A REPORT ON THE PROJECT

E-Commerce Operations with Automated Sentiment Analysis

GENERATIVE AI

AP23110010033-M.PRERANA REDDY

AP23110010581-K.SRAVYA

AP23110010584-B.KARTHIKEYA

AP23110010427-N.SRINITHYA

Project Description:

Elevate Retail Solutions, an e-commerce platform specializing in consumer electronics, receives thousands of product reviews every day. Currently, feedback analysis is performed manually, which is slow, inaccurate, and inefficient. Because of these limitations, critical insights often get delayed, affecting customer satisfaction, product improvements, and service quality.

To solve this, the project proposes building a **web-based automated sentiment analysis application** using **Support Vector Machine (SVM)**. The web app will allow real-time sentiment classification of product reviews, helping the company quickly identify customer opinions— positive, negative, or neutral.

Unlike a standalone Python program, this system will run as a **fully interactive website**, enabling users to upload review datasets, submit individual reviews, visualize results, and generate insights instantly. The web interface will make the system accessible to non-technical staff such as customer service teams, product managers, and business analysts.

Project Scenarios:

Scenario 1: Customer Support Monitoring

The customer support team uses the web-based sentiment analysis app to upload daily product reviews. The system quickly classifies them into positive, negative, and neutral groups, helping the team detect sudden spikes in negative feedback. This allows them to address customer issues faster and reduce escalation rates.

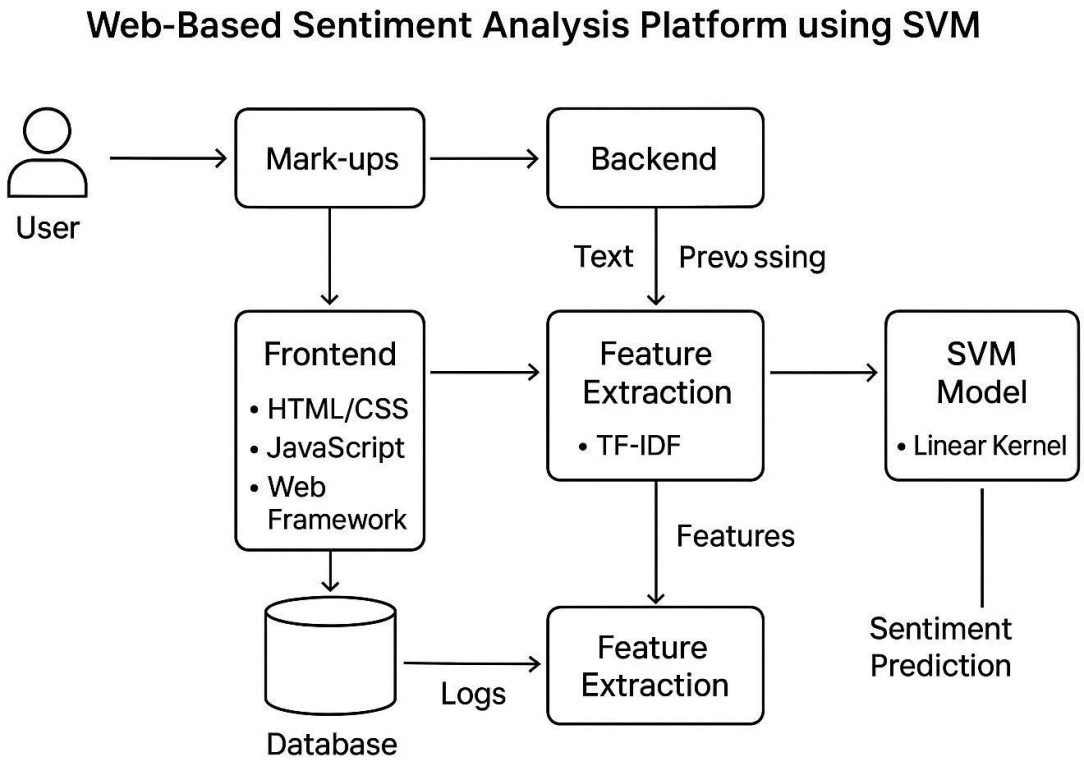
Scenario 2: Product Manager Review Insights

Product managers log into the web platform to study review trends before launching new product versions. By analyzing sentiment charts, they identify what customers liked or disliked in earlier models, enabling data-driven improvements for upcoming releases.

