SENTIMENT ANALYSIS PROJECT DOCUMENTATION

Author:

APDOOLMAJEED HAMZA (apdoolhamza)

Project:

Live Demo | GitHub | Notebook

Table of Contents:

INTRODUCTION

What is Sentiment Analysis?

Sentiment analysis is a technique in machine learning (ML) that helps computers understand whether a piece of text expresses positive, negative, or neutral feelings. Imagine reading thousands of movie reviews manually determining if they're happy or unhappy would take forever. Sentiment analysis automates this by training a model on labeled data (e.g., "This movie is great!" labeled as positive) to predict sentiments in new texts.

This project focuses on binary sentiment analysis: classifying movie reviews as positive or negative. It's a common starting point for natural language processing (NLP) tasks and has real-world applications like monitoring customer feedback on social media, analyzing product reviews, or improving recommendation systems.

Why This Project?

The idea behind this project is to build a simple yet effective sentiment classifier using accessible tools. We use scikit-learn, a popular Python library for ML, because it's fast, explainable, and doesn't require powerful hardware like GPUs. This makes it ideal for beginners or production environments where speed and low resources matter. We'll cover everything from data loading to deployment, ensuring you can replicate and extend it.

This document explains each step in plain language, with code snippets and visuals for clarity. By the end, you'll understand not just *how* it works, but *why* we make certain choices.

PROJECT OBJECTIVES

- **Primary Goal**: Build an accurate model to classify movie reviews as positive or negative.
- **Secondary Goals**:
    - Demonstrate a full ML pipeline: from data cleaning to evaluation.
    - Use professional best practices like cross-validation, tuning, and calibration.
    - Provide deployment tips for real-world use.
- **Success Metrics**: Aim for 85–90% accuracy, high F1-score (balances precision and recall), and well-calibrated probabilities.

DATASET DESCRIPTION

We use the **IMDB Movie Reviews Dataset**, a standard benchmark for sentiment analysis. It contains 50,000 reviews from the Internet Movie Database (IMDB), split evenly:

- 25,000 training reviews.
- 25,000 testing reviews.
- Balanced classes: 50% positive, 50% negative.

Source: Available publicly at this GitHub link.
Format: CSV file with two columns: `review` (text) and `sentiment` (positive/negative).
Example Rows:

- Positive: “This film was a masterpiece with brilliant acting.”
- Negative: “Boring plot and poor direction waste of time.”

Why This Dataset? It’s real-world, diverse (various movie genres), and challenging due to sarcasm, slang, and long texts. No major issues like missing values, but texts need cleaning for noise (e.g., HTML tags).

DATA PREPROCESSING

Preprocessing turns raw text into clean data ready for ML. Raw reviews often have noise or irrelevant symbols that confuse models.

Steps:

1. **Load Data**: Use pandas to read the CSV.
2. **Minimal Cleaning Function**:
    - Remove line breaks and tabs.
    - Remove HTML tags.
    - Normalize multiple spaces to single.
3. **Apply to Dataset**: Create a new column `clean_text`.

Why These Steps? Clean data improves model accuracy by focusing on meaningful words.

Code Snippet:

```
import pandas as pd
import re

def clean_text(text):
    text = re.sub(r'<.*?>', '', text)  # Remove HTML

     text = text.replace("\n", " ").replace("\r", " ").replace("\t", " ")  # Remove line breaks and tabs

     text = re.sub(r'\s+', ' ', text).strip() # Normalize multiple spaces to single

     return text

url_sent = "https://raw.githubusercontent.com/laxmimerit/All-CSV-ML-Data-Files-Download/master/IMDB-Dataset.csv"
df_sent = pd.read_csv(url_sent)
df_sent['clean_text'] = df_sent['review'].apply(professional_clean)
```

EXPLORATORY DATA ANALYSIS (EDA)

EDA helps understand the data before modeling. We visualize to spot patterns, imbalances, or issues.

Key Visuals:

1. **Class Distribution**: Bar plot showing balanced classes (50/50).
2. **Text Length Histogram**: Positive reviews are often shorter; negatives might be longer (people rant more).
3. **Word Clouds**: Common words in positive (e.g., “great”, “love”) vs. negative (e.g., “bad”, “waste”) reviews.

