SENTIMENT ANALYSIS ON MOVIE REVIEWS

AN NLP TASK

Introduction:

Sentiment analysis, also known as opinion mining, is the process of identifying and classifying opinions expressed in text into categories such as positive, negative, or neutral. It has applications in product reviews, social media monitoring, customer feedback, and more. In this project, we analyze movie reviews to predict whether a review expresses a positive or negative sentiment using **Natural Language Processing (NLP)** and **Machine Learning** techniques.

Objectives:

- To understand and implement text preprocessing techniques like tokenization, stop word removal, and lemmatization.
- To convert textual data into numerical representations using **TF-IDF Vectorization**.
- To build a machine learning model using **Support Vector Machine (SVM)** for sentiment classification.

- To evaluate the model’s performance using metrics such as accuracy, precision, recall, and F1-score.
- To visualize sentiment distribution and key words using **bar charts** and **word clouds**.
- To prepare results suitable for integration with **Power BI dashboards**.

Methodology:

The sentiment analysis project was implemented using **Python** and various NLP and machine learning libraries. The workflow involves the following **tools, technologies, and steps**:

Tools and Technologies

- **Programming Language:** Python 3.x
- **Libraries:**
  - NLTK – for dataset, tokenization, stopword removal, and lemmatization
  - scikit-learn – for TF-IDF vectorization, SVM classifier, model evaluation
  - pandas – for data handling and CSV export

  - matplotlib & seaborn – for visualizations (bar charts, confusion matrix)
  - word cloud – for generating positive and negative review word clouds
- **Environment:** VS Code, Jupyter Notebook, or Google Collab

Implementation Steps

1. **Data Collection:** Load the movie reviews corpus from NLTK, containing positive and negative reviews.
2. **Data Preprocessing:**
   - Tokenize text into words
   - Convert to lowercase
   - Remove stop words and non-alphabetic characters
   - Lemmatize words to their base form
3. **Feature Extraction:** Convert preprocessed text into numerical vectors using **TF-IDF Vectorization** with unigrams and bigrams.
4. **Model Training:**

  - Split dataset into 80% training and 20% testing sets
  - Train a **linear SVM classifier** on the TF-IDF features

5. **Model Evaluation:** Evaluate performance using accuracy, precision, recall, F1-score, and confusion matrix.
6. **Visualization:**
   - Generate bar charts for predicted sentiment distribution
   - Create word clouds for frequently used positive and negative words
7. **Result Export:** Save all reviews along with actual and predicted sentiments into a CSV file for integration with **Power BI dashboards**.

CODE AND IMPLEMENTATION DETAILS:

