BERT

Author: Alikhan Semembayev

1. Perform necessary data preprocessing, e.g. removing punctuation and stop words, stemming, lemmatizing. You may use the outputs from previous weekly assignments. (10 points)

In [ ]:
```
from collections import defaultdict
import demoji
import svgling
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import PorterStemmer, WordNetLemmatizer
from nltk import pos_tag
from autocorrect import Speller
import re

# Initialize tools
spell = Speller()
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
stemmer = PorterStemmer()

email_re = r"\b[A-Za-z]+@\S+\b"
ssn_re = r"\b[0-9]{3}-[0-9]{2}-[0-9]{4}\b"
ip_re = r"\b\d{1,3}[.]\d{1,3}[.]\d{1,3}[.]\d{1,3}\b"

street_number_re = r"^\d{1,}"
street_name_re = r"[a-zA-Z0-9\s]+,?"
city_name_re = r" [a-zA-Z]+(\,)?"
state_abbrev_re = r" [A-Z]{2}"
postal_code_re = r" [0-9]{5}$"
address_pattern_re = r"" + street_number_re + street_name_re + city_name_re + st


def clean_text(text):
    # Replace emojis
    text = demoji.replace(text)

    # Remove smart quotes and dashes
    text = text.replace("“", "\"").replace("”", "\"").replace("-", " ").replace(

    # Lowercase text
    text = text.lower()

    # Tokenize text
    words = word_tokenize(text)
    # print(words)

    # Spelling correction + replace all t with not
    words = ['not' if word == 't' else (
```

```
        'ADDRESS' if re.match(address_pattern_re, word)
        else (
            'EMAIL' if re.match(email_re, word)
            else (
                'SSN' if re.match(ssn_re, word)
                else (
                    'IP' if re.match(ip_re, word)
                    else spell(word)
                )
            )
        )
    ) for word in words]

    # Remove stop words and non-alphabetic tokens and punctuation
    words = [word for word in words if word.isalnum() and word not in stop_words

    # POS tagging and Lemmatization
    tagged_words = pos_tag(words)

    tag_map = defaultdict(lambda: "n")
    tag_map["N"] = "n"
    tag_map["V"] = "v"
    tag_map["J"] = "a"
    tag_map["R"] = "r"

    words = [lemmatizer.lemmatize(word, pos=tag_map[tag[0]]) for word, tag in ta

    # Return cleaned words as a single string
    return ' '.join(words)
```

In [ ]:
```
import pandas as pd

data = (pd.read_csv('../../../../data/text/combined_raw.csv'))
data = data.dropna(how='any')

for row in data.values:
    row[0] = clean_text(row[0])

data.to_csv('../../../../data/text/combined_cleaned.csv', index=False)
```

In [43]:
```
import pandas as pd

data = (pd.read_csv('../../../../data/text/combined_cleaned.csv'))
data = data.dropna(how='any')

print(data.head(10))
```

```
                                                text    emotion
0  freshwater fish drink water skin via osmosis s...      happy
1  think everyone must use daily become grained e...    neutral
2  agree google headquarters mountain view califo...    neutral
3  thats funny current ceo sunday ficha didnt kno...    neutral
4  oh yeah not know either also want go google al...  surprised
5                                              say    surprised
6        yeah apparently lol instead hire people row      happy
7  thats funny guess imaginative leave huge tech ...  surprised
8  yeah exactly sure cheap one thing bet not expl...  surprised
9  remember hearing immortality waste jellyfish h...    neutral
```

2. For the binary classification problem you came up previously, build your own model by combining BERT with a classifier. (30 points)

In [45]:
```
import pandas as pd

# Set the maximum rows per label
max_rows_per_label = 10000

# Sample rows for each label
balanced_data = data.groupby('emotion', group_keys=False).apply(lambda x: x.samp

# Save or use the balanced data
balanced_data.to_csv("../../../../data/text/combined_cleaned_balanced_dataset.cs
```

