Task Report: Twitter Text Classification

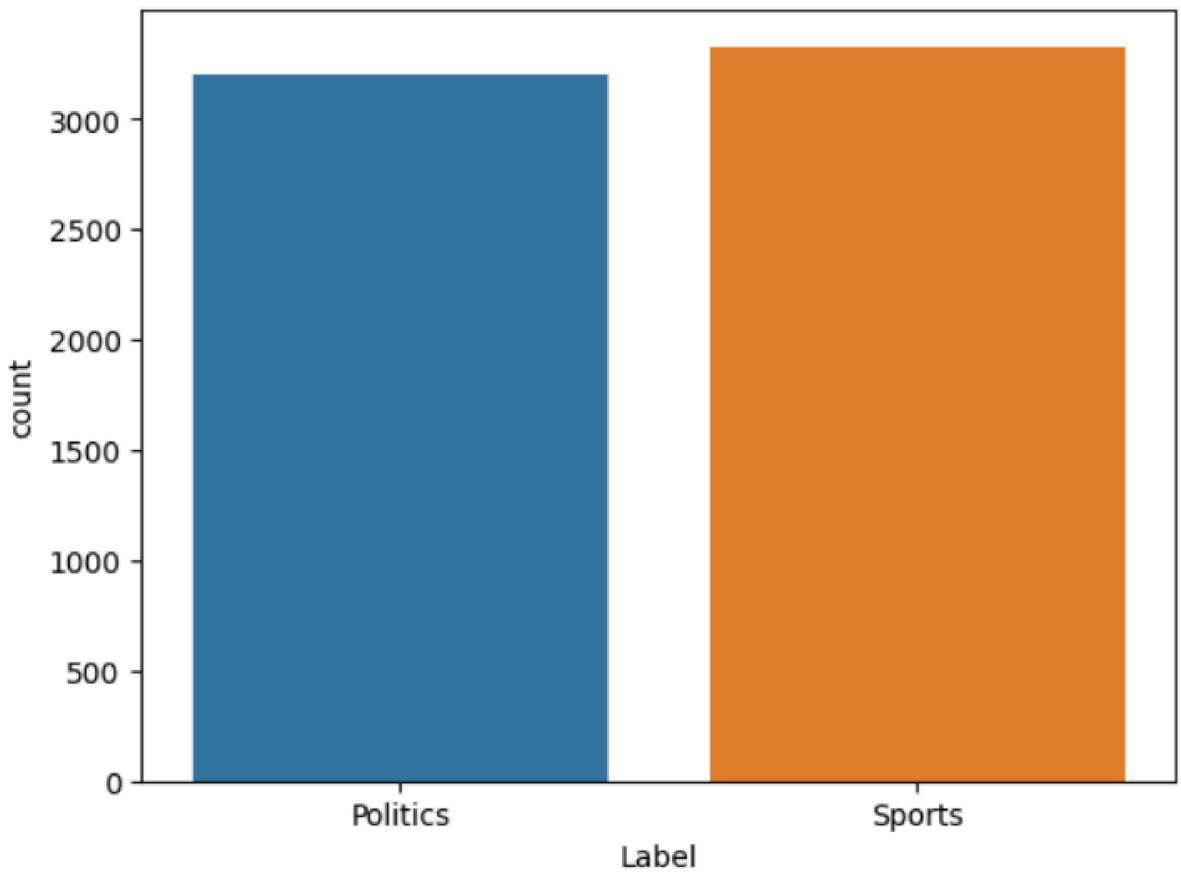
I. Data Exploration:

1. Data Loading and Visualization:

- Loaded the training and test datasets using `pd.read_csv`:

```
train_data = pd.read_csv('/kaggle/input/deeptweets/train.csv')
test_data = pd.read_csv('/kaggle/input/deeptweets/test.csv')
```

- Utilized Seaborn to visualize the distribution of labels in the training set using `sns.countplot`:



II. Data Preprocessing:

1. Word Clouds Visualization:

- Created word clouds for both "Politics" and "Sports" labels to visually explore frequent terms in each category.
- Word clouds were generated using the WordCloud library.

2. Text Processing:

- Defined a text processing function `Text_Processing` to clean and preprocess the tweet text.
- Converted text to lowercase, removed special characters, and URLs using regular expressions.
- Tokenized the text using NLTK's `word_tokenize`.
- Applied stemming using the PorterStemmer to reduce words to their root form.
- Created a new column `ProcessedText` in the training dataset to store the preprocessed text.

```
# Initialization of the stemmer
stemmer = PorterStemmer()

def Text_Processing(text):
    text = text.lower()
    # Remove special characters and URLs
    text = re.sub(r'http\S+', '', text)
    text = re.sub(r'[^a-zA-Z\s]', '', text)
    # Tokenize the text
    text = word_tokenize(text)
    # Stemming words
    text = [stemmer.stem(word) for word in text]
    # Join the processed words back into a sentence
    text = ' '.join(text)
    return text
```

III. Data Splitting:

- Split the dataset into training and testing sets using `train_test_split`:

```
# Splitting the data
X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=42)
```

IV. Model Building:

1. Logistic Regression Pipeline:

- Constructed a machine learning pipeline using `Pipeline` from scikit-learn.
- Utilized a bag-of-words representation with `CountVectorizer` (unigrams and bigrams).
- Employed a logistic regression classifier (`LogisticRegression`) as the predictive model.

```
LR = Pipeline([
    ('bag_of_word' , CountVectorizer(ngram_range=(1,2))),
    ('LR' , LogisticRegression())

])
LR.fit(X_train, y_train)
y_pred = LR.predict(X_test)
print(classification_report(y_test, y_pred))
```

2. Model Training and Evaluation:

- Fitted the pipeline on the training data (`X_train`, `y_train`).
- Made predictions on the test data and printed a classification report using `classification_report`.
- Displayed a confusion matrix to visualize model performance.

