Harikrishna Dev HXD220000

In [ ]:
```
# If in Colab, then import the drive module from google.colab
import sys
if 'google.colab' in str(get_ipython()):
  from google.colab import drive
  # Mount the Google Drive to access files stored there
  drive.mount('/content/drive')

  # Install the latest version of torchtext library quietly without showing

  !pip install torchtext -qq
  # Install the torchinfo library quietly
  !pip install torchinfo -qq
  # !pip install torchtext --upgrade -qq
  !pip install torchmetrics -qq
  !pip install torchinfo -qq
  !pip install fast_ml -qq
  !pip install joblib -qq
  !pip install sklearn -qq
  !pip install pandas -qq
  !pip install numpy -qq
  !pip install scikit-multilearn -qq
  !pip install transformers evaluate wandb accelerate -U -qq
  !pip install pytorch-ignite -qq -U

  basepath = '/content/drive/MyDrive/Colab_Notebooks/BUAN_6342_Applied_Natur
  sys.path.append('/content/drive/MyDrive/Colab_Notebooks/BUAN_6342_Applied_
else:
  basepath = '/Users/harikrishnadev/Library/CloudStorage/GoogleDrive-harikri
  sys.path.append('/Users/harikrishnadev/Library/CloudStorage/GoogleDrive-ha
  # !pip install torchtext -qq
  # # Install the torchinfo library quietly
  # !pip install torchinfo -qq
  # !pip install torchtext --upgrade -qq
  # !pip install torchmetrics -qq
  # !pip install torchinfo -qq
  # !pip install fast_ml -qq
  # !pip install joblib -qq
  # !pip install sklearn -qq
  # !pip install pandas -qq
  # !pip install numpy -qq
  # !pip install scikit-multilearn -qq
  # !pip install transformers evaluate wandb accelerate -U -qq
```

```
Mounted at /content/drive
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 840.4/840.4 kB 8.8 MB/s eta 0:
00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.1/42.1 kB 1.4 MB/s eta 0:0
0:00
  error: subprocess-exited-with-error

  × python setup.py egg_info did not run successfully.
  │ exit code: 1
  ╰─> See above for output.

  note: This error originates from a subprocess, and is likely not a problem
with pip.
  Preparing metadata (setup.py) ... error
error: metadata-generation-failed

× Encountered error while generating package metadata.
╰─> See above for output.

note: This is an issue with the package mentioned above, not pip.
hint: See above for details.
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 89.4/89.4 kB 2.6 MB/s eta 0:0
0:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.5/8.5 MB 19.6 MB/s eta 0:00:
00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 84.1/84.1 kB 11.5 MB/s eta 0:0
0:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 2.2/2.2 MB 39.3 MB/s eta 0:00:
00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 280.0/280.0 kB 30.8 MB/s eta
0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 510.5/510.5 kB 44.7 MB/s eta
0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 116.3/116.3 kB 16.0 MB/s eta
0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 134.8/134.8 kB 17.4 MB/s eta
0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 195.4/195.4 kB 24.4 MB/s eta
0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 258.5/258.5 kB 29.0 MB/s eta
0:00:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.7/62.7 kB 8.8 MB/s eta 0:0
0:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 272.4/272.4 kB 5.0 MB/s eta 0:
00:00
```

Import Libraries

In [ ]:

```python
# Importing PyTorch library for tensor computations and neural network modul
import torch
import torch.nn as nn


# For working with textual data vocabularies and for displaying model summar
from torchtext.vocab import vocab
from torchinfo import summary
```

```
# General-purpose Python libraries for random number generation and numerica
import random
import numpy as np

# Utilities for efficient serialization/deserialization of Python objects ar
import joblib
from collections import Counter

# For creating lightweight attribute classes and for partial function applic
from functools import partial

# For filesystem path handling, generating and displaying confusion matrices
from pathlib import Path
from sklearn.metrics import confusion_matrix
from datetime import datetime

# For plotting and visualization
import matplotlib.pyplot as plt
import seaborn as sns

# Printing: Import the pprint function from the pprint module for formatted
from pprint import pprint
```

Specify Project Folders

In [ ]:
```
# Set the base folder path using the Path class for better path handling
base_folder = Path(basepath)

# Define the data folder path by appending the relative path to the base fol
# This is where the data files will be stored
data_folder = base_folder / '0_Data_Folder'

# Define the model folder path for saving trained models
# This path points to a specific folder designated for NLP models related to
model_folder = data_folder

custom_functions = base_folder / '0_Custom_files'
```

In [ ]:
```
# Create the model folder directory. If it already exists, do nothing.
# The 'parents=True' argument ensures that all parent directories are create
model_folder.mkdir(exist_ok=True, parents=True)

# Create the data folder directory in a similar manner.
data_folder.mkdir(exist_ok=True, parents=True)
```

In [ ]:
```
X_train_cleaned_file = data_folder / 'df_multilabel_hw_cleaned.joblib'

data = joblib.load(X_train_cleaned_file)
```

In [ ]:
```
data.head()
```

Out[ ]:

	cleaned_text	Tags	Tag_Number
0	asp query stre dropdown webpage follow control...	c# asp.net	[0, 9]
1	run javascript code server java code want run ...	java javascript	[1, 3]
2	linq sql throw exception row find change hi li...	c# asp.net	[0, 9]
3	run python script php server run nginx web ser...	php python	[2, 7]
4	advice write function m try write function res...	javascript jquery	[3, 5]

