```
import pandas as pd

# Load the dataset
file_path = '/content/merged_data.csv'
data = pd.read_csv(file_path)

# Display the first few rows to inspect the structure
data.head()
```

	category	sub_category	crimeaditionalinfo
0	Online and Social Media Related Crime	Cyber Bullying Stalking Sexting	I had continue received random calls and abusi...
1	Online Financial Fraud	Fraud CallVishing	The above fraudster is continuously messaging ...
2	Online Gambling Betting	Online Gambling Betting	He is acting like a police and demanding for m...
3	Online and Social Media Related Crime	Online Job Fraud	In apna Job I have applied for job interview f...
4	Online Financial Fraud	Fraud CallVishing	I received a call from lady stating that she w...

```
!pip install nltk
import nltk

# Download 'punkt_tab' specifically
nltk.download('punkt_tab')

# ... (rest of your code remains the same)
```

```
Requirement already satisfied: nltk in /usr/local/lib/python3.10/dist-packages (3.9.1)
Requirement already satisfied: click in /usr/local/lib/python3.10/dist-packages (from nltk) (8.1.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.10/dist-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.10/dist-packages (from nltk) (2024.9.11)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from nltk) (4.66.6)
[nltk_data] Downloading package punkt_tab to /root/nltk_data...
[nltk_data]   Package punkt_tab is already up-to-date!
True
```

```
# Install and download NLTK resources
import nltk
nltk.download('punkt')  # For tokenization
nltk.download('stopwords')  # For stop words
nltk.download('wordnet')  # For lemmatization

# Import necessary libraries
import pandas as pd
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

# Load the dataset (update the file path to your uploaded file)
file_path = "/content/merged_data.csv"  # Change if needed
data = pd.read_csv(file_path)

# Initialize lemmatizer and stop words
lemmatizer = WordNetLemmatizer()
stop_words = set(stopwords.words('english'))

# Define text preprocessing function
def preprocess_text(text):
    text = text.lower()  # Convert text to lowercase
    tokens = word_tokenize(text)  # Tokenize text
    tokens = [lemmatizer.lemmatize(word) for word in tokens if word.isalpha() and word not in stop_words]  # Remove stop words & lemmat
    return ' '.join(tokens)

# Apply preprocessing to the specific column
data['processed_text'] = data['crimeaditionalinfo'].astype(str).apply(preprocess_text)

# Save the processed data
data.to_csv("/content/processed_dataset.csv", index=False)

# Display the first few rows of processed data
data[['crimeaditionalinfo', 'processed_text']].head()
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
```

	crimeaditionalinfo	processed_text
0	I had continue received random calls and abusi...	continue received random call abusive message ...
1	The above fraudster is continuously messaging ...	fraudster continuously messaging asking pay mo...
2	He is acting like a police and demanding for m...	acting like police demanding money adding sect...
3	In apna Job I have applied for job interview f...	apna job applied job interview telecalling res...
4	I received a call from lady stating that she w...	received call lady stating send new phone vivo...

```
# from sklearn.model_selection import train_test_split

# # Define the feature and target variables
# # Assuming 'processed_text' is the feature and another column, like 'label', is the target
# # Replace 'label' with the actual column name if your dataset has one
# X = data['processed_text']  # Features
# y = data['label'] if 'label' in data.columns else None  # Replace 'label' with actual target column, if present

# # Split into train and test sets (80-20 split)
# if y is not None:
#     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
# else:
#     # If no labels, split only the feature set
#     X_train, X_test = train_test_split(X, test_size=0.2, random_state=42)
#     y_train, y_test = None, None

# # Save the splits into separate files for reuse
# X_train.to_csv('/content/X_train.csv', index=False)
# X_test.to_csv('/content/X_test.csv', index=False)

# if y_train is not None:
#     pd.DataFrame(y_train).to_csv('/content/y_train.csv', index=False)
#     pd.DataFrame(y_test).to_csv('/content/y_test.csv', index=False)

# print("Train-test split completed. Files saved as 'X_train.csv', 'X_test.csv', and optionally 'y_train.csv' and 'y_test.csv'")
```

```
from sklearn.model_selection import train_test_split

# Assuming the processed dataset is ready and stored in `data`
# If you have a 'label' column for the target, update it accordingly
# Replace 'processed_text' with the feature column and 'label' with the target column if needed
features = data['processed_text']
target = data['label'] if 'label' in data.columns else None  # Replace 'label' with your target column if it exists

# Split the dataset (80% training, 20% testing)
if target is not None:
    X_train, X_test, y_train, y_test = train_test_split(features, target, test_size=0.2, random_state=42)
else:
    X_train, X_test = train_test_split(features, test_size=0.2, random_state=42)
    y_train, y_test = None, None

# Combine the splits for easy ML application
train_data = pd.DataFrame({'processed_text': X_train})
test_data = pd.DataFrame({'processed_text': X_test})

if y_train is not None:
    train_data['label'] = y_train
    test_data['label'] = y_test

# Save to CSV
train_data.to_csv('/content/train_data.csv', index=False)
test_data.to_csv('/content/test_data.csv', index=False)

print("Dataset split completed.")
print("Train data saved to 'train_data.csv'.")
print("Test data saved to 'test_data.csv'.")
```

```
Dataset split completed.
Train data saved to 'train_data.csv'.
Test data saved to 'test_data.csv'.
```

```
# from sklearn.feature_extraction.text import TfidfVectorizer
# from sklearn.model_selection import train_test_split
# from sklearn.linear_model import LogisticRegression
# from sklearn.metrics import accuracy_score, classification_report

# # Load the dataset
# file_path = '/content/processed_dataset.csv'  # Use the processed dataset
# data = pd.read_csv(file_path)

# # Check if the dataset has labels
# if 'label' not in data.columns:
#     raise ValueError("No target column ('label') found in the dataset. Please ensure your dataset includes labels.")

# # Split data into features and target
# X = data['processed_text']
# y = data['categories']  # Replace 'label' with your target column if necessary

# # Split into training and testing sets
# X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# # Vectorize the text data
# vectorizer = TfidfVectorizer(max_features=5000)  # You can adjust max_features
# X_train_vec = vectorizer.fit_transform(X_train)
# X_test_vec = vectorizer.transform(X_test)

# # Train a Logistic Regression model
# model = LogisticRegression()
# model.fit(X_train_vec, y_train)

# # Predict on test data
# y_pred = model.predict(X_test_vec)

# # Evaluate the model
# accuracy = accuracy_score(y_test, y_pred)
# report = classification_report(y_test, y_pred)

# print(f"Accuracy: {accuracy:.2f}")
# print("Classification Report:")
# print(report)
```

