Unlocking the Power of GPT-2 for News Headline Analysis

Sunidhi Ashtekar
12 min read · Mar 15, 2024

Image from Article

In this article, we delve into an exciting exploration of the GPT-2 model from Hugging Face transformer library, repurposing it for text classification tasks with News Category Dataset.

Text classification stands as a pivotal task in the realm of natural language processing (NLP), enabling the systematic organisation and categorisation of textual data into predefined classes. NLP's significance is underscored by its ability to bridge human communication and computational analysis, facilitating advancements across a myriad of applications from sentiment analysis to topic identification.

Introduction to Transformers

The advent of transformer models like BERT and GPT has significantly advanced text classification, offering unprecedented efficiency and accuracy. By understanding language nuances through deep learning, these models have transformed text interpretation, capturing contextual relationships to improve performance and broaden NLP capabilities. This shift emphasizes the continuous evolution and impact of machine learning in processing and structuring extensive textual data.

While GPT-2 might not wear the crown of the latest transformer technology, especially with the arrival of its successors GPT-3, GPT-3.5 and GPT-4, it remains a cornerstone in the landscape of natural language processing.

BERT and GPT-2, as transformer models, are well-recognized for their NLP prowess, differentiated by their design: GPT-2's decoder blocks excel in text generation, while BERT's encoder blocks enhance text understanding. However, GPT-2's capacity for text classification, despite being less explored compared to BERT's versatility, offers a promising research opportunity.

GPT-2's architecture uniquely predicts the next token using the final token of a sequence, a feature that can be cleverly repurposed for text classification. This allows the model to use the importance of the last token in classifying text snippets effectively.

Despite the advancements of GPT-3 and GPT-4, their closed-source nature and demand for substantial computational resources limit accessibility and practicality for custom experimentation. Hence, we pivot to GPT-2, which offers a middle ground in linguistic sophistication and operational viability, making it a suitable option for text classification. This choice aims to showcase GPT-2's versatility and untapped potential within natural language processing.

Getting Started

For classifying news headlines with GPT-2, follow these structured steps:

- Get the Dataset
- Environment setup
- Verify and Understand the Dataset
- Data Tokenisation
- Data Pre-processing
- Model Training
- Model Evaluation
- Save the Model Weights
- Model Inference

About the Dataset

The dataset hails from Kaggle's "News Category Dataset," a comprehensive collection that spans a spectrum of 42 categories, each offering a glimpse into different facets of global news stories. It is richly detailed, featuring elements such as headlines, short descriptions, authors, and publication dates for each article.

Due to the dataset's expansive nature, focusing on all 42 categories would dilute the specificity and efficiency of the model training process. Therefore, the project narrows its scope to the five most populated categories: Politics, Sports, Technology, Entertainment, and Business. This decision is driven by the intention to harness a more manageable and representative subset of data, ensuring a robust learning process and more accurate classification outcomes. By concentrating on these categories, the project optimizes the balance between variety and volume, setting a solid foundation for effective news headline classification with GPT-2.

Environment Setup

A Google account is required to access Google Colab, a free cloud service providing necessary computational resources for deep learning tasks, including access to GPUs and TPUs up to 16 GB.

Let's get started, we first ensure availability and readiness of the GPU for processing our transformer model and mount the Google drive to utilize the dataset stored followed by installing the necessary Python packages.

```
! nvidia-smi

drive.mount('/content/drive')

! pip install datasets transformers torchvision

import argparse
import json
import logging
import re

import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import torch
import torch.nn as nn
import torch.optim as optim
from google.colab import drive
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from torch.nn.utils import clip_grad_norm_
```

```
from torch.optim.lr_scheduler import ReduceLROnPlateau, StepLR
from torch.utils.data import DataLoader
from torch.utils.data.dataloader import default_collate
from torchvision import transforms
from tqdm import tqdm

from transformers import (AutoModelForSequenceClassification, DataCollatorWithP
                          GPT2Model, GPT2Tokenizer, Trainer, TrainingArguments)
```

Loading and Displaying the Dataset

Once the environment is ready we read the News_Category_Dataset_v3, a JSON file from drive and store it in a pandas DataFrame. Then we will display few entries from the dataset. Here's how the it looks like, there are 6 columns from which we will choose 'Headlines' as our text input and 'Category' as label.

