Opinion Mining Task Report

Panpan Zhang

October 13, 2024

Contents

1 Problem Understanding

This task focuses on Opinion Mining, which involves extracting opinions, sentiments, and specific entities from text data. The goal is to identify not only the sentiment (positive, neutral, negative) expressed in the text but also to recognize specific products or parts of products mentioned. This problem can be framed as a multi-output machine learning task combining: **Named Entity Recognition (NER)** and **Text Classification (Sentiment Analysis)**.

2 Data Research

To tackle the first task, we can use **GPT-3.5-turbo** to generate a dataset of at least 1000 sentences that are annotated with Named Entity Recognition (NER) tags of 'Product' and 'Part' as well as class labels like 'Positive', 'Neutral', and 'Negative'. I will use a **one-shot prompt engineering** approach to guide the model in generating structured output.

When generating the data, try to simulate real-world data as much as possible by adding elements such as **emojis, special characters, extra spaces**, etc. This will ensure that the generated data requires data cleaning operations.

Although there may be some bias in the annotated data, the majority of the annotations generated by the GPT model are accurate. Therefore, we can proceed with using this data directly.

3 Solution Design

This solution is designed to address the task of Opinion Mining by combining **Named Entity Recognition (NER)** and **Sentiment Classification** into a single model.

3.1 Data Collection

- Generate the dataset using **GPT-3.5-turbo** models.
- **One-Shot Prompt Engineering**

3.2 Data Preprocessing

- Remove unnecessary characters (e.g., special symbols, excessive spaces).
- Drop duplicated data.
- Remove invalid data.

3.3 Model Selection

- A BERT base model (**bert-base-uncased**) was used as the core.
- **Loss Functions**:
  - NER Loss: Use `CrossEntropyLoss` for token-level classification.

- Sentiment Loss: Use `CrossEntropyLoss` for sentence-level classification.
- The total loss is the **sum** of both losses, ensuring balanced learning between both tasks.

3.4 Training Strategy

- Split the dataset into 80% training and 20% testing.

3.5 Evaluation and Metrics

- **NER Task**:
  - **Precision**, **Recall**, and **F1-Score** for each label.
  - **Confusion matrix** to visualize misclassifications between the different NER tags.
- **Sentiment Classification Task**:
  - **Accuracy**, **Precision**, **Recall**, and **F1-Score** for the sentiment labels (Positive, Neutral, Negative).
  - Use a **confusion matrix** to understand common misclassifications between the sentiment categories.

4 Model Design

The task involves building a **multi-task learning model** for both Named Entity Recognition (NER) and sentiment classification using a pre-trained BERT model. It uses the framework of PyTorch. Here's how the model was constructed and trained:

4.1 Data Preparation:

- In the previous steps, it was explained that the dataset was generated using **GPT-3.5-turbo**, and the annotations were also produced by GPT-3.5-turbo.
- The dataset consists of sentences annotated with **NER tags** (Product, Part, or both) and **class labels** (Positive, Neutral, Negative).
- The sentences were tokenized using **BERT's tokenizer**, which converts them into input_ids (tokenized format), attention_masks, and corresponding labels for both NER and sentiment.
- The dataset was split into training and test sets (80% for training and 20% for testing).

