Natural Language Processing using Python Programming

Notebook 07.2: Advanced Sentiment Analysis (Machine Learning Approach)

Part of the comprehensive learning series: Natural Language Processing using Python Programming

Learning Objectives:

- Master machine learning approaches to sentiment analysis for domain-specific learning
- Implement complete ML pipeline: data splitting, vectorization, training, and evaluation
- Train and compare Logistic Regression and Multinomial Naive Bayes classifiers
- Understand critical ML practices: fit_transform vs transform for train-test data
- Evaluate model performance using accuracy, precision, recall, and F1-score metrics

---

- While lexicon-based methods (Chapter 7.1) are fast, they lack the flexibility to learn domain-specific sentiment (e.g., 'slow' might be negative for a server but positive for a relaxation app).
- The **Machine Learning (ML) Approach** overcomes this by training a classifier on labeled data.
- This notebook covers the complete ML pipeline for sentiment classification using our prepared TF-IDF features.

1. Setting up: Data and Libraries

- We will use the **cleaned data** (`cleaned_review`) and the **sentiment label** (`sentiment`) prepared in Chapter 3.2, and the vectorizers from Chapter 6.2.

In [1]:
```
# Import necessary libraries
import pandas as pd
from sklearn.model_selection import train_test_split                # for splittin
from sklearn.feature_extraction.text import TfidfVectorizer         # for TF-IDF j
from sklearn.linear_model import LogisticRegression                 # for Logistic
from sklearn.naive_bayes import MultinomialNB                       # for Naive Ba
from sklearn.metrics import classification_report, accuracy_score   # for evaluati

FILE_PATH = '../../data/processed/processed_reviews.csv'
```

```
try:
    df = pd.read_csv(FILE_PATH)
    df = df.dropna(subset=['cleaned_review'])

    # Note: We will restrict this example to Binary Classification (positive/negat
    df_binary = df[df['sentiment'].isin(['positive', 'negative'])].copy()

    X = df_binary['cleaned_review']
    y = df_binary['sentiment']

    print("Data loaded and filtered for binary classification.")
    print(f"Total data points: {len(df_binary)}")
except FileNotFoundError:
    print(f"ERROR: Processed data not found at {FILE_PATH}. Please run 03_2_using_
```

```
Data loaded and filtered for binary classification.
Total data points: 4
```

---

2. Data Preparation: Split and Vectorize

CRITICAL NOTE ON DATASET SIZE

- **The sample data used here is too small to build a meaningful, generalizable model.**
- For practical work, learners should use a large dataset, such as the full **IMDB 50k dataset** or a similar public resource.
- We use this small set here *only* to ensure the code executes and the concept is clear.

2.1 Splitting the Data

- We divide the data into training (for model learning) and testing (for model evaluation) sets.

In [2]:
```
# Split into training (80%) and testing (20%)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_st

print(f"Train size: {len(X_train)} | Test size: {len(X_test)}")
```

```
Train size: 3 | Test size: 1
```

2.2 TF-IDF Feature Extraction (Chapter 6.2 Review)

- We use TF-IDF to convert our text data into numerical vectors.
- **Remember:** `fit_transform` **on train,** `transform` **on test!**

In [3]:
```
# Initialize TF-IDF Vectorizer with both unigrams and bigrams, and limit max featu
tfidf_vectorizer = TfidfVectorizer(ngram_range=(1, 2), max_features=5000) # Limit

# 1. Train Data: FIT and TRANSFORM
X_train_vectors = tfidf_vectorizer.fit_transform(X_train)

# 2. Test Data: TRANSFORM ONLY
```

```
X_test_vectors = tfidf_vectorizer.transform(X_test)

print(f"TF-IDF Matrix Shape (Train): {X_train_vectors.shape}")
print(f"TF-IDF Matrix Shape (Test):  {X_test_vectors.shape}")
```

```
TF-IDF Matrix Shape (Train): (3, 38)
TF-IDF Matrix Shape (Test):  (1, 38)
```

---

3. Model Training and Prediction

- We will use two classic, highly effective classification algorithms for text: **Logistic Regression** and **Multinomial Naive Bayes**.

3.1 Training Logistic Regression

- **Logistic Regression** is a fast, linear model that works exceptionally well on sparse data like TF-IDF vectors, providing high interpretability.

In [4]:
```
# Initialize Logistic Regression model and create instance
lr_model = LogisticRegression(max_iter=1000, random_state=42)

# Train the model on the TF-IDF vectors and training labels
lr_model.fit(X_train_vectors, y_train)

# Predict on the test set
lr_predictions = lr_model.predict(X_test_vectors)

print("Logistic Regression Model Trained.")
```

```
Logistic Regression Model Trained.
```

3.2 Training Multinomial Naive Bayes

- **Multinomial Naive Bayes** is a probabilistic classifier based on word count (or frequency) features, assuming word features are conditionally independent.
- It is often the baseline standard for text classification.

