|  |  |
| --- | --- |
|  | **AIR UNIVERSITY** |
| **DEPARTMENT OF COMPUTER SCIENCE** |
| **Lab Task 5** |

**Student Name: Hamza Umer Farooq Reg. No: 200789**

**Subject: Data Science Semester: VIII**

**Objective: LLM**

**ASSESSMENT:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attributes** | **Excellent**  **(5)** | **Good**  **(4)** | **Average**  **(3)** | **Satisfactory**  **(2)** | **Unsatisfactory (1)** |
| **Ability to Conduct**  Task |  |  |  |  |  |
| **Ability to assimilate the results** |  |  |  |  |  |
| **Effective use of theorems/postulates/formulas** |  |  |  |  |  |

Total Marks:

Obtained Marks:

**REPORT ASSESSMENT:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Attributes** | **Excellent**  **(5)** | **Good**  **(4)** | **Average**  **(3)** | **Satisfactory**  **(2)** | **Unsatisfactory**  **(1)** |
| **Data presentation** |  |  |  |  |  |
| **Experimental results** |  |  |  |  |  |
| **Conclusion** |  |  |  |  |  |

# -\*- coding: utf-8 -\*-

"""SentimentAnalysis\_BERT.ipynb

Automatically generated by Colaboratory.

Original file is located at

    https://colab.research.google.com/drive/1X6tGiNStBrQzQmsgaxn5qHDikXMF2F1r

"""

import torch

import pandas as pd

from tqdm.notebook import tqdm

import re

import string

import nltk

nltk.download("punkt")

nltk.download('wordnet')

nltk.download("stopwords")

from nltk.corpus import stopwords

from sklearn.feature\_extraction.text import TfidfTransformer

from sklearn.feature\_extraction.text import CountVectorizer

df = pd.read\_csv('./smileannotationsfinal.csv', names=['id', 'text', 'category'])

df.set\_index('id', inplace=True)

df.head()

df.category.value\_counts()

df['labels'] = df.apply(lambda x: "unhappy" if x['category'] != "happy" else "happy", axis = 1)

df.labels.value\_counts()

df = df[~df.labels.str.contains('\|')]

# df = df[df.labels != 'nocode']

df.labels.value\_counts()

df['text'] = df['text'].apply(lambda x: x.lower() if isinstance(x, str) else x)

df.head()

def preprocess(text):

  text = text.lower()

  text = re.sub(r'\d+','',text)

  text = re.sub(r'[^\w\s]','',text)

  tokens = nltk.word\_tokenize(text)

  return tokens

def remove\_stopwords(tokens):

  stop\_words  =set(stopwords.words('english'))

  filtered\_tokens = [word for word in tokens if word not in stop\_words]

  return filtered\_tokens

def perform\_lemmatization (tokens) :

  lemmatizer = nltk.WordNetLemmatizer()

  lemmatized\_tokens = [lemmatizer.lemmatize(token) for token in tokens]

  return  lemmatized\_tokens

def clean\_text(text):

  tokens = preprocess(text)

  filtered\_tokens = remove\_stopwords(tokens)

  lemmatized\_tokens = perform\_lemmatization(filtered\_tokens)

  clean\_text = ' '.join(lemmatized\_tokens)

  return clean\_text

df['clean\_text'] = df['text'].apply(lambda x : clean\_text(x))

df.head()

possible\_labels = df.labels.unique()

label\_dict = {}

for index, possible\_label in enumerate(possible\_labels):

    label\_dict[possible\_label] = index

df['label'] = df.labels.replace(label\_dict)

df.head()

from sklearn.model\_selection import train\_test\_split

df.index.values

df.labels.values

X\_train, X\_val, y\_train, y\_val = train\_test\_split(df.clean\_text.values,

                                                  df.labels.values,

                                                  test\_size=0.15,

                                                  random\_state=17,

                                                  stratify=df.label.values)

X\_train

df['data\_type'] = ['not\_set']\*df.shape[0]

df['data\_type']

df.loc[X\_train, 'data\_type'] = 'train'

df.loc[X\_val, 'data\_type'] = 'val'

df.groupby(['labels','label', 'data\_type']).count()

from transformers import BertTokenizer

from torch.utils.data import TensorDataset

tokenizer = BertTokenizer.from\_pretrained('bert-base-uncased',

                                          do\_lower\_case=True)

encoded\_data\_train = tokenizer.batch\_encode\_plus(

    df[df.data\_type=='train'].text.values,

    add\_special\_tokens=True,

    return\_attention\_mask=True,

    pad\_to\_max\_length=True,

    max\_length=256,

    return\_tensors='pt'

)

encoded\_data\_val = tokenizer.batch\_encode\_plus(

    df[df.data\_type=='val'].text.values,

    add\_special\_tokens=True,

    return\_attention\_mask=True,

    pad\_to\_max\_length=True,

    max\_length=256,

    return\_tensors='pt'

)

input\_ids\_train = encoded\_data\_train['input\_ids']

attention\_masks\_train = encoded\_data\_train['attention\_mask']

labels\_train = torch.tensor(df[df.data\_type=='train'].label.values)

input\_ids\_val = encoded\_data\_val['input\_ids']

attention\_masks\_val = encoded\_data\_val['attention\_mask']

labels\_val = torch.tensor(df[df.data\_type=='val'].label.values)

dataset\_train = TensorDataset(input\_ids\_train, attention\_masks\_train, labels\_train)

dataset\_val = TensorDataset(input\_ids\_val, attention\_masks\_val, labels\_val)

len(dataset\_train)

len(dataset\_val)

from transformers import BertForSequenceClassification

model = BertForSequenceClassification.from\_pretrained("bert-base-uncased",

                                                      num\_labels=len(label\_dict),

                                                      output\_attentions=False,

                                                      output\_hidden\_states=False)

