jcpeetnfv

December 16, 2024

1 1

[1]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

1.1 Dataset Description: DailyDialog

Source: DailyDialog Dataset on Kaggle

The **DailyDialog** dataset consists of daily conversations, making it suitable for training models on dialogue-based tasks. The dataset is specifically designed for emotion detection in dialogues, containing conversations across various emotional contexts.

1.2 Key Information:

- **Purpose**: The dataset is used for training models to recognize emotions in text-based dialogues.
- **Content**:
  - The dataset contains conversations where each dialogue is associated with an emotion label.
  - The emotions in the dataset include common categories such as **neutral**, **joy**, and others (e.g., sadness, anger).
- **File Structure**:
  - **daily_dialog_train_cleaned.csv**: The training set with dialogues and corresponding emotion labels.
  - **daily_dialog_val_cleaned.csv**: The validation set to evaluate the model during training.
  - **daily_dialog_test_cleaned.csv**: The test set for final model evaluation.
- **Columns**:
  - **Text**: The dialogue text in the conversation.
  - **Emotion**: The label representing the emotion in the dialogue (e.g., neutral, joy).
- **Emotion Distribution**:

  - **Neutral**: 83% of the dataset
  - **Joy**: 13% of the dataset
  - **Other Emotions**: 4% of the dataset

- **Size**:

  - The dataset contains **9,624** dialogues in total.

1.3 Example Entries:

- **Neutral**:
  - *"I really think you are stubborn about some things, but here let us look at the new balance shoes."*
  - *"Where else have not I been to yet?"*
- **Joy**:
  - *"Um well actually we had a fantastic time last night, he was amazing."*
  - *"No, I am free. I will be there. What time is the thing starting?"*

1.4 Use Case:

This dataset is ideal for training dialogue systems or models for tasks such as: - **Emotion detection in conversations**: Predict the emotional state of the speaker from the dialogue text. - **Sentiment analysis**: Classify text based on emotional tone.

It can be used for various natural language processing (NLP) applications, including chatbots, virtual assistants, and sentiment analysis tools.

[2]:
```
from transformers import BartForConditionalGeneration, BartTokenizer, Trainer,␣
  ↪TrainingArguments
from datasets import Dataset, DatasetDict
from sklearn.preprocessing import LabelEncoder
import pandas as pd
```

[3]:
```
# Load dataset
train_df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data/
  ↪daily_dialog_train_cleaned.csv')
val_df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data/
  ↪daily_dialog_val_cleaned.csv')
test_df = pd.read_csv('/content/drive/MyDrive/Colab Notebooks/data/
  ↪daily_dialog_test_cleaned.csv')

# Replace missing values in the 'Text' column with an empty string or␣
  ↪placeholder
train_df["Text"] = train_df["Text"].fillna("")
val_df["Text"] = val_df["Text"].fillna("")
test_df["Text"] = test_df["Text"].fillna("")

# Take a subset of the training and validation datasets
```

```
train_df = train_df.sample(n=10000, random_state=42)  # 10,000 samples for␣
    ↪training
val_df = val_df.sample(n=2000, random_state=42) # 2,000 samples for validation
```

[4]:
```
# Step 3: Label Encoding (Emotion labels)
label_encoder = LabelEncoder()

# Fit label encoder on the training data's Emotion column and transform it
train_df['label'] = label_encoder.fit_transform(train_df['Emotion'])
val_df['label'] = label_encoder.transform(val_df['Emotion'])
test_df['label'] = label_encoder.transform(test_df['Emotion'])
```

2 2

[5]:
```
# Convert to Hugging Face Dataset format
train_dataset = Dataset.from_pandas(train_df[['Text']])
val_dataset = Dataset.from_pandas(val_df[['Text']])
test_dataset = Dataset.from_pandas(test_df[['Text']])
```

[6]:
```
from transformers import AutoTokenizer, AutoModelForCausalLM, Trainer,␣
    ↪TrainingArguments

# Load GPT-2 tokenizer and model
tokenizer = AutoTokenizer.from_pretrained("gpt2")
model = AutoModelForCausalLM.from_pretrained("gpt2")

# Fix the padding issue by using the eos_token as pad_token
tokenizer.pad_token = tokenizer.eos_token
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_auth.py:94:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(
```

[7]:
```
def tokenize_function(examples):
    # Tokenize the text and create input_ids and labels
    tokenized = tokenizer(
        examples["Text"],  # Column name from the dataset
        truncation=True,
        padding="max_length",
        max_length=128
    )
    # Labels are identical to input_ids for causal language modeling
    tokenized["labels"] = tokenized["input_ids"].copy()
    return tokenized

# Tokenize train and validation datasets
train_dataset = train_dataset.map(tokenize_function, batched=True)
val_dataset = val_dataset.map(tokenize_function, batched=True)
test_dataset = test_dataset.map(tokenize_function, batched=True)
```

```
Map:   0%|          | 0/10000 [00:00<?, ? examples/s]

Map:   0%|          | 0/2000 [00:00<?, ? examples/s]

Map:   0%|          | 0/10298 [00:00<?, ? examples/s]
```