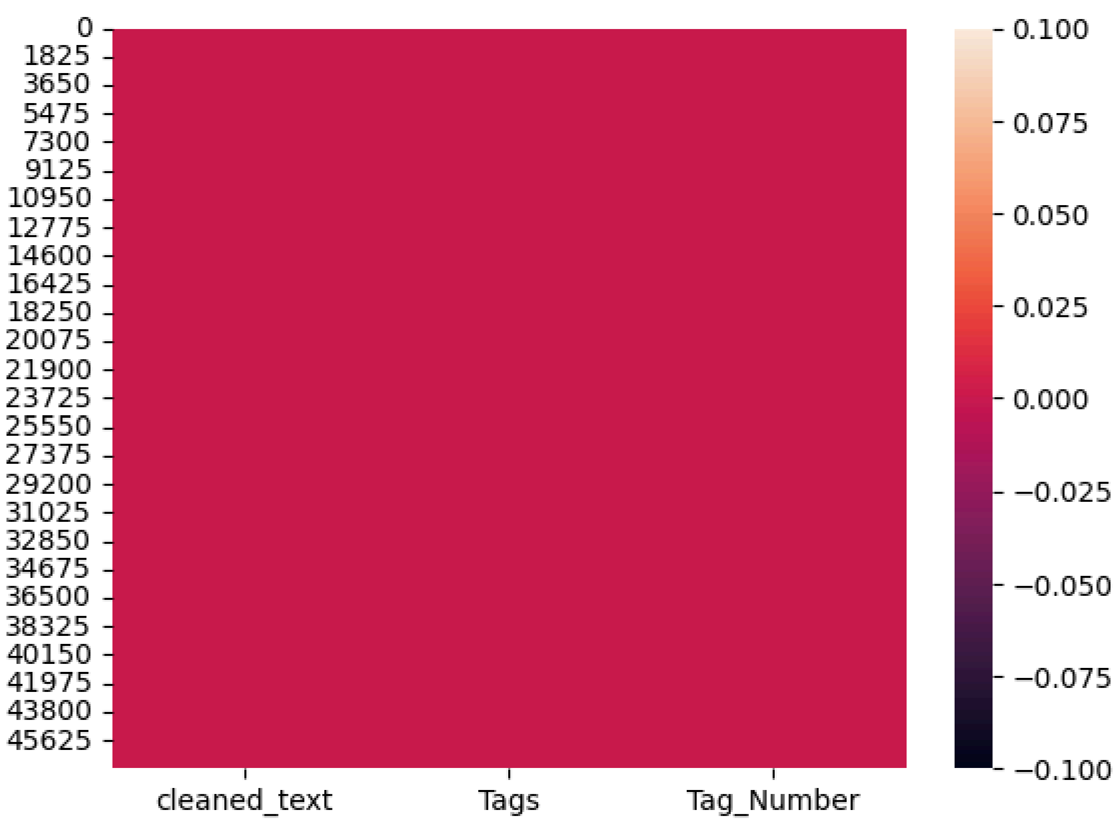
In [ ]:
```
data.describe()
```

Out[ ]:

	cleaned_text	Tags	Tag_Number
count	47427	47427	47427
unique	36481	176	176
top	cause error targetcontrolid valid value null...	javascript jquery	[3, 5]
freq	3	19989	19989

In [ ]:
```
import seaborn as sns
sns.heatmap(data.isnull())
```

Out[ ]: <Axes: >



In [ ]:
```
data['Tag_Number'].info()
```

```
<class 'pandas.core.series.Series'>
RangeIndex: 47427 entries, 0 to 47426
Series name: Tag_Number
Non-Null Count  Dtype
--------------  -----
47427 non-null  object
dtypes: object(1)
memory usage: 370.6+ KB
```

In [ ]:
```
data.columns
```

Out[ ]:
```
Index(['cleaned_text', 'Tags', 'Tag_Number'], dtype='object')
```

In [ ]:
```
# def extract_and_combine(row):
#     langs = row['Tags'].split()
#     tags = row['Tag_Number']
#     return [f'{lang} {tag}' for lang, tag in zip(langs, tags)]
# result = data.apply(extract_and_combine, axis=1)
# result
```

In [ ]:
```
import ast
y_tag = []
x = []

for i in range(data.shape[0]):
    y_tag.append(ast.literal_eval(data['Tag_Number'][i]))
    x.append(str(data['cleaned_text'][i]))

x = np.array(x).reshape(-1,1)
```

In [ ]:
```
x[:5]
```

Out[ ]:
```
array([['asp query stre dropdown webpage follow control relevance    dropdo
wnlist value hyperlink redirect page call   page cancel button redirect use
r menu page like user click hyperlink edit page index dropdownlist preserve
query string page follow aspx code sure proceed < asp hyperlink      id="ln
kedit      navigateurl=\'<% + eval("userid + sure > < /asp hyperlink >   <
asp dropdownlist      id="mydropdown      < asp listitems/ > < /asp dropdow
nlist >   edit clarify m navigateurl query string eval determine user id'],
       ['run javascript code server java code want run javascript code serv
er want manipulate result return javascript inside java code'],
       ['linq sql throw exception row find change hi linq sql get error row
find change update table help linq query show error unable figure problem w
ork get permanent solution fix problem twtmob_campainincomedetails_tb incom
edetails = datacontext.twtmob_campainincomedetails_tbs single(twtincome = >
= = tempincome                  decimal temppayout = decimal parse(lblpertw
eet text                  decimal temptotal = temppayout + tempmoneyearne
= convert tostring(temptotal                  = temptweet + 1
= lblbonus text                  = tempbudurl                  datacontext
submitchanges    twtmob_user_tb twtuserdetails = datacontext.twtmob_user_tb
s single(twtdetail = > = = tempuserid                       float temppayou
t = float parse(lblpertweet text              float tempoutstandingtotal =
temppayout+tempoutstanding              = tempoutstandingtotal
datacontext submitchanges           '],
       ["run python script php server run nginx web server php cgi like kno
w possible execute python script inside php page allow language combine att
empt briefly work sure file < body >       < p > hello message < php exec
('python > < /p >   < /body >   print hello world   clue appreciate"],
       ["advice write function m try write function resize css width elemen
t browser window resize right not get thing right love help m go wrong code
far function windowresize              $ content').css('width $ windowwidth
$ div.mysection').css('width $ windowwidth              $ div.mysection .st
ory').css('width $ windowwidth            $ document).ready(function
var $ windowwidth = $ window).width                              $ window).
resize(function                  windowresize                          m su
re place correct place advise code well great thank kyle"]],
      dtype='<U30141')
```

