Documentation: Data Preprocessing and Model Training

This code performs the following tasks:

1. Data Preparation: Loads and preprocesses a dataset containing text comments and their toxicity labels.

2. Model Training: Fine-tunes a pre-trained transformer model (`roberta-base`) for sequence classification to detect toxic/harmful comments.

3. Evaluation: Evaluates the model on a test set and generates alternative phrasings for detected toxic comments.

The code is designed to work with a CSV file containing text data (`body`), toxicity labels (`toxicity\_level`), and a split column (`split`) indicating whether the data belongs to the `train` or `test` set.

2. Data Preprocessing

2.1. Loading the Dataset

- Input File: The dataset is loaded from a CSV file (`final\_labels.csv`).

- Columns Used:

- body: The text of the comment.

- `toxicity\_level`: The label indicating whether the comment is toxic (e.g., `Misogynistic`) or non-toxic (e.g., `Nonmisogynistic`).

- `split`: Indicates whether the data belongs to the `train` or `test` set.

Code:

file\_path = "/content/final\_labels.csv"

df = pd.read\_csv(file\_path)

2.2. Handling Missing Values

- Missing values in the `body`, `toxicity\_level`, and `split` columns are dropped to ensure clean data for training and evaluation.

Code:

df = df[['body', 'toxicity\_level', 'split']].dropna()

2.3. Label Encoding

- The `toxicity\_level` column is encoded into numerical labels using `LabelEncoder` from `scikit-learn`.

- The available labels are expected to be `['Misogynistic', 'Nonmisogynistic']`.

Code:

from sklearn.preprocessing import LabelEncoder

label\_encoder = LabelEncoder()

df['label'] = label\_encoder.fit\_transform(df['toxicity\_level'])

2.4. Splitting the Dataset

- The dataset is split into `train` and `test` sets based on the `split` column.

Code:

train\_df = df[df['split'] == 'train']

test\_df = df[df['split'] == 'test']

3. Model Training

3.1. Custom Dataset Class

- A custom `Dataset` class (`ToxicityDataset`) is created to handle tokenization and formatting of the text data for the transformer model.

- The `\_\_getitem\_\_` method tokenizes the text using the provided tokenizer and returns `input\_ids`, `attention\_mask`, and `labels`.

Code:

class ToxicityDataset(Dataset):

def \_\_init\_\_(self, texts, labels, tokenizer, max\_length=128):

self.texts = texts.tolist()

self.labels = labels.tolist()

self.tokenizer = tokenizer

self.max\_length = max\_length

def \_\_getitem\_\_(self, idx):

encoding = self.tokenizer(

self.texts[idx],

truncation=True,

padding='max\_length',

max\_length=self.max\_length,

return\_tensors='pt'

)

return {

"input\_ids": encoding["input\_ids"].squeeze(),

"attention\_mask": encoding["attention\_mask"].squeeze(),

"labels": torch.tensor(self.labels[idx], dtype=torch.long),

}

3.2. Model and Tokenizer Setup

- The `roberta-base` model and tokenizer are loaded using the `transformers` library.

- The model is configured for sequence classification with the number of labels set to 2 (`Misogynistic` and `Nonmisogynistic`).

Code:

from transformers import AutoTokenizer, AutoModelForSequenceClassification

model\_name = "roberta-base"

tokenizer = AutoTokenizer.from\_pretrained(model\_name)

model = AutoModelForSequenceClassification.from\_pretrained(model\_name, num\_labels=num\_labels)

3.3. Dataset Objects

- The `ToxicityDataset` class is used to create `train\_dataset` and `test\_dataset` objects for training and evaluation.

Code:

train\_dataset = ToxicityDataset(train\_df["body"], train\_df["label"], tokenizer)

test\_dataset = ToxicityDataset(test\_df["body"], test\_df["label"], tokenizer)

3.4. Metrics Calculation

- A function `compute\_metrics` is defined to calculate evaluation metrics such as accuracy, precision, recall, and F1-score.

Code:

from sklearn.metrics import accuracy\_score, precision\_recall\_fscore\_support

def compute\_metrics(pred):

labels = pred.label\_ids

preds = pred.predictions.argmax(-1)

precision, recall, f1, \_ = precision\_recall\_fscore\_support(

labels, preds, average='weighted', zero\_division=0

)

acc = accuracy\_score(labels, preds)

return {

"accuracy": acc,

"precision": precision,

"recall": recall,

"f1": f1

}

3.5. Training Arguments

- Training arguments are configured using `TrainingArguments` from the `transformers` library.

- Key parameters include:

- Batch size (`per\_device\_train\_batch\_size`, `per\_device\_eval\_batch\_size`)

- Number of epochs (`num\_train\_epochs`)

- Weight decay (`weight\_decay`)

- Logging and evaluation strategies (`logging\_steps`, `eval\_strategy`, `save\_strategy`)

Code:

training\_args = TrainingArguments(

output\_dir="./results",

eval\_strategy="epoch",

save\_strategy="epoch",

per\_device\_train\_batch\_size=8,

per\_device\_eval\_batch\_size=8,

num\_train\_epochs=3,

weight\_decay=0.01,

logging\_dir="./logs",

logging\_steps=10,

load\_best\_model\_at\_end=True,

metric\_for\_best\_model="f1",

greater\_is\_better=True,

)

3.6. Trainer Setup

- The `Trainer` class is used to handle the training loop, evaluation, and saving of the model.

Code:

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=train\_dataset,

eval\_dataset=test\_dataset,

compute\_metrics=compute\_metrics

)

3.7. Fine-Tuning the Model

- The model is fine-tuned using the `trainer.train()` method.

Code

trainer.train()

4. Evaluation and Toxic Comment Detection

4.1. Model Evaluation

- The model is evaluated on the test set using the `trainer.predict()` method.

- Predicted labels are added to the `test\_df` DataFrame for further analysis.

predictions = trainer.predict(test\_dataset)

pred\_labels = predictions.predictions.argmax(-1)

test\_df.loc[:, 'predicted\_label'] = pred\_labels

4.2. Detecting Toxic Comments

- Toxic comments are identified by filtering the test set for rows where the `predicted\_label` matches the encoded value of the target label (`Misogynistic`).

Code:

toxic\_label\_index = list(label\_encoder.classes\_).index(target\_label)

toxic\_comments\_df = test\_df[test\_df['predicted\_label'] == toxic\_label\_index]

5. Generating Alternative Phrasings

5.1. Text Generation Model

- A `t5-base` model is used to generate alternative phrasings for toxic comments.

- The `text2text-generation` pipeline is employed for this task.

Code:

from transformers import pipeline

generation\_model = pipeline("text2text-generation", model="t5-base")

5.2. Generating Alternatives

- For each toxic comment, a prompt is created to rewrite the comment in a more respectful and constructive manner.

Code:

def generate\_alternative(text):

prompt = f"Rewrite the following comment to be more respectful and constructive: {text}"

result = generation\_model(prompt, max\_length=128, truncation=True)

alternative = result[0]['generated\_text']

return alternative