Twitter Sentiment Analysis Report

This report summarizes the Jupyter Notebook, which implements a sentiment analysis system specifically for Twitter data. The notebook covers data loading from multiple sources, comprehensive text preprocessing tailored for social media, model training using a Linear Support Vector Classifier, and deployment of an interactive web interface using Gradio.

1. Project Objective

The primary objective of this project is to build and evaluate a machine learning model capable of classifying the sentiment of tweets into one of four categories: Positive, Neutral, Negative, or Irrelevant. A secondary goal is to provide an intuitive web-based tool for users to input tweets and receive immediate sentiment predictions.

2. Setup and Library Imports

The notebook begins by setting up the environment and importing necessary Python libraries.

- **NLTK Downloads**: Essential NLTK data packages (punkt_tab, stopwords, wordnet) are downloaded for text processing tasks such as tokenization, stop word removal, and lemmatization.

```
import nltk
nltk.download('punkt_tab')
nltk.download('stopwords')
nltk.download('wordnet')
```

- **Core Libraries**: Standard libraries for data manipulation, regular expressions, text feature extraction, machine learning models, and evaluation metrics are imported:

```
import pandas as pd
import numpy as np
import re
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import train_test_split
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, classification_report
```

3. Data Loading and Preparation

The project utilizes two separate datasets for training and validation, which are then combined.

- **Data Loading**: The twitter_training.csv and twitter_validation.csv files are loaded. It's noted that ISO-8859-1 encoding is used, which is common for datasets with special characters.

```
train_df = pd.read_csv('/content/twitter_training.csv', encoding='ISO-8859-1')
valid_df = pd.read_csv('/content/twitter_validation.csv', encoding='ISO-8859-1')
```

- **Column Renaming**: Columns in both DataFrames are renamed to ensure consistency and clarity: 'ID', 'Entity', 'Sentiment', and 'Text'.

```
train_df.columns = ['ID', 'Entity', 'Sentiment', 'Text']
valid_df.columns = ['ID', 'Entity', 'Sentiment', 'Text']
```

- **Data Concatenation**: The training and validation DataFrames are concatenated into a single DataFrame (df) for unified processing.

```
df = pd.concat([train_df, valid_df], ignore_index=True)
```

- **Missing Value Handling**: Any rows containing missing values (NaN) across the combined dataset are removed.

```
df.dropna(inplace=True)
```

4. Text Preprocessing

A specialized preprocess_text function is defined to clean and normalize tweet text, addressing common characteristics of social media data.

- **Preprocessing Steps**:
  1. **Remove URLs**: Removes any web links (e.g., http://..., https://...).

2. **Remove Mentions**: Removes Twitter mentions (e.g., @username).
3. **Remove Hashtags**: Removes hashtags (e.g., #hashtag).
4. **Remove Non-Alphanumeric Characters**: Keeps only word characters and spaces, removing punctuation and special symbols.
5. **Lowercase Conversion**: Converts all text to lowercase.
6. **Tokenization**: Splits the text into individual words using nltk.word_tokenize.
7. **Stop Word Removal**: Filters out common English stop words.
8. **Lemmatization**: Reduces words to their base form using WordNetLemmatizer.
9. **Join Tokens**: Reconstructs the cleaned tokens into a single string.

```
def preprocess_text(text):
    text = str(text) # Ensure text is string
    text = re.sub(r'http\S+', '', text) # Remove URLs
    text = re.sub(r'@[A-Za-z0-9_]+', '', text) # Remove mentions
    text = re.sub(r'#[A-Za-z0-9_]+', '', text) # Remove hashtags
    text = re.sub(r'[^a-zA-Z\s]', '', text) # Keep only alphabetic characters and
spaces
    text = text.lower() # Convert to lowercase

    tokens = nltk.word_tokenize(text) # Tokenize
    stopwords_list = set(stopwords.words('english'))
    tokens = [token for token in tokens if token not in stopwords_list] # Remove
stopwords

    lemmatizer = WordNetLemmatizer()
    tokens = [lemmatizer.lemmatize(token) for token in tokens] # Lemmatize

    return ' '.join(tokens)

df['clean_text'] = df['Text'].apply(preprocess_text)
```

The clean_text column is created to store the processed tweet content.

5. Data Splitting and Feature Extraction

The cleaned text data is prepared for training the machine learning model.