4.2 Model Architecture:

- A BERT base model (**bert-base-uncased**) was used as the core. It outputs two types of embeddings:
  - **Sequence-level embeddings**: For token-based predictions (used for NER tagging).
  - **Pooled embeddings**: For sentence-level predictions (used for sentiment classification).
- Two custom classifiers were added:
  - **NER Classifier**: To predict the NER tags.
    - **Input**: The sequence-level embeddings from BERT (for each token in the input sentence).
    - **Layer**: A simple linear layer (`self.ner_classifier`), which takes the token-level embeddings from BERT and maps them to NER label predictions.
    - **Output**: The NER classifier outputs predictions for each token, classifying it as belonging to one of the NER classes.
  - **Sentiment Classifier**: To predict the sentiment class of each sentence.
    - **Input**: The pooled output from BERT, which is the representation of the entire sentence.
    - **Layer**: A linear layer (`self.sentiment_classifier`) that takes the pooled output and maps it to sentiment classes (Positive, Neutral, Negative).
    - **Output**: The sentiment classifier produces a single label for the entire sentence.
- Forward Pass:
  - **BERT Processing**: In the forward function, the input tokens are passed through the BERT model to obtain the contextual embeddings.
  - **NER Prediction**: The token-level embeddings (`sequence_output`) are passed through the NER classifier to predict the NER labels.
  - **Sentiment Prediction**: The sentence-level pooled output is passed through the sentiment classifier to predict the sentiment label.
- Loss Functions and Optimization:
  - **Loss Calculation**:
    - The NER loss and sentiment loss are computed separately using **cross-entropy loss**.
    - The combined loss is calculated by **summing** these two losses.
  - **Backpropagation**: The total loss is backpropagated, and the model weights are updated using the **AdamW optimizer**.

4.3 Training Process:

- The training loop used the **AdamW optimizer** and a batch size of 20.
- For each batch, the forward pass calculated two types of predictions:
  - **NER predictions** at the token level.
  - **Sentiment predictions** at the sentence level.
- The losses for NER and sentiment were computed separately using cross-entropy, then combined and minimized together.
- The model was trained for 10 epochs, updating the weights in each step based on the combined loss from both tasks.

4.4 Evaluation:

The evaluation framework is structured to assess both the **Named Entity Recognition (NER)** and **Sentiment Classification** tasks in a multi-task learning model. Below is the summary of the design:

- **Model Preparation**:
  - The model is switched to **evaluation mode** using `model.eval()` to ensure proper inference behavior. This disables training-specific behaviors like dropout and batch normalization.
- **Disable Gradient Calculation**:
  - The evaluation process runs under `torch.no_grad()` to prevent unnecessary gradient computations, optimizing both memory usage and computation time.
- **NER Evaluation**:
  - NER predictions are compared against true NER labels using **precision**, **recall**, and **F1-score**, computed via the `classification_report` from `sklearn.metrics`.
- **Sentiment Classification Evaluation**:
  - The sentiment predictions are evaluated by calculating **accuracy**, and a **weighted F1-score** is also computed to provide a balanced performance measure across all sentiment classes.
  - **Precision**, **recall**, and **F1-scores** for each sentiment class are calculated and printed.

5 Results

The model achieved an overall **accuracy** of 84.00% and an **F1-Score** of 84.79%, indicating solid performance across both tasks.

For the NER task, the model demonstrates high precision and recall for the "Product" label, while performance on the "Part" label is moderately strong with higher recall

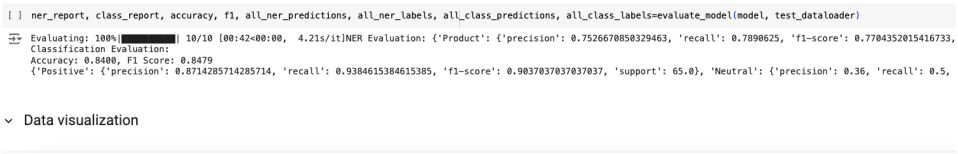

Figure 1: Model Performance with Accuracy and F1-Score

than precision. However, the model faces challenges distinguishing the "Product, Part" category, largely due to data imbalance, as reflected in the confusion matrix, where we observe frequent misclassifications between these labels.

On the sentiment classification side, the model performs well in identifying positive and negative sentiments, with precision and recall nearing 0.9. The neutral sentiment category, however, exhibits lower precision (˜0.45) and recall (˜0.55), which is impacted by the smaller volume of neutral-labeled data.

5.1 NER Classification Report

- From the graphical information in **Question 2**, we can observe the following NER Tag distribution:
  - The number of **Product, Part** labels is approximately 300.
  - The number of **Product** labels is around 200.
  - The **Part** label has the highest count, exceeding 500.