Code Snippet:

```
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud

# Class balance
sns.countplot(data=df_sent, x='sentiment')
plt.title("Sentiment Class Distribution")
plt.show()

# Text length
df_sent['text_length'] = df_sent['clean_text'].str.len()
sns.histplot(df_sent, x='text_length', hue='sentiment', bins=50,
kde=True)
plt.title("Text Length Distribution by Sentiment")
plt.show()

# Word clouds (function as in code)
pos_texts = df_sent[df_sent['sentiment']=='positive']['clean_text']
generate_wordcloud(pos_texts, "Positive Reviews Word Cloud")  # Repeat
for negative
```

Insights: No severe imbalance, but word clouds reveal sentiment-specific vocabulary, guiding feature engineering.

FEATURE ENGINEERING

Feature:

TF-IDF Vectorization: Converts text to numbers. TF-IDF (Term Frequency-Inverse Document Frequency) weighs words by importance (common words like “movie” get low weight; rare ones like “masterpiece” get high).

- Parameters: Up to 30,000 features, bigrams (word pairs), min_df=5 (ignore rare words).

MODEL ARCHITECTURE

We use a Pipeline with:

- TF-IDF for text features.
- LinearSVC (Support Vector Classifier): A linear model good for text classification. It’s fast and handles high-dimensional data.

Train/Test Split: 80/20, stratified for balance.

Code Snippet:

```
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.pipeline import Pipeline

X = df_sent['clean_text']
y = df_sent['sentiment'].map({'positive': 1, 'negative': 0})
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, stratify=y)

pipe_sent = Pipeline([
    ('tfidf', TfidfVectorizer(max_features=30000, ngram_range=(1,2),
min_df=5, max_df=0.9, sublinear_tf=True)),
    ('clf', LinearSVC(C=0.8, dual=False, class_weight='balanced',
```

```
max_iter=2000, random_state=42))
])
```

Why LinearSVC? It's efficient for sparse text data and often beats more complex models on benchmarks.

HYPERPARAMETER TUNING

Tuning finds the best settings. We use **HalvingGridSearchCV**: Efficient grid search that halves candidates early.

Parameters Tuned:

- TF-IDF: ngram_range, max_features.
- SVC: C (regularization).

Code Snippet:

```
from sklearn.model_selection import HalvingGridSearchCV

param_grid = {
    'tfidf__ngram_range': [(1,1), (1,2)],
    'tfidf__max_features': [20000, 30000],
    'clf__C': [0.5, 0.8, 1.2]
}

search = HalvingGridSearchCV(pipe_sent, param_grid, cv=3,
scoring='f1', n_jobs=-1)
search.fit(X_train, y_train)
best_pipe = search.best_estimator_
```

Cross-Validation: 5-fold CV ensures robust scores.

MODEL CALIBRATION

Models output probabilities, but they might be overconfident. **Isotonic Calibration** adjusts them to match true likelihoods.

Code Snippet:

```
from sklearn.calibration import CalibratedClassifierCV

calibrated = CalibratedClassifierCV(best_pipe, method='isotonic',
cv=3)
calibrated.fit(X_train, y_train)
```

Why? Better for decisions, e.g., flag reviews with >80% confidence.

MODEL EVALUATION

Use metrics like accuracy, precision, recall, F1, ROC-AUC. Visuals: Confusion matrix, ROC curve.

Code Snippet:

```
from sklearn.metrics import classification_report, roc_auc_score,
roc_curve, confusion_matrix

y_pred = calibrated.predict(X_test)
y_proba = calibrated.predict_proba(X_test)[:,1]

print(classification_report(y_test, y_pred))
print(f"ROC-AUC: {roc_auc_score(y_test, y_proba):.4f}")

# Confusion Matrix & ROC Plot (as in code)
```

Typical Results: ~88% accuracy, high AUC (~0.97).

MODEL DEPLOYMENT AND USAGE

1. **Save Model**: Use joblib for efficiency.

   ```
   import joblib
   joblib.dump(calibrated, "sentiment_model_calibrated.joblib")
   ```

2. **Inference**: Load and predict.

```
model = joblib.load("sentiment_model_calibrated.joblib")
prob = model.predict_proba(["Great movie!"])[0][1]
```

3. **Gradio UI**: Quick demo interface.

CONCLUSION

This project demonstrates a complete, professional sentiment analysis pipeline using scikit-learn. It achieves strong results on the IMDB dataset while being efficient and explainable. Extend it by adding more features or fine-tuning transformers.

REFERENCES

- IMDB Dataset: Maas et al., 2011.
- scikit-learn Documentation: https://scikit-learn.org