[8]:
```
# Set the format for PyTorch (required for Trainer)
train_dataset.set_format(type="torch", columns=["input_ids", "attention_mask",␣
 ↪"labels"])
val_dataset.set_format(type="torch", columns=["input_ids", "attention_mask",␣
 ↪"labels"])
test_dataset.set_format(type="torch", columns=["input_ids", "attention_mask",␣
 ↪"labels"])
```

[9]:
```
import wandb
from transformers import Trainer, TrainingArguments

# Initialize wandb
wandb.init(
    project="AML_HW_4_2",  # Name your project here
    config={
        "learning_rate": 2e-5,
        "epochs": 3,
        "batch_size": 16,
    }
)
```

```
wandb: Using wandb-core as the SDK backend.  Please refer to
https://wandb.me/wandb-core for more information.
wandb: Currently logged in as: talasilavenkatesh2
(talasilavenkatesh2-indiana-university). Use `wandb login

--relogin` to force relogin

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>
```

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

```
[9]: <wandb.sdk.wandb_run.Run at 0x793bc5562e90>
```

```
[10]: from transformers import Trainer, TrainingArguments

# Step 5: Define the training arguments
training_args = TrainingArguments(
    output_dir="./results",  # Directory to save the model
    evaluation_strategy="epoch",  # Evaluate the model after every epoch
    save_strategy="epoch",  # Save the model after every epoch
    learning_rate=2e-5,  # Learning rate
    per_device_train_batch_size=16,  # Batch size for training
    per_device_eval_batch_size=16,  # Batch size for evaluation
    num_train_epochs=3,  # Number of training epochs
    weight_decay=0.01,  # Weight decay to avoid overfitting
    logging_dir="./logs",  # Directory to save logs
    logging_steps=10,  # Log every 10 steps
    push_to_hub=False  # Set to True if pushing to Hugging Face Model Hub
)

# Step 6: Set up the Trainer
trainer = Trainer(
    model=model,  # The GPT-2 model
    args=training_args,  # Training arguments
    train_dataset=train_dataset,  # Training dataset
    eval_dataset=val_dataset,  # Validation dataset
    tokenizer=tokenizer  # Tokenizer for data processing
)

# Step 7: Train the model
trainer.train()
```

```
/usr/local/lib/python3.10/dist-packages/transformers/training_args.py:1568:
FutureWarning: `evaluation_strategy` is deprecated and will be removed in
version 4.46 of  Transformers. Use `eval_strategy` instead
  warnings.warn(
<ipython-input-10-339f33d74f81>:19: FutureWarning: `tokenizer` is deprecated and
will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class`
instead.
  trainer = Trainer(
wandb: WARNING The `run_name` is currently set to the same
value as `TrainingArguments.output_dir`. If this was not intended, please
specify a different run name by setting the `TrainingArguments.run_name`
parameter.
```

<IPython.core.display.HTML object>

```
[10]: TrainOutput(global_step=1875, training_loss=0.37476914825439456,
metrics={'train_runtime': 1170.8772, 'train_samples_per_second': 25.622,
'train_steps_per_second': 1.601, 'total_flos': 1959690240000000.0, 'train_loss':
0.37476914825439456, 'epoch': 3.0})
```

```
[11]: # Evaluate on the test dataset
metrics = trainer.evaluate(test_dataset)
print(metrics)
```

```
<IPython.core.display.HTML object>

{'eval_loss': 0.33578893542289734, 'eval_runtime': 107.3908,
'eval_samples_per_second': 95.893, 'eval_steps_per_second': 5.997, 'epoch': 3.0}
```

```
[12]: # Step 8: Save the trained model
trainer.save_model("/content/drive/MyDrive/Colab Notebooks/data/
 ↪fine_tuned_gpt2")
tokenizer.save_pretrained("/content/drive/MyDrive/Colab Notebooks/data/
 ↪fine_tuned_gpt2")
```

```
[12]: ('/content/drive/MyDrive/Colab
Notebooks/data/fine_tuned_gpt2/tokenizer_config.json',
 '/content/drive/MyDrive/Colab
Notebooks/data/fine_tuned_gpt2/special_tokens_map.json',
 '/content/drive/MyDrive/Colab Notebooks/data/fine_tuned_gpt2/vocab.json',
 '/content/drive/MyDrive/Colab Notebooks/data/fine_tuned_gpt2/merges.txt',
 '/content/drive/MyDrive/Colab
Notebooks/data/fine_tuned_gpt2/added_tokens.json',
 '/content/drive/MyDrive/Colab Notebooks/data/fine_tuned_gpt2/tokenizer.json')
```

