Praful_Patil_HW4B

March 4, 2024

####**Explaination for why we do not need collate function** The main reason to use collate function is to perform operations such as padding, so that each batch has tensors of the same shape. But in case of Bag of Words (BoW) approach with TF-IDF vectorization text documents are converted into fixed-length feature vectors. By specifying max_features i.e in our case max_features = 5000 and due to this there's no need for padding or other size adjustments typically handled by a collate function.

###**Setting up the environment**

[54]:
```
from pathlib import Path
if 'google.colab' in str(get_ipython()):
    from google.colab import drive
    drive.mount("/content/drive")
    !pip install datasets transformers evaluate wandb accelerate -U -qq
    base_folder = Path("/content/drive/MyDrive")
else:
    base_folder = Path("/home/harpreet/Insync/google_drive_shaannoor/data")

from sklearn.model_selection import train_test_split
import evaluate
import torch
from torch.utils.data import Dataset, DataLoader
import ast
import joblib
import torch.nn as nn
from collections import Counter
import numpy as np
from sklearn.preprocessing import MultiLabelBinarizer
!pip install torchmetrics
from torchmetrics import HammingDistance
from torchmetrics.classification import  MultilabelHammingDistance
from  torchmetrics.functional.classification import multilabel_f1_score,␣
↪multilabel_hamming_distance
from torch.nn.utils import clip_grad_value_
import pandas as pd
from functools import partial
from types import SimpleNamespace
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
```

```
drive.mount("/content/drive", force_remount=True).
Requirement already satisfied: torchmetrics in /usr/local/lib/python3.10/dist-packages (1.3.1)
Requirement already satisfied: numpy>1.20.0 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (1.25.2)
Requirement already satisfied: packaging>17.1 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (23.2)
Requirement already satisfied: torch>=1.10.0 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (2.1.0+cu121)
Requirement already satisfied: lightning-utilities>=0.8.0 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (0.10.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from lightning-utilities>=0.8.0->torchmetrics) (67.7.2)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from lightning-utilities>=0.8.0->torchmetrics) (4.10.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetrics) (3.13.1)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetrics) (1.12)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetrics) (3.2.1)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetrics) (3.1.3)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetrics) (2023.6.0)
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.10.0->torchmetrics) (2.1.0)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.10.0->torchmetrics) (2.1.5)
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.10.0->torchmetrics) (1.3.0)
```

##**Loading Dataset**

[55]:
```
base_folder = Path('/content/drive/MyDrive/NLP/HW_4')
data_folder = base_folder
custom_functions = base_folder/'custom-functions'
```

[56]:
```
file_name = 'df_multilabel_hw_cleaned.joblib'
file_path = data_folder / file_name

# Load the dataset using joblib
df_multilabel = joblib.load(file_path)
```

[57]:
```
df_multilabel.head(10)
```

[57]:
```
                                        cleaned_text                Tags  \
0  asp query stre dropdown webpage follow control…           c# asp.net
1  run javascript code server java code want run …     java javascript
2  linq sql throw exception row find change hi li…           c# asp.net
3  run python script php server run nginx web ser…           php python
4  advice write function m try write function res…   javascript jquery
5  jquery auto resize function cause jump browser…   javascript jquery
6  php page redirect operation page php grid subp…      php javascript
7  advice need expert asp.net usercontrol usage n…           c# asp.net
8  revert style apply focus blur user focus text …   javascript jquery
9  hack work look android source develop app app …        java android

  Tag_Number
0     [0, 9]
1     [1, 3]
2     [0, 9]
3     [2, 7]
4     [3, 5]
5     [3, 5]
6     [2, 3]
7     [0, 9]
8     [3, 5]
9     [1, 4]
```

[58]:
```python
# Convert the 'Tag_Number' column from string representations of lists to
↪actual lists of integers
df_multilabel['Tag_Number'] = df_multilabel['Tag_Number'].apply(lambda x: ast.
↪literal_eval(x))
```

[59]:
```python
# Initialize MultiLabelBinarizer
mlb = MultiLabelBinarizer()

# Fit and transform the 'Tag_Number' column to one-hot encoded format
one_hot_labels = mlb.fit_transform(df_multilabel['Tag_Number'])

# Convert one-hot encoded labels to a DataFrame
one_hot_labels_df = pd.DataFrame(one_hot_labels, columns=mlb.classes_)

# Concatenate the one-hot encoded labels DataFrame with the original DataFrame
df_multilabel = pd.concat([df_multilabel, one_hot_labels_df], axis=1)
```

