Build A Text Summariser Using LLMs with Hugging Face

Gayathri Jujjuru — Published On July 21, 2023

Beginner Classification Generative AI LLMs NLP Sentiment Analysis Streamlit

Introduction

Text Summariser using LLMs has drawn a lot of interest lately because they are now necessary tools for many different natural language processing (NLP) applications. These models, like GPT-3 and T5, are pre trained models that are capable of producing text that resembles that of a human being as well as text classification, summarization, translation, and other tasks. Hugging Face is one of the well-liked libraries for using LLMs.

This article will examine LLM capabilities with a particular emphasis on Hugging Face and how you can apply to handle challenging NLP issues. We will also go over how to use Hugging Face and LLMs to build a text-summarising application for Streamlit. Let's first look into our Learning objectives for this article.

Learning objectives

- Explore the features and functionalities of Hugging Face as a platform for working with LLMs and Transformers.
- Learn how to leverage pre-trained models and pipelines provided by Hugging Face for various NLP tasks like chatbots.
- Develop a practical understanding of text summarization using Hugging Face and LLMs.
- Create an interactive Streamlit application for text summarisation.

This article was published as a part of the Data Science Blogathon.

Table of contents

Understanding Large Language Models (LLMs)

Train the LLM models on massive amounts of text data. These models predict the next word in a sentence based on the previous context, enabling them to capture complex language patterns and generate coherent text.

LLMs are trained on large amounts of datasets, which contain billions of parameters. The vast amount of training data allows LLMs to learn the intricacies of language and provide impressive language generation capabilities.

LLMs have significantly impacted the field of NLP by enabling breakthroughs in various tasks such as

machine translation, text generation, question-answering, sentiment analysis, and many more.

DataHour: Securing LLM-Based Applications

Date: 2 Nov 2023 **Time:** 6:00 PM – 7:00 PM IST

RSVP!

These models have demonstrated remarkable performance on benchmarks and have become go-to tools for many NLP tasks.

Hugging Face

Hugging Face is a platform and library for working with LLMs and transformers. It provides a comprehensive ecosystem that simplifies the usage of LLMs for NLP tasks.

This library offers a wide range of pre-trained models, datasets, and tools, making it easy to leverage LLMs for various applications.

so we need not to train the models, they have trained for us, Let's delve into some key aspects of Hugging Face and how it enhances the usage of LLMs.

Features

1. Pre-trained Models

One of the best features of Hugging Face, it provides a vast collection of pre-trained LLMs. These models are trained on massive datasets and fine-tuned for specific NLP tasks.

For example, models like GPT-3 and T5 are readily available for tasks like text generation, summarization, and translation.

Hugging Face offers models with different architectures, sizes, and performance trade-offs, allowing users to choose the model that best fits their requirements.

2. Easy Model Loading and Fine-tuning

When we talk about the features of the Hugging Face the far most feature is simplicity, it simplifies the process of loading and fine-tuning pre-trained models.

With just a few lines of code, any user can download and initialize a pre-trained model.

3. Datasets and Tokenizers

Working with NLP often involves handling large datasets and preprocessing text. Hugging Face provides datasets and tokenizers that facilitate data loading, preprocessing, and tokenization tasks.

The datasets module offers access to various datasets, including popular benchmark datasets, making it easy to train and evaluate models.

The tokenizers provided by Hugging Face enable efficient text tokenization, allowing users to convert raw text into suitable input formats for LLMs.

4. Training and Inference Pipelines

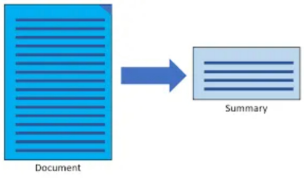
Hugging Face simplifies the usage of LLMs through its training and inference pipelines. These pipelines provide high-level interfaces for common NLP tasks, such as text classification, named entity recognition, sentiment analysis, and summarization.

Users can easily create pipelines and utilize LLMs for specific tasks without delving into low-level implementation details.

For example, the pipeline("summarization") function creates a summarization pipeline that abstracts away the complexities of model loading, tokenization, and inference, allowing users to generate summaries with just a few lines of code.

Summarization with Hugging Face LLMs

Summarization is a common NLP task that involves condensing a piece of text into a concise summary while preserving the main points.



LLMs, when combined with Hugging Face, offer powerful capabilities for both extractive and abstractive summarization.

Extractive summarization involves selecting the most important sentences or phrases from the original text, while abstractive summarization generates new text that captures the essence of the original content.

Hugging Face provides pre-trained models, such as T5, which can be used for both extractive and abstractive summarization tasks.

