```
# Install necessary libraries
!pip install transformers tqdm seaborn matplotlib scikit-learn

from sklearn.metrics import confusion_matrix, roc_curve, auc, precision_recall_curve
from sklearn.preprocessing import label_binarize

# Import necessary libraries
import torch
from torch.utils.data import Dataset, DataLoader
from torch import nn
from transformers import BertTokenizer, BertForSequenceClassification, AdamW, get_linear_schedule_with_warmup
from tqdm import tqdm
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder, label_binarize
from sklearn.metrics import confusion_matrix, roc_curve, auc, precision_recall_curve
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Check if GPU is available and set the device
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Load your dataset (you can upload your CSV to Colab)
from google.colab import files
uploaded = files.upload()

# Load the dataset (replace 'your_file.csv' with the actual filename after uploading)
df = pd.read_csv('MN-DS-news-classification_combined.csv', encoding='ISO-8859-1')

# Preprocess the data: Select relevant columns, clean up missing values, and encode labels
df = df[['title', 'category_level_1']].dropna()

# Encode the labels into numerical format
label_encoder = LabelEncoder()
df['label'] = label_encoder.fit_transform(df['category_level_1'])

# Plot class distribution
def plot_class_distribution(df, label_encoder):
    plt.figure(figsize=(10, 5))
    sns.countplot(x=df['category_level_1'])
    plt.title('Class Distribution')
    plt.xticks(rotation=45)
    plt.show()

plot_class_distribution(df, label_encoder)

# Split the data into training and validation sets
train_texts, val_texts, train_labels, val_labels = train_test_split(
    df['title'].tolist(), df['label'].tolist(), test_size=0.2, random_state=42
)

# Define a Dataset class for PyTorch
class NewsDataset(Dataset):
    def __init__(self, texts, labels, tokenizer, max_len):
        self.texts = texts
        self.labels = labels
```

```
        self.tokenizer = tokenizer
        self.max_len = max_len

    def __len__(self):
        return len(self.texts)

    def __getitem__(self, idx):
        text = self.texts[idx]
        label = self.labels[idx]
        encoding = self.tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=self.max_len,
            return_token_type_ids=False,
            padding='max_length',
            truncation=True,
            return_attention_mask=True,
            return_tensors='pt',
        )

        return {
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'labels': torch.tensor(label, dtype=torch.long)
        }

# Initialize the tokenizer (using BERT for demonstration)
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

# Set maximum length for tokenized sequences
MAX_LEN = 128

# Create DataLoader for training and validation
train_dataset = NewsDataset(train_texts, train_labels, tokenizer, MAX_LEN)
val_dataset = NewsDataset(val_texts, val_labels, tokenizer, MAX_LEN)

train_loader = DataLoader(train_dataset, batch_size=16, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=16)

# Define a basic model using BERT for sequence classification
class NewsClassifier(nn.Module):
    def __init__(self, n_classes):
        super(NewsClassifier, self).__init__()
        self.bert = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=n_classes)

    def forward(self, input_ids, attention_mask):
        return self.bert(input_ids=input_ids, attention_mask=attention_mask)

# Get the number of classes
num_classes = df['label'].nunique()

# Instantiate the model
model = NewsClassifier(num_classes).to(device)

# Set up the optimizer and learning rate scheduler
optimizer = AdamW(model.parameters(), lr=2e-5, correct_bias=False)