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, classification_report

# Load the dataset
file_path = '/content/processed_dataset.csv'  # Use the processed dataset
data = pd.read_csv(file_path)

# Ensure there are no missing values in the target column
data = data.dropna(subset=['sub_category'])  # Replace 'sub_category' with your actual target column name

# Split data into features and target
X = data['processed_text']
y = data['sub_category']  # Replace with your actual target column name

# Handle NaN values in the feature column
X = X.fillna('')

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Vectorize the text data
vectorizer = TfidfVectorizer(max_features=5000)  # You can adjust max_features
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Train a Logistic Regression model
model = LogisticRegression()
model.fit(X_train_vec, y_train)

# Predict on test data
y_pred = model.predict(X_test_vec)

# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)

print(f"Accuracy: {accuracy:.2f}")
print("Classification Report:")
print(report)
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
Accuracy: 0.56
Classification Report:
                                                                  precision    recall  f1-score   support

                Against Interest of sovereignty or integrity of India       0.00      0.00      0.00         1
                             Business Email CompromiseEmail Takeover       0.43      0.04      0.07        80
                                        Cheating by Impersonation       0.19      0.03      0.05       525
                                   Computer Generated CSAM/CSEM       0.00      0.00      0.00         1
                                          Cryptocurrency Fraud       0.52      0.49      0.50       111
                                 Cyber Bullying  Stalking  Sexting       0.62      0.68      0.65      1135
                                                  Cyber Terrorism       0.00      0.00      0.00        42
                             Damage to computer computer systems etc       0.00      0.00      0.00        32
                                                Data Breach/Theft       0.14      0.13      0.14       127
                                DebitCredit Card FraudSim Swap Fraud       0.72      0.66      0.69      2818
                                               DematDepository Fraud       0.00      0.00      0.00       186
Denial of Service (DoS)/Distributed Denial of Service (DDOS) attacks       0.19      0.19      0.19       145
                                                     EMail Phishing       0.50      0.04      0.08        49
                                               EWallet Related Fraud       0.66      0.37      0.48      1087
                                                      Email Hacking       0.67      0.34      0.45       104
                                        FakeImpersonating Profile       0.56      0.42      0.48       609
                                                Fraud CallVishing       0.32      0.24      0.27      1488
                                               Hacking/Defacement       0.14      0.26      0.18       144
                                              Impersonating Email       0.00      0.00      0.00        13
                                   Internet Banking Related Fraud       0.69      0.55      0.61      2376
                                              Intimidating Email       0.00      0.00      0.00         6
                                                  Malware Attack       0.09      0.08      0.09       144
                                       Online Gambling  Betting       0.67      0.04      0.07       113
                                                Online Job Fraud       0.32      0.16      0.22       228
                                       Online Matrimonial Fraud       0.00      0.00      0.00        37
                                             Online Trafficking       0.00      0.00      0.00        51
                                                          Other       0.36      0.46      0.40      2878
                                   Profile Hacking Identity Theft       0.57      0.42      0.49       610
                                Provocative Speech for unlawful acts       0.71      0.12      0.21        97
```

```
                                                  SQL Injection       0.17      0.14      0.16       154
                                Tampering with computer source documents       0.14      0.11      0.12       147
                                                   UPI Related Frauds       0.64      0.85      0.73      7206
                                       Unauthorised AccessData Breach       0.29      0.19      0.23       273
                                            Website DefacementHacking       0.00      0.00      0.00        23

                                                            accuracy                           0.56     23218
                                                           macro avg       0.29      0.20      0.21     23218
                                                        weighted avg       0.54      0.56      0.54     23218

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
!pip install xgboost catboost
```