In [62]:
```
import pandas as pd

data = (pd.read_csv('../../../../data/text/combined_cleaned_multilabel.csv'))
data = data.dropna(how='any')

print(data.head(10))
```

```
                                                text  is_happy  is_surprised  \
0  freshwater fish drink water skin via osmosis s...         1             0
1  think everyone must use daily become grained e...         0             0
2  agree google headquarters mountain view califo...         0             0
3  thats funny current ceo sunday ficha didnt kno...         0             0
4  oh yeah not know either also want go google al...         0             1
5                                                say         0             1
6        yeah apparently lol instead hire people row         1             0
7  thats funny guess imaginative leave huge tech ...         0             1
8  yeah exactly sure cheap one thing bet not expl...         0             1
9  remember hearing immortality waste jellyfish h...         0             0

   is_neutral  is_sad  is_fear  is_angry  is_disgust
0           0       0        0         0           0
1           1       0        0         0           0
2           1       0        0         0           0
3           1       0        0         0           0
4           0       0        0         0           0
5           0       0        0         0           0
6           0       0        0         0           0
7           0       0        0         0           0
8           0       0        0         0           0
9           1       0        0         0           0
```

In [74]:
```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import Dataset, DataLoader
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from transformers import BertTokenizer, BertForSequenceClassification
import pandas as pd
from transformers import BertModel
```

```
# Custom Dataset Class
class EmotionDataset(Dataset):
    def __init__(self, texts, labels):
        self.texts = texts
        self.labels = labels

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

    def __getitem__(self, idx):
        return {
            'input_ids': self.texts[idx],
            # 'attention_mask': self.attention_masks[idx],
            'labels': self.labels[idx]
        }

class EmotionClassifier(nn.Module):
    def __init__(self, num_classes):
        super(EmotionClassifier, self).__init__()

        self.bert = BertForSequenceClassification.from_pretrained('bert-base-unc

        # Freeze BERT parameters
        for param in self.bert.bert.parameters():
            param.requires_grad = False

        self.dropout = nn.Dropout(0.3)

    def forward(self, input_ids, attention_mask=None):
        outputs = self.bert(input_ids=input_ids, attention_mask=attention_mask)
        return outputs.logits
```

In [75]:

```
# Preprocess labels
label_encoder = LabelEncoder()
data['label'] = data['is_happy']
num_classes = 2

# Tokenization
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
max_length = 50

def tokenize_texts(texts):
    encodings = tokenizer(
        list(texts),
        padding='max_length',
        truncation=True,
        max_length=max_length,
        return_tensors='pt'
    )
    return encodings['input_ids'].tolist()

# Convert texts to token IDs
data['input_ids'] = tokenize_texts(data['text'])

# Split the data
X_train, X_test, y_train, y_test = train_test_split(
    data['input_ids'].tolist(), data['label'].tolist(), test_size=0.2, random_st
)

# Convert data to PyTorch tensors
```

```
X_train = torch.tensor(X_train, dtype=torch.long)
X_test = torch.tensor(X_test, dtype=torch.long)
y_train = torch.tensor(y_train, dtype=torch.long)
y_test = torch.tensor(y_test, dtype=torch.long)
```

In [76]:
```
# Create PyTorch datasets
train_dataset = EmotionDataset(X_train, y_train)
test_dataset = EmotionDataset(X_test, y_test)

# DataLoader
train_loader = DataLoader(train_dataset, batch_size=128, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=128)
```