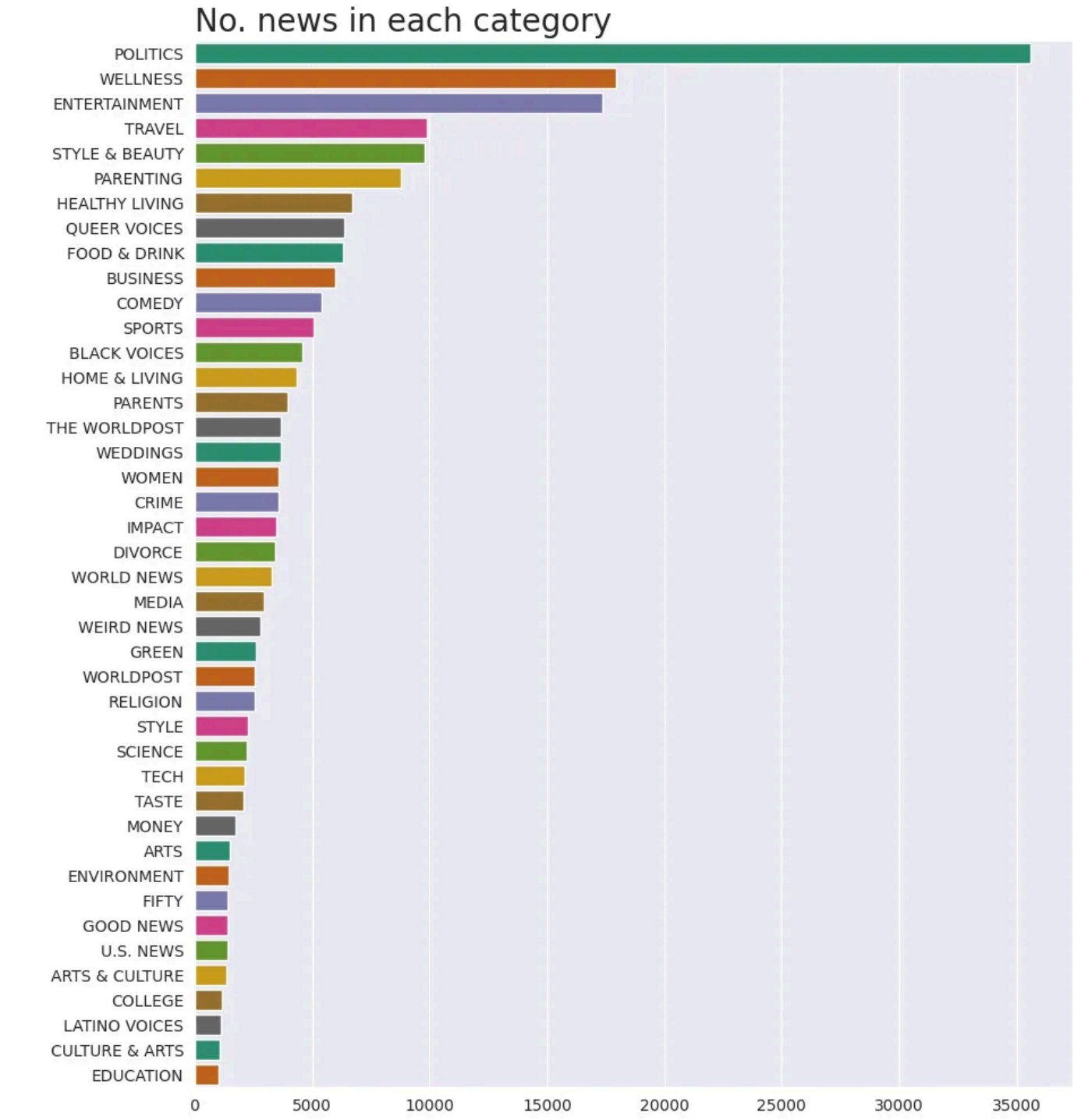
```
df = pd.read_json('/content/drive/My Drive/News_Category_Dataset_v3.json', line
print('No. of rows and columns :', df.shape)
df.head()
```