```
Requirement already satisfied: xgboost in /usr/local/lib/python3.10/dist-packages (2.1.2)
Requirement already satisfied: catboost in /usr/local/lib/python3.10/dist-packages (1.2.7)
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.26.4)
Requirement already satisfied: nvidia-nccl-cu12 in /usr/local/lib/python3.10/dist-packages (from xgboost) (2.23.4)
Requirement already satisfied: scipy in /usr/local/lib/python3.10/dist-packages (from xgboost) (1.13.1)
Requirement already satisfied: graphviz in /usr/local/lib/python3.10/dist-packages (from catboost) (0.20.3)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (from catboost) (3.8.0)
Requirement already satisfied: pandas>=0.24 in /usr/local/lib/python3.10/dist-packages (from catboost) (2.2.2)
Requirement already satisfied: plotly in /usr/local/lib/python3.10/dist-packages (from catboost) (5.24.1)
Requirement already satisfied: six in /usr/local/lib/python3.10/dist-packages (from catboost) (1.16.0)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2.8
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas>=0.24->catboost) (2024.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.3.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (4.55.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (1.4.7)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (24.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (11.0.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib->catboost) (3.2.0)
Requirement already satisfied: tenacity>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from plotly->catboost) (9.0.0)
```

```
# from sklearn.feature_extraction.text import TfidfVectorizer
# from sklearn.model_selection import train_test_split
# from sklearn.preprocessing import LabelEncoder
# from sklearn.metrics import accuracy_score, classification_report
# from xgboost import XGBClassifier
# from catboost import CatBoostClassifier

# # Load the dataset
# file_path = '/content/processed_dataset.csv'
# data = pd.read_csv(file_path)

# # Ensure there are no missing values in the target column
# data = data.dropna(subset=['sub_category'])  # Replace 'sub_category' with your actual target column name

# # Split data into features and target
# X = data['processed_text']
# y = data['sub_category']  # Replace with your actual target column name

# # Handle NaN values in the feature column
# X = X.fillna('')

# # Encode target labels to ensure they start from 0
# label_encoder = LabelEncoder()
# y_encoded = label_encoder.fit_transform(y)  # Convert string labels to numeric

# # Split into training and testing sets
# X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

# # Vectorize the text data
# vectorizer = TfidfVectorizer(max_features=5000)
# X_train_vec = vectorizer.fit_transform(X_train)
# X_test_vec = vectorizer.transform(X_test)

# # ===================
# # XGBoost Classifier
# # ===================
# xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state=42)
# xgb_model.fit(X_train_vec, y_train)

# # Predict and evaluate
# xgb_pred = xgb_model.predict(X_test_vec)
# xgb_accuracy = accuracy_score(y_test, xgb_pred)
# print(f"XGBoost Accuracy: {xgb_accuracy:.2f}")
```

```
# print("XGBoost Classification Report:")
# print(classification_report(y_test, xgb_pred, target_names=label_encoder.classes_))

# # ===================
# # CatBoost Classifier
# # ===================
# cat_model = CatBoostClassifier(verbose=0, random_state=42)
# cat_model.fit(X_train_vec, y_train)

# # Predict and evaluate
# cat_pred = cat_model.predict(X_test_vec)
# cat_accuracy = accuracy_score(y_test, cat_pred)
# print(f"CatBoost Accuracy: {cat_accuracy:.2f}")
# print("CatBoost Classification Report:")
# print(classification_report(y_test, cat_pred, target_names=label_encoder.classes_))



# # Debug: Inspect unique labels before encoding
# print("Unique labels before encoding:", data['sub_category'].unique())

# # Encode target labels to ensure they start from 0
# label_encoder = LabelEncoder()
# y_encoded = label_encoder.fit_transform(y)  # Convert string labels to numeric

# # Debug: Inspect unique labels after encoding
# print("Unique labels after encoding:", sorted(set(y_encoded)))

# # Check for gaps in the encoded labels
# expected_labels = list(range(len(label_encoder.classes_)))
# actual_labels = sorted(set(y_encoded))

# # Debug: Compare actual vs. expected labels
# print("Expected labels:", expected_labels)
# print("Actual labels:", actual_labels)

# # Ensure all expected labels are present
# if actual_labels != expected_labels:
#     raise ValueError("Mismatch between actual and expected labels after encoding.")

# # Proceed with train-test split
# X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

# # Vectorize the text data
# vectorizer = TfidfVectorizer(max_features=5000)
# X_train_vec = vectorizer.fit_transform(X_train)
# X_test_vec = vectorizer.transform(X_test)

# # ===================
# # XGBoost Classifier
# # ===================
# xgb_model = XGBClassifier(use_label_encoder=False, eval_metric='mlogloss', random_state=42)
# xgb_model.fit(X_train_vec, y_train)

# # Predict and evaluate
# xgb_pred = xgb_model.predict(X_test_vec)
# xgb_accuracy = accuracy_score(y_test, xgb_pred)
# print(f"XGBoost Accuracy: {xgb_accuracy:.2f}")
# print("XGBoost Classification Report:")
# print(classification_report(y_test, xgb_pred, target_names=label_encoder.classes_))

# # ===================
# # CatBoost Classifier
# # ===================
# cat_model = CatBoostClassifier(verbose=0, random_state=42)
# cat_model.fit(X_train_vec, y_train)

# # Predict and evaluate
# cat_pred = cat_model.predict(X_test_vec)
# cat_accuracy = accuracy_score(y_test, cat_pred)
# print(f"CatBoost Accuracy: {cat_accuracy:.2f}")
# print("CatBoost Classification Report:")
# print(classification_report(y_test, cat_pred, target_names=label_encoder.classes_))