Scenario 3: Marketing Campaign Feedback

The marketing team uses the web app's manual input feature to instantly evaluate customer reactions to recent promotional campaigns. Positive and negative sentiments are displayed in real time, helping them refine marketing strategies and address campaign-related concerns quickly.

Technical Diagram:



Prerequisites:

- VS Code or Anaconda Navigator.
- Python packages required are:
  - numpy
  - pandas
  - scikit-learn
  - nltk
  - flask

Prior Knowledge:

You must have the prior knowledge of the following topics before completing this project.

- Python Programming
- Machine Learning Fundamentals(Supervised learning, features and labels etc)
- NLP Basics
- SVM Basics
- Web Development Fundamentals
- Python ML Libraries

Project Flow:

- User Interaction
- Backend Request Handling
- Text preprocessing
- Feature Extraction
- Model Prediction
- Response sent back to frontend.

Project Activities:

- Requirement Gathering & Understanding
    - Identify the need for automated sentiment analysis.
    - Understand user roles (customer support, product managers, etc.).
    - Decide the scope: web application + ML model.
- Dataset Collection & Preparation
    - Gather product reviews dataset (CSV or simulated).
    - Inspect dataset structure and sentiment labels.
    - Clean incorrect or missing entries.
- Text Preprocessing
    - Convert reviews to lowercase.
    - Remove punctuation, numbers, special characters.
    - Remove stopwords using NLTK.
    - Perform tokenization / optional stemming.
    - Prepare cleaned text for feature extraction.
- Feature Extraction
    - Apply **TF-IDF (TfidfVectorizer)** to transform text into numerical format.
    - Generate TF-IDF matrix for training the model.

- Model Development
  - Split dataset into train-test sets.
  - Train **SVM (Support Vector Machine)** classifier with linear kernel.
  - Evaluate the model using accuracy, precision, recall, and confusion matrix.
- Backend API Development

  (Flask) Build Flask routes

  for:

  - Homepage
  - Review input
  - CSV upload
  - Prediction

  Load trained model and vectorizer during startup.

  Format predictions in JSON or HTML response.

- Frontend Development
  - Design UI using HTML, CSS, and optionally Bootstrap.
  - Create forms for manual review input.
  - Create upload section for CSV datasets.
  - Display predictions clearly in the UI.
- Integration of ML Model with Web Interface
  - Connect frontend forms to backend endpoints.
  - Send user reviews to backend for processing.
  - Display predicted sentiment in the browser.
- Testing
  - Test UI functionality (manual/form input, file upload).
  - Test backend responses and error handling.
  - Test model predictions for accuracy and stability.
  - Perform browser compatibility tests.

Data Collection:

Data collection is fundamental to machine learning, providing the raw material for training algorithms and making predictions. This process involves gathering relevant information from various sources such as databases, surveys, sensors, and web scraping. The quality, quantity, and diversity of collected data significantly impact the performance and accuracy of ML models.

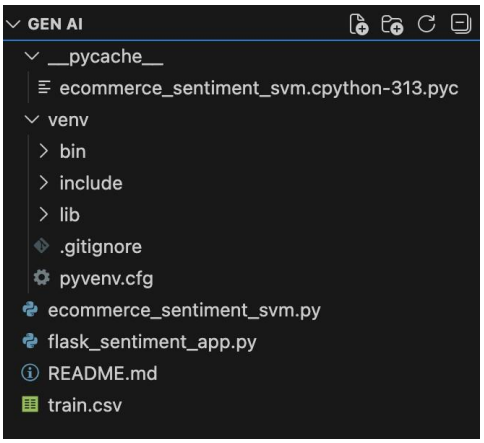
There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository, etc.

In this project, we have used .csv data. This data is downloaded from kaggle.com. Please refer to

the link given below to download the dataset.
Link: Data Set

Project Structure:

Below is the project structure of our project:



Project Structure Explanation:

This structure represents a clean separation between:

1. **Machine Learning Logic** → ecommerce_sentiment_svm.py
2. **Web App Logic** → flask_sentiment_app.py
3. **Data** → train.csv
4. **Environment** → venv/
5. **Compiled Python cache** → __pycache__/

Model Deployment:

Frontend Implementation:

This design combines **ML + Flask + HTML/CSS** in a single, clean workflow.