[60]:
```python
df_multilabel.head(10)
```

[60]:
```
                                        cleaned_text                Tags  \
0  asp query stre dropdown webpage follow control…           c# asp.net
1  run javascript code server java code want run …     java javascript
2  linq sql throw exception row find change hi li…           c# asp.net
```

```
3  run python script php server run nginx web ser…         php python
4  advice write function m try write function res…  javascript jquery
5  jquery auto resize function cause jump browser…  javascript jquery
6  php page redirect operation page php grid subp…    php javascript
7  advice need expert asp.net usercontrol usage n…         c# asp.net
8  revert style apply focus blur user focus text …  javascript jquery
9  hack work look android source develop app app …       java android

  Tag_Number  0  1  2  3  4  5  6  7  8  9
0     [0, 9]  1  0  0  0  0  0  0  0  0  1
1     [1, 3]  0  1  0  1  0  0  0  0  0  0
2     [0, 9]  1  0  0  0  0  0  0  0  0  1
3     [2, 7]  0  0  1  0  0  0  0  1  0  0
4     [3, 5]  0  0  0  1  0  1  0  0  0  0
5     [3, 5]  0  0  0  1  0  1  0  0  0  0
6     [2, 3]  0  0  1  1  0  0  0  0  0  0
7     [0, 9]  1  0  0  0  0  0  0  0  0  1
8     [3, 5]  0  0  0  1  0  1  0  0  0  0
9     [1, 4]  0  1  0  0  1  0  0  0  0  0
```

##**Splitting the dataset**

[61]:
```python
X = df_multilabel['cleaned_text'].values
y = one_hot_labels
```

[62]:
```python
# Split the data into train, validation, and test sets (60%, 20%, 20%)
X_train, X_temp, y_train, y_temp = train_test_split(X, y, test_size=0.4, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

##**tfidf vectorizer with max features: 5000**

[63]:
```python
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer(max_features=5000)
```

[64]:
```python
X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)

# Only transforming the validation and test data
X_val_tfidf = tfidf_vectorizer.transform(X_val)
X_test_tfidf = tfidf_vectorizer.transform(X_test)
```

##**CustomDataset class for loading data and labels.**

[89]:
```python
class CustomDataset(Dataset):
    """
    Custom Dataset class for loading data and labels.
    """
```

```
    def __init__(self, X, y):
        self.X = X
        self.y = y

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

    def __getitem__(self, idx):
        texts = torch.tensor(self.X[idx], dtype=torch.float32)
        labels = torch.tensor(self.y[idx], dtype=torch.float32)
        return texts, labels
```

[80]:
```
train_tfidf_dataset = CustomDataset(X_train_tfidf.toarray(), y_train)
val_tfidf_dataset = CustomDataset(X_val_tfidf.toarray(), y_val)
test_tfidf_dataset = CustomDataset(X_test_tfidf.toarray(), y_test)
```

[81]:
```
# Get the first item from each dataset
train_features, train_labels = train_tfidf_dataset[0]
val_features, val_labels = val_tfidf_dataset[0]
test_features, test_labels = test_tfidf_dataset[0]

# Print the shape of features and labels for each dataset
print(f"Train Features Shape: {train_features.shape}, Train Labels Shape:␣
 ↪{train_labels.shape}")
print(f"Validation Features Shape: {val_features.shape}, Validation Labels␣
 ↪Shape: {val_labels.shape}")
print(f"Test Features Shape: {test_features.shape}, Test Labels Shape:␣
 ↪{test_labels.shape}")
```

```
Train Features Shape: (5000,), Train Labels Shape: (10,)
Validation Features Shape: (5000,), Validation Labels Shape: (10,)
Test Features Shape: (5000,), Test Labels Shape: (10,)
```

##**Creating custom model class**

[67]:
```
import torch.nn as nn

class SimpleMLP(nn.Module):
    def __init__(self, input_dim, hidden_dim1, drop_prob1, hidden_dim2,␣
 ↪drop_prob2, num_outputs):
        super().__init__()

        #␣
 ↪Hidden_Layer1->ReLU->Dropout_Layer1->BatchNorm_Layer1->Hidden_Layer2->ReLU->DropoutLayer2->
 ↪Layer

        # First Linear layer
```

```
        self.linear1 = nn.Linear(input_dim, hidden_dim1)
        # ReLU activation function
        self.relu1 = nn.ReLU()
        # Dropout for first linear layer
        self.dropout1 = nn.Dropout(p=drop_prob1)
        # Batch normalization for first linear layer
        self.batchnorm1 = nn.BatchNorm1d(num_features=hidden_dim1)