total_steps = len(train_loader) * 6  # 3 epochs changed to 6 to test

scheduler = get_linear_schedule_with_warmup(
    optimizer, num_warmup_steps=0, num_training_steps=total_steps
```

```
)

# Define loss function (CrossEntropyLoss is used for classification tasks)
loss_fn = nn.CrossEntropyLoss().to(device)

# Training function
def train_epoch(model, data_loader, loss_fn, optimizer, device, scheduler):
    model.train()
    losses = 0
    correct_predictions = 0

    for d in tqdm(data_loader, desc="Training"):
        input_ids = d["input_ids"].to(device)
        attention_mask = d["attention_mask"].to(device)
        labels = d["labels"].to(device)

        optimizer.zero_grad()

        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        logits = outputs.logits
        _, preds = torch.max(logits, dim=1)
        loss = loss_fn(logits, labels)

        loss.backward()
        optimizer.step()
        scheduler.step()

        correct_predictions += torch.sum(preds == labels)
        losses += loss.item()

    return correct_predictions.double() / len(data_loader.dataset), losses / len(data_loader)

# Evaluation function
def eval_model(model, data_loader, loss_fn, device):
    model.eval()
    losses = 0
    correct_predictions = 0

    with torch.no_grad():
        for d in tqdm(data_loader, desc="Validation"):
            input_ids = d["input_ids"].to(device)
            attention_mask = d["attention_mask"].to(device)
            labels = d["labels"].to(device)

            outputs = model(input_ids=input_ids, attention_mask=attention_mask)
            logits = outputs.logits
            _, preds = torch.max(logits, dim=1)
            loss = loss_fn(logits, labels)

            correct_predictions += torch.sum(preds == labels)
            losses += loss.item()

    return correct_predictions.double() / len(data_loader.dataset), losses / len(data_loader)

# Plotting training and validation accuracy and loss
def plot_training_history(history):
    epochs = range(1, len(history['train_acc']) + 1)

    # Plot training & validation accuracy
    plt.figure(figsize=(12, 5))
```

```
    plt.subplot(1, 2, 1)
    plt.plot(epochs, history['train_acc'], label='Training Accuracy')
    plt.plot(epochs, history['val_acc'], label='Validation Accuracy')
    plt.title('Training and Validation Accuracy')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()

    # Plot training & validation loss
    plt.subplot(1, 2, 2)
    plt.plot(epochs, history['train_loss'], label='Training Loss')
    plt.plot(epochs, history['val_loss'], label='Validation Loss')
    plt.title('Training and Validation Loss')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()

    plt.show()

# Training the model for 3 epochs
history = {'train_acc': [], 'train_loss': [], 'val_acc': [], 'val_loss': []}
epochs = 6

for epoch in range(epochs):
    print(f"Epoch {epoch + 1}/{epochs}")

    # Train
    train_acc, train_loss = train_epoch(model, train_loader, loss_fn, optimizer, device, scheduler)
    train_acc = train_acc.cpu().item()  # Move accuracy to CPU and convert to Python float
    train_loss = float(train_loss)  # Ensure the loss is a float

    # Validate
    val_acc, val_loss = eval_model(model, val_loader, loss_fn, device)
    val_acc = val_acc.cpu().item()  # Move accuracy to CPU and convert to Python float
    val_loss = float(val_loss)  # Ensure the loss is a float

    # Append the metrics to the history dictionary
    history['train_acc'].append(train_acc)
    history['train_loss'].append(train_loss)
    history['val_acc'].append(val_acc)
    history['val_loss'].append(val_loss)

    print(f"Train loss: {train_loss}, Train accuracy: {train_acc}")
    print(f"Validation loss: {val_loss}, Validation accuracy: {val_acc}")

