightforward Python technique. The likelihood of each word being eliminated is determined by the *p* parameter. We can generate a huge number of new samples of text data that are identical to the original text but have different word selections and sentence structures by randomly deleting words from the text.

5. Word Embedding-based Synonym Replacement

The process of word embedding converts each word into a high-dimensional vector in a semantic space. This enables us to locate words in the same region of the semantic space that have similar meanings. We can replace terms in the text with their synonyms in the same area of the semantic space by employing word embedding-based synonym substitution. Utilizing libraries like *spaCy* or *Gensim* will enable this. Here is an illustration of how to use spaCy to achieve word embedding-based synonym replacement:

```
import spacy
```

```
nlp = spacy.load("en_core_web_md")

def replace_synonyms(text):
    doc = nlp(text)
    new_doc = []
    for token in doc:
        if token.has_vector and token.pos_ in ["NOUN", "VERB", "ADJ", "ADV"]:
            synonyms = [t for t in nlp.vocab if t.has_vector and t.similarity(to
            if synonyms:
                new_token = random.choice(synonyms)
                new_doc.append(new_token.text)
            else:
                new_doc.append(token.text)
        else:
            new_doc.append(token.text)
    return " ".join(new_doc)

text = "The quick brown fox jumps over the lazy dog"
synonym_replaced_text = replace_synonyms(text)
print(synonym_replaced_text)
```

In this illustration, we are utilizing the ***spaCy*** library to change textual words with synonyms that are located in the same semantic space. Finding terms with comparable meanings is done using the similarity approach, and choosing a synonym at random is done using the ***random.choice*** method. We can generate a huge number of new samples of text data that are identical to the original text but use different word choices by using word embedding-based synonym substitution.

6. Contextual Augmentation

Adding or deleting context from the original text results in additional examples of text data, which is known as contextual augmentation. This can be accomplished by modifying the original content by adding or deleting sentences or paragraphs. If the source text is a news article, for instance, we can add or remove sentences to produce new versions of the same item. The

nltk package for Python can be used for this. Here is an illustration of how to use nltk for contextual augmentation:

```
import nltk

def add_context(text, n_sentences=1):
    sentences = nltk.sent_tokenize(text)
    if len(sentences) > n_sentences:
        return " ".join(sentences[:n_sentences])
    else:
        return text
def remove_context(text, n_sentences=1):
    sentences = nltk.sent_tokenize(text)
    if len(sentences) > n_sentences:
        return " ".join(sentences[n_sentences:])
    else:
        return text
text = "The quick brown fox jumps over the lazy dog. The dog doesn't seem to car
augmented_text = add_context(text, n_sentences=2)
print(augmented_text)
```

In this case, we are altering the original text's context by utilizing the ***nltk*** package. The first ***n_sentences*** of the text are added to the context using the ***add_context*** function, whereas they are removed using the ***remove_context*** function. We can generate additional samples of text data that are identical to the original text but have distinct circumstances by employing contextual augmentation.

7. Text Generation using Language Models

Deep learning models called language models can produce text that is comparable to the input text. We can build a huge number of new examples of text data that are similar to the original text but with various word selections and sentence structures by utilizing language models to generate

new text data. GPT-2 or GPT-3 libraries can be used for this. The following is an illustration of how to create text using GPT-3:

```
import openai
openai.api_key = "YOUR_API_KEY"

def generate_text(prompt):
    response = openai.Completion.create(
        engine="davinci",
        prompt=prompt,
        temperature=0.5,
        max_tokens=1024,
        n=1,
        stop=None,
        timeout=30,
    )
    return response.choices[0].text.strip()

prompt = "There lived a certain man in Russia long ago"
generated_text = generate_text(prompt)
print(generated_text)
```

In this instance, we create fresh text data that is identical to the supplied text using the OpenAI API. The ***temperature***, ***max_tokens***, and ***n*** parameters regulate the randomness and length of the created text, while the prompt parameter sets the prompt that will be used for text generation. We can build a huge number of new examples of text data that are similar to the original text but with various word selections and sentence structures by utilizing language models to generate new text data.