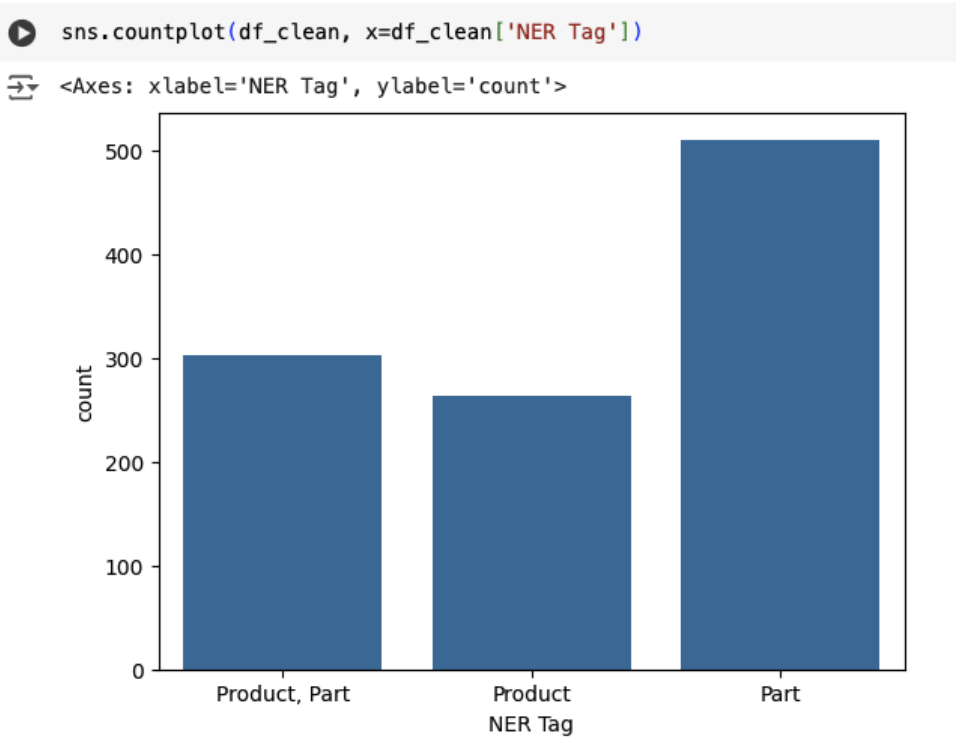

Figure 2: NER Tag distribution

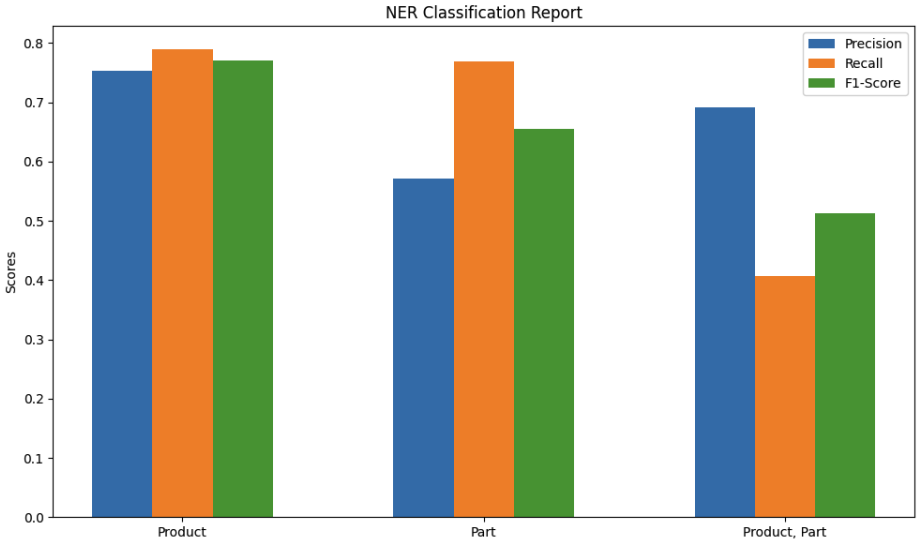

Figure 3: NER Classification Report

- The amount of data for the **Part** label is significantly higher than for the **Product** and **Product, Part** labels. This imbalance may affect the model's performance, particularly for categories with fewer data (such as **Product** and **Product, Part**), leading the model to more likely predict the **Part** category.
- The large quantity of **Part** labels likely explains why the model performs relatively well on this category in the NER task, with a high recall (approximately 0.75).
- On the other hand, the smaller data size for the **Product** and especially **Product, Part** categories may contribute to the lower recall and F1-Score for these categories.

5.2 Sentiment Classification Report

- From the graphical information in **Question 2**, we can observe the following sentiment class distribution:
  - The number of **Positive** labels is the largest, reaching approximately 600.
  - The number of **Negative** labels is around 400.
  - The **Neutral** label has the fewest instances, far fewer than both **Positive** and **Negative** labels, with only around 100.
- For the **Neutral** category:
  - **Precision**: Approximately 0.45, indicating that only 45% of the samples predicted as **Neutral** are actually correct.
  - **Recall**: Around 0.55, meaning the model is only able to identify about 55% of the actual **Neutral** samples, showing a relatively high miss rate.
  - **F1-Score**: About 0.5, suggesting that the model's overall performance on the **Neutral** category is relatively poor. This could be due to the data imbalance or difficulty distinguishing features of this category.

```
[ ] sns.countplot(df_clean, x=df_clean['Class'])
```

```
<Axes: xlabel='Class', ylabel='count'>
```