flask_sentiment_app.py

```python
"""
Flask front end for the Elevate Retail sentiment analysis demo.

Run with:
    cd /Users/pavan/python
    source .venv/bin/activate
    flask --app flask_sentiment_app run --reload
"""

from __future__ import annotations

from pathlib import Path

from flask import Flask, render_template_string, request

from ecommerce_sentiment_svm import (
    ASPECT_KEYWORDS,
    build_vectorizer,
    ensure_stopwords,
    load_dataset,
    prepare_features,
    train_svm,
)

DATA_PATH = Path(__file__).with_name("data").joinpath("ecommerce_reviews.csv")

# Train the TF-IDF + SVM pipeline once at startup so requests just reuse it.
# Using smaller sample (2000 per class) for faster Flask startup
_dataset = load_dataset(DATA_PATH, max_per_class=2000)
_stop_words = ensure_stopwords()
_vectorizer = build_vectorizer(_stop_words)
_X_train, _X_test, _y_train, _y_test = prepare_features(_dataset, _vectorizer)
_svm_model = train_svm(_X_train, _y_train)

app = Flask(__name__)
```

```
PAGE_TEMPLATE = """
<!doctype html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>E-Commerce Sentiment Analysis | Elevate Retail Solutions</title>
  <style>
    * { box-sizing: border-box; margin: 0; padding: 0; }
    body {
      font-family: -apple-system, BlinkMacSystemFont, 'Segoe UI', Roboto, 'Helvetica Neue', Arial,
      background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
      min-height: 100vh;
      padding: 2rem 1rem;
      color: #333;
    }
    .container {
      max-width: 800px;
      margin: 0 auto;
      background: white;
      border-radius: 16px;
      box-shadow: 0 20px 60px rgba(0,0,0,0.3);
      overflow: hidden;
    }
    .header {
      background: linear-gradient(135deg, #667eea 0%, #764ba2 100%);
      color: white;
      padding: 2rem;
      text-align: center;
    }
    .header h1 {
      font-size: 2rem;
      margin-bottom: 0.5rem;
      font-weight: 700;
    }
    .header p {
      opacity: 0.9;
      font-size: 1rem;
    }
```

```
@app.route("/", methods=["GET", "POST"])
def index():
    review_text = ""
    prediction = None
    aspects_triggered = []

    if request.method == "POST":
        review_text = request.form.get("review", "").strip()
        if review_text:
            features = _vectorizer.transform([review_text])
            prediction = _svm_model.predict(features)[0]
            lowered = review_text.lower()
            aspects_triggered = [
                aspect
                for aspect, keywords in ASPECT_KEYWORDS.items()
                if any(word in lowered for word in keywords)
            ]

    return render_template_string(
        PAGE_TEMPLATE,
        review=review_text,
        prediction=prediction,
        aspects=aspects_triggered,
        dataset_size=len(_dataset.reviews),
    )


if __name__ == "__main__":
    app.run(debug=True)
```

Our frontend file performs:

- Loads and trains the SVM model at startup.
- Provides a web interface for entering product reviews.
- Sends the user input to the backend for processing.
- Uses TF-IDF to convert text into features.
- Predicts sentiment using the trained SVM model.
- Detects product-related aspects present in the review.
- Displays beautifully formatted results back to the user.

Backend Implementation:

This module contains **all core machine learning logic** of your project. It loads datasets, preprocesses text, extracts TF-IDF features, trains an SVM classifier, evaluates performance, and provides utilities to power the Flask web application.