In [ ]:
```python
y_tag[:5]
```

Out[ ]:
```
[[0, 9], [1, 3], [0, 9], [2, 7], [3, 5]]
```

In [ ]:
```python
from sklearn.preprocessing import MultiLabelBinarizer
mlb = MultiLabelBinarizer()

y = mlb.fit_transform(y_tag)

print(type(y) , y.shape)
print(type(x) , x.shape)
```

```
<class 'numpy.ndarray'> (47427, 10)
<class 'numpy.ndarray'> (47427, 1)
```

In [ ]:
```python
y[:5]
```

```
Out[ ]: array([[1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
               [0, 1, 0, 1, 0, 0, 0, 0, 0, 0],
               [1, 0, 0, 0, 0, 0, 0, 0, 0, 1],
               [0, 0, 1, 0, 0, 0, 0, 1, 0, 0],
               [0, 0, 0, 1, 0, 1, 0, 0, 0, 0]])
```

```
In [ ]: type(x)
```

```
Out[ ]: numpy.ndarray
```

```
In [ ]: from sklearn.model_selection import train_test_split

        # Split the data into training and testing sets
        X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.2, rar

        # Further split the testing set into validation and testing sets
        X_valid, X_test, y_valid, y_test = train_test_split(X_test, y_test, test_siz
```

```
In [ ]: import CustomPreprocessorSpacy as cp
```

```
In [ ]: import spacy
        nlp = spacy.load('en_core_web_sm')
        cpp = cp.SpacyPreprocessor(model = 'en_core_web_sm', batch_size=1000)
```

```
In [ ]: from sklearn.feature_extraction.text import TfidfVectorizer
        # X_train_processed = cpp.transform([str(x) for x in X_train.tolist()])
        vectorizer = TfidfVectorizer(analyzer='word', token_pattern=r"[\S]+",max_fea
        vectorizer.fit(cpp.transform([str(x) for x in X_train.tolist()]))
```

```
/content/drive/MyDrive/Colab_Notebooks/BUAN_6342_Applied_Natural_Language_Pr
ocessing/0_Custom_files/CustomPreprocessorSpacy.py:83: MarkupResemblesLocato
rWarning: The input looks more like a filename than markup. You may want to
open this file and pass the filehandle into Beautiful Soup.
  soup = BeautifulSoup(text, "html.parser")
/usr/lib/python3.10/html/parser.py:170: XMLParsedAsHTMLWarning: It looks lik
e you're parsing an XML document using an HTML parser. If this really is an
HTML document (maybe it's XHTML?), you can ignore or filter this warning. If
it's XML, you should know that using an XML parser will be more reliable. To
parse this document as XML, make sure you have the lxml package installed, a
nd pass the keyword argument `features="xml"` into the BeautifulSoup constru
ctor.
  k = self.parse_starttag(i)
/usr/local/lib/python3.10/dist-packages/spacy/util.py:1740: UserWarning: [W1
11] Jupyter notebook detected: if using `prefer_gpu()` or `require_gpu()`, i
nclude it in the same cell right before `spacy.load()` to ensure that the mo
del is loaded on the correct device. More information: http://spacy.io/usag
e/v3#jupyter-notebook-gpu
  warnings.warn(Warnings.W111)
```

```
Out[ ]: ▾ TfidfVectorizer
        TfidfVectorizer(max_features=5000, token_pattern='[\\S]+')
```

In [ ]:
```
from scipy.sparse import csr_matrix
class CustomDataset(torch.utils.data.Dataset):
    """
    Custom Dataset class for loading text and labels.

    Attributes:
        X (numpy.ndarray): Feature data, an array of texts.
        y (list or array-like): Target labels.
        vectorizer (TfidfVectorizer): The TF-IDF vectorizer used to transfor
    """

    def __init__(self, X, y, vectorizer,cpp):
        """
        Initialize the dataset with feature and target data.

        Args:
            X (list or array-like): The feature data (texts).
            y (list or array-like): The target labels.
            vectorizer (TfidfVectorizer): The TF-IDF vectorizer used to tran
        """
        X = [str(x) for x in X.tolist()]

        # Storing feature data (texts)
        self.X = cpp.transform(X)

        # Storing the target labels
        self.y = y

        # Storing the TF-IDF vectorizer
        self.vectorizer = vectorizer

        # Transforming the texts to TF-IDF vectors
        self.X_tfidf = self.vectorizer.transform(self.X)

    def __len__(self):
        """
        Return the number of samples in the dataset.

        Returns:
            int: The total number of samples.
        """
        return len(self.X)

    def __getitem__(self, idx):
        """
        Fetch and return a single sample from the dataset at the given index

        Args:
            idx (int): Index of the sample to fetch.

        Returns:
            tuple: A tuple containing the label and the TF-IDF vector for th
        """
        # Retrieve the TF-IDF vector and corresponding label from the datase
        tfidf_vector = csr_matrix.toarray(self.X_tfidf[idx])
```

```
        label = self.y[idx]

        # Convert label to tensor of type float
        label = torch.tensor(label, dtype=torch.float)

        # Convert TF-IDF vector to tensor
        tfidf_vector = torch.tensor(tfidf_vector, dtype=torch.float)

        # Packing them into a tuple before returning

        return tfidf_vector, label
```

In [ ]:
```
type(X_train)
```

Out[ ]:
```
numpy.ndarray
```

In [ ]:
```
# Create an instance of the CustomDataset class for the training set
# This uses the cleaned training data and corresponding labels
trainset = CustomDataset(X_train, y_train,vectorizer,cpp)

# Create an instance of the CustomDataset class for the validation set
# This uses the cleaned validation data and corresponding labels
validset = CustomDataset(X_valid, y_valid,vectorizer,cpp)

# Create an instance of the CustomDataset class for the test set
# This uses the cleaned test data and corresponding labels
testset = CustomDataset(X_test, y_test,vectorizer,cpp)
```

```
/content/drive/MyDrive/Colab_Notebooks/BUAN_6342_Applied_Natural_Language_Pr
ocessing/0_Custom_files/CustomPreprocessorSpacy.py:83: MarkupResemblesLocato
rWarning: The input looks more like a filename than markup. You may want to
open this file and pass the filehandle into Beautiful Soup.
  soup = BeautifulSoup(text, "html.parser")
/usr/lib/python3.10/html/parser.py:170: XMLParsedAsHTMLWarning: It looks lik
e you're parsing an XML document using an HTML parser. If this really is an
HTML document (maybe it's XHTML?), you can ignore or filter this warning. If
it's XML, you should know that using an XML parser will be more reliable. To
parse this document as XML, make sure you have the lxml package installed, a
nd pass the keyword argument `features="xml"` into the BeautifulSoup constru
ctor.
  k = self.parse_starttag(i)
/usr/local/lib/python3.10/dist-packages/spacy/util.py:1740: UserWarning: [W1
11] Jupyter notebook detected: if using `prefer_gpu()` or `require_gpu()`, i
nclude it in the same cell right before `spacy.load()` to ensure that the mo
del is loaded on the correct device. More information: http://spacy.io/usag
e/v3#jupyter-notebook-gpu
  warnings.warn(Warnings.W111)
```

