Individual Final Project Report

NLP for Financial Consumer Protection

Name: Lasya Raghvendra
Group Members: Lasya Raghvendra, Vaijayanti Deshmukh, Amogh Ramagiri

1. Introduction

Project Overview
Our group project aimed to automate the classification of consumer complaints for the Consumer Financial Protection Bureau (CFPB). We utilized Natural Language Processing (NLP) to route unstructured complaint narratives into specific product categories (e.g., "Mortgage," "Credit Card"), thereby reducing manual triage costs.

Outline of Shared Work
The group collaboratively performed the initial data acquisition, cleaning, and Exploratory Data Analysis (EDA). We worked together to define the scope, selecting the specific "Product" target variable and handling the significant class imbalance present in the dataset.
Outline of Individual Contribution.

My specific contributions to this project are:

1. Implemented the fine-tuned DistilBERT classifier used as our high-accuracy model.
2. Designed and built the FastAPI backend, including endpoints for scraping, EDA output, baseline metrics, and transformer predictions.
3. Integrated the saved HuggingFace model into a production-ready API.
4. Implemented real-time transformer inference for our Next.js playground interface.
5. Coordinated model-to-frontend deployment, CORS configuration, and error handling.

This report explains the transformer approach, backend implementation, and key lessons from developing a deployable NLP system.

2. Transformer Modeling Approach

2.1 Model Selection and Rationale

We selected DistilBERT, a compact version of BERT, because it balances high accuracy with lower computational requirements. It preserves ~95% of BERT's performance while being 40% smaller and 60% faster, making it ideal for API deployment.

2.2 Data Preparation for Transformers

Unlike TF-IDF models that treat text as bags of words, transformers require tokenized inputs:

1. We used the DistilBertTokenizerFast to convert text into subword tokens.
2. Added CLS/SEP tokens automatically.
3. Truncated sequences >512 tokens.
4. Preserved casing consistency using the uncased model.

The dataset was split into:

1. 2.8 million training samples.
2. 707,000 validation samples.

2.3 Fine-Tuning Process

The model was fine-tuned using:

- Learning rate: 2e-5
- Batch size: 32
- Epochs: 3
- Warmup: 500 steps
- Weight decay: 0.01
- bf16 mixed-precision training

We optimized the cross-entropy loss across 19 product categories.

2.4 Model Performance Summary

From the training logs:

- Accuracy: 91.04%
- Macro F1: 0.6041
- Improved performance significantly compared to the Naive Bayes baseline (macro F1 = 0.3589). The transformer model handled long narratives and context-rich complaints more effectively than the baseline.

3. Backend Architecture (FastAPI)

3.1 API Design Overview

I built a modular FastAPI backend with clearly separated endpoints:

Endpoint	**Purpose**
/scrape	Downloads latest CFPB data (.zip) and extracts CSV
/eda/plots	Returns all EDA visualizations as JSON for frontend charts
/baseline/metrics	Serves baseline performance results
/transformer/predict	Returns DistilBERT predictions + confidence

/transformer/metrics	Displays transformer training summary

3.2 Loading and Serving the Transformer Model

HuggingFace models cannot be saved as .pkl, so we used the standard format:

```
models/distilbert/
  ├── pytorch_model.bin
  ├── config.json
  ├── tokenizer.json
  ├── vocab.txt
```

On server startup, the backend loads:

```
tokenizer = DistilBertTokenizerFast.from_pretrained("models/distilbert")
model = DistilBertForSequenceClassification.from_pretrained("models/distilbert")
model.eval()
```

This enables instant inference during frontend requests.

3.3 Inference Endpoint

The /transformer/predict endpoint:

1. Accepts text input (JSON)
2. Tokenizes and passes it through DistilBERT
3. Computes softmax probabilities
4. Selects the most likely label
5. Maps the integer class to a human-readable product (via a hard-coded dictionary)

This endpoint powers the interactive Playground page.

3.4 Scraper Integration

I implemented a robust web scraper that:

- Fetches the CFPB complaint page
- Locates the CSV download link via BeautifulSoup
- Downloads and extracts the ZIP file
- Saves data into data/input/ for EDA and modeling

The frontend uses a simulated progress bar, and the backend responds once data is fully downloaded.

4. Deployment & Full-Stack Integration

4.1 CORS and API Communication

A major challenge was communication between the Next.js frontend (localhost:3000) and the FastAPI backend (localhost:8000).

I configured CORS to allow safe cross-origin API requests.

4.2 EDA Visualization API

I transformed complex DataFrame outputs into JSON-friendly series, enabling Recharts (React) to render:

- Product class imbalance
- State-level complaint trends
- Submission methods
- Monthly complaint timelines
- Narrative length distributions
- Word count statistics

These form the backbone of our EDA dashboard in the frontend.

4.3 Playground Integration

I built the backend transformer API and connected it to a clean UI where users can enter a complaint and instantly see:

- Model prediction
- Confidence score
- Transformed text pipeline

This demonstrates real-world deployment feasibility.

5. Summary and Conclusions

5.1 Summary

My work focused on building the high-performance transformer model and implementing the complete backend architecture. The DistilBERT model outperformed the baseline significantly, proving the value of contextual language understanding in financial complaint classification.

The backend provides a production-ready API capable of:

- Serving predictions
- Delivering EDA results
- Exposing model metrics
- Integrating seamlessly with a modern frontend

5.2 Lessons Learned

- **Model Deployment:** Transformers require careful handling, not simple .pkl serialization. HuggingFace's save/load system is essential.
- **Backend Design:** Creating reliable FastAPI endpoints is critical for real-world NLP systems.
- **Data Handling:** Working with millions of rows requires incremental loading, efficient preprocessing, and memory-aware tokenization.
- **API + UI Integration:** Backend models only become useful when tied to a clean, responsive user interface.

6. Code Originality Calculation

Transformer Model Script:

- Total lines: ~180
- Taken from documentation/templates: ~50
- Custom fine-tuning logic + modifications: ~130
- Originality: ~72%

FastAPI Backend:

- Total backend lines: ~350
- Boilerplate (imports, Pydantic models): ~80
- Custom endpoints, scraper, transformer integration: ~270
- Originality: ~77%

Overall Estimated Originality: ~74–76%

7. References

1. **HuggingFace Transformers Documentation:** https://huggingface.co/docs/transformers/en/index
2. **FastAPI Documentation:** https://fastapi.tiangolo.com/
3. **CFPB Complaint Database:** https://www.consumerfinance.gov/data-research/consumer-complaints/
4. **PyTorch Documentation:** https://docs.pytorch.org/docs/stable/index.html