

CASE STUDY

Sentiment Analysis Program in Python on Amazon Reviews Dataset

HU22CSEN0100287

Eswaraprasad Sai Ganesh

Overview of the Tool: Sentiment Analysis

Sentiment analysis, also known as opinion mining, is a powerful tool in the field of Natural Language Processing (NLP) that involves analyzing and extracting subjective information from text data. It enables businesses, researchers, and individuals to gain insights into opinions, feelings, and emotional tones conveyed in written language. By categorizing sentiments as positive, negative, or neutral, sentiment analysis provides a means to quantify qualitative data, making it especially valuable for applications like customer feedback analysis, brand reputation monitoring, and market research.

In this project, we build a sentiment analysis model in Python to classify customer reviews from Amazon into "positive" or "negative" categories. With millions of online reviews available, Amazon's vast collection of feedback provides a robust dataset to train and evaluate sentiment analysis models. Our goal is to leverage the information within customer reviews to assess the general sentiment, which could then serve various applications, such as identifying common pain points or recognizing what customers appreciate most about products.

Applications of Sentiment Analysis

Sentiment analysis is used across numerous industries and has a range of applications:

1. **Customer Feedback:** In e-commerce, sentiment analysis is crucial for analyzing customer reviews and feedback. It helps businesses understand the strengths and weaknesses of their products or services from the customer's perspective.
2. **Social Media Monitoring:** Companies monitor social media platforms to track public sentiment about their brands. Sentiment analysis helps identify trends, monitor brand reputation, and gauge customer loyalty.
3. **Market Research:** Researchers can gain insights into consumer preferences and trends by analyzing customer reviews and survey data, informing product development and marketing strategies.

4. **Financial Markets:** Sentiment analysis is increasingly used in financial markets to gauge public sentiment on companies or economic events, aiding in stock market prediction and investment decisions.
5. **Customer Support:** Sentiment analysis helps customer service teams prioritize responses by identifying negative or urgent feedback from large volumes of inquiries.

Technical Foundations of Sentiment Analysis

Sentiment analysis models can range from rule-based systems to advanced machine learning models. Here are a few foundational approaches:

1. **Lexicon-based Analysis:** This approach relies on predefined dictionaries of words associated with positive, negative, or neutral sentiment. It assigns a sentiment score based on the frequency and intensity of sentiment-laden words. Lexicon-based approaches, while straightforward, struggle with nuanced language or context-dependent meanings.
2. **Machine Learning Models:** Machine learning models for sentiment analysis are trained on labeled data, allowing them to learn patterns associated with different sentiment classes. Models such as Naive Bayes, Support Vector Machines (SVM), and Logistic Regression are often used as baseline classifiers due to their simplicity and interpretability.
3. **Deep Learning Models:** Advanced models such as Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTM) networks, and Transformers (e.g., BERT) capture more complex patterns and contextual relationships in text, resulting in more accurate sentiment classification.

Tool Selection for Sentiment Analysis of Amazon Reviews

For this project, a machine learning approach is chosen for simplicity, effectiveness, and speed. We specifically utilize:

- **Naive Bayes Classifier:** Known for its effectiveness in text classification tasks, this probabilistic model is ideal for quick and interpretable sentiment classification.
- **TF-IDF Vectorization:** TF-IDF (Term Frequency-Inverse Document Frequency) is a feature extraction technique that quantifies the importance of words in the text relative to the document set. By applying TF-IDF, we transform text data into a numerical form suitable for model training while reducing the effect of commonly occurring words (like “the” and “is”).

How Sentiment Analysis Works in this Tool

The sentiment analysis tool leverages Amazon's customer reviews dataset in two files: train.csv and test.csv. The train.csv file is used to train the model, while test.csv provides data for evaluating the model's performance. Below are the stages involved in building the tool:

1. **Data Collection and Preparation:** The dataset is loaded, and each review is assigned a sentiment label (1 for positive, 0 for negative).
2. **Data Cleaning:** Preprocessing includes handling missing values, standardizing text formats, and removing unnecessary words.
3. **Text Vectorization:** TF-IDF vectorization transforms the text into numerical features, allowing the model to understand the relative importance of words.
4. **Model Training:** The Naive Bayes model is trained on the transformed data, learning patterns associated with positive and negative sentiments.
5. **Model Evaluation:** The trained model is tested on unseen data to evaluate its accuracy and robustness. Evaluation metrics like accuracy, precision, recall, and F1-score are used to measure performance.
6. **User Interaction:** The tool allows user input to test review sentiment, making it versatile for real-time analysis.