In [ ]:
```
trainset[:5]
```

Out[ ]:
```
(tensor([[0., 0., 0.,  ..., 0., 0., 0.],
         [0., 0., 0.,  ..., 0., 0., 0.],
         [0., 0., 0.,  ..., 0., 0., 0.],
         [0., 0., 0.,  ..., 0., 0., 0.],
         [0., 0., 0.,  ..., 0., 0., 0.]]),
 tensor([[0., 0., 0., 1., 0., 1., 0., 0., 0., 0.],
         [0., 0., 0., 1., 0., 1., 0., 0., 0., 0.],
         [0., 1., 0., 0., 1., 0., 0., 0., 0., 0.],
         [0., 0., 1., 1., 0., 0., 0., 0., 0., 0.],
         [0., 1., 0., 0., 0., 0., 0., 1., 0., 0.]]))
```

In [ ]:
```
batch_size = 2
check_loader = torch.utils.data.DataLoader(dataset=trainset,
                                           batch_size=batch_size,
                                           shuffle=True,
                                           )
```

In [ ]:
```
torch.manual_seed(22)
for tfidf_vector, label in check_loader:
    print(tfidf_vector, label)
    break
```

```
tensor([[[0.0597, 0.0000, 0.0000,  ..., 0.0000, 0.0000, 0.0000]],

        [[0.1569, 0.0616, 0.0000,  ..., 0.0000, 0.0000, 0.0000]]]) tensor
([[0., 0., 0., 1., 0., 1., 0., 0., 0., 0.],
        [0., 0., 0., 1., 0., 1., 0., 0., 0., 0.]])
```

In [ ]:
```
class CustomModel(nn.Module):
    def __init__(self, input_dim, hidden_dim1, hidden_dim2, drop_prob1, drop
        super().__init__()
        self.hidden1 = nn.Linear(input_dim, hidden_dim1)
        self.relu1 = nn.ReLU()
        self.dropout1 = nn.Dropout(p=drop_prob1)
        self.batchnorm1 = nn.BatchNorm1d(num_features=hidden_dim1)
        self.hidden2 = nn.Linear(hidden_dim1, hidden_dim2)
        self.relu2 = nn.ReLU()
        self.dropout2 = nn.Dropout(p=drop_prob2)
        self.batchnorm2 = nn.BatchNorm1d(num_features=hidden_dim2)
        self.output = nn.Linear(hidden_dim2, output_dim)

    def forward(self, x):
        x = self.hidden1(x)
        x = self.relu1(x)
        x = self.dropout1(x)
        x = self.batchnorm1(x)
        x = self.hidden2(x)
        x = self.relu2(x)
        x = self.dropout2(x)
        x = self.batchnorm2(x)
        x = self.output(x)
        return x
```

In [ ]:
```
INPUT_DIM=5000
HIDDEN_DIM1=200
HIDDEN_DIM2=100
```

```python
DROP_PROB1=0.5
DROP_PROB2=0.5
NUM_OUTPUTS = 10
EPOCHS=5
BATCH_SIZE=128
LEARNING_RATE=0.001
WEIGHT_DECAY=0.000
CLIP_VALUE = 10
PATIENCE = 5
dropout_p = 0.3
```

In [ ]:
```python
import torch.optim as optim
from torch.utils.data import DataLoader
from tqdm import tqdm

# Define the model
# input_size=5000
# Define the sequential model
model = CustomModel(input_dim=INPUT_DIM,
                       hidden_dim1=HIDDEN_DIM1,
                       hidden_dim2=HIDDEN_DIM2,
                       drop_prob1=0.5,
                       drop_prob2=0.5,
                       output_dim=NUM_OUTPUTS)
```

In [ ]:
```python
summary(model,(1, 5000))
```

Out[ ]:
```
===========================================================================
===============
Layer (type:depth-idx)                   Output Shape              Param #
===========================================================================
===============
CustomModel                              [1, 10]                   --
├─Linear: 1-1                            [1, 200]                  1,000,20
0
├─ReLU: 1-2                              [1, 200]                  --
├─Dropout: 1-3                           [1, 200]                  --
├─BatchNorm1d: 1-4                       [1, 200]                  400
├─Linear: 1-5                            [1, 100]                  20,100
├─ReLU: 1-6                              [1, 100]                  --
├─Dropout: 1-7                           [1, 100]                  --
├─BatchNorm1d: 1-8                       [1, 100]                  200
├─Linear: 1-9                            [1, 10]                   1,010
===========================================================================
===============
Total params: 1,021,910
Trainable params: 1,021,910
Non-trainable params: 0
Total mult-adds (M): 1.02
===========================================================================
===============
Input size (MB): 0.02
Forward/backward pass size (MB): 0.00
Params size (MB): 4.09
Estimated Total Size (MB): 4.11
===========================================================================
===============
```

In [ ]:
```
# Define the device
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

# Move the model to the device
model = model.to(device)

# Generate some dummy input data and offsets, and move them to the device
data = trainset[0][0].squeeze(1).to(device)
```

In [ ]:
```
trainset[0][0].shape
```

Out[ ]:
```
torch.Size([1, 5000])
```

In [ ]:
```
output = model(data)

print(output)
```

```
tensor([[-0.0735, -0.0666, -0.0945, -0.0256, -0.0607,  0.0868, -0.0687, -0.0
840,
          0.0552, -0.0071]], device='cuda:0', grad_fn=<AddmmBackward0>)
```

