Aspect-Based Product Review Analysis Engine

Technical Documentation

Supervisor: Prof. Ehab El-Shazly

TA: Eng. Esraa Abdelrazek

Date: June 19, 2025

Team Members:

Name	ID
Abdulrahman Mahmoud Riyad	120220193
Abanoub Samy Farhan	120220108
Abdallah Adel Abdallah	120220107
Shahd Ammar	120220098
Yasmeen Sameh	120220143

Contents

1 Introduction and Project Goal

The core objective of this project is to build an end-to-end system that automatically analyzes customer product reviews to provide actionable insights. The system is designed to identify specific product features (aspects), determine the sentiment expressed towards them, and generate a concise summary of pros and cons. This solves the real-world problem of information overload for consumers and provides a scalable feedback analysis tool for businesses.

This document outlines the development of the core machine learning component: a fine-tuned Transformer model for **joint Aspect-Based Sentiment Analysis (ABSA)**.

2 System Architecture and Tools

The project is designed with a microservices architecture to separate concerns and facilitate independent development and deployment.

- **ML Service (This Document):** A Python service built with **FastAPI** responsible for running the fine-tuned model and exposing its predictions via a REST API.
- **Orchestration Backend (Node.js):** A separate service (developed by a teammate) that handles requests from the client, calls the scraping utility, orchestrates calls to the ML Service API, and potentially calls another LLM for final summarization.
- **Client (Chrome Extension):** A browser extension that allows a user to trigger the analysis on a product page and view the final summary.

2.1 Key Libraries and Technologies

- **Python 3.11:** The primary language for the ML pipeline.
- **Pandas & NumPy:** For data loading and manipulation.
- **PyTorch:** The backend deep learning framework.
- **Hugging Face Transformers:** For accessing the pre-trained `bert-base-uncased` model, tokenizer, and the Trainer API for fine-tuning.
- **Hugging Face Datasets:** For efficient data handling and preprocessing.
- **Seqeval:** For sequence labeling evaluation (Precision, Recall, F1-score).
- **FastAPI & Uvicorn:** For building and serving the ML model as a REST API.
- **Hugging Face Hub:** Used for hosting the final fine-tuned model for easy access and deployment.
- **Docker:** For containerizing the FastAPI service for deployment on Hugging Face Spaces.

3 Phase 1: Data Preparation Pipeline (Steps 1-3)

3.1 Purpose

To load, clean, and transform the raw SemEval datasets (Laptops and Restaurants) into a structured format suitable for training a joint aspect-sentiment model. The goal is to produce a single, clean dataset where each row corresponds to a unique sentence and contains a list of all aspect terms and their associated polarities.

3.2 Key Operations

1. **Data Loading and Combination**: Two separate CSV files were loaded using pandas. A `domain` column ('laptop' or 'restaurant') was added to each before concatenating them into a single DataFrame. This provided more data diversity for training.
2. **Cleaning and Standardization**:
   - Column names were standardized to lowercase with underscores (e.g., "Aspect Term" → `aspect_term`).
   - Rows with missing essential data (aspect term, polarity, or character offsets) were dropped using `dropna()`.
   - Character offset columns (`from`, `to`) were converted to integers.
   - The 'conflict' sentiment was mapped to 'neutral' to simplify the label set for the token classification task, as a single token cannot easily represent conflicting sentiments.
3. **Aggregation and Conversion**:
   - A unique ID was created for each sentence by combining its domain and original ID (e.g., `laptop_123`) to resolve ID conflicts across the two datasets.
   - The data was transformed from a "one row per aspect" format to a "one row per unique sentence" format using `groupby('unique_id').apply()`. The output for each sentence contains a list of dictionaries, where each dictionary holds an aspect term, its polarity, and its character offsets.
   - The final aggregated pandas DataFrame was converted into a Hugging Face `Dataset` object for efficient processing in the next phase.

