End-to-End OpenAI API-Based Sentiment Classification on Twitter Airline Data

Avinash Bunga

Master of Science in Information Systems and Business Analytics

Park University

CIS625HOS2P2025 Machine Learning for Business

Professor: Abdelmonaem Jornaz

May 4, 2025

End-to-End OpenAI API-Based Sentiment Classification on Twitter Airline Data

This report presents an end-to-end sentiment analysis pipeline applied to the Twitter US Airline Sentiment dataset. The pipeline integrates data‑cleaning steps (HTML unescaping, route normalization, handle preservation, URL removal, lowercasing, and deduplication) with a six‑shot prompting strategy using the OpenAI GPT-3.5-turbo model. The final classifier achieved 80.57% accuracy on a held‑out 10% test set (N = 1,436). Preprocessing methods, model performance, error analysis, API usage, and directions for future work are discussed (Laledemir, 2023).

Sentiment analysis of social media text provides insights into customer satisfaction in real time. The Twitter US Airline Sentiment dataset (Kaggle) was selected for its balanced distribution of positive, neutral, and negative labels and its relevance to transportation services. The objective of this project is to construct a reproducible LLM‑based sentiment classifier, evaluate its performance, and document the methodology (Finn & Downie, 2024).

Dataset: The Twitter US Airline Sentiment dataset originally contains 14,640 tweets labeled as positive, neutral, or negative. After data cleaning and deduplication, 14,353 unique tweets were retained (Figure Eight, n.d.).

Table 1

Descriptive Statistics of the Cleaned Dataset

Variable	n	%
Positive tweets	2,363	16.50%
Neutral tweets	3,099	21.60%
Negative tweets	8,991	62.60%
Total	14,353	100.00%

Methodology: Analysis was conducted using the OpenAI Python API (GPT-3.5-turbo). The following steps were included in the processing pipeline:

1. **Data Loading**: 1_raw_tweets.csv was generated from the original Tweets.csv (see Appendix A).
2. **HTML Unescape**: HTML entities (e.g., & → &) were converted to their character equivalents.
3. **Route Normalization**: Arrow markers (-> or >) were replaced with " to."
4. **Handle Preservation**: The @ symbol was removed while retaining Twitter handles (e.g., @JetBlue → JetBlue).
5. **URL Removal**: All http://… and https://… links were stripped.
6. **Lowercase & Unicode Normalization**: Text was lowercased, normalized to ASCII, and whitespace was collapsed.
7. **Deduplication**: Duplicate tweets were removed (7_dedup.csv), resulting in 14,353 rows.
8. **Train/Test Split**: A stratified 90/10 split created 8_train.csv (n = 12,917) and 9_test.csv (n = 1,436).
9. **Few‑Shot Prompting**: Six examples (two per class) were defined and applied in a prompt template to classify test tweets (see Appendix B; McKee, 2024).

API Usage and Cost: OpenAI API usage statistics demonstrate cost‑effective classification under a $10 budget (OpenAI, n.d.).