In [ ]:
```
from torchmetrics import HammingDistance

def step(inputs, targets, model, device, loss_function=None, optimizer=None,
    """
```

```
    Performs a forward and backward pass for a given batch of inputs and tar

    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 m

    Returns:
    - loss (float): The computed loss value (only if loss_function is not No
    - outputs (torch.Tensor): The predictions from the model.
    - train_hamming_distance (torchmetrics.HammingDistance): The Hamming dis
    """
    # Move the model and data to the device
    model = model.to(device)
    inputs, targets = inputs.squeeze(1).to(device), targets.to(device)

    # 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 2b: Update Hamming Distance metric
    train_hamming_distance = HammingDistance(task="multilabel", num_labels=1
    y_pred = (outputs > 0.5).float()
    train_hamming_distance.update(y_pred, targets)

    # Step 3 and 4: Perform backward pass and update model parameters if an
    if optimizer:
        optimizer.zero_grad()
        loss.backward()
        if clip_type == 'value':
            torch.nn.utils.clip_grad_value_(model.parameters(), clip_value)
        optimizer.step()

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

In [ ]:
```
def train_epoch(train_loader, model, device, loss_function, optimizer):
    """
    Trains the model for one epoch using the provided data loader and update

    Parameters:
    - train_loader (torch.utils.data.DataLoader): DataLoader object for the
    - 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 param
```

```
    Returns:
    - train_loss (float): Average training loss for the epoch.
    - epoch_hamming_distance (float): Hamming distance for the epoch.
    """
    # Set the model to training mode
    model.train()

    # Initialize variables to track running training loss and correct predic
    running_train_loss = 0.0
    running_train_correct = 0

    # Initialize Hamming Distance metric
    hamming = HammingDistance(task="multilabel", num_labels=10).to(device)

    # Iterate over all batches in the training data
    for inputs, targets in train_loader:
        # Move data to the appropriate device
        inputs, targets = inputs.squeeze(1).to(device), targets.to(device)

        # Perform a forward and backward pass, updating model parameters
        loss, _, _ = step(inputs, targets, model, device, loss_function, opt

        # Update running loss
        running_train_loss += loss.item()

        # Compute Hamming Distance for the epoch
        y_pred = (model(inputs) > 0.5).float()
        hamming.update(y_pred, targets)

    # Compute average loss for the entire training set
    train_loss = running_train_loss / len(train_loader)

    # Compute Hamming Distance for the epoch
    epoch_hamming_distance = hamming.compute()

    return train_loss, epoch_hamming_distance
```

In [ ]:

```
from torchmetrics import HammingDistance

def val_epoch(valid_loader, model, device, loss_function):
    """
    Validates the model for one epoch using the provided data loader.

    Parameters:
    - valid_loader (torch.utils.data.DataLoader): DataLoader object for the
    - model (torch.nn.Module): The neural network model to be validated.
    - device (torch.device): The computing device (CPU or GPU).
    - loss_function (torch.nn.Module): The loss function to evaluate the moc

    Returns:
    - val_loss (float): Average validation loss for the epoch.
    - val_hamming_distance (float): Hamming distance for the epoch.
    """
    # Set the model to evaluation mode
    model.eval()
```

```
    # Initialize variables to track running validation loss and Hamming Dist
    running_val_loss = 0.0
    val_hamming_distance = HammingDistance(task="multilabel", num_labels=10)

    # Disable gradient computation
    with torch.no_grad():
        # Iterate over all batches in the validation data
        for inputs, targets in valid_loader:
            # Move data to the appropriate device
            inputs, targets = inputs.squeeze(1).to(device), targets.to(devic

            # Perform a forward pass to get loss and number of correct predi
            outputs = model(inputs)
            loss = loss_function(outputs, targets)

            # Update running loss
            running_val_loss += loss.item()

            # Update Hamming Distance metric
            val_hamming_distance.update(torch.round(torch.sigmoid(outputs)),

    # Compute average loss and Hamming Distance for the entire validation se
    val_loss = running_val_loss / len(valid_loader)
    val_hamming_distance = val_hamming_distance.compute()

    return val_loss, val_hamming_distance
```

7.4. train() function

In [ ]:

```
def train(train_loader, valid_loader, model, optimizer, loss_function, epoch
    """
    Trains and validates the model, and returns history of train and validat

    Parameters:
    - train_loader (torch.utils.data.DataLoader): DataLoader for the trainin
    - valid_loader (torch.utils.data.DataLoader): DataLoader for the validat
    - model (torch.nn.Module): Neural network model to train.
    - optimizer (torch.optim.Optimizer): Optimizer algorithm.
    - loss_function (torch.nn.Module): Loss function to evaluate the model.
    - epochs (int): Number of epochs to train the model.
    - device (torch.device): The computing device (CPU or GPU).
    - patience (int): Number of epochs to wait for improvement before early

    Returns:
    - train_loss_history (list): History of training loss for each epoch.
    - train_hamm_history (list): History of training Hamming distance for ea
    - valid_loss_history (list): History of validation loss for each epoch.
    - valid_hamm_history (list): History of validation Hamming distance for
    """

    # Initialize lists to store metrics for each epoch
    train_loss_history = []
    valid_loss_history = []
    train_hamm_history = []
```

```python
    valid_hamm_history = []

    # Initialize variables for early stopping
    best_valid_loss = float('inf')
    no_improvement = 0

    # Loop over the number of specified epochs
    for epoch in range(epochs):
        # Train model on training data and capture metrics
        train_loss, train_hamm = train_epoch(
            train_loader, model, device, loss_function, optimizer)

        # Validate model on validation data and capture metrics
        valid_loss, valid_hamm = val_epoch(
            valid_loader, model, device, loss_function)

        # Store metrics for this epoch
        train_loss_history.append(train_loss)
        valid_loss_history.append(valid_loss)
        train_hamm_history.append(train_hamm)
        valid_hamm_history.append(valid_hamm)

        # Output epoch-level summary
        print(f"Epoch {epoch+1}/{epochs}")
        print(f"Train Loss: {train_loss:.4f} | Train Hamming Distance: {trai
        print(f"Valid Loss: {valid_loss:.4f} | Valid Hamming Distance: {vali
        print()

