Fine Tuning Large Language Models:
Finance AI Chatbot

Alex Apostolu
Rosie Khurmi
Ishreet Grewal
Rhea Gokhale

Aug 16 2024

Introduction

Background

Large Language Models (LLMs) have transformed the landscape of artificial intelligence by enabling machines to understand and generate human-like text. These models have a wide range of applications, from natural language processing (NLP) tasks like text summarization, translation, and question-answering to domain-specific implementations such as legal research, financial forecasting, and customer service automation.

Objective

The goal of this project was to fine-tune a Large Language Model specifically for the financial domain, enhancing its capability to provide accurate, context-aware answers to finance-related queries. The motivation behind this fine-tuning was to create a personal finance chatbot that could leverage both legal and financial information, offering users insightful and precise guidance.

Scope

The project focused on fine-tuning a pre-trained LLM using finance-related data, including legal frameworks, investment strategies, real-time stock data, and terminology. The model, initially trained on general language tasks, was adapted to specialize in finance to improve its performance in answering domain-specific questions.

Project Overview

Dataset

The dataset used for fine-tuning consisted of finance-related information, including legal documentation, investment strategies, and real-time stock data. The data was sourced from publicly available financial databases, regulatory websites, and curated datasets. Preprocessing steps included data cleaning, normalization, and the conversion of raw text data into a structured format suitable for model training.

Tools and Environment

The fine-tuning process was conducted using Hugging Face's Transformers library in a Python environment. Google Collab was used as the primary development platform due to its accessible GPU resources. The project utilized Python 3.x, along with libraries such as pandas for data manipulation, PyMongo for database interaction, as well as bleu score and RAG triad for evaluation metrics.

Fine-tuning Methodology

The fine-tuning process involved using the dataset to retrain the model's weights, with a focus on improving its understanding of finance-related queries. Techniques such as Transfer Learning and Retrieval-Augmented Generation (RAG) were employed to enhance the model's ability to retrieve relevant information from the MongoDB database and generate accurate responses.

Challenges

Several challenges arose during the project, including managing data quality and acceptance, addressing overfitting, and overcoming computational limitations. Ensuring the relevance and accuracy of the financial data used for fine-tuning was critical, as was optimizing the model's performance without overfitting to the training data. The code structure was being consistently changed to be able to be accepted by hugging face to host the model.

Implementation

Step-by-Step Process:

Data Preparation:

To prepare the data for fine-tuning we firstly started off by web scraping personal finance data from the web. This included public websites, reddit posts, stock market information, etc. We then cleaned-up the data by removing irrelevant information that was caught in the process of web scraping. Then we structured the data into a question answer format which then we had to format into a yaml file.

Model Configuration:

To finetune this LLM we used InstructLab. We used the InstructLab's command-line interface to download a pre-trained LLM, generate new synthetic training data, retrain the LLM, as well as chat with the LLM.

To begin with, we choose the pre-trained model merlinite-7b as our base model. We choose to specifically focus on fine tuning and modifying the finance data. So for this, we simply added a new folder under taxonomy->knowledge->finance and added our data under qna.yaml.

To generate the new data, we used “ilab generate” which created synthetic dataset based on your newly added finance data

Training:

To train the model we used the command “ilab model train” which produces a new model in numpy compressed array format.

Since we used a pre trained model the number of epochs used were probably lower (3-10) since the model weights were already partially trained

The learning rate controls how much the model's weights are adjusted with respect to the loss gradient. Since we are fine-tuning a model I would guess our learning rate would be smaller (around 1e-5 to 1e-4).

Evaluation:

To evaluate the model’s performance we used the RAG Triad. RAG stands for retrieval augmented generation. We attempted to use the RAG Triad evaluation method which evaluates the LLM based on context relevance, groundedness and answer relevance. We attempted to use the framework beyondllm as well as deepeval to evaluate our model, but since our model was too large, we were not successful in implementing the RAG triad. Beyondllm, is an all in one tool kit that allows us to experiment, deploy and evaluate RAG systems which enhance reliability and reduce hallucinations. To implement beyondllm, we use the data, retriever and then pipeline. Then use pipeline.call() for the AI response and then we can evaluate using pipeline.get_rag_triad_evals().

```
# Define a groundedness feedback function
f_groundedness = (
    Feedback(provider.groundedness_measure_with_cot_reasons, name = "Groundedness")
    .on(Select.RecordCalls.retrieve.rets.collect())
    .on_output()
)

# Question/answer relevance between overall question and answer.
f_answer_relevance = (
    Feedback(provider.relevance_with_cot_reasons, name = "Answer Relevance")
    .on_input()
    .on_output()
)

# Context relevance between question and each context chunk.
f_context_relevance = (
    Feedback(provider.context_relevance_with_cot_reasons, name = "Context Relevance")
    .on_input()
    .on(Select.RecordCalls.retrieve.rets[:])
    .aggregate(np.mean) # choose a different aggregation method if you wish
)
```

```
data = fit('before_and_after.txt', dtype='pdf')
```

```
embed_model = FastEmbedEmbeddings()

retriever = auto_retriever(data=data, embed_model=embed_model, type='normal', top_k=3)

llm = HuggingFaceHubModel(model='Ishreet1/FinanceLLM',
token=os.getenv('HUGGINGFACE_ACCESS_TOKEN'))

pipeline = Generate(question='Explain the concept of buy and hold', llm=llm,
retriever=retriever)

print(pipeline.call())

print(pipeline.get_rag_triad_evals())
```