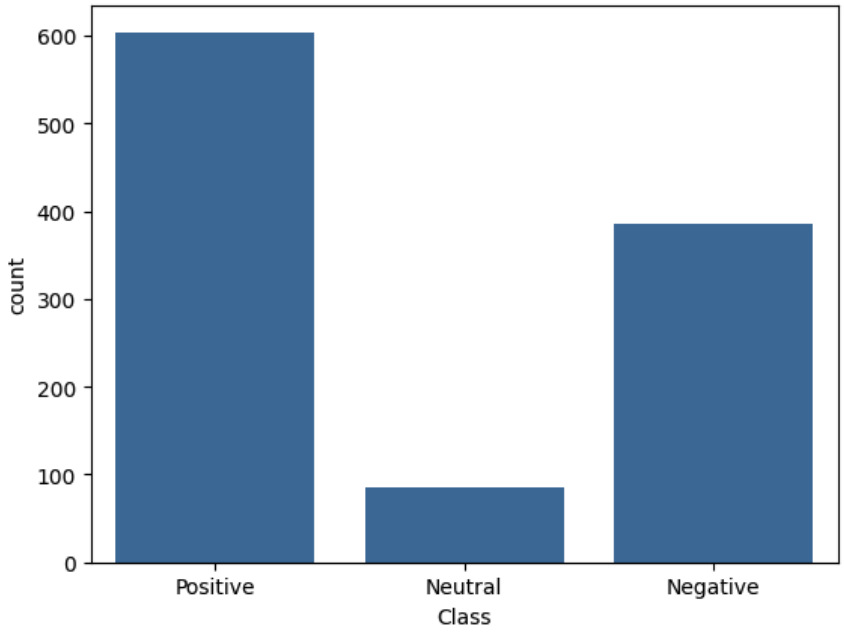

Figure 4: sentiment class distribution

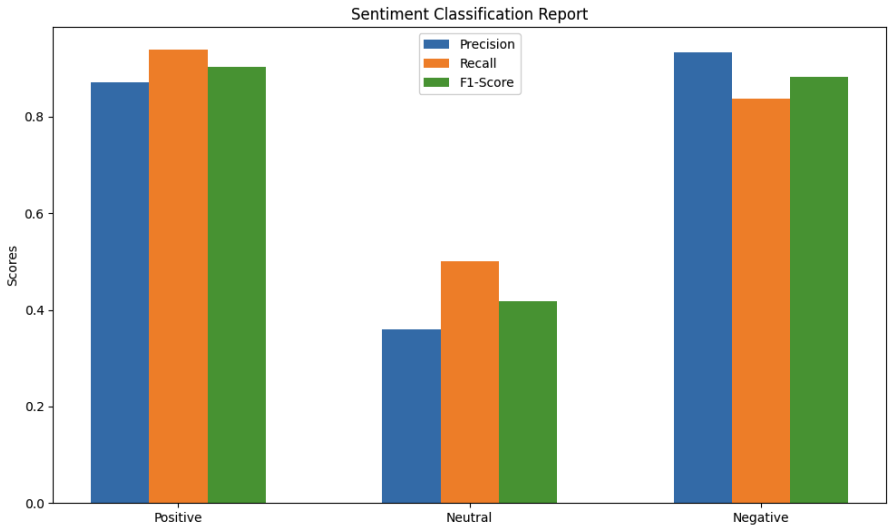

Figure 5: Sentiment Classification Report

5.3 NER Confusion Matrix

The model shows significant misclassification in the **Product, Part** category, with many **Product, Part** entities being incorrectly classified as either **Product** or **Part**. This suggests that the model may struggle to distinguish the features of multi-label entities like **Product, Part**.

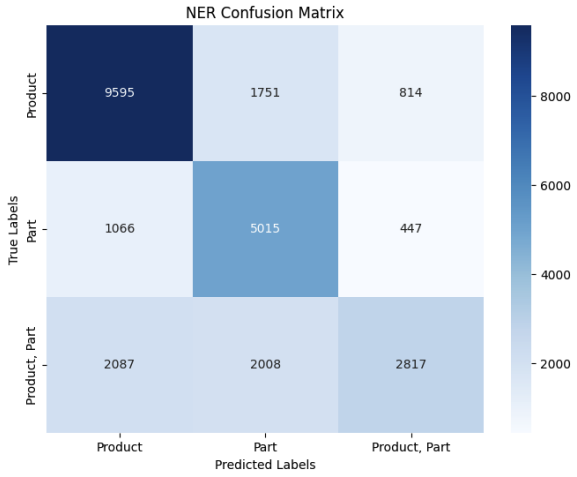

Figure 6: NER Confusion Matrix

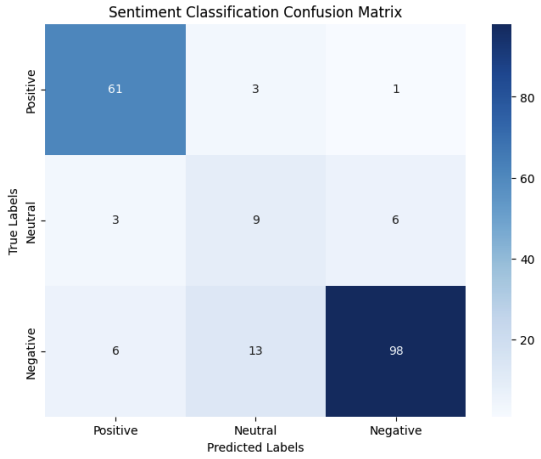

Figure 7: Sentiment Classification Confusion Matrix

5.4 Sentiment Classification Confusion Matrix

- **Classification Evaluation:**
  - **Accuracy**: 0.8400
    - An accuracy of 84% suggests that the model is performing reasonably well overall. However, it doesn't reflect the full picture, as accuracy can be misleading when the data is imbalanced. This is why we also need to consider the F1 score.
  - **F1 Score**: 0.8479
    - With an F1 score of nearly 0.85, we can conclude that the model is doing well in terms of both correctly identifying the relevant classes and minimizing incorrect predictions.
- The **Positive** category performed excellently: most **Positive** instances were correctly classified, and the misclassification rate was very low, indicating that the model performs well in recognizing **Positive** instances.

