Fine-Tuning GPT-4o Mini for Financial Sentiment Analysis

ADVANCED GENERATIVE AI SENTIMENT ANALYSIS

Sentiment analysis in finance is a powerful tool for understanding market trends and investor behavior. However, general sentiment analysis models often fall short when applied to financial texts due to their complexity and nuanced nature. This project proposes a solution by fine-tuning GPT-4o mini, a lightweight language model. By utilizing the TRC2 dataset, a collection of Reuters financial news articles labeled with sentiment classes by the expert model FinBERT, we aim to enhance GPT-4o mini's ability to capture financial sentiment nuances.

This project provides an efficient and scalable approach to financial sentiment analysis, opening the door for more nuanced sentiment-based analysis in finance. By the end, we demonstrate that GPT-4o mini, when fine-tuned with domain-specific data, can serve as a viable alternative to more complex models like FinBERT in financial contexts.

Learning Outcomes

- Understand the process of fine-tuning GPT-4o mini for financial sentiment analysis using domain-specific data.
- Learn how to preprocess and format financial text data for model training in a structured and scalable manner.
- Gain insights into the application of sentiment analysis for financial texts and its impact on market trends.
- Discover how to leverage expert-labeled datasets like FinBERT for improving model performance in financial sentiment analysis.
- Explore the practical deployment of a fine-tuned GPT-4o mini model in real-world financial applications such as market analysis and automated news sentiment tracking.

This article was published as a part of the Data Science Blogathon.

Table of contents

Exploring the Dataset: Essential Data for Sentiment Analysis

For this project, we use the TRC2 (TREC Reuters Corpus, Volume 2) dataset, a collection of financial news articles curated by Reuters and made available through the National Institute of Standards and Technology (NIST). The TRC2 dataset includes a comprehensive selection of Reuters financial news articles, often used in financial language models due to its wide coverage and relevance to financial events.

Accessing the TRC2 Dataset

To obtain the TRC2 dataset, researchers and organizations need to request access through NIST. The dataset is available at NIST TREC Reuters Corpus, which provides details on licensing and usage agreements. You will need to:

- Visit the NIST TREC Reuters Corpus.
- Follow the dataset request process specified on the website.
- Ensure compliance with the licensing requirements to use the dataset in research or commercial projects.

Once you obtain the dataset, preprocess and segment it into sentences for sentiment analysis, allowing you to apply FinBERT to generate expert-labeled sentiment classes.

Research Methodology: Steps to Analyze Financial Sentiment

The methodology for fine-tuning GPT-4o mini with sentiment labels derived from FinBERT consists of the following main steps:

Step 1: FinBERT Labeling

To create the fine-tuning dataset, we leverage FinBERT, a financial language model pre-trained on the financial domain. We apply FinBERT to each sentence in the TRC2 dataset, generating expert sentiment labels across three classes: Positive, Negative, and Neutral. This process produces a labeled dataset

where each sentence from TRC2 is associated with a sentiment, thus providing a foundation for training GPT-4o mini with reliable labels.

Step 2: Data Preprocessing and JSONL Formatting

The labeled data is then preprocessed and formatted into a JSONL structure suitable for OpenAI's fine-tuning API. We format each data point with the following structure:

- A system message specifying the assistant's role as a financial expert.
- A user message containing the financial sentence.
- An assistant response that states the predicted sentiment label from FinBERT.

After labeling, we perform additional preprocessing steps, such as converting labels to lowercase for consistency and stratifying the data to ensure balanced label representation. We also split the dataset into training and validation sets, reserving 80% of the data for training and 20% for validation, which helps assess the model's generalization ability.

Step 3: Fine-Tuning GPT-4o Mini

Using OpenAI's fine-tuning API, we fine-tune GPT-4o mini with the pre-labeled dataset. Fine-tuning settings, such as learning rate, batch size, and number of epochs, are optimized to achieve a balance between model accuracy and generalizability. This process enables GPT-4o mini to learn from domain-specific data and improves its performance on financial sentiment analysis tasks.

Step 4: Evaluation and Benchmarking

After training, the model's performance is evaluated using common sentiment analysis metrics like accuracy and F1-score, allowing a direct comparison with FinBERT's performance on the same data. This benchmarking demonstrates how well GPT-4o mini generalizes sentiment classifications within the financial domain and confirms if it can consistently outperform FinBERT in accuracy.

Step 5: Deployment and Practical Application

Upon confirming superior performance, GPT-4o mini is ready for deployment in real-world financial applications, such as market analysis, investment advisory, and automated news sentiment tracking. This fine-tuned model provides an efficient alternative to more complex financial models, offering robust, scalable sentiment analysis capabilities suitable for integration into financial systems.