        # Second Linear layer
        self.linear2 = nn.Linear(hidden_dim1, hidden_dim2)
        # ReLU activation function
        self.relu2 = nn.ReLU()
        # Dropout for second linear layer
        self.dropout2 = nn.Dropout(p=drop_prob2)
        # Batch normalization for second linear layer
        self.batchnorm2 = nn.BatchNorm1d(num_features=hidden_dim2)

        # Final Linear layer
        self.linear3 = nn.Linear(hidden_dim2, num_outputs)

    def forward(self, x):
        # First linear layer followed by ReLU, BatchNorm, and Dropout
        x = self.linear1(x)
        x = self.relu1(x)
        x = self.dropout1(x)
        x = self.batchnorm1(x)

        # Second linear layer followed by ReLU, BatchNorm, and Dropout
        x = self.linear2(x)
        x = self.relu2(x)
        x = self.dropout2(x)
        x = self.batchnorm2(x)

        # Final linear layer
        x = self.linear3(x)

        return x
```

#Functions to train and evaluate the model

##Step Function

[106]:
```
def step(inputs, targets, model, device, loss_function=None, optimizer=None,␣
 ↪clip_value=10):
    """
    Performs a forward and backward pass for a given batch of inputs and␣
 ↪targets.
```

```
    Parameters:
    - inputs (torch.Tensor): The input data for the model.
    - targets (torch.Tensor): The true labels for the input data.
    - model (torch.nn.Module): The neural network model.
    - device (torch.device): The computing device (CPU or GPU).
    - loss_function (torch.nn.Module, optional): The loss function to use.
    - optimizer (torch.optim.Optimizer, optional): The optimizer to update␣
↪model parameters.

    Returns:
    - loss (float): The computed loss value (only if loss_function is not None).
    - outputs (torch.Tensor): The predictions from the model.
    - correct (int): The number of correctly classified samples in the batch.
    """
    # Move the model and data to the device
    model = model.to(device)
    inputs = inputs.to(device).float()
    targets = targets.to(device).float()

    # Step 1: Forward pass to get the model's predictions
    outputs = model(inputs)

    # Step 2a: Compute the loss using the provided loss function
    if loss_function:
        loss = loss_function(outputs, targets)

    # Step 3 and 4: Perform backward pass and update model parameters if an␣
↪optimizer is provided
    if optimizer:
        optimizer.zero_grad()
        loss.backward()
        clip_grad_value_(model.parameters(), clip_value=clip_value)
        optimizer.step()

    # Return relevant metrics
    if loss_function:
        return loss, outputs
    else:
        return outputs
```

##Train Epoch

[113]:
```
def train_epoch(train_loader, model, device, loss_function, optimizer,␣
↪train_hamming_metric, clip_value=None):
    """
    Trains the model for one epoch using the provided data loader and updates␣
↪the model parameters.
```

```
    Parameters:
    - train_loader (torch.utils.data.DataLoader): DataLoader object for the training set.
    - model (torch.nn.Module): The neural network model to be trained.
    - device (torch.device): The computing device (CPU or GPU).
    - loss_function (torch.nn.Module): The loss function to use for training.
    - optimizer (torch.optim.Optimizer): The optimizer to update model parameters.
    - train_hamming_metric (torchmetrics.Metric): The metric object for computing the hamming distance.
    - clip_value (float, optional): The maximum value for gradient clipping.

    Returns:
    - train_loss (float): Average training loss for the epoch.
    - train_hamming_score (float): Average hamming distance for the epoch.
    """
    model.train()  # Set the model to training mode
    running_train_loss = 0.0  # Initialize variable to track running training loss

    # Ensure the metric is on the correct device
    train_hamming_metric = train_hamming_metric.to(device)
    train_hamming_metric.reset()  # Reset the metric at the start of each epoch

    for inputs, targets in train_loader:
        inputs = inputs.to(device).float()
        targets = targets.to(device).float()

        optimizer.zero_grad()  # Clear gradients
        outputs = model(inputs)  # Forward pass: compute predicted outputs by passing inputs to the model
        loss = loss_function(outputs, targets)  # Calculate the loss
        loss.backward()  # Backward pass: compute gradient of the loss with respect to model parameters

        if clip_value is not None:  # If gradient clipping is specified
            torch.nn.utils.clip_grad_norm_(model.parameters(), clip_value)  # Clip gradients to avoid exploding gradients

        optimizer.step()  # Perform a single optimization step (parameter update)
        running_train_loss += loss.item()  # Update running loss

        with torch.no_grad():  # Update metric without tracking gradients
            probabilities = torch.sigmoid(outputs)
```