3. Train your own model by fine-tuning BERT. And save your model and use it to classify sentences (50 points)

In [77]:
```
# Check for CUDA
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Initialize model, loss, and optimizer
model = EmotionClassifier(num_classes).to(device)
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)

# Training loop
num_epochs = 2
for epoch in range(num_epochs):
    model.train()
    for batch_idx, batch in enumerate(train_loader, start=1):
        optimizer.zero_grad()

        # Move data to device
        input_ids = batch['input_ids'].to(device)
        labels = batch['labels'].to(device)

        # Forward pass
        outputs = model(input_ids)
        loss = criterion(outputs, labels)

        # Backward pass and optimization
        loss.backward()
        optimizer.step()

        if batch_idx % 10 == 0 or batch_idx == len(train_loader):
            print(f"Batch {batch_idx}/{len(train_loader)}: Loss = {loss.item()}"

    print(f"Epoch {epoch + 1}/{num_epochs}, Loss: {loss.item()}")
```

```
Using device: cuda
```

```
Some weights of BertForSequenceClassification were not initialized from the model
checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'c
lassifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it f
or predictions and inference.
```

```
Batch 10/922: Loss = 0.5630530714988708
Batch 20/922: Loss = 0.6131772994995117
Batch 30/922: Loss = 0.6327952146530151
Batch 40/922: Loss = 0.5326545834541321
Batch 50/922: Loss = 0.6016024947166443
Batch 60/922: Loss = 0.6733378767967224
Batch 70/922: Loss = 0.5222554802894592
Batch 80/922: Loss = 0.6361863613128662
Batch 90/922: Loss = 0.6165667772293091
Batch 100/922: Loss = 0.6207923889160156
Batch 110/922: Loss = 0.6651461124420166
Batch 120/922: Loss = 0.5742948651313782
Batch 130/922: Loss = 0.607204258441925
Batch 140/922: Loss = 0.6520662903785706
Batch 150/922: Loss = 0.6315717697143555
Batch 160/922: Loss = 0.5946286916732788
Batch 170/922: Loss = 0.6228736639022827
Batch 180/922: Loss = 0.6310967206954956
Batch 190/922: Loss = 0.6357423067092896
Batch 200/922: Loss = 0.6597950458526611
Batch 210/922: Loss = 0.6262190341949463
Batch 220/922: Loss = 0.6458306312561035
Batch 230/922: Loss = 0.5219413042068481
Batch 240/922: Loss = 0.5974372625350952
Batch 250/922: Loss = 0.5697453022003174
Batch 260/922: Loss = 0.56712406873703
Batch 270/922: Loss = 0.5751973390579224
Batch 280/922: Loss = 0.5681144595146179
Batch 290/922: Loss = 0.6196956634521484
Batch 300/922: Loss = 0.5714631676673889
Batch 310/922: Loss = 0.6176434755325317
Batch 320/922: Loss = 0.5966416001319885
Batch 330/922: Loss = 0.6372080445289612
Batch 340/922: Loss = 0.6497365832328796
Batch 350/922: Loss = 0.6117145419120789
Batch 360/922: Loss = 0.6185929775238037
Batch 370/922: Loss = 0.579983115196228
Batch 380/922: Loss = 0.6225438714027405
Batch 390/922: Loss = 0.6509799957275391
Batch 400/922: Loss = 0.6598306894302368
Batch 410/922: Loss = 0.665298342704773
Batch 420/922: Loss = 0.6188740134239197
Batch 430/922: Loss = 0.5753017663955688
Batch 440/922: Loss = 0.5918484926223755
Batch 450/922: Loss = 0.5168279409408569
Batch 460/922: Loss = 0.6281508207321167
Batch 470/922: Loss = 0.550732433795929
Batch 480/922: Loss = 0.6071640849113464
Batch 490/922: Loss = 0.5696375370025635
Batch 500/922: Loss = 0.5995907783508301
Batch 510/922: Loss = 0.5999315977096558
Batch 520/922: Loss = 0.6529277563095093
Batch 530/922: Loss = 0.656372606754303
Batch 540/922: Loss = 0.6001768112182617
Batch 550/922: Loss = 0.5888123512268066
Batch 560/922: Loss = 0.5740253329277039
Batch 570/922: Loss = 0.5580939650535583
Batch 580/922: Loss = 0.6496306657791138
Batch 590/922: Loss = 0.5807874202728271
Batch 600/922: Loss = 0.5930541157722473
```

```
Batch 610/922: Loss = 0.6509097218513489
Batch 620/922: Loss = 0.7108039259910583
Batch 630/922: Loss = 0.6772646307945251
Batch 640/922: Loss = 0.6549835801124573
Batch 650/922: Loss = 0.7101112604141235
Batch 660/922: Loss = 0.6099209785461426
Batch 670/922: Loss = 0.6350319981575012
Batch 680/922: Loss = 0.5944700241088867
Batch 690/922: Loss = 0.5819827318191528
Batch 700/922: Loss = 0.5864325761795044
Batch 710/922: Loss = 0.6136187314987183
Batch 720/922: Loss = 0.620949923992157
Batch 730/922: Loss = 0.5451276302337646
Batch 740/922: Loss = 0.6207963228225708
Batch 750/922: Loss = 0.5374840497970581
Batch 760/922: Loss = 0.5686373710632324
Batch 770/922: Loss = 0.5706128478050232
Batch 780/922: Loss = 0.614002525806427