In [5]:
```
# Initialize Multinomial Naive Bayes model and create instance
nb_model = MultinomialNB()

# Train the model
nb_model.fit(X_train_vectors, y_train)

# Predict on the test set
nb_predictions = nb_model.predict(X_test_vectors)

print("Naive Bayes Model Trained.")
```

```
Naive Bayes Model Trained.
```

---

4. Model Evaluation

- Evaluation tells us how well our model performed on unseen data.
- We use standard metrics like **Accuracy** and the **Classification Report** (which includes Precision, Recall, and F1-score).

4.1 Logistic Regression Evaluation

In [8]:

```
print("--- Logistic Regression Performance ---")
print(f"Accuracy: {accuracy_score(y_test, lr_predictions):.4f}\n")
```

```
--- Logistic Regression Performance ---
Accuracy: 0.0000
```

In [9]:

```
print("Classification Report:")
print(classification_report(y_test, lr_predictions))
```

```
Classification Report:
              precision    recall  f1-score   support

    negative       0.00      0.00      0.00       1.0
    positive       0.00      0.00      0.00       0.0

    accuracy                           0.00       1.0
   macro avg       0.00      0.00      0.00       1.0
weighted avg       0.00      0.00      0.00       1.0
```

```
c:\Users\admin\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\admin\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\admin\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\admin\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\admin\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\admin\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

4.2 Naive Bayes Evaluation

In [10]:
```
print("--- Naive Bayes Performance ---")
print(f"Accuracy: {accuracy_score(y_test, nb_predictions):.4f}\n")
print("Classification Report:")
print(classification_report(y_test, nb_predictions))
```

```
--- Naive Bayes Performance ---
Accuracy: 0.0000

Classification Report:
              precision    recall  f1-score   support

    negative       0.00      0.00      0.00       1.0
    positive       0.00      0.00      0.00       0.0

    accuracy                           0.00       1.0
   macro avg       0.00      0.00      0.00       1.0
weighted avg       0.00      0.00      0.00       1.0
```

```
c:\Users\admin\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\admin\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\admin\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\admin\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\admin\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
c:\Users\admin\AppData\Local\Programs\Python\Python313\Lib\site-packages\sklearn\metrics\_classification.py:1531: UndefinedMetricWarning: Recall is ill-defined and being set to 0.0 in labels with no true samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

Interpretation of Metrics (Preview to Chapter 8.3)

- **Accuracy:** Overall percentage of correct predictions.
- **Precision:** Of all predicted 'positive' reviews, how many were truly positive? (Minimizes False Positives)

- **Recall:** Of all truly 'positive' reviews, how many did the model capture? (Minimizes False Negatives)

Since our dataset is tiny, the metrics here are unreliable, but the process is the perfect foundation for using large data in the next chapter.

5. Summary and Next Steps

- We successfully executed the entire machine learning pipeline for sentiment classification: data splitting, TF-IDF vectorization, model training (Logistic Regression and Naive Bayes), and evaluation.
- This notebook serves as the core template for supervised Text Classification.
- In **Chapter 8**, we will generalize this process and dive deeper into the nuances of **Text Classification**, focusing on metrics, the `Pipeline` object in Scikit-learn, and more comprehensive model evaluation.

Key Takeaways

- **ML Pipeline Mastery:** We successfully implemented the complete machine learning pipeline for sentiment analysis, from data preparation to model evaluation.
- **Algorithm Comparison:** We trained and compared two powerful text classification algorithms - Logistic Regression (linear, interpretable) and Multinomial Naive Bayes (probabilistic, baseline standard).
- **Best Practices Implementation:** We reinforced critical ML practices including proper train-test splitting and the fit_transform/transform distinction for preventing data leakage.
- **Performance Evaluation:** We learned to assess model performance using standard metrics (accuracy, precision, recall, F1-score) for comprehensive evaluation.

Next Notebook Preview

- With ML-based sentiment analysis mastered, we're ready to explore **advanced text classification** techniques and evaluation methods.
- The next notebook will dive into **comprehensive text classification**, featuring scikit-learn pipelines, advanced metrics, and model optimization strategies.

About This Project

This notebook is part of the **Natural Language Processing using Python Programming for Beginners** repository - a comprehensive, beginner-friendly guide for mastering NLP using Python, NLTK, and SpaCy.

Repository: `NLP`

Author

Prakash Ukhalkar

GitHub prakash-ukhalkar