```
            predictions = (probabilities >= 0.5).float()  # Apply threshold to␣
↪get binary predictions
            train_hamming_metric.update(predictions, targets)  # Update metric␣
↪with current batch

    # Compute average loss and metric score for the entire training set
    train_loss = running_train_loss / len(train_loader)
    train_hamming_score = train_hamming_metric.compute()  # Compute the average␣
↪hamming distance over the epoch

    return train_loss, train_hamming_score.item()  # Return average loss and␣
↪hamming score as float
```

##Val Epoch

[114]:
```
def val_epoch(val_loader, model, device, loss_function, val_hamming_metric):
    model.eval()
    running_val_loss = 0.0

    val_hamming_metric.reset()

    with torch.no_grad():
        for inputs, targets in val_loader:
            inputs, targets = inputs.to(device).float(), targets.to(device).
↪float()

            outputs = model(inputs)
            loss = loss_function(outputs, targets)

            running_val_loss += loss.item()

            predictions = torch.sigmoid(outputs) >= 0.5
            val_hamming_metric.update(predictions, targets)

    val_loss = running_val_loss / len(val_loader)
    val_hamming_score = val_hamming_metric.compute()

    return val_loss, val_hamming_score
```

###Train() function

[115]:
```
def train(train_loader, val_loader, model, optimizer, loss_function, epochs,␣
↪device, train_hamming, val_hamming, clip_value=10):
    train_loss_history = []
    val_loss_history = []
    train_hamming_history = []
    val_hamming_history = []
```

```
    for epoch in range(epochs):
        # Training phase
        train_loss, train_hamming_score = train_epoch(
            train_loader, model, device, loss_function, optimizer,␣
↪train_hamming, clip_value
        )

        # Validation phase
        val_loss, val_hamming_score = val_epoch(
            val_loader, model, device, loss_function, val_hamming
        )

        # Store metrics
        train_loss_history.append(train_loss)
        train_hamming_history.append(train_hamming_score)
        val_loss_history.append(val_loss)
        val_hamming_history.append(val_hamming_score)

        # Print epoch summary
        print(f"Epoch {epoch+1}/{epochs}")
        print(f"Train Loss: {train_loss:.4f} | Train Hamming:␣
↪{train_hamming_score:.4f}")
        print(f"Val Loss: {val_loss:.4f} | Val Hamming: {val_hamming_score:.
↪4f}")
        print()

    return train_loss_history, train_hamming_history, val_loss_history,␣
↪val_hamming_history
```

#**Hyperparameters and training config**

[116]:
```
import torch
from torch import nn
from torch.optim import AdamW
from torch.utils.data import DataLoader
from torchmetrics import HammingDistance

# Hyperparameters and configurations
HIDDEN_DIM1 = 200
HIDDEN_DIM2 = 100
BATCH_SIZE = 128
LEARNING_RATE = 0.001
WEIGHT_DECAY = 0.000
EPOCHS = 5
CLIP_VALUE = 10
NUM_OUTPUTS = 10
```

```
DROP_PROB1 = 0.3
DROP_PROB2 = 0.2

# Data loaders
train_loader = DataLoader(train_tfidf_dataset, batch_size=BATCH_SIZE,␣
 ↪shuffle=True)
val_loader = DataLoader(val_tfidf_dataset, batch_size=BATCH_SIZE)
test_loader = DataLoader(test_tfidf_dataset, batch_size=BATCH_SIZE)

# Determine the computing device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')

# Model initialization
input_dim = train_tfidf_dataset[0][0].shape[0]  # Input dimension should match␣
 ↪the TF-IDF vector size
model = SimpleMLP(
    input_dim = input_dim,
    hidden_dim1=HIDDEN_DIM1,
    hidden_dim2=HIDDEN_DIM2,
    drop_prob1=DROP_PROB1,
    drop_prob2=DROP_PROB2,
    num_outputs=NUM_OUTPUTS
)

# Loss and optimizer
loss_function = nn.BCEWithLogitsLoss()
optimizer = AdamW(model.parameters(), lr=LEARNING_RATE,␣
 ↪weight_decay=WEIGHT_DECAY)