	link	headline	category	short_description	authors	date
0	https://www.huffpost.com/entry/covid-boosters-...	Over 4 Million Americans Roll Up Sleeves For O...	U.S. NEWS	Health experts said it is too early to predict...	Carla K. Johnson, AP	2022-09-23
1	https://www.huffpost.com/entry/american-airlin...	American Airlines Flyer Charged, Banned For Li...	U.S. NEWS	He was subdued by passengers and crew when he ...	Mary Papenfuss	2022-09-23
2	https://www.huffpost.com/entry/funniest-tweets...	23 Of The Funniest Tweets About Cats And Dogs ...	COMEDY	"Until you have a dog you don't understand wha...	Elyse Wanshel	2022-09-23
3	https://www.huffpost.com/entry/funniest-parent...	The Funniest Tweets From Parents This Week (Se...	PARENTING	"Accidentally put grown-up toothpaste on my to...	Caroline Bologna	2022-09-23
4	https://www.huffpost.com/entry/amy-cooper-lose...	Woman Who Called Cops On Black Bird-Watcher Lo...	U.S. NEWS	Amy Cooper accused investment firm Franklin Te...	Nina Golgowski	2022-09-22

Visualizing the Distribution of News Categories

To verify the dataset we analyze the distribution of news articles across different categories. This gives us a clear idea of how many news articles are available per category.

```
count_df = pd.DataFrame(df['category'].value_counts()).reset_index()
print('There are', len(count_df), 'news categories')

sns.set_style('darkgrid')
plt.figure(figsize=(10, 12))
sns.barplot(data=count_df, y='index', x='category', palette='Dark2')
plt.title('No. news in each category', loc='left', fontsize=20)
plt.xlabel("")
plt.ylabel("")
plt.show()
```



Initialization of the Tokenizer for Text Processing

Once the data is verified, we initialize the GPT2Tokenizer. The tokenizer is configured to pad sequences from the left side to maintain consistency with GPT-2's expected input format. Additionally, the end-of-string token (eos_token) is assigned as the padding token, ensuring that padded inputs are treated appropriately during model training and inference. This step is crucial for preparing the text data for subsequent processing stages, such as encoding the headlines into a format that can

be processed by the neural network. Tokenisation is basically dividing large sentences in small words so that model can understand.

The input_ids is the text input and attention mask is for handling the padding. These two are text inputs given to the model.

```
# Initialize the tokenizer
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
tokenizer.padding_side = "left"
tokenizer.pad_token = tokenizer.eos_token  # Set pad token

example_text = "I will watch Movie tonight"
gpt2_input = tokenizer(example_text, padding="max_length", max_length=10, trunc

print(gpt2_input['input_ids'])

print(gpt2_input["attention_mask"])
```

Defining a Stopwords List for Text Cleaning

Next action item in the pipeline is the data preprocessing, for that we define a list of common English stopwords sourced from a reputable SEO tool's GitHub repository. Stopwords are words which are commonly removed from text data before processing in natural language tasks as they often carry little informational weight. This list includes pronouns, conjunctions, prepositions, and other words that are frequently used in the English language, but are typically not useful for understanding the context or sentiment of a text. By defining this list, we can preprocess and clean our news headline data to enhance the performance of the classification model by focusing on the most meaningful words.

```
# Stopwords list from https://github.com/Yoast/YoastSEO.js/blob/develop/src/cor
stopwords = [ "a", "about", "above", "after", "again", "against", "all", "am",
```

Balancing the Dataset Across Categories

Moving ahead, we address potential class imbalance by creating a more balanced dataset. For each unique category in our dataset, we subset the DataFrame to include only the data from that category. From each subset, we randomly sample a fixed number of entries (1004 in this case, which is chosen to match the size of the smallest class or for another specific reason). All these balanced subsets are then stored in a list.

```
dfs = []

for category in df['category'].unique():
    temp = df[df['category']==category]
    dfs.append(temp.sample(1004))

df = pd.concat(dfs)
df = df.sample(frac=1).reset_index(drop=True)
df.head()
```

Cleaning and Stopwords Removal

Here are two functions essential for preprocessing the text data, making it more suitable for model training and analysis:

alpha_num: This function removes all characters from the input text that are not alphanumeric (letters and numbers) or spaces. This cleaning step is crucial for standardizing the text data and focusing the model's attention on meaningful words and numbers.

remove_stopwords: Removing stopwords helps in reducing the noise within the data, allowing models to focus on the words that carry significant meaning or sentiment.

