Natural Language Processing using Python Programming

Notebook 08.2: Building a Text Classifier (MNB, LR, and SVM)

Python 3.8+ NLTK Latest SpaCy Latest Scikit-learn Latest License MIT

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

Learning Objectives:

- Master implementation and comparison of three fundamental text classification algorithms
- Build production-ready pipelines for Multinomial Naive Bayes, Logistic Regression, and Linear SVM
- Understand algorithm strengths and weaknesses for text classification tasks
- Learn systematic model comparison and performance evaluation techniques
- Establish foundation for advanced model selection and optimization strategies

---

- This notebook applies the Scikit-learn `Pipeline` (Chapter 8.1) to compare three highly effective and common machine learning algorithms for text classification:

  1. **Multinomial Naive Bayes (MNB):** Excellent baseline, simple, and fast.

  2. **Logistic Regression (LR):** A strong, linear classifier that provides good interpretability.

  3. **Support Vector Machines (SVM):** Historically one of the best performers for sparse, high-dimensional data like TF-IDF vectors.

1. Setting up: Libraries and Data

- We load the multi-class **20 Newsgroups** dataset again and reuse the data split from the previous notebook for consistency.

In [1]:
```
# Import necessary libraries for text classification comparison
import pandas as pd                                                 # Data manipu
from sklearn.datasets import fetch_20newsgroups                     # 20 Newsgrou
from sklearn.model_selection import train_test_split                # Data splitt
from sklearn.pipeline import Pipeline                               # Pipeline fo
```

```
from sklearn.feature_extraction.text import TfidfVectorizer      # TF-IDF Vect
from sklearn.naive_bayes import MultinomialNB                    # Multinomial
from sklearn.linear_model import LogisticRegression              # Logistic Re
from sklearn.svm import LinearSVC                                # Linear Supp
from sklearn.metrics import accuracy_score                       # Accuracy ev

# Fetching the same data subset
categories = ['alt.atheism', 'soc.religion.christian', 'comp.graphics', 'rec.
newsgroups_train = fetch_20newsgroups(
    subset='all',                                                # Use the ent
    categories=categories,                                       # Focus on 4
    shuffle=True,                                                # Shuffle the
    random_state=42                                              # For reprodu
)

X_train, X_test, y_train, y_test = train_test_split(
    newsgroups_train.data, newsgroups_train.target, test_size=0.3, random_sta
)

print(f"Data Split Complete. Training samples: {len(X_train)}")
```

```
Data Split Complete. Training samples: 2634
```

---

2. Model 1: Multinomial Naive Bayes (MNB)

- **MNB** is often the first algorithm tried for text.
- It's fast to train and provides a competitive **baseline** performance against which all other models can be judged.
- We combine the TF-IDF vectorizer and the MNB classifier into one pipeline.

In [2]:

```
# Model 1: Multinomial Naive Bayes (MNB)
# Pipeline Creation with TF-IDF and MNB
mnb_pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('clf', MultinomialNB()),
])

# Training
mnb_pipeline.fit(X_train, y_train)

# Prediction and Evaluation
mnb_predictions = mnb_pipeline.predict(X_test)
mnb_accuracy = accuracy_score(y_test, mnb_predictions)

print(f"MNB Pipeline Trained. Test Accuracy: {mnb_accuracy:.4f}")
```

```
MNB Pipeline Trained. Test Accuracy: 0.9167
```

---

3. Model 2: Logistic Regression (LR)

- **Logistic Regression** is a linear model that estimates the probability of a document belonging to a certain class.
- Because it uses regularization by default, it is highly effective and less prone to overfitting than complex non-linear models on high-dimensional text data.

In [3]:
```
# Model 2: Logistic Regression (LR)
# Pipeline Creation with TF-IDF and LR
lr_pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('clf', LogisticRegression(random_state=42, solver='liblinear')),
])

# Training
lr_pipeline.fit(X_train, y_train)

# Prediction and Evaluation
lr_predictions = lr_pipeline.predict(X_test)
lr_accuracy = accuracy_score(y_test, lr_predictions)

print(f"LR Pipeline Trained. Test Accuracy: {lr_accuracy:.4f}")
```

```
LR Pipeline Trained. Test Accuracy: 0.9522
```

---

4. Model 3: Support Vector Machine (SVM) - LinearSVC

- SVMs, particularly the linear implementation (`LinearSVC`), are known for finding the optimal hyperplane to separate classes.
- Historically, **Linear SVMs** have been considered state-of-the-art for sparse text classification due to their effectiveness in high-dimensional spaces.

In [4]:
```
# Model 3: Support Vector Machine (SVM)
# Pipeline Creation with TF-IDF and SVM
svm_pipeline = Pipeline([
    ('tfidf', TfidfVectorizer()),
    ('clf', LinearSVC(random_state=42, dual=True)),
])

# Training
svm_pipeline.fit(X_train, y_train)

# Prediction and Evaluation
svm_predictions = svm_pipeline.predict(X_test)
svm_accuracy = accuracy_score(y_test, svm_predictions)

print(f"SVM Pipeline Trained. Test Accuracy: {svm_accuracy:.4f}")
```

```
SVM Pipeline Trained. Test Accuracy: 0.9761
```

---

5. Comparison of Algorithm Performance

- We collect the results to compare the models trained with identical features and data splits.

In [5]:
```
# Summary of Results
results = {
    'Model': ['Multinomial Naive Bayes', 'Logistic Regression', 'Linear SVM'],
    'Test Accuracy': [mnb_accuracy, lr_accuracy, svm_accuracy]
}

df_results = pd.DataFrame(results)
df_results = df_results.sort_values(by='Test Accuracy', ascending=False).rese

print("\n--- Model Comparison ---")
print(df_results)
```

```
--- Model Comparison ---
                     Model  Test Accuracy
0               Linear SVM       0.976085
1      Logistic Regression       0.952170
2  Multinomial Naive Bayes       0.916740
```

> **Data Scientist's Insight:** For text classification using TF-IDF, **Linear SVM** and **Logistic Regression** often outperform Naive Bayes, particularly as the complexity of the data increases. Naive Bayes, while fast, makes a strong assumption about feature independence that is often violated in language.

6. Summary and Next Steps

- We successfully built and compared three foundational text classifiers using the efficient Scikit-learn `Pipeline` structure.
- We now have a champion model based on **Accuracy**.
- However, relying solely on accuracy can be misleading, especially with imbalanced data.
- In the next notebook (**8.3**), we will learn how to properly evaluate our models using essential metrics like **Precision, Recall, F1 Score, and the Confusion Matrix**.

Key Takeaways

- **Algorithm Comparison Mastery:** We successfully implemented and compared three fundamental text classification algorithms using identical pipeline structures for fair evaluation.

- **Baseline Establishment:** We learned that Multinomial Naive Bayes serves as an excellent baseline due to its speed and simplicity, while understanding its feature independence assumptions.
- **Advanced Algorithms:** We implemented Logistic Regression and Linear SVM, understanding their strengths in high-dimensional text data and regularization capabilities.
- **Performance Insights:** We discovered that Linear SVM and Logistic Regression often outperform Naive Bayes for complex text classification tasks.

Next Notebook Preview

- With multiple models trained and compared, we're ready to dive deeper into **comprehensive model evaluation**.
- The next notebook will explore **advanced evaluation metrics** including Precision, Recall, F1-Score, and Confusion Matrix analysis for thorough model assessment.

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

Built with ❤️ for the Python community