Additionally, you can add the prompt in **CHATGPT** as well as in the above code as a prompt as -

> generate 5 paraphrased samples for the given sentence.- “your sentence”

So that it will generate **5X more data** for you.

So the question would arise in your mind, which method should you choose ❓

It can be difficult to select the optimum strategy for text augmentation because results depend on the dataset and the particular problem. There are benefits and drawbacks to various strategies. While rapid, word substitution and synonym replacement could not result in meaningful improvements.

Although they demand more resources, generative models like the GPT-3 offer sophisticated variants. Though not always, contextual augmentation can alter meaning. Although it can produce new content, back translation is not always accurate.

The most effective strategy for increasing text variations for minority class designations is frequently combining different approaches. The best approach depends on the dataset, and it could take some trial and error to find it.

Some important stats for text augmentation and its effect on models ✅

- Machine learning model accuracy can be increased by up to 4% using data augmentation approaches, including text augmentation — ***Google.***

- Word substitution and synonym replacement as text augmentation approaches can improve the performance of natural language processing models and boost the F1 score by up to **7%** — ***University of California, Berkeley.***
- Generative models like GPT-3 employed for text augmentation can increase machine learning model accuracy by up to **13%** — ***Carnegie Mellon University.***
- Contextual augmentation techniques like adding or removing sentences can improve the performance of machine learning models and raise the F1 score by up to **8%** — ***Carnegie Mellon University.***
- Back translation for text augmentation can increase machine learning model accuracy by up to **6%** — ***IBM.***

In Conclusion, there is truly a need for text augmentation methodology to deal with minority classes so as to improve the model's performance. There may be other methods also, but combining them with this method would surely benefit a lot.

Happy learning 😁!

If you like the content, **claps** 👏 are appreciated, and **follow me** ✅ for more informative content 💡

Gpt 3 | Data Science | AI | NLP | Data

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 Harshmeet Singh Chandhok

Following

1.2K Followers · Writer for Towards AI

AI Master's Student at @UNSW Australia 📈 Medium Blogger 🖋 Future Skynet whisperer 🤖 Lets Collaborate 💡 https://linktr.ee/techno_paji_

More from Harshmeet Singh Chandhok and Towards AI

Harshmeet Singh Chandhok in Data And Beyond

How to Use Data Science in Marketing ?🚀

"The best marketing doesn't feel like marketing."

✦ · 6 min read · Jan 19

470 2

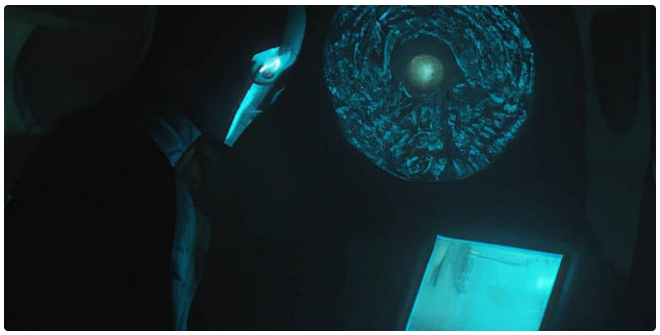
Nour Islam Mokhtari in Towards AI

How to Extract Key Information from Business Documents using...

A quick guide on how to use LayoutLMv3 to streamline business documents,...

✦ · 3 min read · Aug 16

166 1



Pere Martra in Towards AI

Create Your Own Data Analyst Assistant With Langchain Agents

Allow me to share my personal opinion on LLM Agents: They are going to revolutionize...

13 min read · Aug 5

176 2

Harshmeet Singh Chandhok in Data And Beyond

ChatGPT vs BARD: The Battle of the AI Assistants

"Language models will be one of the biggest drivers of progress in AI over the next...