Together, these functions streamline the dataset by removing irrelevant characters and words, thereby improving the efficiency and performance of subsequent natural language processing tasks.

```
def alpha_num(text):
    return re.sub(r'[^A-Za-z0-9 ]', '', text)

def remove_stopwords(text):
    final_text = []
    for i in text.split():
        if i.strip().lower() not in stopwords:
            final_text.append(i.strip())
    return " ".join(final_text)

df['headline'] = df['headline'].str.lower()
df['headline'] = df['headline'].apply(alpha_num)
df['headline'] = df['headline'].apply(remove_stopwords)

df.head()
```

Narrowing Down to Selected News Categories

We refine our dataset using the `filter_categories` function to include only headlines from five categories: ‘POLITICS’, ‘SPORTS’, ‘TECH’, ‘BUSINESS’, ‘ENTERTAINMENT’. This targeted approach sharpens the model's focus and potentially boosts its performance on these specific topics, streamlining the data for subsequent processing and training. You can select any categories of your choice from 42!

```
def filter_categories(df, categories_to_keep):
    """
    Filters the DataFrame to keep only the rows with the specified categories.

    Parameters:
    - df (pandas.DataFrame): The original DataFrame.
    - categories_to_keep (list of str): The list of categories to retain.

    Returns:
    - pandas.DataFrame: A DataFrame with only the rows of the specified categor
    """
    return df[df['category'].isin(categories_to_keep)]

# Define the categories to keep
categories_to_keep = ['POLITICS', 'SPORTS', 'TECH', 'BUSINESS', 'ENTERTAINMENT'
```

```
df = filter_categories(df, categories_to_keep)
```

Ensuring Data Quality by Removing Empty Headlines

In this crucial step, we meticulously cleanse our dataset, discarding any entries where the ‘headline’ field is missing or filled with mere whitespace. This purification step ensures our model trains on meaningful, high-quality data, laying the groundwork for effective machine learning outcomes.

```
df = df[df['headline'].notnull() & (df['headline'].str.strip() != '')]

# unique news category
df['category'].unique()
```

Creating a Custom Dataset for News Headline Classification

Now, I will create a custom Dataset class for PyTorch, designed to refine our news headlines dataset for efficient training and evaluation. It includes label encoding to map categories to integers, initialization to standardize headlines and organize them into lists of encoded labels and tokenized texts, and utility methods to facilitate data retrieval and compatibility with PyTorch’s DataLoader. This class is pivotal for streamlined data management, optimizing our dataset for the GPT-2 classification task.

```
labels = {category: i for i, category in enumerate([
    'POLITICS', 'SPORTS', 'TECH', 'ENTERTAINMENT', 'BUSINESS'])}

class Dataset(torch.utils.data.Dataset):
    def __init__(self, df):
        self.labels = [labels[label] for label in df['category']]
        self.texts = [tokenizer(text,
                                padding='max_length',
                                max_length=128,
```

```
                                    truncation=True,
                                    return_tensors="pt") for text in df['headline']]
        for i, enc in enumerate(self.texts):
            if enc['input_ids'].size(1) == 0:
                print(f"Empty input at index: {i}, text: {df.iloc[i]['headline'


    def classes(self):
        return self.labels

    def __len__(self):
        return len(self.labels)

    def get_batch_labels(self, idx):
        return np.array(self.labels[idx])

    def get_batch_texts(self, idx):
        return self.texts[idx]

    def __getitem__(self, idx):
        batch_texts = self.get_batch_texts(idx)
        batch_y = self.get_batch_labels(idx)
        return batch_texts, batch_y
```

Splitting the Dataset into Training, Validation, and Test Sets

The last step in data preprocessing before we dive into our transformer model!

```
np.random.seed(112)
df_train, df_val, df_test = np.split(df.sample(frac=1, random_state=35),
                                     [int(0.8*len(df)), int(0.9*len(df))])

print(len(df_train), len(df_val), len(df_test))
```

Defining GPT-2 Based Sequence Classifier

By integrating the pre-trained GPT-2 model for feature extraction and adding a linear classification layer on top of it, this architecture excels in categorizing text, such as news headlines, into specific groups.

```
class GPT2SequenceClassifier(nn.Module):
    def __init__(self, hidden_size: int, num_classes:int ,max_seq_len:int, gpt_
        super(GPT2SequenceClassifier,self).__init__()
        self.gpt2model = GPT2Model.from_pretrained(gpt_model_name)
        self.fc1 = nn.Linear(hidden_size*max_seq_len, num_classes)


    def forward(self, input_id, mask):
        gpt_out, _ = self.gpt2model(input_ids=input_id, attention_mask=mask, re
        batch_size = gpt_out.shape[0]
        linear_output = self.fc1(gpt_out.view(batch_size,-1))
        return linear_output
```