# Plot the training history
plot_training_history(history)

# Plot confusion matrix
def plot_confusion_matrix(model, data_loader, label_encoder, device):
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for d in data_loader:
            input_ids = d["input_ids"].to(device)
            attention_mask = d["attention_mask"].to(device)
            labels = d["labels"].to(device)
```

```
            labels = d["labels"].to(device)

            outputs = model(input_ids=input_ids, attention_mask=attention_mask)
            logits = outputs.logits
            _, preds = torch.max(logits, dim=1)

            all_preds.extend(preds.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    # Create confusion matrix
    cm = confusion_matrix(all_labels, all_preds)

    # Plot the confusion matrix
    plt.figure(figsize=(10, 7))
    sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=label_encoder.classes_, yticklabels=label_encoder.classes_)
    plt.xlabel('Predicted')
    plt.ylabel('Actual')
    plt.title('Confusion Matrix')
    plt.show()

# Call the function to plot confusion matrix after validation
plot_confusion_matrix(model, val_loader, label_encoder, device)

# Plot ROC Curve for Binary/Multiclass Classification
def plot_roc_curve(model, data_loader, device, num_classes):
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for d in data_loader:
            input_ids = d["input_ids"].to(device)
            attention_mask = d["attention_mask"].to(device)
            labels = d["labels"].to(device)

            outputs = model(input_ids=input_ids, attention_mask=attention_mask)
            logits = outputs.logits

            # Calculate probabilities
            probs = torch.nn.functional.softmax(logits, dim=1)

            all_preds.extend(probs.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    # Convert to numpy arrays for easier handling
    all_preds = np.array(all_preds)
    all_labels = np.array(all_labels)

   # Plot ROC Curve for Binary/Multiclass Classification
def plot_roc_curve(model, data_loader, device, num_classes):
    model.eval()
    all_preds = []
    all_labels = []

    with torch.no_grad():
        for d in data_loader:
            input_ids = d["input_ids"].to(device)
            attention_mask = d["attention_mask"].to(device)
            labels = d["labels"].to(device)
```

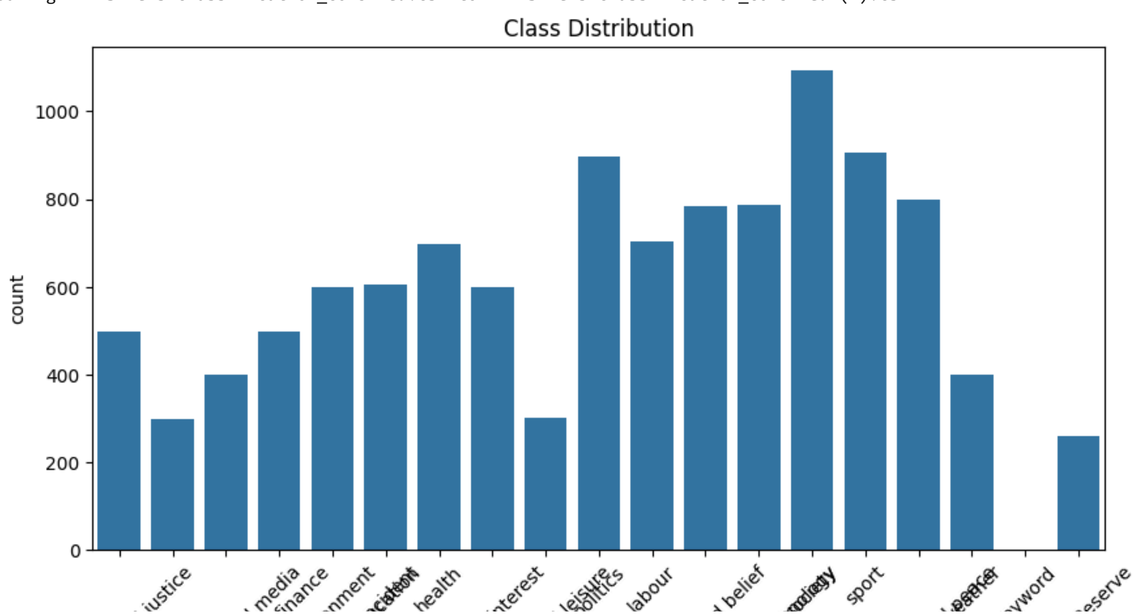
```
            outputs = model(input_ids=input_ids, attention_mask=attention_mask)
            logits = outputs.logits

            # Calculate probabilities
            probs = torch.nn.functional.softmax(logits, dim=1)

            all_preds.extend(probs.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())

    # Convert to numpy arrays for easier handling
    all_preds = np.array(all_preds)
    all_labels = np.array(all_labels)

    # Plot ROC curves for each class
    plt.figure(figsize=(10, 7))
    for i in range(num_classes):
        fpr, tpr, _ = roc_curve(label_binarize(all_labels, classes=range(num_classes))[:, i], all_preds[:, i])
        roc_auc = auc(fpr, tpr)
        plt.plot(fpr, tpr, label=f'Class {i} (AUC = {roc_auc:.2f})')

    plt.plot([0, 1], [0, 1], 'k--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC Curve for Multiclass Classification')
    plt.legend(loc="lower right")
    plt.show()

# Call the function after training
plot_roc_curve(model, val_loader, device, num_classes)
```

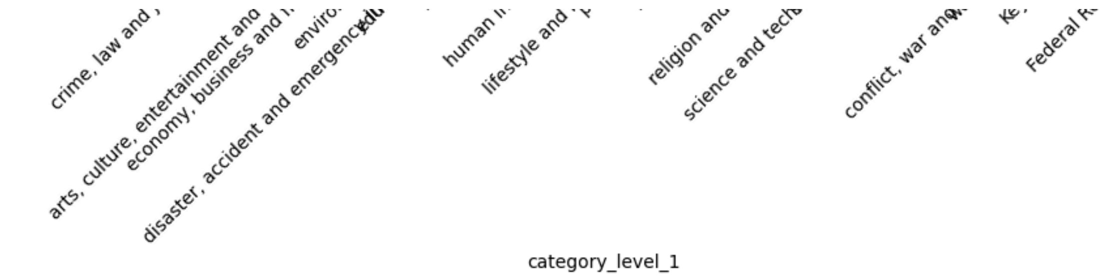
```
Requirement already satisfied: transformers in /usr/local/lib/python3.10/dist-packages (4.42.4)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (4.66.5)
Requirement already satisfied: seaborn in /usr/local/lib/python3.10/dist-packages (0.13.1)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-packages (3.7.1)