!pip install datasets
```

```
Requirement already satisfied: datasets in /usr/local/lib/python3.10/dist-packages (3.1.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from datasets) (3.16.1)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from datasets) (1.26.4)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (17.0.0)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (0.3.8)
```

```
Requirement already satisfied: pandas in /usr/local/lib/python3.10/dist-packages (from datasets) (2.2.2)
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.10/dist-packages (from datasets) (2.32.3)
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.10/dist-packages (from datasets) (4.66.6)
Requirement already satisfied: xxhash in /usr/local/lib/python3.10/dist-packages (from datasets) (3.5.0)
Requirement already satisfied: multiprocess<0.70.17 in /usr/local/lib/python3.10/dist-packages (from datasets) (0.70.16)
Requirement already satisfied: fsspec<=2024.9.0,>=2023.1.0 in /usr/local/lib/python3.10/dist-packages (from fsspec[http]<=2024.9.0,
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from datasets) (3.11.2)
Requirement already satisfied: huggingface-hub>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from datasets) (0.26.2)
Requirement already satisfied: packaging in /usr/local/lib/python3.10/dist-packages (from datasets) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from datasets) (6.0.2)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (2.4.3)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (24.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (6.1.0)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (0.2.0)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (1.17.2)
Requirement already satisfied: async-timeout<6.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->datasets) (4.0.3)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub>=0.23.0-
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.32.2->datasets
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.32.2->datasets) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.32.2->datasets) (2.2
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.32.2->datasets) (202
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2024.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.10/dist-packages (from pandas->datasets) (2024.2)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.8.2->pandas->datasets)
```

```
# import pandas as pd
# import torch
# from sklearn.model_selection import train_test_split
# from sklearn.preprocessing import LabelEncoder
# from transformers import BertTokenizer, BertForSequenceClassification, Trainer, TrainingArguments
# from datasets import Dataset
# import os
# os.environ["WANDB_DISABLED"] = "true"

# # Load your dataset
# file_path = '/content/processed_dataset.csv'
# data = pd.read_csv(file_path)

# # Ensure there are no missing values
# data = data.dropna(subset=['processed_text', 'sub_category'])

# # Preprocess the target column (label encoding)
# label_encoder = LabelEncoder()
# data['encoded_labels'] = label_encoder.fit_transform(data['sub_category'])

# # Split data into train and test
# X_train, X_test, y_train, y_test = train_test_split(data['processed_text'], data['encoded_labels'], test_size=0.2, random_state=42)

# # Tokenizer and Encoding
# tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# # Encode the text data
# def encode_text(text):
#     return tokenizer(text, padding='max_length', truncation=True, max_length=128)

# X_train_enc = [encode_text(text) for text in X_train]
# X_test_enc = [encode_text(text) for text in X_test]

# # Convert to Hugging Face Dataset format
# train_dataset = Dataset.from_dict({
#     'input_ids': [item['input_ids'] for item in X_train_enc],
#     'attention_mask': [item['attention_mask'] for item in X_train_enc],
#     'labels': y_train.tolist()
# })

# test_dataset = Dataset.from_dict({
#     'input_ids': [item['input_ids'] for item in X_test_enc],
#     'attention_mask': [item['attention_mask'] for item in X_test_enc],
#     'labels': y_test.tolist()
# })

# # Load the pre-trained BERT model for sequence classification
# model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=len(label_encoder.classes_))

# # Define the training arguments
# training_args = TrainingArguments(
#     output_dir='./results',            # output directory for model predictions and checkpoints
#     num_train_epochs=3,                # number of training epochs
#     per_device_train_batch_size=16,    # batch size for training
#     per_device_eval_batch_size=64,     # batch size for evaluation
```

```
#     warmup_steps=500,                 # number of warmup steps for learning rate scheduler
#     weight_decay=0.01,                # strength of weight decay
#     logging_dir='./logs',             # directory for storing logs
#     logging_steps=10,
# )

# # Initialize the Trainer
# trainer = Trainer(
#     model=model,                        # the pre-trained model
#     args=training_args,                 # training arguments
#     train_dataset=train_dataset,        # training dataset
#     eval_dataset=test_dataset,          # evaluation dataset
# )

# # Train the model
# trainer.train()

# # Evaluate the model
# results = trainer.evaluate()

# # Print evaluation re sults
# print(f"Test Accuracy: {results['eval_accuracy']:.2f}")
# print("Classification Report:")
# print(classification_report(y_test, trainer.predict(test_dataset).predictions.argmax(axis=-1), target_names=label_encoder.classes_))


import pandas as pd
import numpy as np
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report
from imblearn.over_sampling import SMOTE

# Assume your dataset is already loaded in X and y
# Example:
# X = df['text']  # Feature column
# y = df['label']  # Target column

# Ensure there are no NaN values
X = X.dropna()
y = y[X.index]  # Align y with X after removing NaN

# ============================
# Option 1: Remove Rare Classes
# ============================
class_counts = y.value_counts()
valid_classes = class_counts[class_counts > 1].index
X_filtered = X[y.isin(valid_classes)]
y_filtered = y[y.isin(valid_classes)]

# ============================
# Option 2: Non-Stratified Split
# ============================
# Comment this out if using Option 1 or 3
X_train, X_test, y_train, y_test = train_test_split(
    X_filtered, y_filtered, test_size=0.2, random_state=42
)