V. Results and Analysis:

1. Classification Report:

- Examined precision, recall, and F1-score for both "Politics" and "Sports" labels.
- Assessed the overall model performance on the test set.

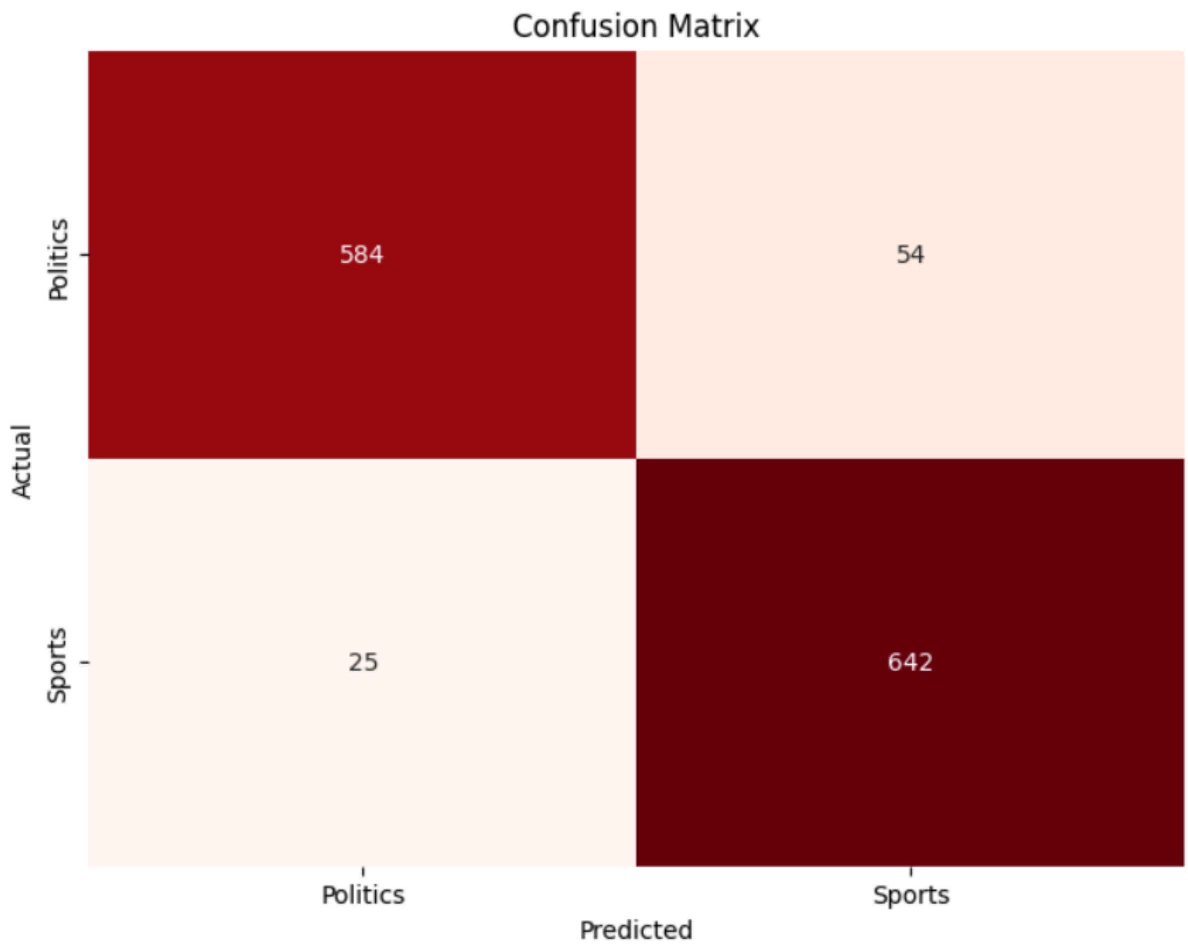
```
              precision    recall  f1-score   support

    Politics       0.96      0.92      0.94       638
      Sports       0.92      0.96      0.94       667

    accuracy                           0.94      1305
   macro avg       0.94      0.94      0.94      1305
weighted avg       0.94      0.94      0.94      1305
```

2. Confusion Matrix Visualization:

- Visualized the confusion matrix using Seaborn's heatmap to understand model predictions.
- The heatmap displays actual vs. predicted labels.



VI. Conclusion:

- Summarized key findings, including model performance metrics.
- Provided insights into the effectiveness of the chosen model and preprocessing techniques.
- Suggested potential areas for improvement, such as experimenting with different vectorization techniques or exploring other classification algorithms.