        # Check for early stopping
        if valid_loss < best_valid_loss:
            best_valid_loss = valid_loss
            no_improvement = 0
        else:
            no_improvement += 1
            if no_improvement == patience:
                print(f"No improvement for {patience} epochs. Early stopping
                break

    return train_loss_history, train_hamm_history, valid_loss_history, valic
```

In [ ]:
```python
# training
EPOCHS=5
BATCH_SIZE=128
LEARNING_RATE=0.001
WEIGHT_DECAY=0.0
PATIENCE=10
```

In [ ]:
```python
# Fixing the seed value for reproducibility across runs
SEED = 2345
random.seed(SEED)                      # Set seed for Python's 'random' modul
np.random.seed(SEED)                   # Set seed for NumPy's random number g
torch.manual_seed(SEED)                # Set seed for PyTorch's CPU operation
torch.cuda.manual_seed(SEED)           # Set seed for PyTorch's CUDA (GPU) op
torch.backends.cudnn.deterministic = True  # Ensure deterministic behavior i
```

```python
# Define the device for model training (use CUDA if available, else CPU)
device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')

# Data Loaders for training, validation, and test sets
# These loaders handle batching, shuffling, and data processing using the cu
train_loader = torch.utils.data.DataLoader(trainset, batch_size = BATCH_SIZE
valid_loader = torch.utils.data.DataLoader(validset, batch_size=BATCH_SIZE,
test_loader = torch.utils.data.DataLoader(testset, batch_size=BATCH_SIZE, sh

# Define the loss function for the model, using cross-entropy loss
loss_function = nn.BCEWithLogitsLoss()

# Define the model with specified hyperparameters
model = CustomModel(input_dim=INPUT_DIM,
                       hidden_dim1=HIDDEN_DIM1,
                       hidden_dim2=HIDDEN_DIM2,
                       drop_prob1=0.5,
                       drop_prob2=0.5,
                       output_dim=NUM_OUTPUTS)

model = model.to(device)

# Define the optimizer
optimizer = optim.AdamW(model.parameters(), lr=LEARNING_RATE, weight_decay=W
```

In [ ]:

```python
for inputs, targets in train_loader:
  print(type(inputs))
  print(inputs.shape)
  print(targets.shape)
  break
```

```
<class 'torch.Tensor'>
torch.Size([128, 1, 5000])
torch.Size([128, 10])
```

In [ ]:

```python
for inputs, targets in train_loader:
    # Move inputs and targets to the CPU.
    inputs = inputs.squeeze(1).to(device)
    targets = targets.to(device)
    model = model.to(device)
    model.eval()
    # Forward pass
    with torch.no_grad():  # Ensure no gradients are calculated since this i
        output = model(inputs)
        loss = loss_function(output, targets)
        print(f'Actual loss: {loss.item()}')
    break

print(f'Expected Theoretical loss: {np.log(2)}')
```

```
Actual loss: 0.6782360076904297
Expected Theoretical loss: 0.6931471805599453
```

In [ ]:

```python
CLIP_VALUE = 10
# Call the train function to train the model
train_losses, train_hamm, valid_losses, valid_hamm = train(
```

```
    train_loader, valid_loader, model, optimizer, loss_function, EPOCHS, dev
)
```

```
Epoch 1/5
Train Loss: 0.2995 | Train Hamming Distance: 0.0783
Valid Loss: 0.1288 | Valid Hamming Distance: 0.0469

Epoch 2/5
Train Loss: 0.1249 | Train Hamming Distance: 0.0451
Valid Loss: 0.1083 | Valid Hamming Distance: 0.0408

Epoch 3/5
Train Loss: 0.1016 | Train Hamming Distance: 0.0368
Valid Loss: 0.1010 | Valid Hamming Distance: 0.0365

Epoch 4/5
Train Loss: 0.0877 | Train Hamming Distance: 0.0320
Valid Loss: 0.0960 | Valid Hamming Distance: 0.0348

Epoch 5/5
Train Loss: 0.0798 | Train Hamming Distance: 0.0292
Valid Loss: 0.0949 | Valid Hamming Distance: 0.0339
```

Plot losses and metrics

In [ ]:

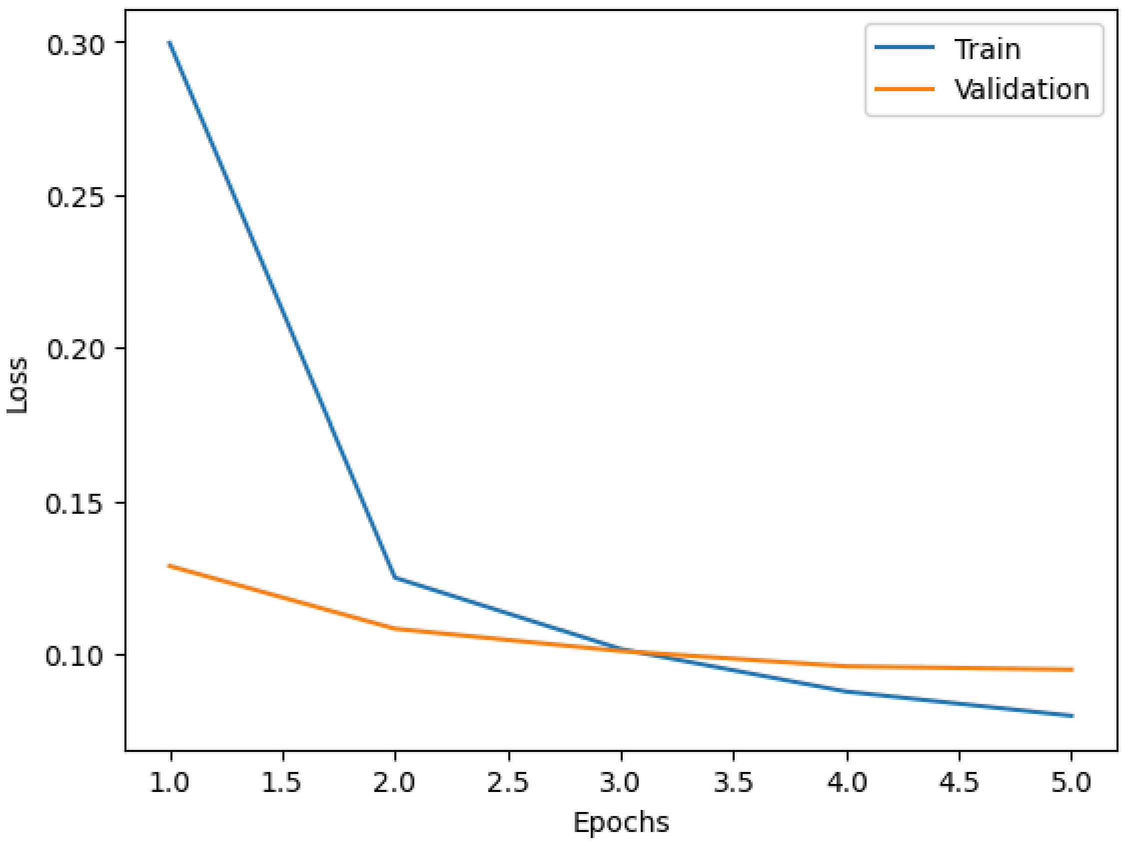
```
def plot_history(train_losses, train_metrics, val_losses=None, val_metrics=N
    """
    Plot training and validation loss and metrics over epochs.

