Konfuzio "AI Comedy Club" Challenge

Project Report

Introduction

Developing an AI bot that can understand humour and generate jokes can be a challenging task due to the abstract and subjective nature of humour. Humour often relies on cultural references, wordplay, and timing, making it even more challenging to capture and replicate in an AI bot. That said, the potential benefits and applications of such an AI bot are vast, ranging from enhancing social interactions to providing entertainment and assistance in various domains. In this project, we delve into the interesting and challenging world of developing AI bots that can understand humour and generate and rate jokes.

Data Insights

The Short Jokes dataset from Kaggle is a large collection of short jokes in English that includes both one-liners and longer jokes. The dataset contains 231,657 jokes in CSV format. The Joke Dataset from taivop/joke-dataset on GitHub includes two separate datasets: Stupidstuff and Wocka. The Stupidstuff dataset contains 3,770 English plaintext jokes scraped from stupidstuff.org, while the Wocka dataset contains 10,000 English plaintext jokes scraped from wocka.com. Both datasets are in JSON format and contain additional fields such as category, title, and rating. The total number of jokes in all three datasets combined is 245,427 jokes. We are going to work with a subset of the combined dataset.

Table 1: Joke Datasets Information

Dataset Name	Description	Source	Number of Jokes	Format
Short Jokes	A collection of short jokes in English	Kaggle	231,657	CSV
Joke Dataset - Stupidstuff	A dataset of English plaintext jokes from stupidstuff.org	GitHub	3,770	JSON
Joke Dataset - Wocka	A dataset of English plaintext jokes from wocka.com	GitHub	10,000	JSON

Data Pre-processing

To pre-process jokes data, several steps are followed. Firstly, special characters are removed from the jokes. Next, any numbers present in the jokes are eliminated. Leading and trailing whitespace are stripped to ensure consistency. Additionally, newlines and carriage returns are removed to maintain a clean format. Finally, a check is performed to determine if the joke is considered free of profanity after the pre-processing steps have been applied. These pre-processing steps help to standardise the text and ensure that the jokes are suitable for further analysis or processing without any offensive content.

Model Selection

There are several models that we can leverage to create a bot that can generate fresh and original jokes. Fine-tuning allows us to customise the model's behaviour and align it specifically for the task of generating jokes. By feeding the model with a carefully curated dataset of jokes, it can learn the patterns, structures, and comedic elements that make jokes amusing. Here are a few models that can be fine-tuned for joke generation:

- **bert-base-uncased:** BERT (Bidirectional Encoder Representations from Transformers) is a widely-used transformer-based model that has achieved state-of-the-art performance on various natural language processing tasks. The "base" variant refers to its medium-sized configuration, offering a balance between model size and performance. "uncased" indicates that the model treats all text as lowercase, disregarding capitalization. BERT incorporates a deep bidirectional transformer encoder, capturing contextual information from both preceding and following words. It is pretrained on a large corpus and can be fine-tuned for specific tasks.

- **distilbert-base-uncased:** DistilBERT is a distilled version of BERT, striking a good balance between performance and efficiency. It retains competitive performance while being smaller and faster than the original BERT model. Like BERT, "uncased" signifies that the model operates with lowercase text. DistilBERT achieves efficiency gains through techniques such as knowledge distillation and parameter reduction. Its reduced size makes it more manageable and quicker to fine-tune, particularly in scenarios with limited computational resources or smaller training datasets.

- **GPT-2:** GPT-2 (Generative Pre-trained Transformer 2) is a cutting-edge language model explicitly designed for text generation tasks. Renowned for its ability to produce high-quality and coherent text, GPT-2 is particularly well-suited for joke generation. Built upon a transformer architecture with a substantial number of parameters, GPT-2 captures long-range dependencies in input text effectively.

We will start with fine-tuning the GPT-2 (124M parameter) model for joke generation. This particular variant of the model is computationally less demanding compared to larger versions, making it an ideal starting point. In addition to the joke generation model, we will also utilise another model from the Hugging Face library as our joke rater.

Setup

Before we start the training process, we need to set up the GPT-2 language model and tokeniser. We initialise the GPT2LMHeadModel from the pre-trained "gpt2" model and create a GPT2Tokenizer using the same pre-trained model.

To accommodate our specific requirements, we define custom special tokens: the beginning of sequence token (bos), end of sequence token (eos), and padding token (pad). These tokens will play a crucial role in shaping the model's understanding of joke structures. We add these custom tokens to the tokeniser, ensuring that the tokeniser recognises and handles them appropriately during the tokenisation process.

Next, we create a model configuration, specifying the custom tokens and setting output_hidden_states to False, as we do not require hidden states for our training process. The GPT2LMHeadModel is loaded with the pre-trained "gpt2" model and the custom configuration. Additionally, the token embeddings are resized to include the new tokens, ensuring compatibility between the model and the tokeniser.

To tokenise our data, we define a function that takes in the data and a maximum length parameter. This function maps over the data and tokenises the "jokes" field, applying padding and truncation to ensure a maximum length of 512 tokens. The data is processed in batches for efficiency. We then tokenise our training, validation, and test jokes. These tokenised datasets will be used during the training and evaluation phases.

Training Approach

We define the training arguments for our training process. The TrainingArguments object is initialised with various parameters. The model is trained for 4 epochs. The batch size for training and evaluation is both set to 8 to accommodate resource constraints. We set the learning rate to 2e-5 and the weight decay to 0.01 to optimise the training process and improve the model's generalisation performance.

To facilitate language modelling, we create a DataCollatorForLanguageModeling object, which takes in the tokeniser and sets mlm (masked language modelling) to False. In this case, the model will learn to predict the next token in the sentence based on the context provided by the preceding tokens, without the additional step of masking and predicting the masked tokens.

We then instantiate a Trainer object, providing it with the model, training arguments, data collator, and tokenised training and evaluation datasets. We also include the EarlyStoppingCallback with a patience of 2 to prevent overfitting and improve efficiency.

Results and Analysis

The training process showed a consistent decrease in both training and validation loss as the number of steps increased. The validation loss consistently improved throughout the training process. After training the model for 4 epochs, the loss was 3.97 on the test set.

Here are a few sample jokes generated by the model:

1. Theres no good way to describe the world I live in without being very funny.
2. A man was walking down the street when he saw two people trying to hide in his car.
3. Im going to be the next Harry Potter.
4. My brother told me to go buy some cheese. I said, No nooope!
5. How about when you have your birthday party... and all of the guests are in there, laughing.
6. A boy walks into an elevator.
7. A man walks into a grocery store and asks the clerk, What kind of food do you like?
8. I was driving through the woods and saw an owl flying across my road.
9. So yeah, my dad would be the president.

We can clearly see that the performance of the model leaves a lot to be desired. In my experiments with fine-tuning models for joke generation, I attempted to train larger variants of the GPT-2 model, such as GPT-2 Medium (345M parameters) and GPT-2 Large (774M parameters) as well. However, due to limited computational resources, I had to revert to using the GPT-2 base model (117M parameters).

Concluding Remarks

It is important to acknowledge that the performance of the model in generating jokes could potentially be improved with a larger model. With increased computational resources, the model could be trained on a larger scale, allowing for better joke generation capabilities. Furthermore, the availability of better and more diverse data could significantly enhance the model's performance. Lastly, the development of a better bot goes beyond just the technical aspects. While there are challenges and potential areas for improvement, the development of AI bots capable of generating jokes holds great promise.