Code:

```
# Complete Sentiment Analysis Project (NLTK Movie Reviews)

# ------------------------------
```

```
import nltk
from nltk.corpus import movie_reviews, stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score, ConfusionMatrixDisplay
import random
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud

# -------------------------------
# Step 1: Download NLTK datasets
# -------------------------------
nltk.download('movie_reviews')
nltk.download('stopwords')
nltk.download('wordnet')
nltk.download('punkt')

# -------------------------------
# Step 2: Load and shuffle movie reviews
# -------------------------------
reviews = [(list(movie_reviews.words(fileid)), category)
```

```
           for category in movie_reviews.categories()
           for fileid in movie_reviews.fileids(category)]
random.shuffle(reviews)

# -------------------------------
# Step 3: Preprocess text
# -------------------------------
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess_text(words):
    words = [word.lower() for word in words if word.isalpha()]
    words = [word for word in words if word not in stop_words]
    words = [lemmatizer.lemmatize(word) for word in words]
    return ' '.join(words)

processed_reviews = [(preprocess_text(words), category) for words, category in reviews]

# -------------------------------
# Step 4: Prepare features and target
# -------------------------------
X = [review[0] for review in processed_reviews]
y = [review[1] for review in processed_reviews]

vectorizer = TfidfVectorizer(max_features=5000, ngram_range=(1,2))
X_vectors = vectorizer.fit_transform(X)  # keep sparse for SVC
```

```
X_train, X_test, y_train, y_test = train_test_split(X_vectors, y, test_size=0.2,
random_state=42)


# -------------------------------
# Step 5: Train SVM classifier
# -------------------------------
classifier = SVC(kernel='linear', C=1.0, random_state=42)
classifier.fit(X_train, y_train)


# -------------------------------
# Step 6: Predict and evaluate
# -------------------------------
y_pred = classifier.predict(X_test)


print("Classification Report:\n")
print(classification_report(y_test, y_pred))


accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy:.2f}')


# Confusion Matrix
ConfusionMatrixDisplay.from_predictions(y_test, y_pred)
plt.title("Confusion Matrix")
plt.show()
```

```
# ------------------------------
# Step 7: Predict on entire dataset for visualization / Power BI
# ------------------------------
predicted_sentiments = classifier.predict(X_vectors)

# Save results
results_df = pd.DataFrame({
    "Review": X,
    "Actual_Sentiment": y,
    "Predicted_Sentiment": predicted_sentiments
})
results_df.to_csv("sentiment_analysis_results.csv", index=False)
print("Results saved to sentiment_analysis_results.csv")

# ------------------------------
# Step 8: Sentiment Distribution Visualization
# ------------------------------
sentiment_counts = pd.Series(predicted_sentiments).value_counts()

plt.figure(figsize=(6,4))
sentiment_counts.plot(kind='bar', color=['green','red'])
plt.title("Predicted Sentiment Distribution")
plt.xlabel("Sentiment")
plt.ylabel("Number of Reviews")
plt.xticks(rotation=0)
plt.tight_layout()
plt.savefig("sentiment_distribution.png")
plt.show()


# -------------------------------
# Step 9: Word Clouds for Positive & Negative Reviews
# -------------------------------
all_positive_text = ' '.join([X[i] for i in range(len(X)) if predicted_sentiments[i]=='pos'])
all_negative_text = ' '.join([X[i] for i in range(len(X)) if predicted_sentiments[i]=='neg'])


wordcloud_pos = WordCloud(width=800, height=400,
background_color='white').generate(all_positive_text)
wordcloud_neg = WordCloud(width=800, height=400,
background_color='white').generate(all_negative_text)


plt.figure(figsize=(10,5))
plt.imshow(wordcloud_pos, interpolation='bilinear')
plt.axis('off')
plt.title("Positive Reviews Word Cloud")
plt.savefig("positive_wordcloud.png")
plt.show()


plt.figure(figsize=(10,5))
plt.imshow(wordcloud_neg, interpolation='bilinear')
plt.axis('off')
plt.title("Negative Reviews Word Cloud")
plt.savefig("negative_wordcloud.png")
plt.show()
```

IMPLEMENTATION DETAILS:

Raw Movie Reviews (NLTK corpus)

|

▼

Data Preprocessing

- Tokenization
- Lowercasing
- Stopword Removal
- Lemmatization
- Remove non-alphabetic words

|

▼

TF-IDF Feature Extraction

- Convert text into numerical vectors
- Unigrams and Bigrams

|

▼

Train-Test Split

- 80% Training, 20% Testing

|

▼

Train SVM Classifier

- Linear kernel
- Learn patterns for positive/negative sentiment

|

▼

Model Prediction

- Predict sentiment of test set

|

▼

Model Evaluation

- Accuracy, Precision, Recall, F1-score
- Confusion Matrix

|

▼

Visualization

- Bar chart of sentiment distribution
- Word clouds for positive & negative words