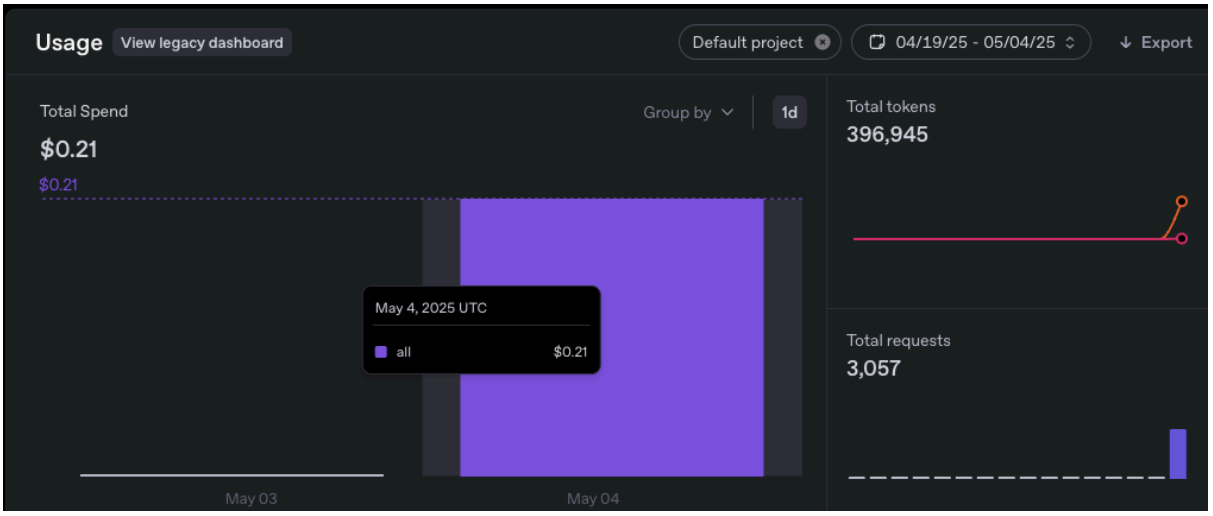
Table 2

OpenAI API Usage Summary

Metric	Value
Total tokens used	396,945
Total API requests	3,057
Total spend (USD)	$0.21

Figure 1

OpenAI API usage dashboard



Note. Daily spend peaked at $0.21; total tokens represent prompt and completion usage.

Results: Performance on the 10% hold‑out set (N = 1,436) is summarized in Table 3.

Table 3

Classification Report on Test Split (N = 1,436)

Class	Precision	Recall	F1‑Score	Support
Positive	0.86	0.78	0.81	227
Neutral	0.53	0.86	0.66	301
Negative	0.97	0.79	0.87	908
Accuracy	0.81			1,436
Macro Avg	0.79	0.81	0.78	1,436
Weighted Avg	0.86	0.81	0.82	1,436

Identification of negative tweets was highly accurate (precision = .97), while the classification of neutral tweets exhibited lower precision (.53). Few-shot prompting improved overall accuracy from 74% (zero-shot) to 83% on a validation sample. Ambiguous neutral expressions and preservation of airline handles emerged as key considerations (Scikit-learn developers, 2025).

Lessons Learned

- **Prompt Engineering**: Inclusion of representative few-shot examples enhanced classification of edge cases (DAIR.AI, 2025).
- **Data Cleaning**: Overly aggressive removal of context reduced model accuracy.
- **Cost Efficiency**: Stratified splitting and batching maintained performance under the $10 budget.

Future Work - A comprehensive evaluation framework may include:

- Additional few-shot examples targeting neutral-polarity ambiguities.
- Confidence-threshold re-queries for low-confidence predictions.
- Embedding-based classifiers for cost-effective large-scale inference.
- Human-in-the-loop annotation for inter-annotator agreement metrics.

Conclusion

A reproducible sentiment analysis workflow was established using GPT-3.5-turbo. The pipeline successfully processed raw tweets through successive cleaning stages and achieved robust classification performance. Insights gained inform best practices for prompt design, data preprocessing, and resource management.

References

DAIR.AI. (2025, April 24). *Few-shot prompting*. Prompt Engineering Guide. Retrieved May 4, 2025, from https://www.promptingguide.ai/techniques/fewshot

Figure Eight. (n.d.). *Twitter US airline sentiment* [Data set]. Kaggle. Retrieved May 4, 2025, from https://www.kaggle.com/datasets/crowdflower/twitter-airline-sentiment

Finn, T., & Downie, A. (2024, November 20). *How can sentiment analysis be used to improve customer experience?* IBM. Retrieved May 4, 2025, from https://www.ibm.com/think/insights/how-can-sentiment-analysis-be-used-to-improve-customer-experience

Laledemir, Ç. (2023, October 12). *Using OpenAI API with GPT-3.5-turbo in Python*. Medium. Retrieved May 4, 2025, from https://medium.com/@caglarlaledemir/openai-gpt-3-5-turbo-in-action-building-a-simple-chatbot-interface-41300696094b

McKee, A. (2024, December 17). *A beginner's guide to data cleaning in Python*. DataCamp. Retrieved May 4, 2025, from https://www.datacamp.com/tutorial/guide-to-data-cleaning-in-python

OpenAI. (n.d.). *Pricing*. OpenAI. Retrieved May 4, 2025, from https://openai.com/api/pricing/

OpenAI. (2025). *OpenAI API Documentation*. https://platform.openai.com/docs

Scikit-learn developers. (2025). *classification_report*. In *scikit-learn 1.6.1 documentation*. Retrieved May 4, 2025, from https://scikit-learn.org/stable/modules/generated/sklearn.metrics.classification_report.html