Advantages and Challenges

Advantages:

- **Scalability:** Sentiment analysis can process massive amounts of text data, providing insights that would be impractical to gain through manual review.
- **Real-time Analysis:** Once trained, sentiment analysis models can classify sentiments in real time, which is valuable for applications like social media monitoring.
- **Quantitative Insight from Qualitative Data:** Sentiment analysis transforms qualitative feedback into quantifiable metrics, aiding decision-making.

Challenges:

- **Context and Sarcasm:** Basic machine learning models may struggle to understand context or detect sarcasm, which can skew results.
- **Data Imbalance:** Sentiment datasets may have more positive or negative reviews, leading to biased models. Balancing data or using advanced models helps address this issue.

- **Complex Sentiment:** Some reviews express mixed feelings, which can be difficult to classify as strictly positive or negative.

CODE

```
# Import necessary libraries
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, accuracy_score

# Load the datasets (adjust the paths as needed)
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')

# Rename columns for clarity
train_data.columns = ['Sentiment', 'Review_Title', 'Review_Text']
test_data.columns = ['Sentiment', 'Review_Title', 'Review_Text']

# Prepare the training and testing features and labels
X_train = train_data['Review_Text'].fillna('') # Handle any NaN values
y_train = train_data['Sentiment'].replace({1: 0, 2: 1}) # Encode
# sentiment: 1 (negative) -> 0, 2 (positive) -> 1

X_test = test_data['Review_Text'].fillna('')
y_test = test_data['Sentiment'].replace({1: 0, 2: 1})

# Define the pipeline with TF-IDF vectorizer and Naive Bayes classifier
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words='english',
max_features=5000)),
    ('nb_classifier', MultinomialNB())
])

# Train the model on the training data
pipeline.fit(X_train, y_train)

# Predict on the test set
y_pred = pipeline.predict(X_test)

# Show the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")

# Classification report
```

```

print("\nClassification Report:\n", classification_report(y_test,
y_pred, target_names=['Negative', 'Positive']))

# Function for predicting sentiment of new reviews
def predict_sentiment(review_text):
    prediction = pipeline.predict([review_text])
    sentiment = "Positive" if prediction[0] == 1 else "Negative"
    print(f"\nThe review sentiment is: {sentiment}")

# Example interactive input
while True:
    review = input("\nEnter a review (or type 'exit' to quit): ")
    if review.lower() == 'exit':
        break
    predict_sentiment(review)

```

PROCEDURE

CODE SEGMENTS FOR EACH RESPECTIVE STEPS

Step 1: Importing Libraries

```

python
Copy code
# Import necessary libraries
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, accuracy_score

```

In this step, we import the essential libraries:

- **pandas:** For data handling and manipulation.
- **TfidfVectorizer:** To convert text data into numerical features.
- **MultinomialNB:** A Naive Bayes classifier suitable for text classification.
- **Pipeline:** To streamline the model workflow.
- **accuracy_score** and **classification_report:** For evaluating model performance.

Step 2: Loading and Inspecting the Datasets

```

python
Copy code
# Load the datasets (adjust the paths as needed)
train_data = pd.read_csv('train.csv')

```

```
test_data = pd.read_csv('test.csv')

# Rename columns for clarity
train_data.columns = ['Sentiment', 'Review_Title', 'Review_Text']
test_data.columns = ['Sentiment', 'Review_Title', 'Review_Text']
```

Here, we load the `train.csv` and `test.csv` datasets. We then rename columns to make them easier to reference. The dataset includes:

- **Sentiment:** The label of each review (1 for negative, 2 for positive).
 - **Review_Title** and **Review_Text:** Text features used to analyze sentiment.
-