Requirement already satisfied: scikit-learn in /usr/local/lib/python3.10/dist-packages (1.3.2)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.15.4)
Requirement already satisfied: huggingface-hub<1.0,>=0.23.2 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.23.5)
Requirement already satisfied: numpy<2.0,>=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) (24.1)
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) (2024.5.15)
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) (0.4.4)
Requirement already satisfied: tokenizers<0.20,>=0.19 in /usr/local/lib/python3.10/dist-packages (from transformers) (0.19.1)
Requirement already satisfied: pandas>=1.2 in /usr/local/lib/python3.10/dist-packages (from seaborn) (2.1.4)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (4.53.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: scipy>=1.5.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.13.1)
Requirement already satisfied: joblib>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (1.4.2)
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.10/dist-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.23.2->transformers) (2024.6.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.23.2->transformers) (4.12.2)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: tzdata>=2022.1 in /usr/local/lib/python3.10/dist-packages (from pandas>=1.2->seaborn) (2024.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.10/dist-packages (from python-dateutil>=2.7->matplotlib) (1.16.0)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2024.7.4)
Using device: cuda
```

Choose Files MN-DS-ne…ombined.csv

- **MN-DS-news-classification_combined.csv**(text/csv) - 47957005 bytes, last modified: 8/22/2024 - 100% done

```
Saving MN-DS-news-classification_combined.csv to MN-DS-news-classification_combined (2).csv
```






```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:591: FutureWarning: This implementation of AdamW is deprecated and will be removed in a future version. Use the PyTor
  warnings.warn(
Epoch 1/6
Training: 100%|██████████| 557/557 [03:01<00:00,  3.06it/s]
Validation: 100%|██████████| 140/140 [00:15<00:00,  9.13it/s]
Train loss: 1.0977015212258487, Train accuracy: 0.6978651685393259
Validation loss: 0.6396755676716566, Validation accuracy: 0.8301886792452831
Epoch 2/6
Training: 100%|██████████| 557/557 [03:01<00:00,  3.07it/s]
Validation: 100%|██████████| 140/140 [00:15<00:00,  9.14it/s]
Train loss: 0.45776185402696823, Train accuracy: 0.8807865168539326
Validation loss: 0.5633253753717457, Validation accuracy: 0.844115004492363
Epoch 3/6
Training: 100%|██████████| 557/557 [03:01<00:00,  3.07it/s]
Validation: 100%|██████████| 140/140 [00:15<00:00,  9.12it/s]
Train loss: 0.29795931372624324, Train accuracy: 0.9219101123595506
Validation loss: 0.5383324213325977, Validation accuracy: 0.8566936208445642
Epoch 4/6
Training: 100%|██████████| 557/557 [03:01<00:00,  3.07it/s]
Validation: 100%|██████████| 140/140 [00:15<00:00,  9.08it/s]
Train loss: 0.2071278276995386, Train accuracy: 0.9479775280898877
Validation loss: 0.5647310524247586, Validation accuracy: 0.8499550763701708
Epoch 5/6
Training: 100%|██████████| 557/557 [03:01<00:00,  3.07it/s]
Validation: 100%|██████████| 140/140 [00:15<00:00,  9.13it/s]
Train loss: 0.15650326027275738, Train accuracy: 0.9589887640449438
Validation loss: 0.5713281507337732, Validation accuracy: 0.8526504941599282
Epoch 6/6
Training: 100%|██████████| 557/557 [03:01<00:00,  3.07it/s]
Validation: 100%|██████████| 140/140 [00:15<00:00,  9.12it/s]
Train loss: 0.12487516906346517, Train accuracy: 0.9671910112359551
Validation loss: 0.5769467598492546, Validation accuracy: 0.8557951482479785
```