2.1 Afer Finetuning the model

```
[13]: from transformers import GPT2LMHeadModel, GPT2Tokenizer

# Load the fine-tuned model and tokenizer
model = GPT2LMHeadModel.from_pretrained("/content/drive/MyDrive/Colab Notebooks/
 ↪data/fine_tuned_gpt2")
tokenizer = GPT2Tokenizer.from_pretrained("/content/drive/MyDrive/Colab␣
 ↪Notebooks/data/fine_tuned_gpt2")

# Generate responses for custom prompts
prompts = [
    "congratulations",
    "good afternoon madam",
]

for prompt in prompts:
    inputs = tokenizer(prompt, return_tensors="pt")
    outputs = model.generate(inputs["input_ids"], max_length=50, num_beams=5,␣
  ↪early_stopping=True)
    print(tokenizer.decode(outputs[0], skip_special_tokens=True))
```

```
The attention mask and the pad token id were not set. As a consequence, you may
observe unexpected behavior. Please pass your input's `attention_mask` to obtain
reliable results.
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
The attention mask is not set and cannot be inferred from input because pad
token is same as eos token. As a consequence, you may observe unexpected
behavior. Please pass your input's `attention_mask` to obtain reliable results.
The attention mask and the pad token id were not set. As a consequence, you may
observe unexpected behavior. Please pass your input's `attention_mask` to obtain
reliable results.
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.
```

```
congratulations to you sir
good afternoon madam i am sorry for the delay
```

2.2 Before Fine Tuning the model.

[14]:
```
from transformers import AutoTokenizer, AutoModelForCausalLM

# Load the fine-tuned model and tokenizer
tokenizer = AutoTokenizer.from_pretrained("gpt2")  # Replace with the path to␣
  ↪your fine-tuned model
model = AutoModelForCausalLM.from_pretrained("gpt2")  # Replace with the path␣
  ↪to your fine-tuned model

# Set pad_token to eos_token (since GPT-2 doesn't have a dedicated padding␣
  ↪token)
tokenizer.pad_token = tokenizer.eos_token  # Setting the pad token to eos_token
model.config.pad_token_id = model.config.eos_token_id  # Ensuring the model␣
  ↪config uses eos_token for padding

# Define the prompt
prompt = "Congratulations"

# Tokenize input with padding and attention mask
inputs = tokenizer(prompt, return_tensors="pt", padding=True, truncation=True,␣
  ↪max_length=50)

# Generate text using the model, ensuring attention mask is passed
outputs = model.generate(
    inputs["input_ids"],
    max_length=50,
    num_return_sequences=1,
    attention_mask=inputs["attention_mask"],  # Pass attention mask to avoid␣
 ↪padding issues
    top_p=0.95,  # Nucleus sampling for more varied generation
    temperature=0.7,  # Temperature setting to introduce randomness into␣
 ↪generation
    do_sample=True  # Enable sampling to make use of top_p and temperature
)

# Decode and print the generated text
generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
print(generated_text)
```

```
Setting `pad_token_id` to `eos_token_id`:None for open-end generation.

Congratulations as you do. I'd like to thank you for your time, and I hope
you'll be able to share more of this information with your family, friends and
colleagues.
```

2.2.1 Observations:

Before Fine-tuning:

- The model's output was not very relevant to the prompt.
- It seemed to pull random content or context that wasn't aligned with the input, such as references to forums, websites, and disjointed responses.

After Fine-tuning:

- The outputs are much more contextually aligned with the input prompt.
- For example, the response to "congratulations" is now a more suitable and relevant phrase like "congratulations to you sir," and the response to "good afternoon madam" is more coherent with "good afternoon madam, I am sorry for the delay."
- Fine-tuning has clearly helped the model generate responses that are more in line with the desired tone and context for conversational text.

2.2.2 Conclusion:

- Fine-tuning the model on your specific dataset has led to a more appropriate and focused response for conversational prompts, moving away from the random, less relevant outputs before fine-tuning. This aligns the model with the language patterns and behavior that you aimed to train it on.

- The interesting difference observed is that after fine-tuning, the model's responses are **more coherent, polite, and contextually appropriate**, in contrast to the disjointed and seemingly random outputs it produced before fine-tuning.

If you didn't observe a similar difference, there could be several factors to check:

- **Insufficient training data:** The dataset may not have been large or diverse enough to allow the model to generalize well.
- **Training duration:** If the model wasn't trained for long enough, it may not have had sufficient time to learn the desired patterns.
- **Model size and complexity:** If using a smaller model or suboptimal configurations, the fine-tuning might not have led to a significant improvement in output quality.