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: Overview of Joke Datasets

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. For our joke rating task, we utilise the VADER (Valence Aware Dictionary and sEntiment Reasoner) sentiment analyser to generate sentiment scores for a subset of jokes and map the compound scores in the range [-1, 1] to a 1-10 rating scale. The scaled sentiment scores can be considered as target ratings for this data subset.

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 fine-tune the distilbert-base-uncased model for our joke rating task. By fine-tuning the model specifically for joke rating, we can leverage its language comprehension capabilities to generate ratings. The generated ratings can provide a helpful reference but should not be taken as an absolute measure of a joke's comedic worth.

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.

In addition, the setup and training process for fine-tuning the distilbert-base-uncased model for our jokes rating task mostly follow a similar approach.

Results and Analysis

Table 2 illustrates the progression of training and validation loss during the model training process. The training loss represents the error between predicted and actual values on the training dataset, while the validation loss indicates the performance of the model on a separate validation dataset.

At the beginning of training, the training loss is relatively high at 6.30, indicating a significant initial error. However, as the training progresses, the model's performance improves. Similarly, the validation loss starts at 4.22 and gradually decreases throughout training.

After training the model for 4 epochs, the loss was 3.97 on the test set.

Table 2: Training and Validation Loss Progression During Model Training

Step	Training Loss	Validation Loss
500	6.303400	4.218644
1000	4.233000	4.113698
1500	4.008900	4.073078
2000	3.984900	4.045985
2500	3.855800	4.033286
3000	3.801600	4.024386
3500	3.764100	4.024865
4000	3.719600	4.019519

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 (124M parameter).

Table 3 shows the progression of training and validation loss during the joke rater model training process as well as Mean Squared Error (MSE), Root Mean Squared Error (RMSE), and Mean Absolute Error (MAE) metrics at different steps of the training process.

Table 3: Model Evaluation Metrics - Training Loss, Validation Loss, and Error Analysis

Step	Training Loss	Validation Loss	Mse	Rmse	Mae
500	4.687900	2.334007	6.614991	2.571963	2.069519
1000	2.087100	1.648909	6.806559	2.608938	2.062700
1500	1.405600	1.420838	7.791146	2.791262	2.205038
2000	1.167900	1.378124	8.595682	2.931839	2.321136
2500	0.945800	1.154988	8.322764	2.884920	2.288404
3000	0.680900	1.027863	8.051106	2.837447	2.238993
3500	0.670800	0.977394	7.807900	2.794262	2.202772
4000	0.497000	0.965505	8.102925	2.846564	2.251322
4500	0.457300	0.956578	8.111990	2.848155	2.248373

Here are a few sample jokes, each rated by the model:

1. Why don't scientists trust atoms? Because they make up everything! 5.09
2. Why don't skeletons fight each other? They don't have the guts! 4.69
3. What do you call a fish with no eyes? Fsh! 3.09
4. Why did the scarecrow win an award? Because he was outstanding in his field! 9.26
5. Why don't eggs tell jokes? Because they might crack up! 5.26

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 more advanced bot capable of generating jokes and providing precise ratings extends 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.