Step 3: Preparing Data for Training and Testing

```
python
Copy code
# Prepare the training and testing features and labels
X_train = train_data['Review_Text'].fillna('') # Handle any NaN values
y_train = train_data['Sentiment'].replace({1: 0, 2: 1}) # Encode
sentiment: 1 (negative) -> 0, 2 (positive) -> 1

X_test = test_data['Review_Text'].fillna('')
y_test = test_data['Sentiment'].replace({1: 0, 2: 1})
```

We prepare our features and labels:

- **X_train** and **X_test:** Contain the review text for training and testing.
 - **y_train** and **y_test:** Labels for the sentiment. We encode them as binary values where 0 = negative and 1 = positive. This encoding simplifies classification.
-

Step 4: Defining the Pipeline

```
python
Copy code
# Define the pipeline with TF-IDF vectorizer and Naive Bayes classifier
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words='english', max_features=5000)),
    ('nb_classifier', MultinomialNB())
])
```

Here, we define a **Pipeline** that combines:

1. **TF-IDF Vectorizer:** Converts text to numeric data, removes common English stop words, and limits the vocabulary to the top 5,000 words.
2. **MultinomialNB Classifier:** A Naive Bayes classifier tailored for text data.

The pipeline structure ensures that these steps are applied in sequence, making the workflow efficient and easier to manage.

Step 5: Training the Model

```
python
Copy code
# Train the model on the training data
pipeline.fit(X_train, y_train)
```

This line trains the model on the training data using the pipeline. The model learns the patterns between text features and the sentiment labels from `train.csv`.

Step 6: Model Evaluation

```
python
Copy code
# Predict on the test set
y_pred = pipeline.predict(X_test)

# Show the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")

# Classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred,
target_names=['Negative', 'Positive']))
```

In this step, we evaluate the model:

1. **Predict on Test Set:** The model makes predictions on `X_test`.
 2. **Accuracy Score:** We calculate the accuracy, which shows the percentage of correct predictions.
 3. **Classification Report:** The report includes precision, recall, and F1-score for each class, giving a more comprehensive view of model performance.
-

Step 7: Defining a Function for User Input

```
python
Copy code
# Function for predicting sentiment of new reviews
def predict_sentiment(review_text):
    prediction = pipeline.predict([review_text])
    sentiment = "Positive" if prediction[0] == 1 else "Negative"
    print(f"\nThe review sentiment is: {sentiment}")
```

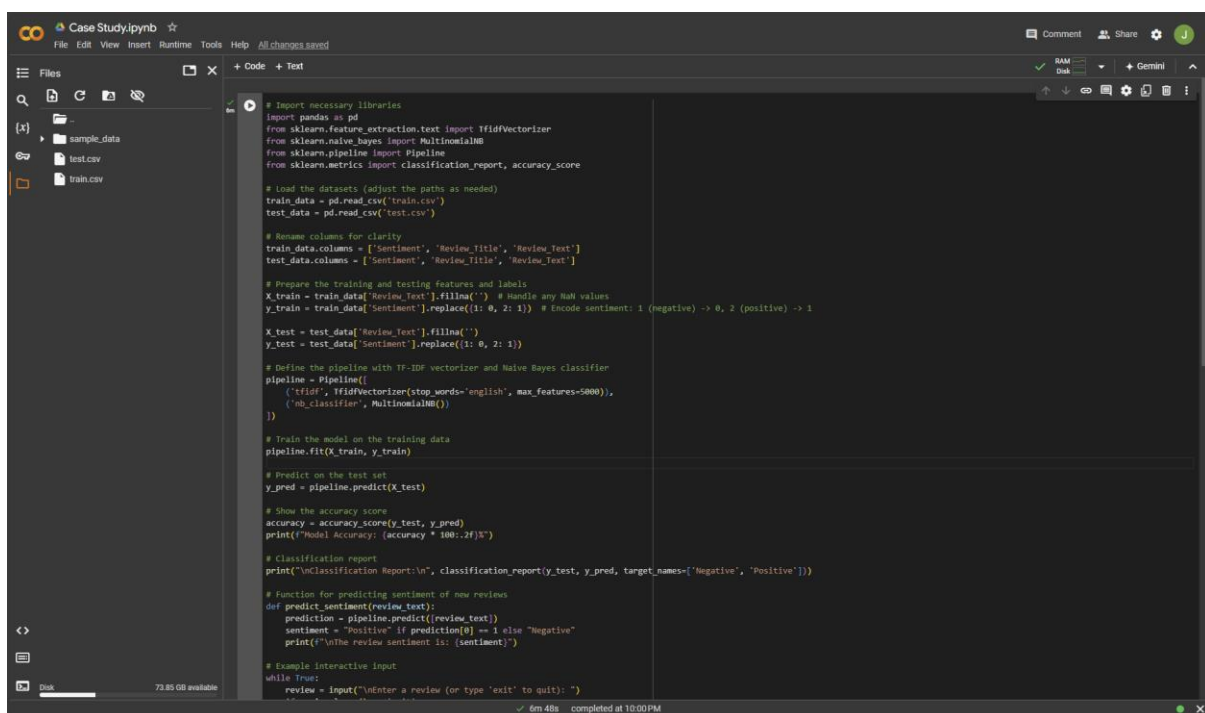
This function takes a review as input, predicts the sentiment, and outputs either "Positive" or "Negative." It uses the trained pipeline, allowing for immediate predictions without retraining the model.