ecommerce_sentiment_svm.py

ecommerce_sentiment_svm.py > ...

```python
"""
Automated sentiment analysis for Elevate Retail Solutions.

This script simulates a small e-commerce review dataset, performs text
preprocessing, extracts TF-IDF features, trains a linear SVM model, and
evaluates its performance. Run with `python ecommerce_sentiment_svm.py`.
"""

from __future__ import annotations

import re
from dataclasses import dataclass
from pathlib import Path
from typing import Dict, List, Optional, Tuple

import numpy as np
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC

import nltk
from nltk.corpus import stopwords


@dataclass
class SentimentDataset:
    """Container for review text and labels."""

    reviews: pd.Series
    labels: pd.Series


DATA_DIR = Path(__file__).with_name("data")
CURATED_DATASET = DATA_DIR.joinpath("ecommerce_reviews.csv")
# Updated: train.csv is now in the same directory as the script
EXTERNAL_DATASET = Path(__file__).with_name("train.csv")
```

```python

def load_external_reviews(
    csv_path: Path, max_per_class: Optional[int] = 10000
) -> pd.DataFrame:
    """Load Amazon-style CSV with rating/title/review into review/sentiment columns."""
    # Optimize: Read in chunks and sample early to avoid loading entire file
    sentiment_map = {1: "Negative", 2: "Positive"}
    chunk_size = 50000
    collected = {sent: [] for sent in sentiment_map.values()}
    target_per_class = max_per_class if max_per_class else float('inf')

    print(f"Loading dataset in chunks (max {max_per_class} per class)...")
    for chunk in pd.read_csv(
        csv_path,
        header=None,
        names=["rating", "title", "review"],
        chunksize=chunk_size,
        low_memory=False
    ):
        chunk = chunk[chunk["rating"].isin(sentiment_map)].dropna(subset=["review"])
        if len(chunk) == 0:
            continue
        chunk["sentiment"] = chunk["rating"].map(sentiment_map)
        chunk["review"] = (
            chunk["title"].fillna("").astype(str).str.strip() + " " + chunk["review"].ast
        ).str.strip()

        # Collect samples per sentiment
        for sent, group in chunk.groupby("sentiment"):
            current_count = sum(len(df) for df in collected[sent])
            if current_count < target_per_class:
                needed = int(target_per_class - current_count)
                collected[sent].append(group.head(needed))

        # Early exit if we have enough samples
        if max_per_class and all(
            sum(len(df) for df in collected[sent]) >= target_per_class
            for sent in sentiment_map.values()
        ):
```

```
        # Early exit if we have enough samples
        if max_per_class and all(
            sum(len(df) for df in collected[sent]) >= target_per_class
            for sent in sentiment_map.values()
        ):
            break

    # Combine collected samples
    frames = []
    for sent in sentiment_map.values():
        if collected[sent]:
            combined = pd.concat(collected[sent], ignore_index=True)
            if max_per_class and len(combined) > max_per_class:
                combined = combined.sample(max_per_class, random_state=42)
            frames.append(combined)

    df = pd.concat(frames, ignore_index=True) if frames else pd.DataFrame(columns=["review", "
    return df[["review", "sentiment"]]


def load_curated_reviews(csv_path: Path) -> pd.DataFrame:
    if not csv_path.exists():
        raise FileNotFoundError(
            f"Expected curated dataset at {csv_path}. Please recreate it."
        )
    df = pd.read_csv(csv_path)
    required_cols = {"review", "sentiment"}
    if not required_cols.issubset(df.columns):
        raise ValueError(f"{csv_path} must contain columns {required_cols}")
    return df[["review", "sentiment"]]


def load_dataset(csv_path: Path, max_per_class: Optional[int] = 10000) -> SentimentDataset:
    """Load reviews/sentiment columns from curated + external CSVs."""
    frames: List[pd.DataFrame] = []

    if EXTERNAL_DATASET.exists():
        print(f"Loading external dataset from {EXTERNAL_DATASET} ...")
        frames.append(load_external_reviews(EXTERNAL_DATASET, max_per_class))
```

```
    else:
        print(f"No external dataset found at {EXTERNAL_DATASET}.")

    if csv_path.exists():
        frames.append(load_curated_reviews(csv_path))

    if not frames:
        raise FileNotFoundError(
            f"No datasets found. Please provide either:\n"
            f"  - External dataset at {EXTERNAL_DATASET}\n"
            f"  - Curated dataset at {csv_path}"
        )

    df = pd.concat(frames, ignore_index=True)
    return SentimentDataset(reviews=df["review"], labels=df["sentiment"])


def clean_text(text: str) -> str:
    """Light text cleanup: lowercase, remove non-word chars, collapse spaces."""
    text = text.lower()
    text = re.sub(r"[^a-z0-9\s]", " ", text)
    text = re.sub(r"\s+", " ", text).strip()
    return text


def build_vectorizer(stop_words: list[str]) -> TfidfVectorizer:
    """Create and configure the TF-IDF vectorizer."""
    return TfidfVectorizer(
        preprocessor=clean_text,
        stop_words=stop_words,
        ngram_range=(1, 2),
        min_df=1,
    )
```

```python
def prepare_features(
    dataset: SentimentDataset,
    vectorizer: TfidfVectorizer,
    test_size: float = 0.3,
    random_state: int = 42,
) -> Tuple[np.ndarray, np.ndarray, pd.Series, pd.Series]:
    """Split data and transform text into TF-IDF matrices."""
    X_train_text, X_test_text, y_train, y_test = train_test_split(
        dataset.reviews,
        dataset.labels,
        test_size=test_size,
        stratify=dataset.labels,
        random_state=random_state,
    )
    vectorizer.fit(X_train_text)
    X_train = vectorizer.transform(X_train_text)
    X_test = vectorizer.transform(X_test_text)
    return X_train, X_test, y_train.reset_index(drop=True), y_test.reset_index(drop=True)


def train_svm(X_train: np.ndarray, y_train: pd.Series) -> SVC:
    """Train a linear-kernel SVM classifier."""
    model = SVC(kernel="linear", probability=False, random_state=42)
    model.fit(X_train, y_train)
    return model


def evaluate_model(model: SVC, X_test: np.ndarray, y_test: pd.Series) -> None:
    """Print evaluation metrics for the trained model."""
    predictions = model.predict(X_test)
    accuracy = accuracy_score(y_test, predictions)
    report = classification_report(y_test, predictions, digits=3)
    conf_matrix = confusion_matrix(y_test, predictions, labels=["Positive", "Negative", "Neutral"]
```