from torch.utils.data import DataLoader, RandomSampler, SequentialSampler

batch\_size = 32

dataloader\_train = DataLoader(dataset\_train,

                              sampler=RandomSampler(dataset\_train),

                              batch\_size=batch\_size)

dataloader\_validation = DataLoader(dataset\_val,

                                   sampler=SequentialSampler(dataset\_val),

                                   batch\_size=batch\_size)

from transformers import AdamW, get\_linear\_schedule\_with\_warmup

optimizer = AdamW(model.parameters(),

                  lr=1e-5,

                  eps=1e-8)

epochs = 3

scheduler = get\_linear\_schedule\_with\_warmup(optimizer,

                                            num\_warmup\_steps=0,

                                            num\_training\_steps=len(dataloader\_train)\*epochs)

import numpy as np

from sklearn.metrics import f1\_score

def f1\_score\_func(preds, labels):

    preds\_flat = np.argmax(preds, axis=1).flatten()

    labels\_flat = labels.flatten()

    return f1\_score(labels\_flat, preds\_flat, average='weighted')

def accuracy\_per\_class(preds, labels):

    label\_dict\_inverse = {v: k for k, v in label\_dict.items()}

    preds\_flat = np.argmax(preds, axis=1).flatten()

    labels\_flat = labels.flatten()

    for label in np.unique(labels\_flat):

        y\_preds = preds\_flat[labels\_flat==label]

        y\_true = labels\_flat[labels\_flat==label]

        print(f'Class: {label\_dict\_inverse[label]}')

        print(f'Accuracy: {len(y\_preds[y\_preds==label])}/{len(y\_true)}\n')

import random

seed\_val = 17

random.seed(seed\_val)

np.random.seed(seed\_val)

torch.manual\_seed(seed\_val)

torch.cuda.manual\_seed\_all(seed\_val)

device = torch.device('cuda' if torch.cuda.is\_available() else 'cpu')

model.to(device)

print(device)

def evaluate(dataloader\_val):

    model.eval()

    loss\_val\_total = 0

    predictions, true\_vals = [], []

    for batch in dataloader\_val:

        batch = tuple(b.to(device) for b in batch)

        inputs = {'input\_ids':      batch[0],

                  'attention\_mask': batch[1],

                  'labels':         batch[2],

                 }

        with torch.no\_grad():

            outputs = model(\*\*inputs)

        loss = outputs[0]

        logits = outputs[1]

        loss\_val\_total += loss.item()

        logits = logits.detach().cpu().numpy()

        label\_ids = inputs['labels'].cpu().numpy()

        predictions.append(logits)

        true\_vals.append(label\_ids)

    loss\_val\_avg = loss\_val\_total/len(dataloader\_val)

    predictions = np.concatenate(predictions, axis=0)

    true\_vals = np.concatenate(true\_vals, axis=0)

    return loss\_val\_avg, predictions, true\_vals

for epoch in tqdm(range(1, epochs+1)):

    model.train()

    loss\_train\_total = 0

    progress\_bar = tqdm(dataloader\_train, desc='Epoch {:1d}'.format(epoch), leave=False, disable=False)

    for batch in progress\_bar:

        model.zero\_grad()

        batch = tuple(b.to(device) for b in batch)

        inputs = {'input\_ids':      batch[0],

                  'attention\_mask': batch[1],

                  'labels':         batch[2],

                 }

        outputs = model(\*\*inputs)

        loss = outputs[0]

        loss\_train\_total += loss.item()

        loss.backward()

        torch.nn.utils.clip\_grad\_norm\_(model.parameters(), 1.0)

        optimizer.step()

        scheduler.step()

        progress\_bar.set\_postfix({'training\_loss': '{:.3f}'.format(loss.item()/len(batch))})

    torch.save(model.state\_dict(), f'finetuned\_BERT\_epoch\_{epoch}.model')

    tqdm.write(f'\nEpoch {epoch}')

    loss\_train\_avg = loss\_train\_total/len(dataloader\_train)

    tqdm.write(f'Training loss: {loss\_train\_avg}')

    val\_loss, predictions, true\_vals = evaluate(dataloader\_validation)

    val\_f1 = f1\_score\_func(predictions, true\_vals)

    tqdm.write(f'Validation loss: {val\_loss}')

    tqdm.write(f'F1 Score (Weighted): {val\_f1}')

model = BertForSequenceClassification.from\_pretrained("bert-base-uncased",

                                                      num\_labels=len(label\_dict),

                                                      output\_attentions=False,

                                                      output\_hidden\_states=False)

model.to(device)

\_, predictions, true\_vals = evaluate(dataloader\_validation)

accuracy\_per\_class(predictions, true\_vals)

**Tasks performed:**

**Data preprocessing:**

* Lowercased, removed digits, and punctuation.
* Tokenized, removed stopwords, and lemmatized using NLTK.

**Data cleaning:**

* Applied text cleaning steps.

**Data encoding:**

* Encoded text using BERT tokenizer.
* Added special tokens, attention masks, and padding.

**Model training:**

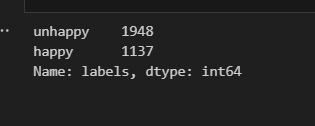
* Fine-tuned BERT for sequence classification.
* Used AdamW optimizer and a linear scheduler with warmup.

**Evaluation:**

* Evaluated model performance on validation data after each epoch.
* Calculated validation loss and F1 score.

**Post-training evaluation:**

* Loaded best-performing model.
* Calculated accuracy per class on validation set.

Snippets:  
A screenshot of a computer

Description automatically generatedA screenshot of a computer program

Description automatically generatedA computer screen with white text

Description automatically generatedA screen shot of a computer program

Description automatically generated