Batch 790/922: Loss = 0.5065705180168152
Batch 800/922: Loss = 0.561711847782135
Batch 810/922: Loss = 0.6020426750183105
Batch 820/922: Loss = 0.6365953683853149
Batch 830/922: Loss = 0.6307429075241089
Batch 840/922: Loss = 0.5569747686386108
Batch 850/922: Loss = 0.6081017851829529
Batch 860/922: Loss = 0.5835798978805542
Batch 870/922: Loss = 0.5943648815155029
Batch 880/922: Loss = 0.6243953108787537
Batch 890/922: Loss = 0.6308507919311523
Batch 900/922: Loss = 0.5880237817764282
Batch 910/922: Loss = 0.6128373146057129
Batch 920/922: Loss = 0.6365931034088135
Batch 922/922: Loss = 0.5860567092895508
Epoch 1/2, Loss: 0.5860567092895508
Batch 10/922: Loss = 0.6269892454147339
Batch 20/922: Loss = 0.6112962365150452
Batch 30/922: Loss = 0.5761814117431641
Batch 40/922: Loss = 0.605475664138794
Batch 50/922: Loss = 0.6240124106407166
Batch 60/922: Loss = 0.5338467359542847
Batch 70/922: Loss = 0.5576759576797485
Batch 80/922: Loss = 0.575230598449707
Batch 90/922: Loss = 0.594041645526886
Batch 100/922: Loss = 0.5847200155258179
Batch 110/922: Loss = 0.6131572723388672
Batch 120/922: Loss = 0.5728012919425964
Batch 130/922: Loss = 0.6537070870399475
Batch 140/922: Loss = 0.5722397565841675
Batch 150/922: Loss = 0.5651401281356812
Batch 160/922: Loss = 0.5597229599952698
Batch 170/922: Loss = 0.6052383184432983
Batch 180/922: Loss = 0.5780014395713806
Batch 190/922: Loss = 0.5872494578361511
Batch 200/922: Loss = 0.5713818669319153
Batch 210/922: Loss = 0.6001211404800415
Batch 220/922: Loss = 0.5609968900680542
Batch 230/922: Loss = 0.639348566532135
Batch 240/922: Loss = 0.565445601940155
Batch 250/922: Loss = 0.5864381790161133
Batch 260/922: Loss = 0.6099804043769836
```

```
Batch 270/922: Loss = 0.6042872667312622
Batch 280/922: Loss = 0.5995222926139832
Batch 290/922: Loss = 0.5948017239570618
Batch 300/922: Loss = 0.6525886058807373
Batch 310/922: Loss = 0.6476280093193054
Batch 320/922: Loss = 0.5826666355133057
Batch 330/922: Loss = 0.5883219242095947
Batch 340/922: Loss = 0.5911024212837219
Batch 350/922: Loss = 0.5833261609077454
Batch 360/922: Loss = 0.6494131088256836
Batch 370/922: Loss = 0.5831775069236755
Batch 380/922: Loss = 0.6331817507743835
Batch 390/922: Loss = 0.563279390335083
Batch 400/922: Loss = 0.5506247282028198
Batch 410/922: Loss = 0.6066303849220276
Batch 420/922: Loss = 0.6082770824432373
Batch 430/922: Loss = 0.6092134714126587
Batch 440/922: Loss = 0.613429844379425
Batch 450/922: Loss = 0.5280771255493164
Batch 460/922: Loss = 0.6420609951019287
Batch 470/922: Loss = 0.5771382451057434
Batch 480/922: Loss = 0.5836032629013062
Batch 490/922: Loss = 0.621660590171814
Batch 500/922: Loss = 0.5207904577255249
Batch 510/922: Loss = 0.5878470540046692
Batch 520/922: Loss = 0.5718967318534851
Batch 530/922: Loss = 0.6531387567520142
Batch 540/922: Loss = 0.6168565154075623
Batch 550/922: Loss = 0.592318058013916
Batch 560/922: Loss = 0.586536169052124
Batch 570/922: Loss = 0.592185378074646
Batch 580/922: Loss = 0.6248476505279541
Batch 590/922: Loss = 0.594023585319519
Batch 600/922: Loss = 0.6537906527519226
Batch 610/922: Loss = 0.5451487898826599
Batch 620/922: Loss = 0.5406287908554077
Batch 630/922: Loss = 0.5762225985527039
Batch 640/922: Loss = 0.5909875631332397
Batch 650/922: Loss = 0.6116880774497986
Batch 660/922: Loss = 0.5797199010848999
Batch 670/922: Loss = 0.6687448024749756
Batch 680/922: Loss = 0.559087872505188
Batch 690/922: Loss = 0.6240326166152954
Batch 700/922: Loss = 0.5507158041000366
Batch 710/922: Loss = 0.5524781346321106
Batch 720/922: Loss = 0.5526093244552612
Batch 730/922: Loss = 0.6186273097991943
Batch 740/922: Loss = 0.5482545495033264
Batch 750/922: Loss = 0.5862537622451782
Batch 760/922: Loss = 0.5555471181869507
Batch 770/922: Loss = 0.602100670337677
Batch 780/922: Loss = 0.5909079313278198
Batch 790/922: Loss = 0.6019454002380371
Batch 800/922: Loss = 0.5035797953605652
Batch 810/922: Loss = 0.5216773152351379
Batch 820/922: Loss = 0.5819045901298523
Batch 830/922: Loss = 0.5503303408622742
Batch 840/922: Loss = 0.6687129139900208
Batch 850/922: Loss = 0.5829569101333618
Batch 860/922: Loss = 0.5864537358283997
```

```
Batch 870/922: Loss = 0.5838385820388794
Batch 880/922: Loss = 0.45380404591560364
Batch 890/922: Loss = 0.6343964338302612
Batch 900/922: Loss = 0.6097432971000671
Batch 910/922: Loss = 0.5481880307197571
Batch 920/922: Loss = 0.5437068939208984
Batch 922/922: Loss = 0.6516422033309937
Epoch 2/2, Loss: 0.6516422033309937
```