Training and Evaluating the GPT-2 Classifier

In our GPT-2 classifier's training loop, we prepare data using DataLoaders, initialize the model with CrossEntropyLoss and Adam optimizer, and leverage GPU acceleration. Each epoch involves forward passes, loss minimization, and parameter updates. The model's efficacy is highlighted by an 85.2% training accuracy and 97.1% on validation showcasing strong learning and generalization capabilities.

```
def train(model, train_data, val_data, learning_rate, epochs):
    train, val = Dataset(train_data), Dataset(val_data)

    train_dataloader = torch.utils.data.DataLoader(train, batch_size=8, shuffle
    val_dataloader = torch.utils.data.DataLoader(val, batch_size=8)

    use_cuda = torch.cuda.is_available()
    device = torch.device("cuda" if use_cuda else "cpu")

    criterion = nn.CrossEntropyLoss()
    optimizer = optim.Adam(model.parameters(), lr=learning_rate)

    if use_cuda:
        model = model.cuda()
        criterion = criterion.cuda()

    for epoch_num in range(epochs):
        total_acc_train = 0
```

```
        total_loss_train = 0

        for train_input, train_label in tqdm(train_dataloader):
            train_label = train_label.to(device)
            mask = train_input['attention_mask'].to(device)
            input_id = train_input["input_ids"].squeeze(1).to(device)

            model.zero_grad()

            output = model(input_id, mask)

            batch_loss = criterion(output, train_label)
            total_loss_train += batch_loss.item()

            acc = (output.argmax(dim=1)==train_label).sum().item()
            total_acc_train += acc

            batch_loss.backward()
            optimizer.step()

        total_acc_val = 0
        total_loss_val = 0

        with torch.no_grad():

            for val_input, val_label in val_dataloader:
                val_label = val_label.to(device)
                mask = val_input['attention_mask'].to(device)
                input_id = val_input['input_ids'].squeeze(1).to(device)

                output = model(input_id, mask)

                batch_loss = criterion(output, val_label)
                total_loss_val += batch_loss.item()

                acc = (output.argmax(dim=1)==val_label).sum().item()
                total_acc_val += acc

            print(
            f"Epochs: {epoch_num + 1} | Train Loss: {total_loss_train/len(trair
            | Train Accuracy: {total_acc_train / len(train_data): .3f} \
            | Val Loss: {total_loss_val / len(val_data): .3f} \
            | Val Accuracy: {total_acc_val / len(val_data): .3f}")

EPOCHS = 1
model = GPT2SequenceClassifier(hidden_size=768, num_classes=5, max_seq_len=128,
LR = 1e-5

train(model, df_train, df_val, LR, EPOCHS)
```