3.3 Output of Phase 1

The output is a Hugging Face `Dataset` object (`hf_dataset`). Each record in this dataset represents a unique sentence and has the following structure:

```
{
    'unique_id': 'laptop_100',
    'sentence': 'I had of course bought a 3 year warranty...',
    'aspects': [
        {'term': '3 year warranty', 'polarity': 'neutral', 'from': 25, 'to': 40}
    ],
    'domain': 'laptop'
}
```

Listing 1: Structure of a single record in hf_dataset

This structure is the direct input for the model tokenization and label alignment phase.

4 Phase 2: Model Configuration and Data Transformation (Steps 4-5)

4.1 Purpose

This phase prepares the model and transforms the aggregated text data into numerical tensors that the model can understand. This involves defining a rich BIO-Sentiment labeling scheme, aligning these labels with model-specific tokens, and defining the evaluation metrics.

4.2 Key Operations

1. **Labeling Scheme Definition**:
    - A joint Aspect-Sentiment BIO tagging scheme was defined in `src/config.py`. Instead of just "B-ASP", "I-ASP", "O", we created labels for each sentiment: "B-ASP-POS", "I-ASP-POS", "B-ASP-NEG", "I-ASP-NEG", "B-ASP-NEU", "I-ASP-NEU".
    - This results in a 7-class token classification problem, allowing the model to learn both boundary detection and sentiment simultaneously.
2. **Tokenizer and Config Loading**:
    - The pre-trained `bert-base-uncased` tokenizer was loaded using `AutoTokenizer`.
    - An `AutoConfig` instance was loaded and updated with our 7 custom labels and their corresponding ID mappings (`label2id`, `id2label`). This prepares the model to have a classification head with 7 outputs per token.
3. **Tokenization and Label Alignment**:
    - The `tokenize_and_align_labels` function (in `src/tokenization_utils.py`) was applied to the dataset using `.map()`.
    - This function tokenizes each sentence and, for each ground-truth aspect, it uses the character offsets (`from`, `to`) and `polarity` to assign the correct BIO-Sentiment label ID (e.g., `1` for `B-ASP-POS`) to the corresponding tokens.
    - Special tokens (`[CLS]`, `[SEP]`) and subsequent subword tokens are assigned the label `-100`, which is ignored by the loss function during training.
4. **Data Splitting and Collation**:
    - The final tokenized dataset was split into training (80%), validation (10%), and test (10%) sets.
    - A `DataCollatorForTokenClassification` was instantiated to handle dynamic padding of input batches during training.
5. **Evaluation Metric Definition**:
    - The `compute_absa_metrics` function was defined in `src/evaluation_utils.py`.
    - It uses the `seqeval` library to calculate entity-level Precision, Recall, and F1-score. It correctly handles the BIO-Sentiment scheme, reporting metrics for entities like "ASP-POS", "ASP-NEG", and an overall micro-averaged F1 score.

4.3 Output of Phase 2

- `dataset_splits`: A Hugging Face `DatasetDict` containing `train`, `validation`, and `test` splits, with each record tokenized and labeled, ready for training.
- `data_collator`: An object ready to be passed to the `Trainer`.
- `compute_metrics`: The function to be passed to the `Trainer` for evaluation.

5 Phase 3: Model Fine-Tuning and Evaluation (Steps 6-9)

5.1 Purpose

To configure the training process, instantiate the model and `Trainer`, execute the fine-tuning loop, and evaluate the final model's performance on unseen data.