In [79]:
```python
# Evaluation
model.eval()
all_preds = []
all_labels = []
with torch.no_grad():
    for batch in test_loader:
        # Move data to device
        input_ids = batch['input_ids'].to(device)
        labels = batch['labels'].to(device)

        outputs = model(input_ids)
        preds = torch.argmax(outputs, dim=1)

        # Move predictions and labels back to CPU for evaluation
        all_preds.extend(preds.cpu().tolist())
        all_labels.extend(labels.cpu().tolist())

# # Classification Report
# print("Classification Report:")
# # print(classification_report(all_labels, all_preds, target_names=label_encode
# print("Accuracy:", accuracy_score(all_labels, all_preds))

from sklearn.metrics import classification_report

print("Classification Report:")
print("Accuracy:", accuracy_score(all_labels, all_preds))
print(classification_report(all_labels, all_preds, target_names=['Not happy', 'H
```

```
Classification Report:
Accuracy: 0.7088139503324739
              precision    recall  f1-score   support

   Not happy       0.71      1.00      0.83     20896
       Happy       0.46      0.00      0.00      8580

    accuracy                           0.71     29476
   macro avg       0.58      0.50      0.42     29476
weighted avg       0.64      0.71      0.59     29476
```

In [80]:
```python
# Define a path to save the model
model_save_path = "emotion_classifier_model.pth"

# Save the model state dictionary
torch.save(model.state_dict(), model_save_path)
```