Step 8: Adding Interactive User Input

```
python
Copy code
# Example interactive input
while True:
    review = input("\nEnter a review (or type 'exit' to quit): ")
    if review.lower() == 'exit':
        break
    predict_sentiment(review)
```

The loop prompts users to enter reviews. For each input, it calls `predict_sentiment` to predict and display the sentiment. The loop continues until the user types "exit," providing a user-friendly interface for testing the model interactively.

This breakdown covers each part of the sentiment analysis process, from data loading and preprocessing to model training, evaluation, and interactive predictions.



```
# Import necessary libraries
import pandas as pd
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.pipeline import Pipeline
from sklearn.metrics import classification_report, accuracy_score

# Load the datasets (adjust the paths as needed)
train_data = pd.read_csv('train.csv')
test_data = pd.read_csv('test.csv')

# Remove columns for clarity
train_data.columns = ['Sentiment', 'Review_Title', 'Review_Text']
test_data.columns = ['Sentiment', 'Review_Title', 'Review_Text']

# Prepare the training and testing features and labels
X_train = train_data['Review_Text'].fillna('') # Handle any NaN values
y_train = train_data['Sentiment'].replace({1: 0, 2: 1}) # Encode sentiment: 1 (negative) -> 0, 2 (positive) -> 1
X_test = test_data['Review_Text'].fillna('')
y_test = test_data['Sentiment'].replace({1: 0, 2: 1})

# Define the pipeline with TF-IDF vectorizer and Naive Bayes classifier
pipeline = Pipeline([
    ('tfidf', TfidfVectorizer(stop_words='english', max_features=5000)),
    ('nb_classifier', MultinomialNB())
])

# Train the model on the training data
pipeline.fit(X_train, y_train)

# Predict on the test set
y_pred = pipeline.predict(X_test)

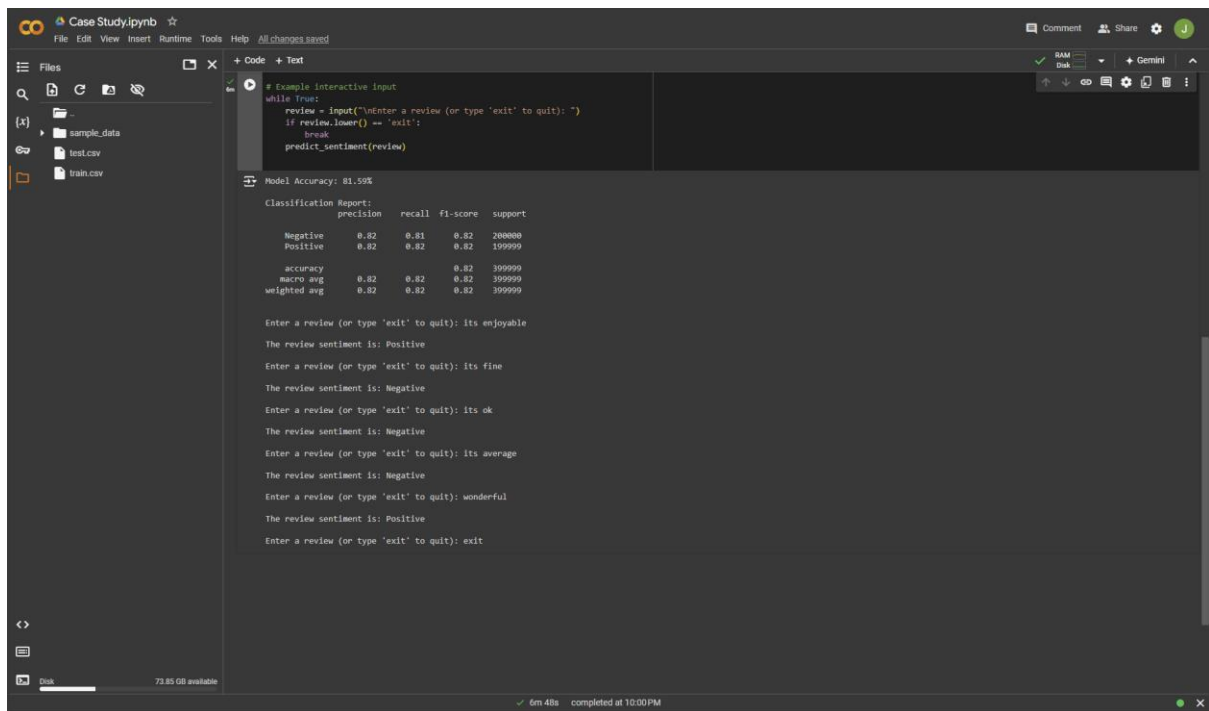
# Show the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"Model Accuracy: {accuracy * 100:.2f}%")

# Classification report
print("\nClassification Report:\n", classification_report(y_test, y_pred, target_names=['Negative', 'Positive']))