# Metric
train_hamming = HammingDistance(task="multilabel", num_labels=NUM_OUTPUTS).
 ↪to(device)
val_hamming = HammingDistance(task="multilabel", num_labels=NUM_OUTPUTS).
 ↪to(device)
```

##**Training**

[117]:
```
# Call the training function to start the training process
train_losses, train_acc, valid_losses, valid_acc = train(
    train_loader, val_loader, model, optimizer, loss_function, EPOCHS, device,
    train_hamming, val_hamming, clip_value= CLIP_VALUE)
```

```
Epoch 1/5
Train Loss: 0.3082 | Train Hamming: 0.1160
Val Loss: 0.1280 | Val Hamming: 0.0434

Epoch 2/5
```

```
Train Loss: 0.1040 | Train Hamming: 0.0356
Val Loss: 0.1069 | Val Hamming: 0.0379

Epoch 3/5
Train Loss: 0.0732 | Train Hamming: 0.0256
Val Loss: 0.1030 | Val Hamming: 0.0366

Epoch 4/5
Train Loss: 0.0551 | Train Hamming: 0.0192
Val Loss: 0.1079 | Val Hamming: 0.0362

Epoch 5/5
Train Loss: 0.0443 | Train Hamming: 0.0157
Val Loss: 0.1116 | Val Hamming: 0.0359
```

[124]:
```python
import torch

def get_acc_pred(data_loader, model, device, threshold=0.5):
    model.to(device)
    model.eval()

    all_predictions = []
    all_targets = []

    with torch.no_grad():
        for inputs, targets in data_loader:
            inputs = inputs.to(device).float()
            targets = targets.to(device)

            outputs = model(inputs)  #
            predicted = (torch.sigmoid(outputs) > threshold).float()

            all_predictions.append(predicted.cpu())
            all_targets.append(targets.cpu())

    all_predictions = torch.cat(all_predictions)
    all_targets = torch.cat(all_targets)

    return all_predictions, all_targets
```

[126]:
```python
predictions_test, target_test = get_acc_pred(test_loader, model, device)
predictions_train, target_train = get_acc_pred(train_loader, model, device)
predictions_valid, target_valid = get_acc_pred(val_loader, model, device)
```

[127]:
```python
#multi-label classification where labels = 10
num_labels = 10
```

```
# Calculate Hamming Distance for the training dataset
train_hd = multilabel_hamming_distance(predictions_train.int(), target_train.
 ↪int(), num_labels=num_labels)

# Calculate Hamming Distance for the validation dataset
valid_hd = multilabel_hamming_distance(predictions_valid.int(), target_valid.
 ↪int(), num_labels=num_labels)

# Calculate Hamming Distance for the test dataset
test_hd = multilabel_hamming_distance(predictions_test.int(), target_test.
 ↪int(), num_labels=num_labels)
```

[128]:
```
print(f"Training Hamming Distance: {train_hd}")
print(f"Validation Hamming Distance: {valid_hd}")
print(f"Test Hamming Distance: {test_hd}")
```

```
Training Hamming Distance: 0.00638881279155612
Validation Hamming Distance: 0.03586716204881668
Test Hamming Distance: 0.036369387060403824
```

##**Understanding model's performance with F1_score** f1_score is called with average='micro' to compute the micro-averaged F1 score, which aggregates the contributions of all classes to compute the average metric.

[129]:
```
from torchmetrics.functional import f1_score

# F1 Score for the training dataset
train_f1_micro = f1_score(predictions_train, target_train,␣
 ↪num_labels=num_labels, average='micro',task='multilabel')

# F1 Score for the validation dataset
valid_f1_micro = f1_score(predictions_valid, target_valid,␣
 ↪num_labels=num_labels, average='micro',task='multilabel')

# F1 Score for the test dataset
test_f1_micro = f1_score(predictions_test, target_test, num_labels=num_labels,␣
 ↪average='micro',task='multilabel')

print(f"Training F1 Score (Micro): {train_f1_micro}")
print(f"Validation F1 Score (Micro): {valid_f1_micro}")
print(f"Test F1 Score (Micro): {test_f1_micro}")
```

```
Training F1 Score (Micro): 0.9845678210258484
Validation F1 Score (Micro): 0.9126527905464172
Test F1 Score (Micro): 0.9114567041397095
```

#####**The Tfidf-based model demonstrates excellent performance on the training**

data, as evidenced by the notably low Hamming Distance and the impressively high F1 score. slight performance reduction on the validation and test sets is within expected bounds and does not significantly detract from the model's overall strong performance.

#####**The Tfidf approach outperforms Dense Embedding, showing lower Hamming Distances and higher F1 scores, indicating more accurate label predictions. Despite a slight performance drop on unseen data, both methods demonstrate good generalization, with Tfidf maintaining a stronger edge.**