If you want to learn the basics of Sentiment Analysis, checkout our article on Sentiment Analysis using Python!

Fine-Tuning GPT-4o Mini for Financial Sentiment Analysis

Follow this structured, step-by-step approach to seamlessly navigate through each stage of the process. Whether you're a beginner or experienced, this guide ensures clarity and successful implementation from start to finish.

Step 1: Initial Setup

Load Required Libraries and Configure the Environment.

```
from transformers import AutoTokenizer, AutoModelForSequenceClassification import torch import pandas as pd from tqdm import tqdm tokenizer = AutoTokenizer.from_pretrained("ProsusAI/finbert") model = AutoModelForSequenceClassification.from_pretrained("ProsusAI/finbert") device = torch.device('cuda' if torch.cuda.is_available() else 'cpu') model.to(device)
```

Step 2: Define a Function to Generate Sentiment Labels with FinBERT

- This function accepts text input, tokenizes it, and uses FinBERT to predict sentiment labels.
- Label Mapping: FinBERT outputs three classes—Positive, Negative, and Neutral.

```
def get_sentiment(text): inputs = tokenizer(text, return_tensors="pt", truncation=True, max_length=512).to(device) with torch.no_grad(): outputs = model(**inputs) logits = outputs.logits sentiment = torch.argmax(logits, dim=1).item() sentiment_label = ["Positive", "Negative", "Neutral"][sentiment] return sentiment_label
```

Step 3: Data Preprocessing and Sampling the TRC2 Dataset

You must carefully preprocess the TRC2 dataset to retain only relevant sentences for fine-tuning. The following steps outline how to read, clean, split, and filter the data from the TRC2 dataset.

Given the constraints of non-disclosure, this section provides a high-level overview of the data preprocessing workflow with pseudocode.

- **Load and Extract Data:** The dataset, provided in a compressed format, was loaded and extracted using standard text handling methods. Relevant sections of each document were isolated to focus on key textual content.
- **Text Cleaning and Sentence Segmentation:** After isolating content sections, each document was cleaned to remove extraneous characters and ensure consistency in formatting. This prepared the content for splitting into sentences or smaller text units, which enhances model performance by providing manageable segments for sentiment analysis.
- **Structured Data Storage:** To facilitate streamlined processing, the data was organized into a structured format where each row represents an individual sentence or text segment. This setup allows for efficient processing, filtering, and labeling, making it suitable for fine-tuning language models.
- **Filter and Screen for Relevant Text Segments:** To maintain high data quality, we applied various criteria to filter out irrelevant or noisy text segments. These criteria included eliminating overly short segments, removing those with specific patterns indicative of non-sentiment-bearing content, and excluding segments with excessive special characters or specific formatting characteristics.
- **Final Preprocessing:** Only the segments that met predefined quality standards were retained for model training. The filtered data was saved as a structured file for easy reference in the fine-tuning workflow.

```
# Load the compressed dataset from file open compressed_file as file: # Read the contents of the file into memory data = read_file(file) # Extract relevant sections of each document for each document in data: extract document_id extract date extract main_text_content # Define a function to clean and segment text content function clean_and_segment_text(text): # Remove unwanted characters and whitespace cleaned_text =
```

```
remove_special_characters(text) cleaned_text = standardize_whitespace(cleaned_text) # Split the cleaned text into sentences or text segments sentences = split_into_sentences(cleaned_text) return sentences # Apply the cleaning and segmentation function to each document's content for each document in data: sentences = clean_and_segment_text(document['main_text_content']) save sentences to structured format # Create a structured data storage for individual sentences initialize empty list of structured_data for each sentence in sentences: # Append sentence to structured data structured_data.append(sentence) # Define a function to filter out unwanted sentences based on specific criteria function filter_sentences(sentence): if sentence is too short: return False if sentence contains specific patterns (e.g., dates or excessive symbols): return False if sentence matches unwanted formatting characteristics: return False return True # Apply the filter to structured data filtered_data = [sentence for sentence in structured_data if filter_sentences(sentence)] # Further filter the sentences based on minimum length or other criteria final_data = [sentence for sentence in filtered_data if meets_minimum_length(sentence)] # Save the final data structure for model training save final_data as structured_file
```

- Load the dataset and sample 1,000,000 sentences randomly to ensure a manageable dataset size for fine-tuning.
- Store the sampled sentences in a DataFrame to enable structured handling and easy processing.

```
df_sampled = df.sample(n=1000000, random_state=42).reset_index(drop=True)
```