# Function for predicting sentiment of new reviews
def predict_sentiment(review_text):
    prediction = pipeline.predict([review_text])
    sentiment = "Positive" if prediction[0] == 1 else "Negative"
    print(f"\nThe review sentiment is: {sentiment}")

# Example interactive input
while True:
    review = input("\nEnter a review (or type 'exit' to quit): ")
```


OUTPUT



The screenshot displays a Jupyter Notebook titled 'CaseStudy.ipynb'. The left sidebar shows a file explorer with 'sample_data', 'test.csv', and 'train.csv'. The main area contains a code cell with a while loop for interactive input and a text cell showing the output.

```
# Example: Interactive Input
while True:
    review = input("\nEnter a review (or type 'exit' to quit): ")
    if review.lower() == 'exit':
        break
    predict_sentiment(review)
```

Model Accuracy: 81.59%

Classification Report:				
	precision	recall	f1-score	support
Negative	0.82	0.81	0.82	200000
Positive	0.82	0.82	0.82	199999
accuracy			0.82	399999
macro avg	0.82	0.82	0.82	399999
weighted avg	0.82	0.82	0.82	399999

Enter a review (or type 'exit' to quit): its enjoyable
The review sentiment is: Positive
Enter a review (or type 'exit' to quit): its fine
The review sentiment is: Negative
Enter a review (or type 'exit' to quit): its ok
The review sentiment is: Negative
Enter a review (or type 'exit' to quit): its average
The review sentiment is: Negative
Enter a review (or type 'exit' to quit): wonderful
The review sentiment is: Positive
Enter a review (or type 'exit' to quit): exit

Conclusion

This sentiment analysis tool on Amazon reviews enables us to capture customer opinions systematically. The program converts qualitative text data into meaningful sentiment scores, assisting in decision-making and customer understanding. By using TF-IDF vectorization and a Naive Bayes classifier, this tool provides a balance between simplicity and effectiveness, demonstrating the power of machine learning in analyzing large-scale textual data.