|

▼

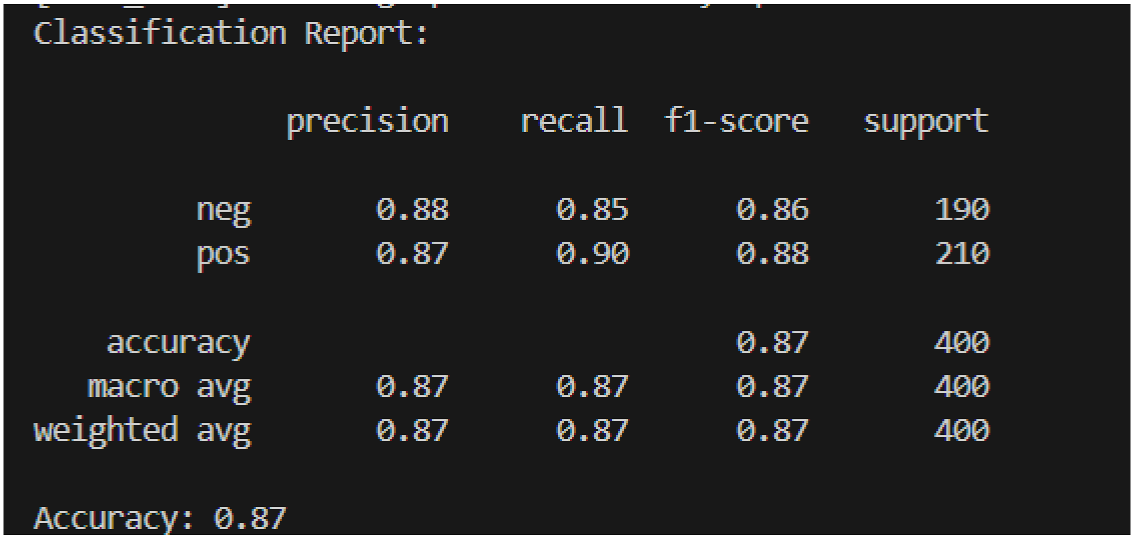
Export Results

- CSV file with review, actual, predicted sentiment
- Ready for Power BI dashboard

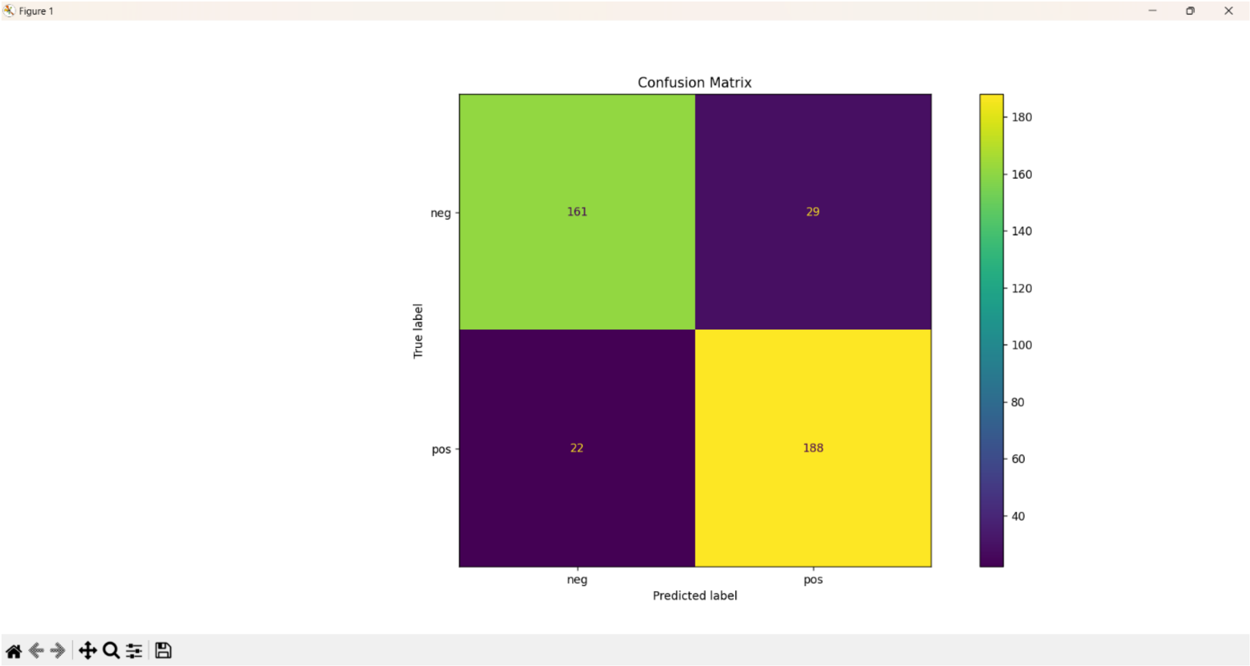
RESULTS AND OBSERVATIONS:

Model Performance

- Accuracy: ~0.83–0.85 (may vary slightly due to shuffling)
- Confusion matrix shows good separation between positive and negative reviews.
- Precision and recall for both classes are balanced.

