Social Media Sentiment Analysis Project

Project Overview

This project aims to analyze public sentiment about various airlines through tweets using Natural Language Processing (NLP) techniques and machine learning. The goal is to classify tweets into three categories: **negative**, **neutral**, and **positive** sentiments. A **Random Forest Classifier** is employed to perform the sentiment classification task. The pipeline covers the entire process from loading data, preprocessing text, building a model, and visualizing results.

In this project, data is sourced from multiple files: one in an SQLite database format and another in CSV format. These are merged, preprocessed, and then used to train the machine learning model.

Table of Contents

---

1. Import Libraries

```
python
Copy code
import warnings
warnings.filterwarnings("ignore")

import sqlite3
import pandas as pd
from sklearn.preprocessing import StandardScaler
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from wordcloud import WordCloud
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk.corpus import stopwords
from nltk import pos_tag
import nltk
```

Explanation:

This section imports the essential libraries required for the project. Libraries like **Pandas** and **NumPy** are used for data manipulation, **Seaborn** and **Matplotlib** for visualizations, and **NLTK** for text preprocessing tasks like tokenization, stop word removal, stemming, and lemmatization.

We also use `nltk.download()` commands to ensure we have access to necessary corpora for tokenization, POS tagging, and stop word filtering.

---

2. Loading and Merging Data

```
python
Copy code
# Load SQLite and CSV data
conn = sqlite3.connect(db_path)
df_sqlite = pd.read_sql_query("SELECT * FROM Tweets", conn)
conn.close()

df_csv = pd.read_csv(csv_file_path)

# Merge datasets on 'tweet_id'
merged_df = pd.merge(df_sqlite, df_csv, on='tweet_id', suffixes=('_sqlite',
'_csv'))

# Save the merged data to a CSV file
merged_df.to_csv(merged_file_path, index=False)
```

Explanation:

We load the data from an **SQLite database** and a **CSV file** containing tweets. These two datasets are merged on the common column `tweet_id`. The resulting merged dataset is saved to a new CSV file for further analysis.

---

3. Data Cleaning and Preprocessing

```
python
Copy code
# Drop duplicates
merged_df.drop_duplicates(inplace=True)

# Handle missing values
threshold = 0.1
for column in merged_df.columns:
    missing_ratio = merged_df[column].isnull().mean()
    if missing_ratio > threshold:
        if merged_df[column].dtype == 'object':
            merged_df[column].fillna('Unknown', inplace=True)
        else:
```

```
            merged_df[column].fillna(merged_df[column].mean(),
inplace=True)
    else:
        merged_df.dropna(subset=[column], inplace=True)

# Convert date columns to datetime
merged_df['tweet_created_sqlite'] =
pd.to_datetime(merged_df['tweet_created_sqlite'], errors='coerce')

# Drop unnecessary columns
columns_to_drop = ['tweet_coord_sqlite', 'tweet_coord_csv']
merged_df.drop(columns=columns_to_drop, inplace=True)
```

Explanation:

This section covers:

1. **Removing duplicates**: We eliminate any duplicate rows in the dataset to ensure data integrity.
2. **Handling missing values**: For columns with more than 10% missing values, imputation strategies are applied. For categorical columns, 'Unknown' is used, while the mean is used for numerical columns.
3. **Datetime conversion**: We convert the column `tweet_created_sqlite` into a proper datetime format.
4. **Column removal**: Unnecessary columns such as `tweet_coord_sqlite` and `tweet_coord_csv` are dropped.

---

4. Exploratory Data Analysis (EDA)

```
python
Copy code
# Sentiment Confidence Score Distribution
sns.histplot(merged_df['airline_sentiment_confidence_sqlite'], bins=30,
kde=True)
plt.title('Distribution of Sentiment Confidence Scores')

# Correlation Heatmap
numeric_cols = merged_df.select_dtypes(include=[np.number]).columns
corr = merged_df[numeric_cols].corr()
sns.heatmap(corr, annot=True, cmap='coolwarm')

# Bar Plot for Categorical Variables
categorical_cols = ['airline_sqlite', 'airline_sentiment_csv']
sns.countplot(data=merged_df, x='airline_sqlite')
```

Explanation:

Exploratory Data Analysis (EDA) helps in understanding the structure and relationships within the data:

- **Sentiment confidence score distribution**: We plot a histogram to examine the spread of confidence levels associated with tweet sentiments.