# ============================
# Option 3: Oversample Rare Classes
# ============================
# Uncomment to use SMOTE
# smote = SMOTE(random_state=42)
# tfidf = TfidfVectorizer()
# X_tfidf = tfidf.fit_transform(X_filtered)
# X_smote, y_smote = smote.fit_resample(X_tfidf, y_filtered)
# X_train, X_test, y_train, y_test = train_test_split(
#     X_smote, y_smote, test_size=0.2, random_state=42
# )

# ============================
# TF-IDF Vectorization
# ============================
tfidf = TfidfVectorizer(max_features=5000)
X_train_vec = tfidf.fit_transform(X_train)
X_test_vec = tfidf.transform(X_test)

# ============================
# Train SVM Model
# ============================
svm_model = SVC(kernel='linear', random_state=42)
svm_model.fit(X_train_vec, y_train)
```

```
# ============================
# Predictions and Evaluation
# ============================
y_pred = svm_model.predict(X_test_vec)
print("Classification Report:")
print(classification_report(y_test, y_pred))

# ============================
# Optional Logistic Regression
# ============================
# from sklearn.linear_model import LogisticRegression

# lr_model = LogisticRegression(random_state=42)
# lr_model.fit(X_train_vec, y_train)
# y_pred_lr = lr_model.predict(X_test_vec)

# print("Classification Report for Logistic Regression:")
# print(classification_report(y_test, y_pred_lr))
```

```
Classification Report:
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
                                                                   precision    recall  f1-score   support

                              Business Email CompromiseEmail Takeover       1.00      0.01      0.02        85
                                         Cheating by Impersonation       0.20      0.01      0.03       543
                                       Computer Generated CSAM/CSEM       0.00      0.00      0.00         1
                                              Cryptocurrency Fraud       0.65      0.56      0.60       124
                                    Cyber Bullying  Stalking  Sexting       0.60      0.69      0.64      1125
                                                     Cyber Terrorism       0.00      0.00      0.00        42
                              Damage to computer computer systems etc       0.25      0.03      0.06        31
                                                   Data Breach/Theft       0.16      0.12      0.14       113
                                 DebitCredit Card FraudSim Swap Fraud       0.70      0.67      0.68      2811
                                                 DematDepository Fraud       0.50      0.01      0.01       190
Denial of Service (DoS)/Distributed Denial of Service (DDOS) attacks       0.19      0.16      0.17       147
                                                       EMail Phishing       0.60      0.05      0.10        55
                                                 EWallet Related Fraud       0.70      0.38      0.49      1129
                                                        Email Hacking       0.63      0.43      0.51       105
                                            FakeImpersonating Profile       0.59      0.42      0.49       606
                                                   Fraud CallVishing       0.32      0.26      0.29      1487
                                                  Hacking/Defacement       0.15      0.60      0.24       138
                                                 Impersonating Email       0.00      0.00      0.00        19
                                       Internet Banking Related Fraud       0.72      0.54      0.62      2366
                                                  Intimidating Email       0.00      0.00      0.00         5
                                                      Malware Attack       0.11      0.01      0.02       142
                                            Online Gambling  Betting       0.88      0.06      0.12       108
                                                    Online Job Fraud       0.32      0.16      0.22       251
                                            Online Matrimonial Fraud       0.67      0.05      0.10        37
                                                 Online Trafficking       0.00      0.00      0.00        50
                                                              Other       0.35      0.47      0.40      2867
                                       Profile Hacking Identity Theft       0.57      0.41      0.48       605
                                 Provocative Speech for unlawful acts       0.65      0.11      0.18       121
                                                         Ransomware       1.00      0.15      0.27        13
                                                  Ransomware Attack       0.18      0.04      0.06       163
                                                      SQL Injection       0.22      0.08      0.12       146
                              Tampering with computer source documents       0.21      0.15      0.18       144
                                                  UPI Related Frauds       0.64      0.85      0.73      7143
                                       Unauthorised AccessData Breach       0.35      0.18      0.24       282
                                            Website DefacementHacking       0.00      0.00      0.00        23

                                                             accuracy                           0.56     23217
                                                            macro avg       0.40      0.22      0.23     23217
                                                         weighted avg       0.56      0.56      0.54     23217

/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score
from imblearn.over_sampling import SMOTE
from xgboost import XGBClassifier
from collections import Counter

# Load your dataset
df = pd.read_csv("/content/processed_dataset.csv")  # Replace with your actual dataset path

# Drop rows with NaN in the text column or fill with a placeholder
df['processed_text'] = df['processed_text'].fillna("missing_text")  # Replace NaN with a placeholder string
```

```
# Check the class distribution
class_counts = df['processed_text'].value_counts()
print("Class distribution:\n", class_counts)

# Filter out classes with only one sample
# Instead of filtering by 'category', filter by the target column 'processed_text'
rare_classes = class_counts[class_counts <= 1].index
df_filtered = df[~df['processed_text'].isin(rare_classes)]

# Split dataset into features and target
X = df_filtered['processed_text']
y = df_filtered['category']

# Encode the target labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Perform the train-test split
# Remove stratify if you want to allow classes with only one sample in the training set
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000, stop_words='english')
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Check the class distribution after splitting
print(f"Original class distribution in training set: {Counter(y_train)}")