5.2 Key Operations

1. **Configure Training Arguments (Step 6)**:
    - An instance of `TrainingArguments` was created in `src/train.py`.
    - Key hyperparameters were set from `config.py`: `num_train_epochs=3`, `learning_rate=2e-5`, `per_device_train_batch_size=16`, and `weight_decay=0.01`.
    - A minimal configuration was used due to environment inconsistencies encountered in Kaggle. `load_best_model_at_end` was set to `False`, and evaluation was planned as a post-training step.
2. **Instantiate Model and Trainer (Step 7)**:
    - The pre-trained `bert-base-uncased` model was loaded using `AutoModelForTokenClassification.from_pretrained()`, passing the modified `config` object. This attached a randomly initialized classification head with 7 outputs per token.
    - The model was moved to the available device (CPU for local run).
    - The `Trainer` was instantiated, bringing together all previously prepared components: the model, training arguments, datasets, data collator, tokenizer, and the `compute_metrics` function.
3. **Execute Training (Step 8)**:
    - The fine-tuning process was started by calling `trainer.train()`.
    - During this process, the `Trainer` managed the training loop, updating the model weights to minimize loss on the training data.
    - The training loss was observed to decrease steadily (from $\sim$0.48 to $\sim$0.24), indicating successful learning.
4. **Evaluate Performance (Step 9)**:
    - After training, `trainer.evaluate()` was called on both the validation and test sets.
    - The `compute_metrics` function was used to calculate performance.

5.3 Results and Outputs

The model demonstrates solid performance, particularly on positive aspects. The overall F1 score is lower than an aspect-only extraction model, which is expected given the increased complexity of the joint task. The low loss on both sets indicates good generalization.

5.3.1 Outputs

- `trainer`: The trained `Trainer` object holding the fine-tuned model.
- `eval_results` & `test_results`: Dictionaries containing the performance metrics.
- Saved model checkpoints and results in the `./saved_models/` directory.

Table 1: Model Performance on Unseen Data (3 Epochs)

Metric	Validation Set	Test Set
Overall F1-Score	0.5943	0.6166
Precision	0.5665	0.5904
Recall	0.6250	0.6453
Loss	0.2009	0.1801
F1 Score (Positive Aspect)	0.7141	0.6974
F1 Score (Negative Aspect)	0.5470	0.6250
F1 Score (Neutral Aspect)	0.2692	0.3896

6 Phase 4: Deployment and Inference (Step 10 and API)

6.1 Purpose

To package the fine-tuned model into a usable format, deploy it as a REST API service, and demonstrate its inference capabilities.

6.2 Key Operations

1. **Model Saving**:
   - The final fine-tuned model and its tokenizer were saved to a clean directory (`final_model_with_sentiment`) using `trainer.save_model()` and `tokenizer.save_pretrained()`.
2. **Model Hosting on Hugging Face Hub**:
   - The saved model directory was pushed to the Hugging Face Model Hub using `huggingface-hub` and Git LFS.
   - This provides a centralized, version-controlled location for the model, making it accessible for deployment. The model is available at: `huggingface.co/AbdulrahmanMahmoud007/bert-absa-reviews-analysis`.
3. **API Service with FastAPI**:
   - An API service was created using FastAPI (`src/ml_api_service/main.py`).
   - On startup, this service loads the fine-tuned BERT model and tokenizer directly from the Hugging Face Hub.
   - It exposes a `/analyze` POST endpoint that accepts a list of review strings.
   - For each review, it uses a `token-classification` pipeline to run inference and extract aspect terms along with their predicted sentiments (positive, negative, neutral).
   - The service returns a structured JSON response containing the per-review results.
4. **Deployment on Hugging Face Spaces**:
   - A `Dockerfile` was created to containerize the FastAPI application.
   - The application was deployed as a Docker Space on Hugging Face. The Space automatically builds the container from the `Dockerfile` and runs the API service.

- The deployed API is publicly accessible, allowing the project's Node.js backend and Chrome extension to interact with it. The live endpoint is located at: `abdulrahmanmahmoud007-absa-fastapi-service.hf.space/analyze`.

6.2.1 Outputs

- A fine-tuned model and tokenizer hosted on the Hugging Face Model Hub.
- A containerized FastAPI application.
- A live, deployed REST API endpoint capable of performing ABSA on new review texts.

7 Conclusion

This project successfully demonstrates the end-to-end process of fine-tuning a BERT model for the complex task of joint Aspect-Based Sentiment Analysis. The data preparation pipeline effectively cleaned and transformed the SemEval dataset, and the model achieved strong evaluation metrics, particularly for identifying positive and negative aspects. The successful deployment of this model as a REST API via FastAPI on Hugging Face Spaces provides a robust and scalable solution for integration with the project's other microservices.