Step 4: Generate Labels and Prepare JSONL Data for Fine-Tuning

- Loop through the sampled sentences, use FinBERT to label each sentence, and format it as JSONL for GPT-4o mini fine-tuning.
- Structure for JSONL: Each entry includes a system message, user content, and the assistant's sentiment response.

```
import json jsonl_data = [] for _, row in tqdm(df_sampled.iterrows(), total=df_sampled.shape[0]): content = row['sentence'] sentiment = get_sentiment(content) jsonl_entry = { "messages": [ {"role": "system", "content": "The assistant is a financial expert."}, {"role": "user", "content": content}, {"role": "assistant", "content": sentiment} ] } jsonl_data.append(jsonl_entry) with open('finetuning_data.jsonl', 'w') as jsonl_file: for entry in jsonl_data: jsonl_file.write(json.dumps(entry) + '\n')
```

Step 5: Convert Labels to Lowercase

- Ensure label consistency by converting sentiment labels to lowercase, aligning with OpenAI's formatting for fine-tuning.

```
with open('finetuning_data.jsonl', 'r') as jsonl_file: data = [json.loads(line) for line in jsonl_file] for entry in data: entry["messages"][2]["content"] = entry["messages"][2]["content"].lower() with open('finetuning_data_lowercase.jsonl', 'w') as new_jsonl_file: for entry in data: new_jsonl_file.write(json.dumps(entry) + '\n')
```

Step 6: Shuffle and Split the Dataset into Training and Validation Sets

- Shuffle the Data: Randomize the order of entries to eliminate ordering bias.

- Split into 80% Training and 20% Validation Sets.

```
import random random.seed(42) random.shuffle(data) split_ratio = 0.8 split_index = int(len(data) * split_ratio) training_data = data[:split_index] validation_data = data[split_index:] with open('training_data.jsonl', 'w') as train_file: for entry in training_data: train_file.write(json.dumps(entry) + '\n') with open('validation_data.jsonl', 'w') as val_file: for entry in validation_data: val_file.write(json.dumps(entry) + '\n')
```

Step 7: Perform Stratified Sampling and Save Reduced Dataset

- To further optimize, perform stratified sampling to create a reduced dataset while maintaining label proportions.
- Use Stratified Sampling: Ensure equal distribution of labels across both training and validation sets for balanced fine-tuning.

```
from sklearn.model_selection import train_test_split data_df = pd.DataFrame({ 'content': [entry["messages"][1]["content"] for entry in data], 'label': [entry["messages"][2]["content"] for entry in data] }) df_sampled, _ = train_test_split(data_df, stratify=data_df['label'], test_size=0.9, random_state=42) train_df, val_df = train_test_split(df_sampled, stratify=df_sampled['label'], test_size=0.2, random_state=42) def df_to_jsonl(df, filename): jsonl_data = [] for _, row in df.iterrows(): jsonl_entry = { "messages": [ {"role": "system", "content": "The assistant is a financial expert."}, {"role": "user", "content": row['content']}, {"role": "assistant", "content": row['label']} ] } jsonl_data.append(jsonl_entry) with open(filename, 'w') as jsonl_file: for entry in jsonl_data: jsonl_file.write(json.dumps(entry) + '\n') df_to_jsonl(train_df, 'reduced_training_data.jsonl') df_to_jsonl(val_df, 'reduced_validation_data.jsonl')
```

Step 8: Fine-Tune GPT-4o Mini Using OpenAI's Fine-Tuning API

- With your prepared JSONL files, follow OpenAI's documentation to fine-tune GPT-4o mini on the prepared training and validation datasets.
- Upload Data and Start Fine-Tuning: Upload the JSONL files to OpenAI's platform and follow their API instructions to initiate the fine-tuning process.

Step 9: Model Testing and Evaluation

To evaluate the fine-tuned GPT-4o mini model's performance, we tested it on a labeled financial sentiment dataset available on Kaggle. This dataset contains 5,843 labeled sentences in financial contexts, which allows for a meaningful comparison between the fine-tuned model and FinBERT.

FinBERT scored an accuracy of 75.81%, while the fine-tuned GPT-4o mini model achieved 76.46%, demonstrating a slight improvement.