    Args:
        train_losses (list): List of training losses for each epoch.
        train_metrics (list): List of training metrics (e.g., accuracy) for 
        val_losses (list, optional): List of validation losses for each epoc
        val_metrics (list, optional): List of validation metrics for each ep

    Returns:
        None
    """
    # Determine the number of epochs based on the length of train_losses
    epochs = range(1, len(train_losses) + 1)

    # Plotting training and validation losses
    plt.figure()
    plt.plot(epochs, train_losses, label="Train")  # Plot training losses
    if val_losses:  # Check if validation losses are provided
        plt.plot(epochs, val_losses, label="Validation")  # Plot validation 
    plt.xlabel("Epochs")
    plt.ylabel("Loss")
    plt.legend()
    plt.show()

    # Plotting training and validation metrics
    if train_metrics[0] is not None:  # Check if training metrics are availa
        plt.figure()
        plt.plot(epochs, train_metrics, label="Train")  # Plot training metr
```

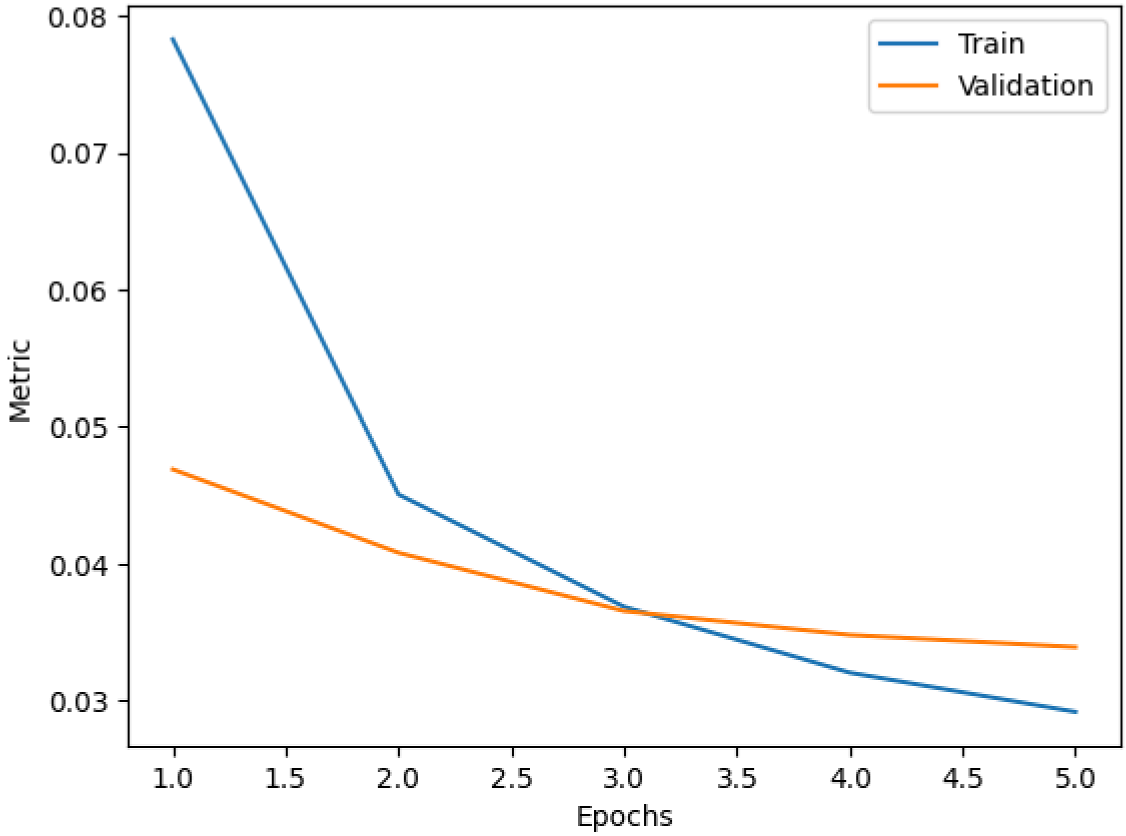
```
        if val_metrics:  # Check if validation metrics are provided
            plt.plot(epochs, val_metrics, label="Validation")  # Plot valida
        plt.xlabel("Epochs")
        plt.ylabel("Metric")
        plt.legend()
        plt.show()
```

In [ ]:
```
train_hamm
```

Out[ ]:
```
[tensor(0.0783, device='cuda:0'),
 tensor(0.0451, device='cuda:0'),
 tensor(0.0368, device='cuda:0'),
 tensor(0.0320, device='cuda:0'),
 tensor(0.0292, device='cuda:0')]
```

In [ ]:
```
import numpy as np
# Plot the training and validation losses and metrics
train_hamm_np = [ham.cpu().numpy() for ham in train_hamm]
valid_hamm_np = [ham.cpu().numpy() for ham in valid_hamm]
plot_history(train_losses, train_hamm_np, valid_losses, valid_hamm_np)
```





Model Checkpointing

In [ ]:
```
# Get the current timestamp in the format "YYYY-MM-DD_HH-MM-SS"
timestamp = datetime.now().strftime("%Y-%m-%d_%H-%M-%S")