- **Train-Test Split**: The dataset is split into training and testing sets. clean_text serves as the features (X), and Sentiment is the target variable (y). A test size of 20% is used, with random_state=42 for consistent splitting.

```
X_train, X_test, y_train, y_test = train_test_split(
    df['clean_text'], df['Sentiment'], test_size=0.2, random_state=42
```

)

- **TF-IDF Vectorization**: TfidfVectorizer is applied to convert the textual data into numerical TF-IDF (Term Frequency-Inverse Document Frequency) feature vectors. This method assigns weights to words based on their frequency in a document and rarity across the entire corpus. max_features is set to 5000 to limit the vocabulary to the most significant terms.

  tfidf_vectorizer = TfidfVectorizer(max_features=5000)
  X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
  X_test_tfidf = tfidf_vectorizer.transform(X_test)

6. Model Training and Evaluation

A Linear Support Vector Classifier (SVC) is employed for the multi-class sentiment classification.

- **Model Initialization and Training**: A LinearSVC model is instantiated and trained using the TF-IDF transformed training data.

  svm_classifier = LinearSVC()
  svm_classifier.fit(X_train_tfidf, y_train)

- **Prediction**: The trained model makes predictions on the unseen test set.

  y_pred = svm_classifier.predict(X_test_tfidf)

- **Evaluation Metrics**: The model's performance is assessed using accuracy score and a detailed classification report, which includes precision, recall, and f1-score for each sentiment class.

  print("Accuracy:", accuracy_score(y_test, y_pred))
  print(classification_report(y_test, y_pred))

  The evaluation results are as follows:
  Accuracy: 0.6955797053136876

	precision	recall	f1-score	support

Irrelevant	0.68	0.55	0.61	2636
Negative	0.69	0.80	0.74	4439
Neutral	0.70	0.64	0.67	3681
Positive	0.70	0.73	0.72	4243
accuracy			0.70	14999
macro avg	0.69	0.68	0.68	14999
weighted avg	0.70	0.70	0.69	14999

The model achieves an overall accuracy of approximately **69.56%**. It shows relatively balanced performance across the "Negative", "Neutral", and "Positive" classes, with slightly lower recall for "Irrelevant" tweets.

7. Interactive Interface with Gradio

To provide an accessible way to interact with the trained model, a web interface is built using Gradio.

- **Sentiment Prediction Function**: The predict_sentiment function takes a raw tweet as input, preprocesses it using the defined preprocess_text function, transforms it using the tfidf_vectorizer, and then uses the svm_classifier to predict the sentiment. The predicted sentiment label is returned directly.

```
def predict_sentiment(tweet):
    processed_tweet = preprocess_text(tweet)
    tweet_tfidf = tfidf_vectorizer.transform([processed_tweet])
    prediction = svm_classifier.predict(tweet_tfidf)[0]
    return f"{prediction}"
```

- **Gradio Interface Setup**: A gr.Interface is configured to create the interactive web application:
  - fn: The predict_sentiment function.
  - inputs: A gr.Textbox for user input, labeled "Enter a Tweet".
  - outputs: A gr.Textbox to display the "Sentiment Prediction".
  - title: "Twitter Sentiment Analysis".
  - description: "Enter a tweet and get its sentiment: Positive, Neutral, Negative or Irrelevant."
  - share=True: This enables the generation of a public, shareable link for the interface.

```
import gradio as gr
interface = gr.Interface(
    fn=predict_sentiment,
    inputs=gr.Textbox(label="Enter a Tweet"),
    outputs=gr.Textbox(label="Sentiment Prediction"),
    title="Twitter Sentiment Analysis",
    description="Enter a tweet and get its sentiment: Positive, Neutral, Negative
or Irrelevant.",
)
interface.launch(share=True)
```

This launches a web application where users can input a tweet and receive its classified sentiment.

8. Conclusion and Future Work

The notebook successfully demonstrates a complete pipeline for Twitter sentiment analysis, from data acquisition and specialized preprocessing to model training and interactive deployment. The LinearSVC model achieves an accuracy of approximately 69.56% in classifying tweet sentiments.

Potential areas for future improvements include:

- **Advanced Feature Engineering**: Exploring more sophisticated features beyond TF-IDF, such as word embeddings (Word2Vec, GloVe, FastText) or contextual embeddings (BERT, RoBERTa) to capture deeper semantic meanings.
- **Deep Learning Models**: Implementing neural network architectures like Recurrent Neural Networks (RNNs), Long Short-Term Memory (LSTMs), or transformer models, which are often more effective for complex natural language processing tasks.
- **Hyperparameter Optimization**: Conducting a more thorough search for optimal hyperparameters for both the TfidfVectorizer and the LinearSVC classifier.
- **Handling Imbalance**: If certain sentiment classes are underrepresented, techniques like oversampling (SMOTE) or undersampling could be applied to balance the dataset and improve minority class performance.
- **Domain-Specific Lexicons**: Integrating sentiment lexicons tailored for social media or specific domains to enhance sentiment detection.
- **Ensemble Methods**: Combining predictions from multiple models to potentially achieve higher accuracy and robustness.