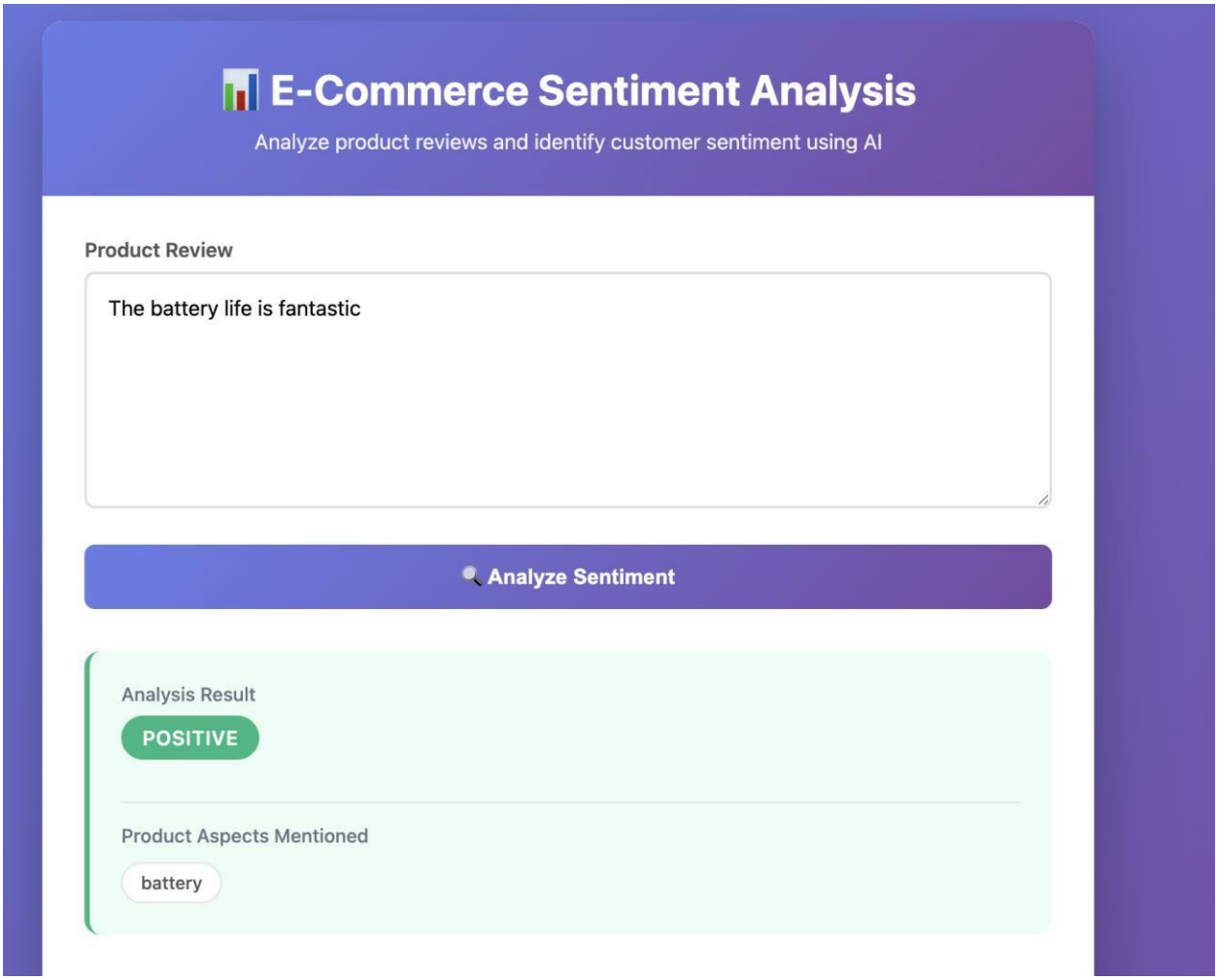
```
    print("\nModel Evaluation")
    print("-----------------")
    print(f"Accuracy: {accuracy:.3f}")
    print("\nClassification Report:")
    print(report)
    print("Confusion Matrix (rows=true, cols=pred):")
    print(
        pd.DataFrame(
            conf_matrix,
            index=["Positive", "Negative", "Neutral"],
            columns=["Positive", "Negative", "Neutral"],
        )
    )


ASPECT_KEYWORDS: Dict[str, List[str]] = {
    "battery": ["battery", "charge", "charging", "charger", "power"],
    "performance": ["performance", "speed", "lag", "slow", "fast", "processor"],
    "shipping": ["shipping", "delivery", "courier", "logistics", "package"],
    "design": ["design", "build", "look", "feel", "style"],
    "audio": ["audio", "sound", "speaker", "noise", "headphone"],
}


def ensure_stopwords() -> list[str]:
    """Download NLTK stopwords once and return the English stopword list."""
    try:
        nltk.data.find("corpora/stopwords")
    except LookupError:
        nltk.download("stopwords")
    return stopwords.words("english")
```