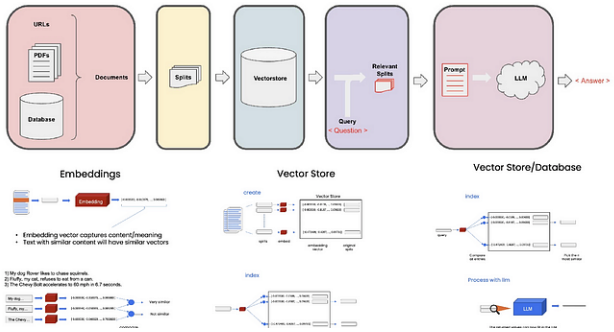
✦ · 5 min read · Feb 23

452 3

See all from Harshmeet Singh Chandhok

See all from Towards AI

Recommended from Medium



TeeTracker

Chat with your PDF (Streamlit Demo)

Conversation with specific files

4 min read · Sep 15

56

Mohit Soni

Training Large Language Model (LLM) on your data

Large Language Models (LLM) have taken the internet by storm in the last few months. The...

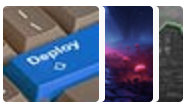
5 min read · Jun 16

103 1

Lists

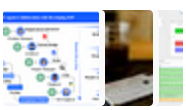
The New Chatbots: ChatGPT, Bard, and Beyond

13 stories · 133 saves



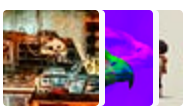
Predictive Modeling w/ Python

20 stories · 452 saves



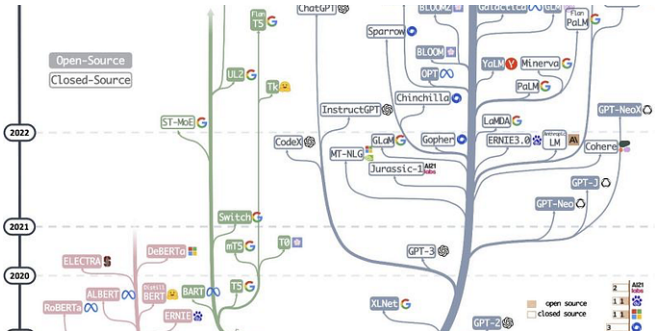
Natural Language Processing

669 stories · 283 saves



What is ChatGPT?

9 stories · 182 saves



Haifeng Li

A Tutorial on LLM

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

15 min read · Sep 14

372

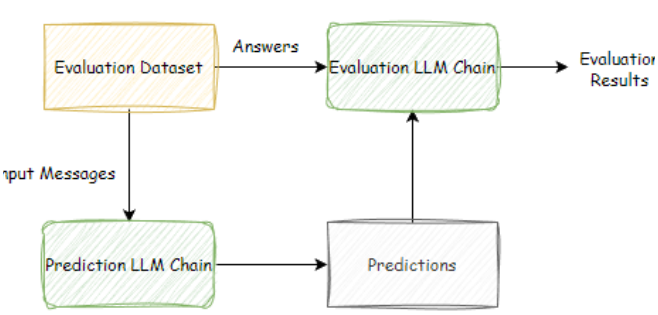
Yvann in Better Programming

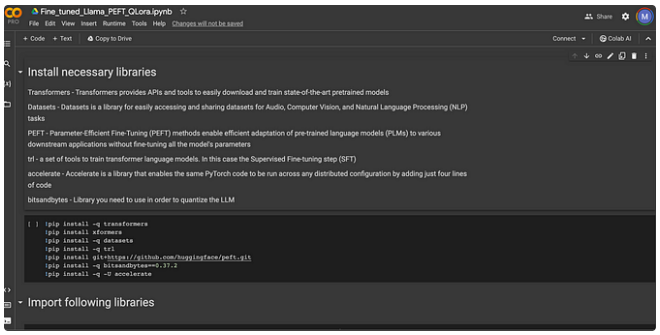
Build a Chatbot on Your CSV Data With LangChain and OpenAI

Chat with your CSV file with a memory chatbot — Made with Langchain ...

5 min read · Jun 2

1.1K 24





Sheldon L.

Evaluation on LLMs

The advent of large language models (LLMs) such as ChatGPT and others, has brought...

· 6 min read · Jun 4

22

Maya Akim

Complete Guide to LLM Fine Tuning for Beginners

Fine-tuning a model refers to the process of adapting a pre-trained, foundational model...

5 min read · Aug 14

112 1