# Define a suffix for the file name
suffix = 'twolayer'

# Combine the timestamp and suffix to create the file path
path = model_folder / f'{timestamp}_{suffix}.pt'
path
```

Out[ ]:
```
PosixPath('/content/drive/MyDrive/Colab_Notebooks/BUAN_6342_Applied_Natural
_Language_Processing/0_Data_Folder/2024-03-04_06-29-41_twolayer.pt')
```

In [ ]:
```
# Save the model's state dictionary to the specified file path
torch.save(model.state_dict(), path)
```

Evaluate model on validation dataset

We will now plot the confusion matrix to understand the performance of our model in more detail, particularly how well it classifies each class. For thet, we first need to get the predictions and labels.

In [ ]:
```
def get_acc_pred(data_loader, model, device):
    """
    Function to get predictions and accuracy for a given data using a traine
    Input: data iterator, model, device
```

```python
    Output: predictions and accuracy for the given dataset
    """
    model = model.to(device)
    # Set model to evaluation mode
    model.eval()

    # Create empty tensors to store predictions and actual labels
    predictions = torch.Tensor().to(device)
    y = torch.Tensor().to(device)

    # Iterate over batches from data iterator
    with torch.no_grad():
        for inputs, targets in data_loader:
            # Process the batch to get the loss, outputs, and correct predic
            outputs, _ = step(inputs, targets, model,
                              device, loss_function=None, optimizer=None)

            # Choose the label with maximum probability
            # Correct prediction using thresholding
            y_pred = (outputs.data>0.5).float()

            # Add the predicted labels and actual labels to their respective
            predictions = torch.cat((predictions, y_pred))
            y = torch.cat((y, targets.to(device)))

    # Calculate accuracy by comparing the predicted and actual labels
    accuracy = (predictions == y).float().mean()

    # Return tuple containing predictions and accuracy
    return predictions, accuracy, y
```

In [ ]:
```python
# Get the prediction and accuracy
predictions_test, acc_test, y_test = get_acc_pred(test_loader, model, device
predictions_train, acc_train, y_train = get_acc_pred(train_loader, model, de
predictions_valid, acc_valid, y_valid = get_acc_pred(valid_loader, model, de
```

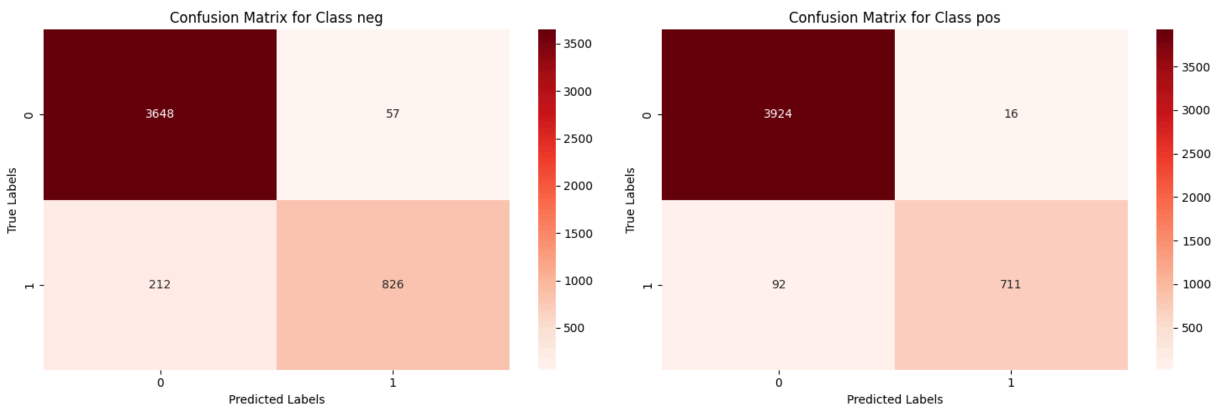
In [ ]:
```python
# Print Test Accuracy
print('Valid accuracy', acc_valid * 100)
```

```
Valid accuracy tensor(96.4790, device='cuda:0')
```

In [ ]:
```python
from sklearn.metrics import multilabel_confusion_matrix

def plot_confusion_matrix(valid_labels, valid_preds, class_labels):
    """
    Plots a confusion matrix.

    Args:
        valid_labels (array-like): True labels of the validation data.
        valid_preds (array-like): Predicted labels of the validation data.
        class_labels (list): List of class names for the labels.
    """
    # Compute the confusion matrix
    cm = multilabel_confusion_matrix(valid_labels, valid_preds)

    # Plot the confusion matrix using Seaborn
```

```
    fig, axs = plt.subplots(1, len(class_labels), figsize=(15, 5))
    for i, (label, matrix) in enumerate(zip(class_labels, cm)):
        sns.heatmap(matrix, annot=True, fmt="d", cmap="Reds", xticklabels=['
        axs[i].set_title(f"Confusion Matrix for Class {label}")
        axs[i].set_xlabel('Predicted Labels')
        axs[i].set_ylabel('True Labels')

    # Display the plot
    plt.tight_layout()
    plt.show()
```

In [ ]:
```
plot_confusion_matrix(y_test.cpu().numpy(), predictions_test.cpu().numpy(),
```



In [ ]:
```
test_hamming_distance = HammingDistance(task="multilabel", num_labels=10).tc
test_hamming_distance.update(y_test, predictions_test)
```

In [ ]:
```
test_hamming_distance.compute()
```

Out[ ]:
```
tensor(0.0337, device='cuda:0')
```

Inference:

While using Sparse embedding seem to given a slightly better result than Embedding bag, Embedding bag method ran faster. It couldn't capture fine-grained semantic information and struggle to generalise as it can't work when it sees words not in the training data.