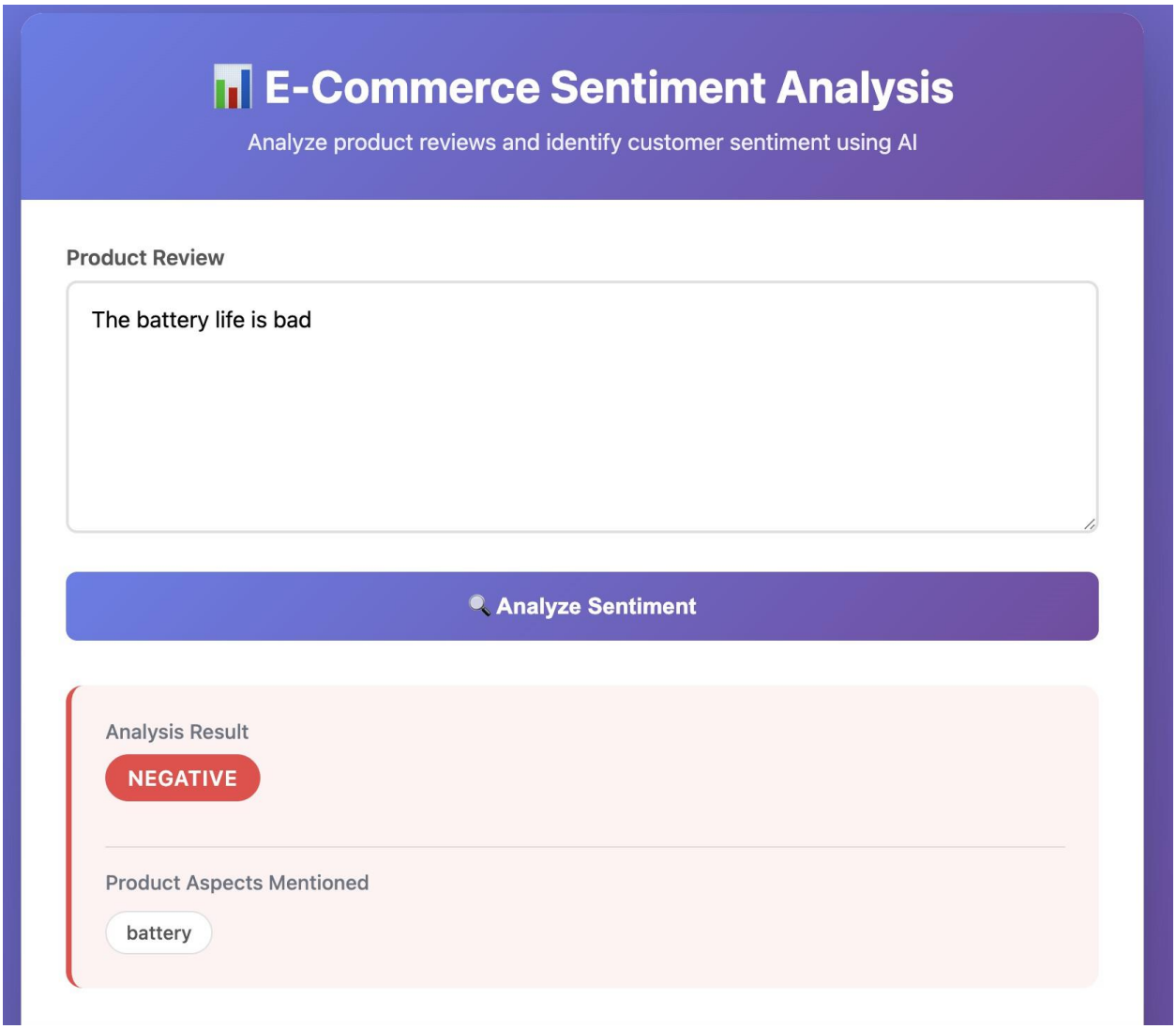
```python
def main() -> None:
    dataset = load_dataset(CURATED_DATASET)
    print("Preview of dataset:")
    preview = pd.DataFrame(
        {"review": dataset.reviews, "sentiment": dataset.labels}
    ).head()
    print(preview)
    print(f"\nTotal samples: {len(dataset.reviews)}")

    stop_words = ensure_stopwords()
    vectorizer = build_vectorizer(stop_words)
    X_train, X_test, y_train, y_test = prepare_features(dataset, vectorizer)

    svm_model = train_svm(X_train, y_train)
    evaluate_model(svm_model, X_test, y_test)


if __name__ == "__main__":
    main()
```