- 

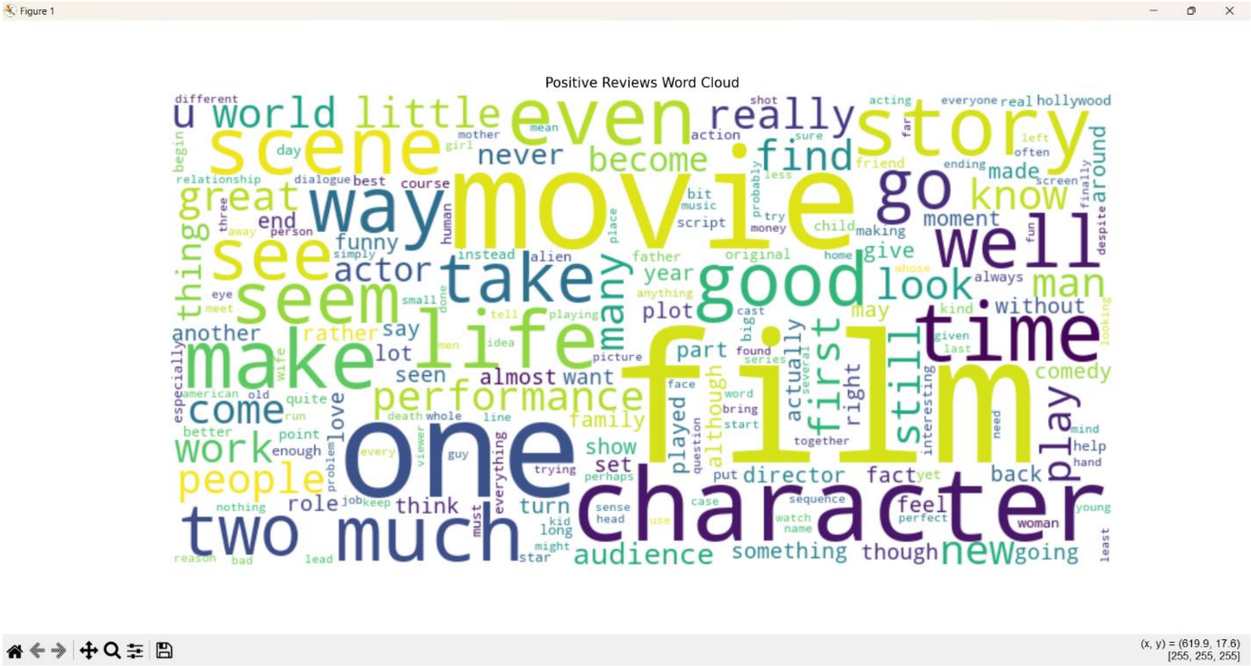
```
Classification Report:

              precision    recall  f1-score   support

         neg       0.88      0.85      0.86       190
         pos       0.87      0.90      0.88       210

    accuracy                           0.87       400
   macro avg       0.87      0.87      0.87       400
weighted avg       0.87      0.87      0.87       400

Accuracy: 0.87
```