# Apply SMOTE for balancing the classes in the training set
# Reduce k_neighbors to a value less than or equal to the minimum number of samples in any class
smote = SMOTE(random_state=42, k_neighbors=min(Counter(y_train).values()) -1 if min(Counter(y_train).values()) > 1 else 1)  # Adjust k_ne
X_train_smote, y_train_smote = smote.fit_resample(X_train_vec, y_train)

# Train the XGBoost Classifier
xgb_model = XGBClassifier(random_state=42, objective="multi:softmax", num_class=len(set(y_encoded)), use_label_encoder=False)
xgb_model.fit(X_train_smote, y_train_smote)

# Make predictions
y_pred = xgb_model.predict(X_test_vec)

# Print the classification report
print("Classification Report for XGBoost:")
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))

# Print the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

```
Class distribution:
 processed_text
respected sir serious matter want inform person running involve shamefull activity using woman many place kolkata gariahat ballygung
financial fraud
missing_text
dear sir please stop fraudulent transaction refund amount source account regard
fraudulent transaction

phone per bat karte huye unhone mujse bar bar kaha ki paisa nahi aye aur main unko send karta gaya mere pas bank se msz late jiske k
lost almost lakh please help student took money account father atught settle life getting job verge suicide hope rather please help
user hooren channel id httpswwwyoutubecomchannelucpckqjdzbgnlopfma used extremely abusive language allah muslim community often seer
froud call came friend phone froudent speaking doctor talk send money processing becouse delivery date friend wife recentlyso belive
saw add facebook job placement want job contacted told good contact trust transferred phone got switch kindly help
Name: count, Length: 104618, dtype: int64
Original class distribution in training set: Counter({6: 14400, 9: 2720, 8: 481, 0: 333, 11: 250, 10: 203, 1: 52, 4: 49, 2: 33, 7: 9
/usr/local/lib/python3.10/dist-packages/xgboost/core.py:158: UserWarning: [17:07:23] WARNING: /workspace/src/learner.cc:740:
Parameters: { "use_label_encoder" } are not used.

  warnings.warn(smsg, UserWarning)
Classification Report for XGBoost:
                                                   precision    recall  f1-score   support

                                Any Other Cyber Crime       0.25      0.35      0.29        94
Child Pornography CPChild Sexual Abuse Material CSAM       0.67      0.63      0.65        19
                                 Cryptocurrency Crime       0.80      1.00      0.89         4
                                      Cyber Terrorism       0.00      0.00      0.00         0
      Hacking  Damage to computercomputer system etc       0.12      0.08      0.10        13
                             Online Cyber Trafficking       0.00      0.00      0.00         1
                               Online Financial Fraud       1.00      0.92      0.96      3547
                             Online Gambling  Betting       0.00      0.00      0.00         5
                 Online and Social Media Related Crime       0.57      0.33      0.42       109
             RapeGang Rape RGRSexually Abusive Content       1.00      0.99      1.00       715
                                Sexually Explicit Act       0.82      0.74      0.78        50
                            Sexually Obscene material       0.82      0.66      0.73        77

                                            accuracy                           0.89      4634
```

```
       macro avg       0.50      0.48      0.48      4634
    weighted avg       0.96      0.89      0.93      4634

Accuracy: 0.8945
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Recall is ill-defined and b
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Recall is ill-defined and b
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Recall is ill-defined and b
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```python
from lightgbm import LGBMClassifier
```

```python
lgbm_model = LGBMClassifier(random_state=42, objective="multiclass", num_class=len(set(y_encoded)))
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score
from imblearn.over_sampling import SMOTE
from lightgbm import LGBMClassifier
from collections import Counter

# Load your dataset
df = pd.read_csv("/content/processed_dataset.csv")  # Replace with your actual dataset path

# Drop rows with NaN in the text column or fill with a placeholder
df['processed_text'] = df['processed_text'].fillna("missing_text")  # Replace NaN with a placeholder string

# Check the class distribution
class_counts = df['processed_text'].value_counts()
print("Class distribution:\n", class_counts)

# Filter out classes with only one sample
rare_classes = class_counts[class_counts <= 1].index
df_filtered = df[~df['processed_text'].isin(rare_classes)]

# Split dataset into features and target
X = df_filtered['processed_text']
y = df_filtered['category']

# Encode the target labels
label_encoder = LabelEncoder()
y_encoded = label_encoder.fit_transform(y)

# Perform the train-test split
X_train, X_test, y_train, y_test = train_test_split(X, y_encoded, test_size=0.2, random_state=42)

# Vectorize the text data using TF-IDF
vectorizer = TfidfVectorizer(max_features=5000, stop_words='english')
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

# Check the class distribution after splitting
print(f"Original class distribution in training set: {Counter(y_train)}")

# Apply SMOTE for balancing the classes in the training set
smote = SMOTE(random_state=42, k_neighbors=min(Counter(y_train).values()) - 1 if min(Counter(y_train).values()) > 1 else 1)  # Adjust k_
X_train_smote, y_train_smote = smote.fit_resample(X_train_vec, y_train)

# Train the LightGBM Classifier
lgbm_model = LGBMClassifier(random_state=42, objective="multiclass", num_class=len(set(y_encoded)))
lgbm_model.fit(X_train_smote, y_train_smote)

# Make predictions
y_pred = lgbm_model.predict(X_test_vec)

# Print the classification report
print("Classification Report for LightGBM:")
print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))
```

```
# Print the accuracy score
accuracy = accuracy_score(y_test, y_pred)
print(f"Accuracy: {accuracy:.4f}")
```

```
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
[LightGBM] [Warning] No further splits with positive gain, best gain: -inf
Classification Report for LightGBM:
                                                      precision    recall  f1-score   support

                               Any Other Cyber Crime       0.28      0.35      0.31        94
Child Pornography CPChild Sexual Abuse Material CSAM       0.86      0.63      0.73        19
                                Cryptocurrency Crime       0.80      1.00      0.89         4
                                     Cyber Terrorism       0.00      0.00      0.00         0
      Hacking  Damage to computercomputer system etc       0.12      0.08      0.10        13
                            Online Cyber Trafficking       0.00      0.00      0.00         1
                              Online Financial Fraud       0.99      0.92      0.96      3547
                            Online Gambling  Betting       0.00      0.00      0.00         5
               Online and Social Media Related Crime       0.59      0.33      0.42       109
           RapeGang Rape RGRSexually Abusive Content       1.00      0.99      1.00       715
                               Sexually Explicit Act       0.88      0.70      0.78        50
                           Sexually Obscene material       0.88      0.75      0.81        77

                                            accuracy                           0.90      4634
                                           macro avg       0.53      0.48      0.50      4634
                                        weighted avg       0.96      0.90      0.93      4634

Accuracy: 0.8994
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Recall is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Recall is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Precision is ill-defined
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.10/dist-packages/sklearn/metrics/_classification.py:1531: UndefinedMetricWarning: Recall is ill-defined ar
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

