Report on Data Preprocessing and Model Training

Data Preprocessing

Data preprocessing is a critical step in any machine learning pipeline. It ensures that the data is clean, consistent, and ready for model training. Below is a detailed breakdown of the data preprocessing steps implemented in this project:

1.1 Data Loading

- The dataset is loaded from a CSV file (`final\_labels.csv`) using `pandas`.

- The dataset contains two key columns:

- `body`: The text data (e.g., subreddit posts or comments).

- `group`: The labels (e.g., subreddit categories).

Command:

df = pd.read\_csv("/content/final\_labels.csv")

1.2 Column Validation

- The code checks for the presence of required columns (`body` and `group`).

- If the columns are missing or incorrectly named, an error is raised with a list of available columns.

Command:

if 'body' not in df.columns or 'group' not in df.columns:

raise KeyError("Required columns 'body' and 'group' not found in dataset.")

1.3 Column Renaming and Cleaning

- Column names are standardized by converting them to lowercase and stripping any leading/trailing whitespace.

- The dataset is filtered to retain only the `body` and `group` columns, which are renamed to `text` and `label` for consistency.

Command:

df.rename(columns=lambda x: x.strip().lower(), inplace=True)

df = df[['body', 'group']].rename(columns={'body': 'text', 'group': 'label'})

1.4 Handling Missing Values

- Rows with missing values (NaN) in either the `text` or `label` columns are dropped to ensure data quality.

df = df.dropna()

Command:

5 Text and Label Formatting

- The `text` column is converted to string format to avoid any inconsistencies.

- Text labels are mapped to integer values using a label mapping dictionary. This is necessary because the model requires numerical labels for training.

Command:

df['text'] = df['text'].astype(str)

label\_mapping = {label: idx for idx, label in enumerate(df['label'].unique())}

df['label'] = df['label'].map(label\_mapping).astype(int)

1.6 Train-Validation Split

- The dataset is split into training and validation sets using an 80-20 split.

- `train\_test\_split` from `sklearn.model\_selection` is used for this purpose.

Command:

train\_texts, val\_texts, train\_labels, val\_labels = train\_test\_split(

df['text'].tolist(), df['label'].tolist(), test\_size=0.2, random\_state=42

)

1.7 Tokenization

- The `DistilBertTokenizer` is used to tokenize the text data.

- The tokenization process includes:

- Converting text into input IDs.

- Adding attention masks.

- Padding and truncating sequences to a fixed length.

Command:

tokenizer = DistilBertTokenizer.from\_pretrained("distilbert-base-uncased")

def tokenize\_function(examples):

return tokenizer(examples["text"], padding="max\_length", truncation=True)

dataset = Dataset.from\_dict({"text": train\_texts, "label": train\_labels})

dataset = dataset.map(tokenize\_function, batched=True)

dataset = dataset.train\_test\_split(test\_size=0.2)

2. Model Training

The model training process involves fine-tuning the DistilBERT model for the text classification task. Below is a detailed breakdown of the training steps:

2.1 Model Initialization

- The `DistilBertForSequenceClassification` model is initialized with the number of labels equal to the number of unique classes in the dataset.

Command:

model = DistilBertForSequenceClassification.from\_pretrained(

"distilbert-base-uncased", num\_labels=len(label\_mapping)

2.2 Training Arguments

- Training arguments are defined using the `TrainingArguments` class from Hugging Face.

- Key parameters include:

- Batch size: 8 (for both training and evaluation).

- Number of epochs: 10.

- Weight decay: 0.01 (for regularization).

- Evaluation strategy: Evaluate after each epoch.

Code:

training\_args = TrainingArguments(

output\_dir="./results",

evaluation\_strategy="epoch",

save\_strategy="epoch",

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

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

num\_train\_epochs=10,

weight\_decay=0.01,

logging\_dir="./logs",

)

2.3 Trainer Setup

- The `Trainer` class is used to handle the training loop.

- The trainer is initialized with the model, training arguments, datasets, and the custom `compute\_metrics` function.

Code:

trainer = Trainer(

model=model,

args=training\_args,

train\_dataset=dataset['train'],

eval\_dataset=dataset['test'],

compute\_metrics=compute\_metrics

)

2.4 Training

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

- Training progresses for 10 epochs, with evaluation performed after each epoch.

Code:

trainer.train()

2.5 Evaluation

- After training, the model is evaluated on the validation set.

- The evaluation metrics (accuracy, precision, recall, F1-score, and loss) are computed and logged.

Code:

metrics = trainer.evaluate()

print(f"Final Evaluation Results - Accuracy: {metrics['eval\_accuracy']}, Precision: {metrics['eval\_precision']}, Recall: {metrics['eval\_recall']}, F1 Score: {metrics['eval\_f1']}

2.6 Model Saving

- The trained model and tokenizer are saved to the directory `./subreddit\_classifier` for future use.

Code:

model.save\_pretrained("./subreddit\_classifier")

tokenizer.save\_pretrained("./subreddit\_classifier")