Evaluating the GPT-2 Classifier on Test Data

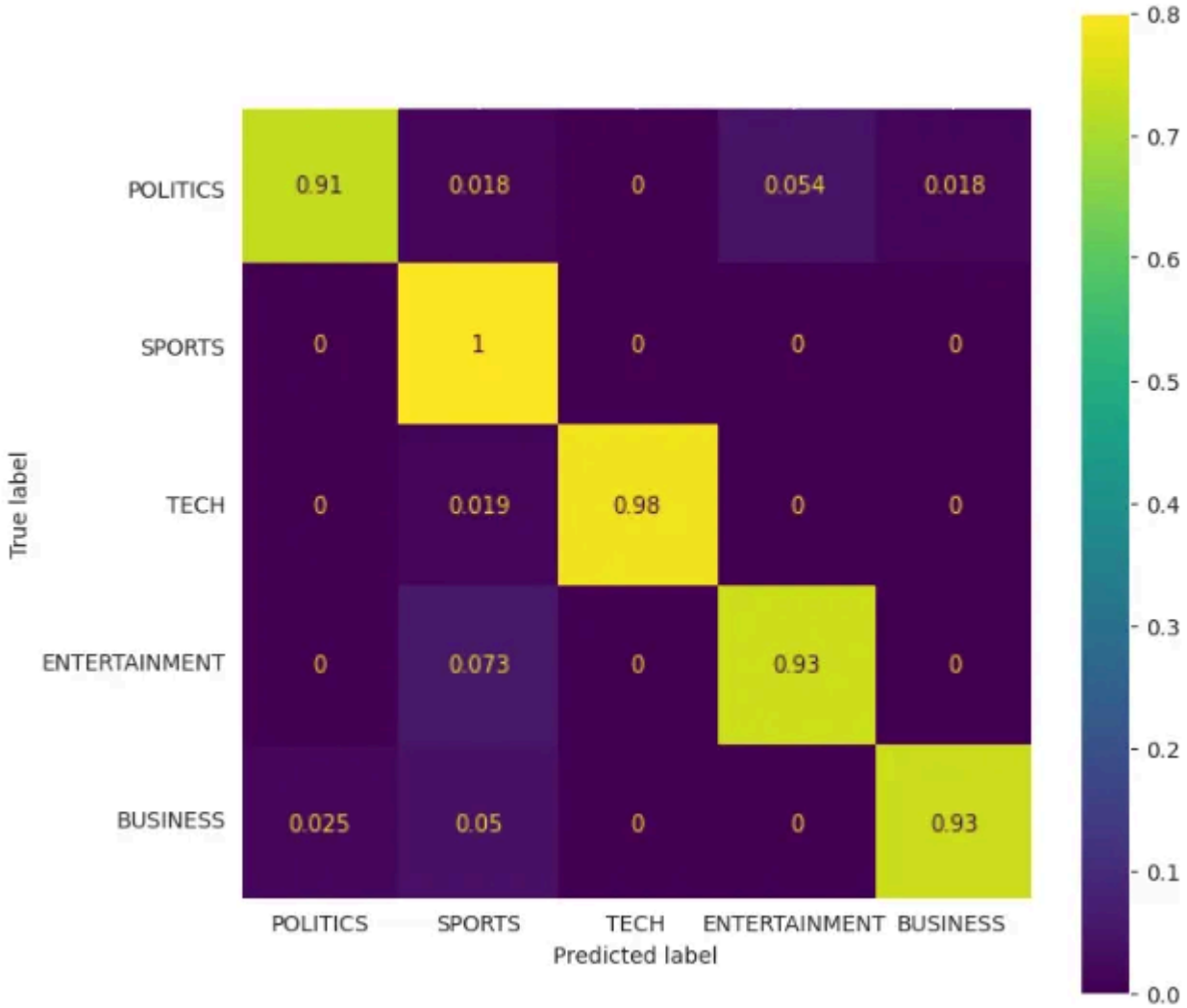
This step involves evaluating our trained GPT-2 sequence classifier on the test dataset, aiming to measure the model's generalization ability on unseen data. This evaluation reflects the model's effectiveness, evidenced by a test accuracy of 94%, indicating robust performance in classifying new examples.

```
def evaluate(model, test_data):

    test = Dataset(test_data)

    test_dataloader = torch.utils.data.DataLoader(test, batch_size=2)

    use_cuda = torch.cuda.is_available()
    device = torch.device("cuda" if use_cuda else "cpu")

    if use_cuda:

        model = model.cuda()

    predictions_labels = []
    true_labels = []

    total_acc_test = 0
    with torch.no_grad():

        for test_input, test_label in test_dataloader:

            test_label = test_label.to(device)
            mask = test_input['attention_mask'].to(device)
            input_id = test_input['input_ids'].squeeze(1).to(device)

            output = model(input_id, mask)

            acc = (output.argmax(dim=1) == test_label).sum().item()
            total_acc_test += acc

            true_labels += test_label.cpu().numpy().flatten().tolist()
            predictions_labels += output.argmax(dim=1).cpu().numpy().flatten().

    print(f'Test Accuracy: {total_acc_test / len(test_data): .3f}')
    return true_labels, predictions_labels

true_labels, pred_labels = evaluate(model, df_test)
```

Visualizing Model Performance with a Confusion Matrix

Then comes the confusion matrix, that we will use to succinctly visualize our GPT-2 classifier's performance, revealing its accuracy across categories and identifying specific areas of strength and misclassification.

```
fig, ax = plt.subplots(figsize=(8, 8))
cm = confusion_matrix(y_true=true_labels, y_pred=pred_labels, labels=range(len(
disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=list(labels.k
disp.plot(ax=ax)
```