```
# from catboost import CatBoostClassifier
```

```
# X_train_vec = vectorizer.fit_transform(X_train).toarray()
# X_test_vec = vectorizer.transform(X_test).toarray()
```

```
# catboost_model = CatBoostClassifier(
#     random_seed=42,
#     iterations=500,
#     learning_rate=0.1,
#     depth=6,
#     loss_function="MultiClass",
#     verbose=100
# )
```

```
pip install transformers tensorflow
```

```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.46.2)
Requirement already satisfied: tensorflow in /usr/local/lib/python3.10/dist-packages (2.17.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.16.1)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.26.2)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.26.4)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (24.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2024.9.11)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.32.3)
Requirement already satisfied: safetensors>=0.4.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.4.5)
Requirement already satisfied: tokenizers<0.21,>=0.20 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.20.3)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.6)
Requirement already satisfied: absl-py>=1.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.4.0)
Requirement already satisfied: astunparse>=1.6.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.6.3)
Requirement already satisfied: flatbuffers>=24.3.25 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (24.3.25)
Requirement already satisfied: gast!=0.5.0,!=0.5.1,!=0.5.2,>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.6
Requirement already satisfied: google-pasta>=0.1.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.2.0)
Requirement already satisfied: h5py>=3.10.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.12.1)
Requirement already satisfied: libclang>=13.0.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (18.1.1)
Requirement already satisfied: ml-dtypes<0.5.0,>=0.3.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0.4.1)
Requirement already satisfied: opt-einsum>=2.3.2 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.4.0)
Requirement already satisfied: protobuf!=4.21.0,!=4.21.1,!=4.21.2,!=4.21.3,!=4.21.4,!=4.21.5,<5.0.0dev,>=3.20.3 in /usr/local/lib/py
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from tensorflow) (75.1.0)
Requirement already satisfied: six>=1.12.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.5.0)
Requirement already satisfied: typing-extensions>=3.6.6 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (4.12.2)
Requirement already satisfied: wrapt>=1.11.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.16.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (1.68.0)
Requirement already satisfied: tensorboard<2.18,>=2.17 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (2.17.1)
Requirement already satisfied: keras>=3.2.0 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (3.5.0)
Requirement already satisfied: tensorflow-io-gcs-filesystem>=0.23.1 in /usr/local/lib/python3.10/dist-packages (from tensorflow) (0
Requirement already satisfied: wheel<1.0,>=0.23.0 in /usr/local/lib/python3.10/dist-packages (from astunparse>=1.6.0->tensorflow) (6
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.23.2->trans
Requirement already satisfied: rich in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (13.9.4)
Requirement already satisfied: namex in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (0.0.8)
Requirement already satisfied: optree in /usr/local/lib/python3.10/dist-packages (from keras>=3.2.0->tensorflow) (0.13.1)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2024.8.3
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow
Requirement already satisfied: tensorboard-data-server<0.8.0,>=0.7.0 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2
Requirement already satisfied: werkzeug>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from tensorboard<2.18,>=2.17->tensorflow
Requirement already satisfied: MarkupSafe>=2.1.1 in /usr/local/lib/python3.10/dist-packages (from werkzeug>=1.0.1->tensorboard<2.18,
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorflow
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.10/dist-packages (from rich->keras>=3.2.0->tensorfl
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.10/dist-packages (from markdown-it-py>=2.2.0->rich->keras>=3.2.6
```