When attempting to use deepeval, we used metrics such as contextual precision, contextual recall, contextual relevance, answer_relevancy, and faithfulness to evaluate the models performance.

```
# Evaluation retrivals

contextual_precision = ContextualPrecisionMetric(model=llm)

contextual_recall = ContextualRecallMetric(model=llm)

contextual_relevancy = ContextualRelevancyMetric(model=llm)

# Evaluation generation

answer_relevancy = AnswerRelevancyMetric(model=llm)

faithfulness = FaithfulnessMetric(model=llm)

# Evaluating RAG

print(evaluate(

   test_cases=[test_case],

   metrics=[

       contextual_precision,

       contextual_recall,

       contextual_relevancy,
```

```
    answer_relevancy,
    faithfulness,
]))
```

Alongside RAG, we manually tested our model. Since we fine-tuned an existing pre-trained model, after adding our data we compared the responses to some sample questions. We checked if the LLM gave more personalized and specific responses in respect to personal finance. We were successful in this area as we noticed a higher quality of responses to finance based questions.

Lastly we also used the command “ilab test” which generated a series of outputs from the model before and after training. This also was a good way to evaluate the model’s performance because we were able to manually see how our data changed the response of the LLM.

Results

Performance Metrics

Although we tried multiple evaluation methods, a lot of the methods were unsuccessful due to our model being very large. So we were not able to get a good set of results for our model. As mentioned before, we attempted using RAG, BLEU score, Instructlab test results, and manually testing our model to get results. The only successful result we were able to attain was our BLEU score code. We had a range of scores between 0 and 1 which tells us that our finetuning was able to produce results similar to our references. The score was arguably better with our fine tuned version of this model than the base.

Comparison

As mentioned before, when we used the“ilab test” command we were able to compare the performance of our fine-tuned model with the previous pre-trained model. Our goal was to compare our results with an open-source LLM like ChatGPT but due to resource limitations and not having access to a free API key we were not able to.

Observations

From manually looking at the results from our fine-tuned model in comparison to the pre-trained model we were able to see a drastic change in the responses to personal finance. We were able to see that the answers were much more specific and to-the-point in comparison to the pre-trained model. For example, if the prompt was “How can I invest 700 dollars?”, the

pre-trained model would respond in a vague way saying that it depends on what resources you have available and so on. Our model gives specific ways to invest your 700 dollars saying, “Ensure your funds are safe by placing them in high-yield savings accounts that offer competitive interest rates, invest in low-risk securities, Diversify your portfolio, Leverage compound interest…”, etc.

Conclusion

Future Work

Future efforts could focus on expanding the dataset to include more diverse financial scenarios, refining the model's handling of ambiguous queries, and exploring other techniques such as reinforcement learning to further enhance the model's accuracy and usability.

Summary

This project successfully fine-tuned a Large Language Model to specialize in the financial domain, resulting in a chatbot capable of providing accurate and context-aware financial advice. The integration of legal and financial data into the model's training process noticeably improved its performance on finance-related tasks.

Citations

Ai deploy - tutorial - deploy an app for sentiment analysis with hugging face and Flask. OVHcloud. (n.d.). https://help.ovhcloud.com/csm/en-ca-public-cloud-ai-deploy-flask-hugging-face-sentiment-analysis?id=kb_article_view&sysparm_article=KB0035507

Hugging face. DeepEval - The Open-Source LLM Evaluation Framework. (n.d.-a). https://docs.confident-ai.com/docs/integrations-huggingface

Priyanka. (2022, November 16). *Evaluation metrics in natural language processing-bleu*. Medium. https://medium.com/@priyankads/evaluation-metrics-in-natural-language-processing-bleu-dc3cfa8faaa5

Rag evaluation. DeepEval - The Open-Source LLM Evaluation Framework. (n.d.-b). https://docs.confident-ai.com/docs/guides-rag-evaluation

Tomar, A. (2024, May 19). *InstructLab: A new approach to fine-tuning large language models*. Medium.

https://medium.com/@263akash/instructlab-a-new-approach-to-fine-tuning-large-language-models-6ffb65aa1d57

transformers. . (n.d.). https://huggingface.co/docs/transformers/index

GITHUB LINK

https://github.com/saapte32/financeLLMilab/tree/main