In [81]:
```python
# Instantiate a new model instance
loaded_model = EmotionClassifier(num_classes).to(device)

# Load the saved state dictionary
loaded_model.load_state_dict(torch.load(model_save_path, map_location=device))
```

```
# Set the model to evaluation mode if you're planning to evaluate or make predic
loaded_model.eval()
```

```
Some weights of BertForSequenceClassification were not initialized from the model
checkpoint at bert-base-uncased and are newly initialized: ['classifier.bias', 'c
lassifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it f
or predictions and inference.
```

Out[81]:

```
EmotionClassifier(
  (bert): BertForSequenceClassification(
    (bert): BertModel(
      (embeddings): BertEmbeddings(
        (word_embeddings): Embedding(30522, 768, padding_idx=0)
        (position_embeddings): Embedding(512, 768)
        (token_type_embeddings): Embedding(2, 768)
        (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (dropout): Dropout(p=0.1, inplace=False)
      )
      (encoder): BertEncoder(
        (layer): ModuleList(
          (0-11): 12 x BertLayer(
            (attention): BertAttention(
              (self): BertSdpaSelfAttention(
                (query): Linear(in_features=768, out_features=768, bias=True)
                (key): Linear(in_features=768, out_features=768, bias=True)
                (value): Linear(in_features=768, out_features=768, bias=True)
                (dropout): Dropout(p=0.1, inplace=False)
              )
              (output): BertSelfOutput(
                (dense): Linear(in_features=768, out_features=768, bias=True)
                (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=Tr
ue)
                (dropout): Dropout(p=0.1, inplace=False)
              )
            )
            (intermediate): BertIntermediate(
              (dense): Linear(in_features=768, out_features=3072, bias=True)
              (intermediate_act_fn): GELUActivation()
            )
            (output): BertOutput(
              (dense): Linear(in_features=3072, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=Tru
e)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
        )
      )
      (pooler): BertPooler(
        (dense): Linear(in_features=768, out_features=768, bias=True)
        (activation): Tanh()
      )
    )
    (dropout): Dropout(p=0.1, inplace=False)
    (classifier): Linear(in_features=768, out_features=2, bias=True)
  )
  (dropout): Dropout(p=0.3, inplace=False)
)
```

In [82]:
```
# Evaluation
model.eval()
all_preds = []
all_labels = []
with torch.no_grad():
    for batch in test_loader:
        # Move data to device
        input_ids = batch['input_ids'].to(device)
        labels = batch['labels'].to(device)

        outputs = model(input_ids)
        preds = torch.argmax(outputs, dim=1)

        # Move predictions and labels back to CPU for evaluation
        all_preds.extend(preds.cpu().tolist())
        all_labels.extend(labels.cpu().tolist())

from sklearn.metrics import classification_report

print("Classification Report:")
print("Accuracy:", accuracy_score(all_labels, all_preds))
print(classification_report(all_labels, all_preds, target_names=['Not happy', 'H
```

```
Classification Report:
Accuracy: 0.7088139503324739
              precision    recall  f1-score   support

   Not happy       0.71      1.00      0.83     20896
       Happy       0.46      0.00      0.00      8580

    accuracy                           0.71     29476
   macro avg       0.58      0.50      0.42     29476
weighted avg       0.64      0.71      0.59     29476
```

4. Summarize what you have learned and discovered from Task 1-3. (10 points)

1. Preprocessing is very important stage of development. Using stemming and lemmatization, along with removing stop words, helps improve text data representation. It decreases the size of dataset by removing unnecessary information, and optimizes it for training.
2. Bart can be used as a layer of MLP. It has its own dropout. But we added another dropout layer. Also, we removed all other fully connected layers.
3. We have very big amount of data. To save time we used high learning rate and small number of epochs for training. THe accuracy of the model predictions is 71%.
4. Binary classification has better accuracy, because of fewer labels(possible outputs). Multicalss problem is more difficult and requires more time.
5. It is possible to save the model with Bert as one of layers and load it for later use.

In [ ]:
```
```