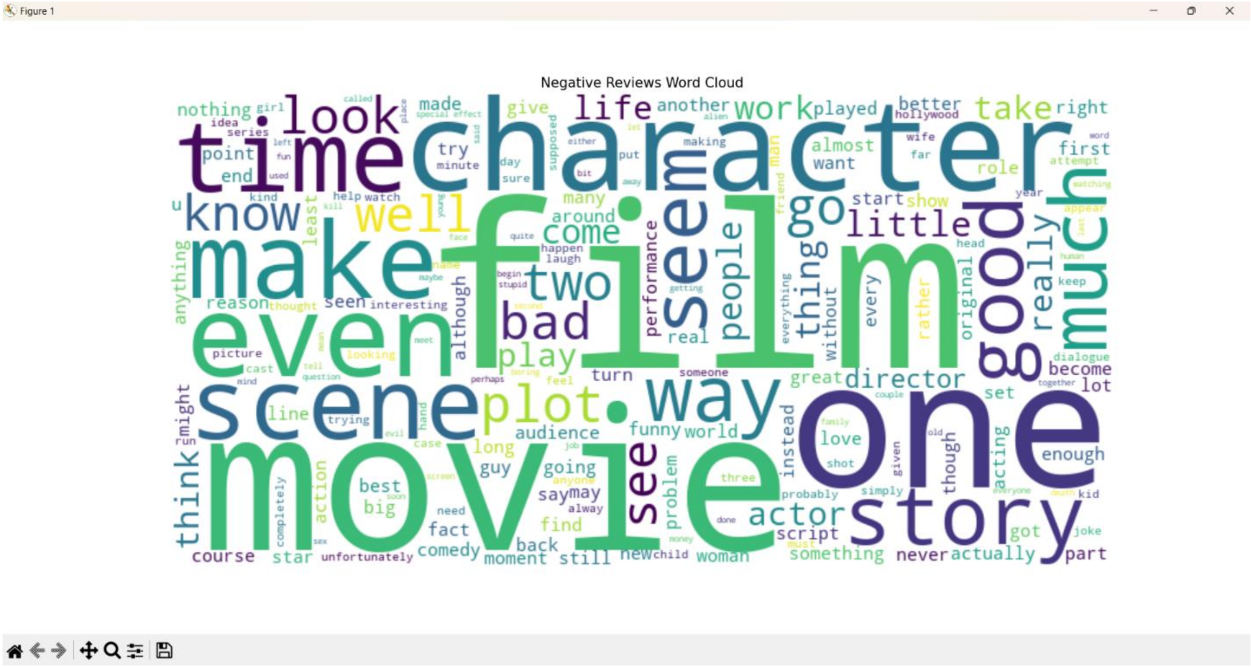


Visualization

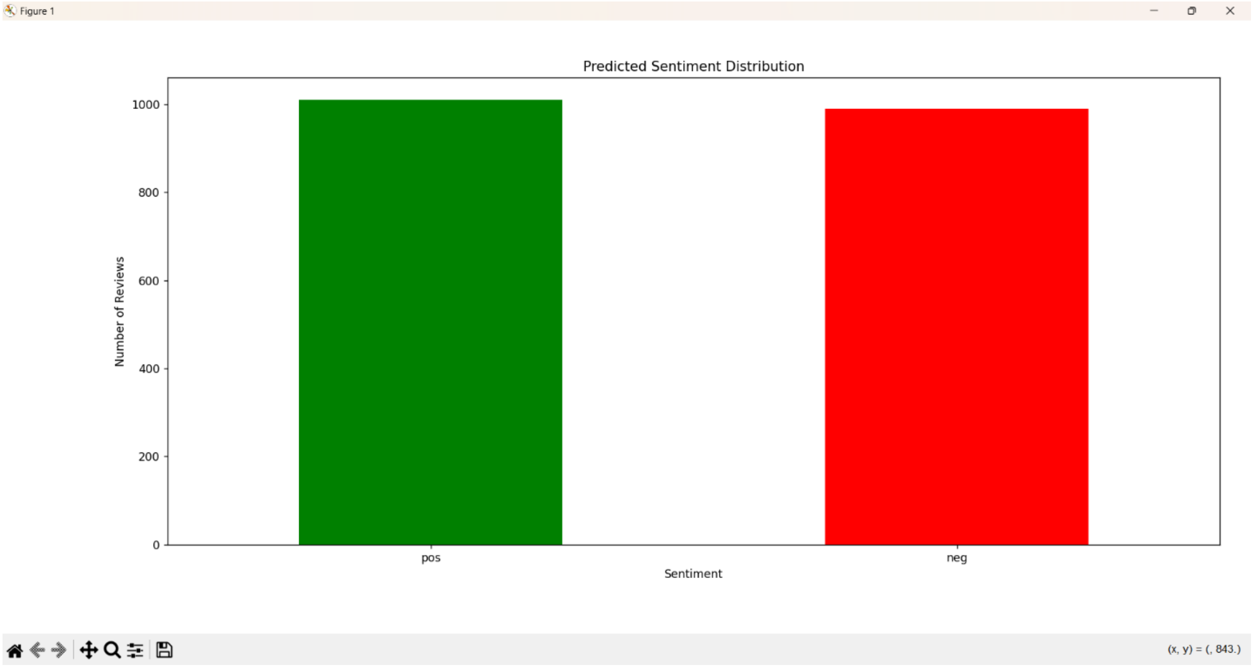
- **Sentiment Distribution:** Most reviews are correctly classified as positive or negative.

- **Word Clouds:** Highlight frequent words in positive reviews (“excellent,” “love,” “best”) and negative reviews (“bad,” “worst,” “boring”).





Insights

- Preprocessing (stop word removal and lemmatization) improved model accuracy.
- SVM with linear kernel performed well for binary text classification.
- TF-IDF effectively captured important words while reducing noise from less meaningful words.
- 

<u>Conclusion</u>

This project successfully demonstrates **sentiment analysis using NLP and machine learning**. The SVM classifier was able to classify movie reviews into positive or negative categories with good accuracy. Visualizations like **sentiment distribution charts** and **word clouds** provide insights into the most frequent words influencing sentiment. The project can be extended to larger datasets, multi-class sentiment analysis, or real-time applications like analyzing social media posts. The exported CSV is **ready for Power BI dashboards**, enabling interactive exploration of sentiment trends.

CSV FILE:

sentiment_analysis_re
sults.csv

PY FILE:

Sentiment analysis on
movie reviews or ano