SENTIMENT ANALYSIS USING TF-IDF VECTORIZER

Problem Statement

To design and implement a ML model to perform sentiment analysis on a given dataset containing textual data and corresponding sentiment labels. The analysis should use TF-IDF vectors to transform the text into numerical features and classify the sentiments effectively.

Data Description

The dataset consists of ~ 1.05million rows and 6 fields:

1. **Polarity** (Column 0):
    - Sentiment label for the tweet:
        - 0: Negative sentiment
        - 2: Neutral sentiment
        - 4: Positive sentiment
2. **Tweet ID** (Column 1):
    - A unique identifier for each tweet.
3. **Date** (Column 2):
    - The timestamp of when the tweet was posted, in the format Day Month Date HH:MM:SS UTC Year (e.g., Sat May 16 23:58:44 UTC 2009).
4. **Query** (Column 3):
    - The search query used to retrieve the tweet. If no query was used, the value is NO_QUERY.
5. **User** (Column 4):
    - The username of the account that posted the tweet (e.g., robotickilldozr).
6. **Text** (Column 5):
    - The content of the tweet, consisting of raw text after emoticons have been removed (e.g., Lyx is cool).

There are no missing values in any of the columns -

```
Missing Values:
 polarity of tweet     0
id of the tweet        0
date of the tweet      0
query                  0
user                   0
text of the tweet      0
```

I cleaned the column, “text of the tweet”, to remove any html tags while keeping only letters, numbers and single spaces. After that the clean text was preprocessed to lemmatize the tokens and then joined into a string.

In the “polarity of tweet” column, there were only 0s and 4s and hence no “neutral” comments. I replaced the 4s with 1s to keep it binary, i.e. 0 = negative, 1 = positive. After splitting the data into 80% train and 20% test data, and using scaling, I used TF-IDF vectorization with max_features = 10000 to reduce dimensionality.

Analysis

To carry out the sentiment analysis, I have used 3 common classifier models for text:

- Logistic regression
- Random Forest
- Support Vector Machine

Initially the SVM model was taking a lot of time to run and hence I switched to LinearSVC, which is optimised for linear kernels and scales much better and thus reduced the operation time. For the performance of the models, I added the accuracy scores and classification report.

Results

Model	Accuracy
Logistic Regression	83.1%
Random Forest	81.6%
Support Vector Machine	82.95%

As we can see from the above table, Logistic Regression performs the best and is just marginally ahead of the SVM model with respect to accuracy. From the classification report below, we observe all 3 models achieved 94% recall for negative sentiment analysis, meaning they correctly identified most negative tweets. Logistic regression had slightly better precision, reducing false positives.

While all 3 models struggled with positive sentiment recall, Logistic regression performed 8% better than Random Forest, with SVM being in the middle at 46%. This suggests, Logistic regression model generalises better to unseen positive tweets. However, the poor positive sentiment recall shows all models missed more than half of the positive tweets, i.e. with recall < 50%.

Classification metrics (by Class)			
Negative sentiment (Class 0)			
Model	**Precision**	**Recall**	**F1-Score**
Logistic Regression	0.86	0.94	0.89
Random Forest	0.84	0.94	0.89
Support Vector Machine	0.85	0.94	0.89
Positive sentiment (Class 1)			
Model	**Precision**	**Recall**	**F1-Score**
Logistic Regression	0.7	0.49	0.58
Random Forest	0.69	0.41	0.51
Support Vector Machine	0.72	0.46	0.56

Model	Macro Avg (F1)	Weighted Avg (F1)
Logistic Regression	0.74	0.82
Random Forest	0.7	0.8
Support Vector Machine	0.73	0.82

Conclusion

One possible reason for such performance on positive sentiments is the class imbalance in the data, i.e. only ~24% of the test samples were positive. Random Forest, despite being an ensemble model, performed worse than Logistoc Regression possibly as TF-IDF may not benefit from Random Forest's feature interactions. Hyperparameter tuning could possibly help, however, I haven't performed it due to the time taken to run these models on the dataset and in a later run I'll try to use different batches to break them down into smaller chunks for faster processing.

I have added a few sample texts ranging from simple ones to more sarcastic tones and at least in 3 of the 4 cases it could correctly identify the sentiment - which seems good performance for a relatively less complex model implementation.

Appendix

```
Logistic Regression Performance:
Accuracy: 0.830908614071478

Classification Report:
              precision    recall  f1-score   support

           0       0.86      0.94      0.89    160000
           1       0.70      0.49      0.58     49715

    accuracy                           0.83    209715
   macro avg       0.78      0.71      0.74    209715
weighted avg       0.82      0.83      0.82    209715
```

```
Random Forest Performance:
Accuracy: 0.8159692916577259

Classification Report:
              precision    recall  f1-score   support

           0       0.84      0.94      0.89    160000
           1       0.69      0.41      0.51     49715

    accuracy                           0.82    209715
   macro avg       0.76      0.68      0.70    209715
weighted avg       0.80      0.82      0.80    209715
```

```
SVM Performance:
Accuracy: 0.8295210166177908

Classification Report:
              precision    recall  f1-score   support

           0       0.85      0.94      0.89    160000
           1       0.72      0.46      0.56     49715

    accuracy                           0.83    209715
   macro avg       0.78      0.70      0.73    209715
weighted avg       0.82      0.83      0.82    209715
```

```
Sample prediction for 'I love this product! It's amazing.':
Positive
>>> print(f"\nSample prediction for '{sample_text2}':")
... print(predict_sentiment(sample_text2, best_model, tfidf_vectorizer, scaler))
...

Sample prediction for 'I work 60 hours a week to be this poor.':
Negative
>>> print(f"\nSample prediction for '{sample_text3}':")
... print(predict_sentiment(sample_text3, best_model, tfidf_vectorizer, scaler))
...

Sample prediction for 'Really, Sherlock? No! You are amazingly clever.':
Positive
>>> sample_text4 = ("Thank you for explaining that my eye cancer isn't going to make me deaf. "
...                 "I feel so fortunate that an intellectual giant like yourself would deign to operate on me.")
>>> print(f"\nSample prediction for '{sample_text4}':")
... print(predict_sentiment(sample_text4, best_model, tfidf_vectorizer, scaler))
...

Sample prediction for 'Thank you for explaining that my eye cancer isn't going to make me deaf. I feel so fortunate that an intellectual giant like
Negative
```