Example

To demonstrate summarization using Hugging Face, let's walk through an example. First, we need to install the required packages:

```
%pip install sacremoses==0.0.53
%pip install datasets
%pip install transformers
%pip install torch torchvision torchaudio
```

These packages, namely sacremoses, datasets, transformers and torch or tensorflow 2.0 are essential for working with the dataset and model in the subsequent code

Next, we import the necessary modules from the installed packages:

```
from datasets import load_dataset
from transformers import pipeline
```

Here, we import the load_dataset function from the datasets package, which enables us to load the dataset, and the pipeline function from the transformers package, which allows us to create a pipeline for text summarization.

To illustrate the process, let's use the xsum dataset, which comprises a collection of BBC articles and summaries. We load the dataset as follows:

```
#loading the dataset
xsum_dataset = load_dataset(
    "xsum",
    version="1.2.0",
    cache_dir='/Documents/Huggin_Face/data'
)  # Note: We specify cache_dir to use predownloaded data.
xsum_dataset
# The printed representation of this object shows the `num_rows`
# of each dataset split.
```

Here, we use the load_dataset function to load the xsum dataset, specifying the version and cache directory where the downloaded dataset files will be stored. The resulting dataset object is assigned to the variable xsum_dataset.

To work with a smaller subset of the dataset, we can select a few examples. For instance, the code snippet below selects the first 10 examples from the training split and displays them as a Pandas DataFrame:

```
xsum_sample = xsum_dataset["train"].select(range(10))

display(xsum_sample.to_pandas())
```

Create Summarization Pipeline

Now that wehave the dataset ready, we can create a summarization pipeline using Hugging Face and perform summarization on a given text. Here's an example:

```
summarizer = pipeline(
    task="summarization",
    model="t5-small",
    min_length=20,
    max_length=40,
    truncation=True,
    model_kwargs={"cache_dir": '/Documents/Huggin_Face/'},
)  # Note: We specify cache_dir to use predownloaded models.
```

In this code snippet, we create a summarization pipeline using the pipeline function from the transformers package.

The task parameter is set to "summarization", indicating that the pipeline's task is text summarization. We specify the pre-trained model to use as "t5-small".

The min_length and max_length parameters define the desired length range for the generated summaries.

We set truncation=True to truncate the input text if it exceeds the maximum length supported by the model. Finally, we use model_kwargs to specify the cache directory for the pre-downloaded models.

To generate a summary for a given document using the created summarization pipeline, we can use the following code:

```
summarizer(xsum_sample["document"][0])
```

In this code snippet, we apply the summarization pipeline to the first document in the xsum_sample dataset. The pipeline generates a summary for the document based on the specified model and length constraints.

Alternatively, if you want to generate a summary directly from user input

```


# Ask the user for input
input_text = input("Enter the text you want to summarize: ")

# Generate the summary
summary = summarizer(input_text, max_length=150, min_length=30, do_sample=False)[0]['su

bullet_points = summary.split(". ")

for point in bullet_points:

    print(f"- {point}")

# Print the generated summary
print("Summary:", summary)
```

In this modified code, we removed the parts related to loading the dataset and displaying the results using a DataFrame. Instead, we directly ask the user for input using the input() function.

The user's input is then passed to the summarization pipeline, which generates a summary based on the provided text. The generated summary is printed to the console.

Feel free to adjust the parameters (max_length and min_length) according to your desired summary length range.

By leveraging Hugging Face and LLMs like T5, you can easily perform text summarization for a variety of applications, such as news articles, research papers, or any other text that requires concise summaries.

Web Application

Streamlit Application for Text Summarization

In addition to discussing LLMs and Hugging Face, let's explore how we can create a Streamlit application for text summarization. Streamlit is a popular Python library that simplifies the development of interactive web applications. By combining Streamlit with Hugging Face, we can create a user-friendly interface where users can easily input text and obtain a summarization output.

Install Necessary Packages

To get started, we need to install the necessary packages:

```
pip install streamlit
```

Once Streamlit is installed, we can create a Python script, let's call it app.py, and import the required modules:

import streamlit as st
from transformers import pipeline

Next, we create a Streamlit application by defining a function and using Streamlit decorators to specify the app layout:

```
import streamlit as st
from transformers import pipeline, AutoTokenizer, AutoModelForSeq2SeqLM

def main():
    st.title("Text Summarization")

    summarizer = pipeline(
        task="summarization",
        model="t5-small",
        min_length=20,
        max_length=40,
        truncation=True,
        model_kwargs={"cache_dir": '/Documents/Huggin_Face/'},
    )

    # User input
    input_text = st.text_area("Enter the text you want to summarize:", height=200)

    # Summarize button
    if st.button("Summarize"):
        if input_text:
            # Generate the summary
            output = summarizer(input_text, max_length=150, min_length=30, do_sample=Fa
            summary = output[0]['summary_text']

            # Display the summary as bullet points
            st.subheader("Summary:")
            bullet_points = summary.split(". ")
            for point in bullet_points:
                st.write(f"- {point}")
        else:
            st.warning("Please enter text to summarize.")

if __name__ == "__main__":
    main()
```

In this code, we define the main function that represents our Streamlit application. We set the title of the application using st.title.