- **Correlation heatmap**: This heatmap shows correlations between numerical features to reveal any significant relationships.
- **Bar plots for categorical data**: Visualizations of counts for airlines and sentiments provide insights into the distribution of sentiment across different airlines.

---

5. Feature Selection and Engineering

```
python
Copy code
# Selecting features
features = ['airline_sentiment_confidence_sqlite', 'retweet_count_sqlite',
'negativereason_confidence_sqlite']

# New Feature: Text Length
merged_df['text_length'] = merged_df['text_sqlite'].apply(len)
X = merged_df[features]
```

Explanation:

We select features that will be used for model training. Additionally, we create a new feature `text_length`, representing the number of characters in each tweet.

---

6. Text Preprocessing

```
python
Copy code
# Tokenization
merged_df['tokenized_text'] = merged_df['text_sqlite'].apply(word_tokenize)

# Stop Word Removal
stop_words = set(stopwords.words('english'))
merged_df['filtered_text'] = merged_df['tokenized_text'].apply(lambda
tokens: [word for word in tokens if word.lower() not in stop_words])

# Stemming and Lemmatization
stemmer = PorterStemmer()
lemmatizer = WordNetLemmatizer()
merged_df['stemmed_text'] = merged_df['filtered_text'].apply(lambda tokens:
[stemmer.stem(token) for token in tokens])
merged_df['lemmatized_text'] = merged_df['filtered_text'].apply(lambda
tokens: [lemmatizer.lemmatize(token) for token in tokens])

# POS Tagging
merged_df['pos_tags'] = merged_df['filtered_text'].apply(lambda tokens:
pos_tag(tokens))
```

Explanation:

Text preprocessing transforms raw tweet text into a format suitable for machine learning:

- **Tokenization**: Splits text into individual words.
- **Stop word removal**: Filters out common words that do not contribute to sentiment analysis (e.g., 'the', 'and').
- **Stemming and Lemmatization**: Reduce words to their base form to ensure consistency (e.g., 'running' becomes 'run').
- **POS Tagging**: Assigns part-of-speech tags to each token for further text analysis.

---

7. Model Training and Evaluation

```
python
Copy code
# Train-Test Split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=42)

# Random Forest Model
model = RandomForestClassifier(n_estimators=100, random_state=42)
model.fit(X_train, y_train)

# Model Evaluation
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
conf_matrix = confusion_matrix(y_test, y_pred)
```

Explanation:

We split the dataset into training and testing sets and train a **Random Forest Classifier** model. After training, we evaluate the model's accuracy using **accuracy_score** and assess its predictions using a **confusion matrix**.

---

8. Hyperparameter Tuning

```
python
Copy code
# Hyperparameter Tuning with GridSearchCV
param_grid = {'n_estimators': [50, 100, 200], 'max_depth': [10, 20, None]}
grid_search = GridSearchCV(estimator=model, param_grid=param_grid, cv=5)
grid_search.fit(X_train, y_train)

# Best Parameters and Improved Accuracy
best_model = grid_search.best_estimator_
y_pred_best = best_model.predict(X_test)
```

Explanation:

We use **GridSearchCV** to tune the model's hyperparameters, such as `n_estimators` and `max_depth`, to improve the performance of the classifier. The best model is identified and re-evaluated.

---

9. Visualizations

```
python
Copy code
# Sentiment Distribution Bar Plot
sns.barplot(x=sentiment_counts.index, y=sentiment_counts.values)

# Pie Chart for Sentiment
sentiment_pie =
merged_df['airline_sentiment_sqlite'].value_counts().plot.pie(autopct="%1.1
f%%", startangle=90)
```

Explanation:

Various visualizations, such as **bar plots** and **pie charts**, are created to represent the distribution of tweet sentiments and help us visualize the overall performance and behavior of the model.

---

10. Conclusion

In this project, we successfully built a pipeline for **social media sentiment analysis** using machine learning techniques. The pipeline covers data loading, cleaning, preprocessing, model training, and evaluation. **Random Forest Classifier** provided good accuracy for sentiment classification, and hyperparameter tuning further improved the model's performance. Visualizations highlighted the key insights and results.