This backend file is responsible for:

- Loads and merges multiple datasets
- Performs efficient chunk-based processing for large files (1.5GB)
- Cleans and preprocesses review text
- Builds a TF-IDF vectorizer with bi-grams
- Splits data into training and testing sets
- Trains a Linear SVM classifier
- Evaluates accuracy and classification performance
- Defines aspect keyword detection for the frontend
- Provides reusable functions that the Flask app imports.

RESULTS:


E-Commerce Sentiment Analysis
Analyze product reviews and identify customer sentiment using AI
Product Review
The battery life is fantastic
Analyze Sentiment
Analysis Result
POSITIVE
Product Aspects Mentioned
battery


📊 E-Commerce Sentiment Analysis
Analyze product reviews and identify customer sentiment using AI
Product Review
The battery life is bad
🔍 Analyze Sentiment
Analysis Result
NEGATIVE
Product Aspects Mentioned
battery

FUTURE IMPLEMENTATIONS:

Future implementations include integrating deep learning models, expanding multilingual support, improving aspect-based sentiment extraction, adding spam detection, deploying the system as a cloud API, creating advanced analytics dashboards, and enabling automated retraining pipelines. These enhancements will make the system more scalable, accurate, explainable, and useful for real-world e-commerce applications.

CONCLUSION:

The web-based sentiment analysis system developed for Elevate Retail Solutions successfully automates the interpretation of large volumes of customer product reviews. By integrating TF- IDF feature extraction with a linear SVM classifier, the system provides fast, accurate, and reliable sentiment predictions for real-world e-commerce data. The implementation of a Flask- based web interface makes the solution accessible to non-technical users, enabling them to analyze reviews instantly through a simple and intuitive interface.

The system not only identifies overall sentiment—Positive, Negative—but also detects key product aspects such as battery, performance, audio quality, and design. This allows businesses to derive deeper and more actionable insights from customer feedback. With optimized handling of large datasets and efficient preprocessing, the model performs well even with extensive review data.

Overall, the project demonstrates how machine learning and natural language processing can significantly enhance customer experience analysis in retail environments. The developed system reduces manual effort, improves decision-making, and provides a scalable foundation for future enhancements such as deep learning models, multilingual support, real-time monitoring, and advanced analytics dashboards. It serves as a practical and impactful tool for transforming raw customer reviews into meaningful business intelligence.