Create Summarization Pipeline Using HuggingFace

Next, we create a summarization pipeline using Hugging Face's pipeline function. This pipeline will handle the text summarization task.

We use st.text_area to create an input text area where the user can paste or type the content they want to summarize. The height parameter sets the height of the text area to 200 pixels.

Create the "Summarize" button using st.button. Click the button and check if the input text is not empty. If it's not empty, we pass the input text to the summarization pipeline, generate the summary, and display it

using st.subheader and st.write. If the input text is empty, we display a warning message using st.warning.

Finally, we execute the main function when the script is run as the main program.

To run the Streamlit application, open a terminal or command prompt, navigate to the directory where the app.py script is located, and run the following command:

```
streamlit run app.py
```

Streamlit will start a local web server and provide a URL where you can access the text summarization application.

Users can then copy and paste the content they want to summarize into the text area, click the "Summarize" button, and the generated summary will appear.

Here is the code Link – GitHub

Conclusion

In this article, we explored the concept of LLMs and their significance in NLP. We introduced Hugging Face as a leading platform and library for working with LLMs, discussing its key features such as pre-trained models, easy model loading, fine-tuning, datasets, tokenizers, training and inference pipelines. We also demonstrated how to create a Streamlit application for text summarization using LLMs and Hugging Face.

With LLMs and Hugging Face, developers and researchers have powerful tools at their disposal to solve complex NLP problems, enhance language generation, and enable more efficient and effective natural language understanding. The continuous advancements in LLMs and the vibrant Hugging Face community ensure that the future of NLP will fill with exciting possibilities.

Key Takeaways

- Large Language Models (LLMs) are powerful models trained on massive amounts of text data that can generate human-like text and perform various NLP tasks.
- Hugging Face offers a wide range of pre-trained models with different architectures, sizes, and performance trade-offs, allowing users to choose the model that best suits their needs.
- Hugging Face provides easy model loading, fine-tuning, and adaptation to custom tasks, empowering users to leverage LLMs for specific applications.
- Hugging Face offers training and inference pipelines for common NLP tasks, providing high-level interfaces for model utilization without requiring low-level implementation details.

Frequently Asked Questions

Q1. Can I use Hugging Face models for tasks other than summarization?

A. Use Hugging Face models for various NLP tasks, such as text classification, named entity recognition, sentiment analysis, machine translation, and more. Hugging Face provides pipelines and tools tailored for different tasks, making it easy to leverage the capabilities of LLMs.

Q2. Are Hugging Face models only available for English text?

A. No, Hugging Face offers models trained on multilingual data, allowing you to work with different languages. Additionally, the community contributes models for specific languages and domains, expanding the available options.

Q3. Can I fine-tune a pre-trained Hugging Face model on my custom dataset?

A. Yes, Hugging Face provides tools and resources for fine-tuning pre-trained models on custom datasets. You can adapt the models to your specific tasks and data by leveraging transfer learning techniques.

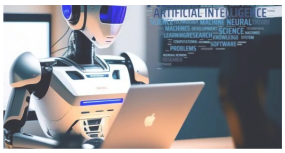
Q4. How can I contribute to the Hugging Face community and Model Hub?

A. The Hugging Face community welcomes contributions. You can share your trained models, submit improvements to existing models, or participate in discussions on the Hugging Face forum or GitHub repository. By sharing your knowledge and expertise, you can contribute to the growth of the NLP community.

Related



Build Your Own Translator with LLMs & Hugging Face



What are Large Language Models (LLMs)?

Text Analysis app using Spacy, Streamlit, and Hugging face Spaces

blogathon dataset datasets hugging face Models NLP streamlit summary

About the Author

Gayathri Jujjuru

Our Top Authors

view more

Download

Analytics Vidhya App for the Latest blog/Article

Previous Post

The Fascinating Evolution of Generative AI

Next Post

Frequentist vs Bayesian Statistics in Data Science

Top Resources

10 Best AI Image Generator Tools to Use in 2023

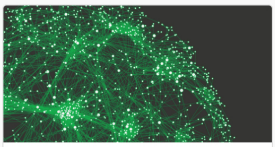
avcontentteam - AUG 17, 2023

Understand Random Forest Algorithms With Examples (Updated 2023)

Sruthi E R - JUN 17, 2021

How to Read and Write With CSV Files in Python?

Harika Bonthu - AUG 21, 2021



The Ultimate Guide to K-Means Clustering: Definition, Methods and Applications

Pulkit Sharma - AUG 19, 2019