EDA-Classification

July 24, 2023

[ ]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
```

Step 1: Data Preparation and Splitting
First, let's load the combined dataset and split it into training and test sets.

[ ]:
```
# Load the dataset
data_files = ['stories_art-et-culture.csv', 'stories_economie.csv',␣
 ↪'stories_politique.csv', 'stories_regions.csv', 'stories_societe.csv',␣
 ↪'stories_sport.csv']
df = pd.concat((pd.read_csv(filename) for filename in data_files))
```

[ ]:
```
# Split the data into features (X) and target labels (y)
X = df['story']
y = df['topic']

# Split the data into training (80%) and test (20%) sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,␣
 ↪random_state=42)
```

Step 2: Text Preprocessing
Before training the classifier, we need to preprocess the text data by converting it into numerical representations. We'll use techniques like TF-IDF vectorization to transform the text into numerical features.

[ ]:
```
from sklearn.feature_extraction.text import TfidfVectorizer

# Initialize the TF-IDF vectorizer
vectorizer = TfidfVectorizer()

# Fit and transform the vectorizer on the training set
X_train_tfidf = vectorizer.fit_transform(X_train)

# Transform the test set using the same vectorizer
X_test_tfidf = vectorizer.transform(X_test)
```

Step 3: Train the Classifier
We'll use a popular classifier such as the Multinomial Naive Bayes classifier for this task.

[ ]:
```
from sklearn.naive_bayes import MultinomialNB

# Initialize the Multinomial Naive Bayes classifier
clf = MultinomialNB()

# Train the classifier on the training data
clf.fit(X_train_tfidf, y_train)
```

[ ]:
```
MultinomialNB()
```

Step 4: Evaluate the Classifier
Now, let's evaluate the classifier's performance on the test set and calculate precision, recall, F1-score, and accuracy for each class and the overall test data.

[ ]:
```
from sklearn.metrics import classification_report, accuracy_score

# Make predictions on the test set
y_pred = clf.predict(X_test_tfidf)

# Calculate classification metrics
class_report = classification_report(y_test, y_pred)
accuracy = accuracy_score(y_test, y_pred)

print("Classification Report:")
print(class_report)
print("Accuracy:", accuracy)
```

```
Classification Report:
               precision    recall  f1-score   support

art-et-culture       0.99      0.93      0.96       215
      economie       0.86      0.89      0.88       222
     politique       0.79      0.94      0.86       176
       regions       0.88      0.66      0.75       204
       societe       0.69      0.79      0.74       189
         sport       0.99      0.97      0.98       194

      accuracy                           0.86      1200
     macro avg       0.87      0.86      0.86      1200
  weighted avg       0.87      0.86      0.86      1200

Accuracy: 0.8625
```

Interpretation of Metrics: - **Precision:** Precision is the ratio of true positive predictions to the total predicted positive instances. It measures the model's ability to avoid false positives. In the context of this task, precision indicates the percentage of correctly predicted articles for each class (topic).

- **Recall:** Recall (also known as sensitivity) is the ratio of true positive predictions to the total

actual positive instances. It measures the model's ability to find all positive instances. In this task, recall shows the percentage of correctly predicted articles out of all articles belonging to each class.

- **F1-score:** The F1-score is the harmonic mean of precision and recall. It provides a balanced measure that considers both precision and recall. It is useful when there is an uneven class distribution.
- **Accuracy:** Accuracy is the overall correct predictions out of all predictions. It provides an overall performance measure for the entire test set.

Enhancements for Better Results: 1. **Hyperparameter Tuning:** Experiment with different hyperparameters for the classifier, vectorizer, or other model components using techniques like GridSearchCV to find the optimal configuration.

2. **Use Advanced Classifiers:** Try using more advanced classifiers like Support Vector Machines (SVM), Random Forest, or Gradient Boosting, as they may capture complex patterns in the text data better.
3. **Feature Engineering:** Explore additional text preprocessing techniques like stemming, lemmatization, or using n-grams to create more informative features.
4. **Handling Imbalanced Data:** If there's an imbalance in the number of examples per class, consider using techniques like oversampling, undersampling, or using class weights to address the class imbalance issue.
5. **Ensemble Methods:** Combine multiple classifiers using ensemble methods like Voting or Stacking to improve overall performance.
6. **Domain-specific Features:** Consider incorporating domain-specific features (e.g., word embeddings or topic features) that can provide valuable information for the classifier.
7. **Error Analysis:** Analyze the misclassified examples to identify patterns or common sources of errors, and make necessary adjustments to the model or data preprocessing.