- The **Neutral** category performed poorly: this category had the most misclassifications, and the model struggled to distinguish the features of **Neutral** instances. Most **Neutral** instances were misclassified as **Positive** or **Negative**, which is primarily related to the smaller amount of data for this category.

5.5 Possible Pros and Cons of This Model

5.5.1 Pros

- **Multi-Task Learning**:

  The model performs both Named Entity Recognition (NER) and Sentiment Classification tasks simultaneously, leveraging shared knowledge from the BERT model for efficient learning of both tasks.

- **Efficient Training Process**:

  The model uses the AdamW optimizer with a learning rate of 5e-5, a commonly used optimizer that helps achieve faster convergence during training.

- **Balanced Loss Calculation**:

  The loss function handles both NER and sentiment classification with separate cross-entropy losses for each task, allowing the model to focus on minimizing errors across both tasks without imbalance.

5.5.2 Cons

- **Data Imbalance Issues**:

  The model's performance is negatively impacted by imbalanced data. Categories like **Neutral** in sentiment classification and **Product, Part** in NER have lower data representation, leading to lower accuracy and recall in these classes.

- **Limited Architecture Complexity**:

  While BERT provides strong contextual understanding, the custom classifier layers are simple linear classifiers.

- **Performance**:

  By examining detailed performance metrics like precision, recall, F1-score, accuracy, and confusion matrices, it is evident that the model's accuracy needs improvement, especially when dealing with imbalanced data.

6 Recommendations for Future Improvement

The model effectively uses BERT for multi-task learning, providing strong performance on most categories. However, it faces challenges with imbalanced data, multi-label classification, and potential limitations in its simple classifier architecture.

- **Address Data Imbalance**
  - Use **data augmentation** to increase the representation of underrepresented classes like **Neutral** in sentiment classification and **Product, Part** in NER.

  - Implement **class-weighting** in the loss function to give more importance to underrepresented classes, helping the model better handle imbalanced datasets.
  - The paper Debiasing Generative Named Entity Recognition by Calibrating Sequence Likelihood addresses bias in generative NER models. The key idea, **Sequence Likelihood Calibration**, redistributes probability mass across multiple candidate sequences rather than concentrating it on the most likely one. The likelihoods are adjusted based on performance metrics, such as F1-scores, using a reranking approach.
  - The paper DAGA: Data Augmentation with a Generation Approach for Low-resource Tagging Tasks introduces a novel data augmentation method for low-resource sequence tagging tasks. They propose a method that trains a language model on **linearized labeled sentences** (where tags are paired with corresponding words). This approach adds diversity to the training data, helping to mitigate overfitting and improving model performance.

- **Enhanced Feature Extraction for Multi-Label Entities**
  - Introduce **multi-head attention mechanisms** specifically designed to handle multi-label classification (e.g., **Product, Part**) to help the model better differentiate between overlapping entities.
  - In the paper titled An End-to-End Progressive Multi-Task Learning Framework for Medical Named Entity Recognition and Normalization, the authors present an **end-to-end progressive multi-task learning model**. The progressive tasks are designed to reduce error propagation through incremental task settings, which means that lower-level tasks receive supervisory signals—apart from errors—from higher-level tasks, leading to performance improvement.

- **Use Advanced Classifiers**
  - Replace the simple linear classifiers with more sophisticated layers like **bi-LSTM** or **CRF (Conditional Random Fields)** or **Multi-Head Attention Mechanism** for NER.
  - Replace the simple linear classifiers with more sophisticated layers like **bi-LSTM** for sentiment classification.
  - The paper Multilingual Part-of-Speech Tagging with Bidirectional Long Short-Term Memory Models and Auxiliary Loss indicates that the **auxiliary loss** is effective at improving the accuracy of rare words. Leveraging bi-LSTMs with auxiliary loss to handle data sparsity and improve rare word tagging could be highly beneficial for NLP models that deal with low-resource languages or domains with limited labeled data.

- **Training**
  - Implement **k-fold cross-validation** during model training to enhance the model's robustness and generalization.