```
#too much time to run

# # Import necessary libraries
# import pandas as pd
# from sklearn.model_selection import train_test_split
# from sklearn.preprocessing import LabelEncoder
# from sklearn.metrics import classification_report, accuracy_score
# from transformers import DistilBertTokenizer, TFDistilBertForSequenceClassification, create_optimizer
# # Import TFTrainer and TFTrainingArguments from accelerate
# #from accelerate import Accelerator # This is not needed for TensorFlow models
# #from transformers import TrainingArguments, Trainer # Trainer is for PyTorch models
# import tensorflow as tf


# # ... (rest of the code remains the same up to model creation) ...


# # Load the DistilBERT model for classification
# model = TFDistilBertForSequenceClassification.from_pretrained(
#     'distilbert-base-uncased',
#     num_labels=len(set(y_encoded))
# )

# # Define optimizer and compile the model (for TensorFlow)
# num_train_steps = len(train_dataset) * training_args.num_train_epochs
# optimizer, lr_schedule = create_optimizer(
#     init_lr=training_args.learning_rate,
#     num_train_steps=num_train_steps,
#     num_warmup_steps=0,
#     weight_decay_rate=0.01,
# )
# model.compile(optimizer=optimizer, loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True), metrics=['accuracy'])

# # Train the model (using TensorFlow's fit method)
# model.fit(train_dataset, epochs=training_args.num_train_epochs)
```

```
# # Evaluate the model
# loss, accuracy = model.evaluate(test_dataset)
# print(f"Evaluation Loss: {loss:.4f}")
# print(f"Evaluation Accuracy: {accuracy:.4f}")

# # Make predictions
# y_pred_logits = model.predict(test_dataset)
# y_pred = tf.argmax(y_pred_logits, axis=1).numpy()

# # Print the classification report
# print("Classification Report for DistilBERT:")
# print(classification_report(y_test, y_pred, target_names=label_encoder.classes_))

# # Print the accuracy score
# accuracy = accuracy_score(y_test, y_pred)
# print(f"Accuracy: {accuracy:.4f}")


!pip install transformers tensorflow datasets
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, accuracy_score
from transformers import DistilBertTokenizerFast, TFDistilBertForSequenceClassification, create_optimizer
import tensorflow as tf
import datasets

# Load the dataset using datasets library (for faster processing)
dataset = datasets.Dataset.from_pandas(pd.read_csv("/content/processed_dataset.csv"))

# Handle missing values in the text column
dataset = dataset.map(lambda examples: {'processed_text': examples['processed_text'] if examples['processed_text'] is not None else "miss

# Filter out classes with only one sample (using datasets API)
class_counts = dataset.unique('processed_text')
rare_classes = [c for c in class_counts if dataset.filter(lambda example: example['processed_text'] == c).num_rows <= 1]
dataset = dataset.filter(lambda example: example['processed_text'] not in rare_classes)

# Split dataset into features and target (using datasets API)
train_testvalid = dataset.train_test_split(test_size=0.2, seed=42)
test_valid = train_testvalid['test'].train_test_split(test_size=0.5, seed=42)
train_dataset = train_testvalid['train']
test_dataset = test_valid['test']
valid_dataset = test_valid['train']

# Encode the target labels
label_encoder = LabelEncoder()
label_encoder.fit(train_dataset['category'])

# Add a 'label' column to the dataset
def add_label_column(example):
    example['label'] = label_encoder.transform([example['category']])[0]  # Transform the 'category' column
    return example

train_dataset = train_dataset.map(add_label_column)
test_dataset = test_dataset.map(add_label_column)
valid_dataset = valid_dataset.map(add_label_column)

# Tokenizer initialization (using DistilBertTokenizerFast for speed)
tokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')

# Tokenize the datasets (using datasets map function for efficiency)
def tokenize_function(examples):
    return tokenizer(examples["processed_text"], padding="max_length", truncation=True)

train_dataset = train_dataset.map(tokenize_function, batched=True)
test_dataset = test_dataset.map(tokenize_function, batched=True)
valid_dataset = valid_dataset.map(tokenize_function, batched=True)

# Format the datasets for TensorFlow
train_dataset = train_dataset.remove_columns(["category", "processed_text"]).with_format("tensorflow")
test_dataset = test_dataset.remove_columns(["category", "processed_text"]).with_format("tensorflow")
valid_dataset = valid_dataset.remove_columns(["category", "processed_text"]).with_format("tensorflow")

# Convert datasets to tf.data.Dataset
tf_train_dataset = train_dataset.to_tf_dataset(
    columns=["attention_mask", "input_ids"],
    label_cols=["label"],  # Use 'label' instead of 'category'
    shuffle=True,
    batch_size=32,
)

tf_test_dataset = test_dataset.to_tf_dataset(
```

```
    columns=["attention_mask", "input_ids"],
    label_cols=["label"],  # Use 'label' instead of 'category'
    shuffle=False,
    batch_size=32,
)

tf_valid_dataset = valid_dataset.to_tf_dataset(
    columns=["attention_mask", "input_ids"],
    label_cols=["label"],  # Use 'label' instead of 'category'
    shuffle=False,
    batch_size=32,
)

# Load the DistilBERT model for classification
model = TFDistilBertForSequenceClassification.from_pretrained(
    'distilbert-base-uncased',
    num_labels=len(label_encoder.classes_)
)

# Define optimizer and compile the model (for TensorFlow)
num_train_steps = len(tf_train_dataset) * 3  #
```