Saving the Trained Model

This cell handles saving the trained GPT-2 classifier's parameters to a file and loading them as needed, utilizing `torch.save` to store the model's state dictionary. This process ensures the model can be reused without retraining, streamlining deployment and further experimentation. Saving and loading the model is a crucial step in preserving its training outcomes for future use.

```
torch.save(model.state_dict(), "/content/drive/My Drive/gpt2-text-classifier-mc

model_new = GPT2SequenceClassifier(hidden_size=768, num_classes=5, max_seq_len=
model_new.load_state_dict(torch.load("/content/drive/My Drive/gpt2-text-classif
model_new.eval()
```

```
GPT2SequenceClassifier(
  (gpt2model): GPT2Model(
    (wte): Embedding(50257, 768)
    (wpe): Embedding(1024, 768)
    (drop): Dropout(p=0.1, inplace=False)
    (h): ModuleList(
      (0-11): 12 x GPT2Block(
        (ln_1): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (attn): GPT2Attention(
          (c_attn): Conv1D()
          (c_proj): Conv1D()
          (attn_dropout): Dropout(p=0.1, inplace=False)
          (resid_dropout): Dropout(p=0.1, inplace=False)
        )
        (ln_2): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
        (mlp): GPT2MLP(
          (c_fc): Conv1D()
          (c_proj): Conv1D()
          (act): NewGELUActivation()
          (dropout): Dropout(p=0.1, inplace=False)
        )
      )
    )
    (ln_f): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
  )
  (fc1): Linear(in_features=98304, out_features=5, bias=True)
)
```

Model Inference

For inference using our trained GPT-2 sequence classifier we will give example text from test dataset so it matches the format expected by the model, allowing us to predict its category. The process culminates in accurately classifying the text as 'POLITICS', demonstrating our model's adeptness at discerning and categorizing textual content.

```
example_text = """
  journalisms challenge unending state emergency
"""
fixed_text = " ".join(example_text.lower().split())
print(fixed_text)

tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
tokenizer.padding_side = "left"
tokenizer.pad_token = tokenizer.eos_token

model_input = tokenizer(fixed_text, padding='max_length', max_length=128, trunc

mask = model_input['attention_mask'].cpu()
input_id = model_input["input_ids"].squeeze(1).cpu()

output = model_new(input_id, mask)

prob = torch.nn.functional.softmax(output, dim=1)[0]
print(prob)

labels_to_ids = {category: i for i, category in enumerate([
    'POLITICS', 'SPORTS', 'TECH', 'ENTERTAINMENT', 'BUSINESS'
])}

# Create a dictionary to map numeric labels to category names
ids_to_labels = {i: category for category, i in labels_to_ids.items()}

pred_label = ids_to_labels[output.argmax(dim=1).item()]
print(pred_label)

"""outputs=POLITICS"""
```

Conclusion

I hope this tutorial has been informative and that you've found success in navigating the complexities of text classification with GPT-2. Through the utilization of the Kaggle "News Category Dataset," we've demonstrated the adaptability of GPT-2 beyond its typical text generation applications, showcasing its potential in classifying news headlines into distinct categories. This endeavour highlights the

versatility of transformer models in addressing diverse NLP challenges, offering insights into their capacity for nuanced language understanding and categorization.

If you have any queries, suggestions, or feedback, I warmly invite you to connect with me on LinkedIn. I'm always open to discussions and keen to engage with your experiences, offer assistance, or explore any challenges you might face.

References

For those looking to delve deeper into the methodologies and tools employed in this project, the following resources are invaluable:

- News Headline Classification Git Repository
- GPT-2 Official Repository
- Hugging Face Transformers Library
- Kaggle News Category Dataset: For accessing a diverse dataset that forms the backbone of this classification task. Visit Kaggle to explore this and other datasets.

This project serves as a practical illustration of leveraging advanced NLP techniques for real-world applications. By employing GPT-2 for news headline classification, we not only expand the scope of tasks that transformer models can perform but also open up new avenues for research and development in natural language processing. Whether for scholarly pursuits, industry use, or personal projects, this approach offers a solid foundation for further exploration and innovation in the field of NLP.

Genarative Ai | Llm | Deep Learning | Computer Vision

Edit profile