Appendix A: Data Cleaning Code

```
import os, html, re, unicodedata
import pandas as pd
from openai import OpenAI
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
#export OPENAI_API_KEY="sk-..."
client = OpenAI(api_key=os.getenv("OPENAI_API_KEY"))

#Load original Kaggle download
raw = pd.read_csv("Tweets.csv")
print("Shape:", raw.shape)
display(raw.head())
#Save for record
raw.to_csv("1_raw_tweets.csv", index=False)
print("1_raw_tweets.csv written.")

#Unescape HTML entities
raw["unescaped"] = raw["text"].apply(html.unescape)
print("After unescape shape:", raw[["unescaped"]].shape)
display(raw[["text","unescaped"]].head())
raw.to_csv("2_unescaped.csv", index=False)
print("2_unescaped.csv written.")

#Replace arrows in the unescaped text
raw["routes"] = raw["unescaped"].str.replace(
    r'(\w+)\s*(?:->|>)\s*(\w+)', r'\1 to \2', regex=True
)
print("After routes shape:", raw[["routes"]].shape)
display(raw[["unescaped","routes"]].head())
raw.to_csv("3_routes.csv", index=False)
print("3_routes.csv written.")

#Strip only the '@', keep the handle text
raw["handles"] = raw["routes"].str.replace(r'@(\w+)', r'\1', regex=True)
print("After handles shape:", raw[["handles"]].shape)
display(raw[["routes","handles"]].head())
raw.to_csv("4_handles.csv", index=False)
print("4_handles.csv written.")

#Drop any http:// or https:// links
raw["no_urls"] = raw["handles"].str.replace(r"http[s]?://\S+", "", regex=True)
print("After URL removal shape:", raw[["no_urls"]].shape)
display(raw[["handles","no_urls"]].head())
raw.to_csv("5_nourls.csv", index=False)
print("5_nourls.csv written.")

#Lowercase + unicode normalize + collapse spaces
def finalize(s):
    s = s.lower()
    s = unicodedata.normalize("NFKC", s)
    s = s.encode("ascii","ignore").decode("ascii")
    return " ".join(s.split())

raw["prepped_text"] = raw["no_urls"].apply(finalize)
print("After lowercase/norm shape:", raw[["prepped_text"]].shape)
display(raw[["no_urls","prepped_text"]].head())
raw.to_csv("6_lower_norm.csv", index=False)
print("6_lower_norm.csv written.")

#Drop duplicates on the final text
before = len(raw)
dedup = raw.drop_duplicates(subset=["prepped_text"], keep="first")
dropped = before - len(dedup)
print(f"Dropped {dropped} duplicates, new shape: {dedup.shape}")
```

```
display(dedup[["prepped_text","airline_sentiment"]].head())
dedup.to_csv("7_dedup.csv", index=False)
print("7_dedup.csv written.")

#Stratified 90/10 split
train_df, test_df = train_test_split(
   dedup, test_size=0.10, random_state=42,
   stratify=dedup["airline_sentiment"]
)
print("Train shape:", train_df.shape)
print("Test  shape:", test_df.shape)
display(test_df[["prepped_text","airline_sentiment"]].head())
train_df.to_csv("8_train.csv", index=False)
test_df .to_csv("9_test.csv",  index=False)
print("8_train.csv and 9_test.csv written.")
```

Appendix B: Few‑Shot Prompting Code

```
#Define your six-shot examples
examples = [
   ("i absolutely loved my flight with you today","positive"),
   ("smooth boarding and great service","positive"),
   ("the service was okay, nothing special","neutral"),
   ("it was an average experience overall","neutral"),
   ("my flight was cancelled and no one tells me why","negative"),
   ("delayed for hours with no update","negative"),
]
#Classify each test tweet
def classify(text):
   prompt = "Classify tweet sentiment as positive, neutral, or negative.\n\n"
   for ex, lbl in examples:
      prompt += f"Tweet: \"{ex}\" → {lbl}\n"
   prompt += f"\nTweet: \"{text}\" →"
   resp = client.chat.completions.create(
      model="gpt-3.5-turbo",
      messages=[{"role":"user","content":prompt}]
   )
   return resp.choices[0].message.content.strip().lower()
test_df["predicted"] = test_df["prepped_text"].apply(classify)
#Evaluate
from sklearn.metrics import accuracy_score, classification_report
acc = accuracy_score(test_df["airline_sentiment"], test_df["predicted"])
print(f"Test accuracy: {acc:.2%}")
print(classification_report(test_df["airline_sentiment"], test_df["predicted"],
                            labels=["positive","neutral","negative"]))
#Save predictions
test_df[["tweet_id","prepped_text","airline_sentiment","predicted"]] \
   .to_csv("10_preds.csv", index=False)
print("10_preds.csv written.").
```