Here's the code used for testing:

```
import pandas as pd import os import openai from dotenv import load_dotenv # Load the CSV file csv_file_path = 'data.csv' # Replace with your actual file path df = pd.read_csv(csv_file_path) # Convert DataFrame to text format with open('sentences.txt', 'w', encoding='utf-8') as f: for index, row in df.iterrows(): sentence = row['Sentence'].strip() # Clean sentence sentiment = row['Sentiment'].strip().lower() # Ensure sentiment is lowercase and clean f.write(f"{sentence} @{sentiment}\n") # Load environment variables load_dotenv() # Set your OpenAI API key openai.api_key = os.getenv("OPENAI_API_KEY") # Ensure OPENAI_API_KEY is set in your environment variables # Path to the dataset text file file_path = 'sentences.txt' # Text file containing sentences and labels # Read sentences and true labels from the dataset sentences = [] true_labels = [] with open(file_path, 'r', encoding='utf-8') as file: lines = file.readlines() # Extract sentences and labels for line in lines: line = line.strip() if '@' in line: sentence, label = line.rsplit('@', 1) sentences.append(sentence.strip()) true_labels.append(label.strip()) # Function to get predictions from the fine-tuned model def get_openai_predictions(sentence, model="your_finetuned_model_name"): # Replace with your model name try: response = openai.ChatCompletion.create( model=model, messages=[ {"role": "system", "content": "You are a financial sentiment analysis expert."}, {"role": "user", "content": sentence} ],
```

```
max_tokens=50, temperature=0.5 ) return response['choices'][0]['message']['content'].strip() except Exception as e: print(f"Error generating prediction for sentence: '{sentence}'. Error: {e}") return "unknown" # Generate predictions for the dataset predicted_labels = [] for sentence in sentences: prediction = get_openai_predictions(sentence) # Normalize the predictions to 'positive', 'neutral', 'negative' if 'positive' in prediction.lower(): predicted_labels.append('positive') elif 'neutral' in prediction.lower(): predicted_labels.append('neutral') elif 'negative' in prediction.lower(): predicted_labels.append('negative') else: predicted_labels.append('unknown') # Calculate the model's accuracy correct_count = sum([pred == true for pred, true in zip(predicted_labels, true_labels)]) accuracy = correct_count / len(sentences) print(f'Accuracy: {accuracy:.4f}') # Expected output: 0.7646
```

Conclusion

By combining the expertise of FinBERT's financial domain labels with the flexibility of GPT-4o mini, this project achieves a high-performance financial sentiment model that surpasses FinBERT in accuracy. This guide and methodology pave the way for replicable, scalable, and interpretable sentiment analysis, specifically tailored to the financial industry.

Key Takeaways

- Fine-tuning GPT-4o mini with domain-specific data enhances its ability to capture nuanced financial sentiment, outperforming models like FinBERT in accuracy.
- The TRC2 dataset, curated by Reuters, provides high-quality financial news articles for effective sentiment analysis training.
- Preprocessing and labeling with FinBERT enable GPT-4o mini to generate more accurate sentiment predictions for financial texts.
- The approach demonstrates the scalability of GPT-4o mini for real-world financial applications, offering a lightweight alternative to complex models.
- By leveraging OpenAI's fine-tuning API, this method optimizes GPT-4o mini for efficient and effective financial sentiment analysis.

Frequently Asked Questions

Q1. Why use GPT-4o mini instead of FinBERT for financial sentiment analysis?

A. GPT-4o mini provides a lightweight, flexible alternative and can outperform FinBERT on specific tasks with fine-tuning. By fine-tuning with domain-specific data, GPT-4o mini can capture nuanced sentiment patterns in financial texts while being more computationally efficient and easier to deploy.

Q2. How do I request access to the TRC2 dataset?

A. To access the TRC2 dataset, submit a request through the National Institute of Standards and Technology (NIST) at this link. Review the website's instructions to complete licensing and usage agreements, typically required for both research and commercial use.

Q3. Can I use a different dataset for financial sentiment analysis?

A. You can also use other datasets like the Financial PhraseBank or custom datasets containing labeled financial texts. The TRC2 dataset suits training sentiment models particularly well, as it includes financial news content and covers a wide range of financial topics.

Q4. How does FinBERT generate the sentiment labels?

A. FinBERT is a financial domain-specific language model that pre-trains on financial data and fine-tunes for sentiment analysis. When applied to the TRC2 sentences, it categorizes each sentence into Positive, Negative, or Neutral sentiment based on the language context in financial texts.

Q5. Why do we need to convert the labels to lowercase in JSONL?

A. Converting labels to lowercase ensures consistency with OpenAI's fine-tuning requirements, which often expect labels to be case-sensitive. It also helps prevent mismatches during evaluation and maintains a uniform structure in the JSONL dataset.

Article Url - https://www.analyticsvidhya.com/blog/2024/11/financial-sentiment-analysis/

Adarsh Balan

Hi! I'm Adarsh, a Business Analytics graduate from ISB, currently deep into research and exploring new frontiers. I'm super passionate about data science, AI, and all the innovative ways they can transform industries. Whether it's building models, working on data pipelines, or diving into machine learning, I love experimenting with the latest tech. AI isn't just my interest, it's where I see the future heading, and I'm always excited to be a part of that journey!