Homework 2, CS678 Spring 2023

This is due on March 16th, 2023. This notebook is to be submitted via Gradescope along with your report and datasets with the naming convention of **Firstname_Lastname_HW2**

IMPORTANT: After copying this notebook to your Google Drive, please paste a link to it below. To get a publicly-accessible link, hit the *Share* button at the top right, then click "Get shareable link" and copy over the result. Alternatively, you can upload the completed .ipynb file along with your completed .pdf report to Gradescope.If you fail to do this, you will receive no credit for this homework!

LINK:

How to submit this problem set:

- Write all the answers in this Colab notebook. Once you are finished, generate a PDF via (File -> Print -> Save as PDF) and upload it to Gradescope.
- **Important:** check your PDF before you submit to Gradescope to make sure it exported correctly. If Colab gets confused about your syntax, it will sometimes terminate the PDF creation routine early.
- **Important:** on Gradescope, please make sure that you tag each page with the corresponding question(s). This makes it significantly easier for our graders to grade submissions, especially with the long outputs of many of these cells. We will take off points for submissions that are not tagged.
- When creating your final version of the PDF to hand in, please do a fresh restart and execute every cell in order. One handy way to do this is by clicking `Runtime -> Run All` in the notebook menu.

Academic honesty

- We will audit the Colab notebooks from a set number of students, chosen at random. The audits will check that the code you wrote actually generates the answers in your PDF. If you turn in correct answers on your PDF without code that actually generates those answers, we will consider this a serious case of cheating. See the course page for honesty policies.

- We will also run automatic checks of Colab notebooks for plagiarism. Copying code from others is also considered a serious case of cheating.

[ ] ↳ 1 cell hidden

Part 1: Data Collection and Annotation

In this homework, you will first collect a labeled dataset of **150** sentences for a text classification task of your choice. This process will include:

1. *Data collection*: Collect 150 sentences from any source you find interesting (e.g., literature, Tweets, news articles, reviews, etc.)
2. *Task design*: Come up with a multilabel sentence-level classification task that you would like to perform on your sentences.
3. On your dataset, collect annotations from **two** classmates for your task on a **second, separate set** of a minimum of **150** sentences. Everyone in this class will need to both create their own dataset and also serve as an annotator for two other classmates. In order to get everything done on time, you need to complete the following steps:

> - Find two classmates willing to label 150 sentences each (use the Piazza "search for teammates" thread if you're having issues finding labelers).
> - Collect the labeled data from each of the two annotators.
> - Sanity check the data for basic cleanliness (are all examples annotated? are all labels allowable ones?)

4. Collect feedback from annotators about the task including annotation time and obstacles encountered (e.g., maybe some sentences were particularly hard to annotate!)
5. Calculate and report inter-annotator agreement.
6. Aggregate output from both annotators to create final dataset (include your first 150 sentences too).
7. Perform NLP experiments on your new dataset!

Question 1.3 (10 points):

Now, compute the inter-annotator agreement between your two annotators. Upload both .tsv files to your Colab session (click the folder icon in the sidebar to the left of the screen). In the code cell below, read the data from the two files and compute both the raw agreement (% of examples for which both annotators agreed on the label) and the Cohen's Kappa. Feel free to use implementations in existing libraries (e.g., sklearn). After you're done, report the raw agreement and Cohen's scores in your report.

If you're curious, Cohen suggested the Kappa result be interpreted as follows: values ≤ 0 as indicating no agreement and 0.01–0.20 as none to slight, 0.21–0.40 as fair, 0.41– 0.60 as moderate, 0.61–0.80 as substantial, and 0.81–1.00 as almost perfect agreement.

```
from google.colab import drive

# Mount Google Drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
import os
os.chdir('/content/drive/MyDrive/Colab Notebooks/Data_NLP')
import pandas as pd
```

```
### WRITE CODE TO LOAD ANNOTATIONS AND
### COMPUTE AGREEMENT + COHEN'S KAPPA HERE!

import pandas as pd
import itertools

# Read the 5 tsv files into separate pandas dataframes
df1 = pd.read_csv('Annotator1_final.tsv', delimiter='\t',encoding='latin1')
df2 = pd.read_csv('Annotator2_final.tsv', delimiter='\t',encoding='latin1')
df3 = pd.read_csv('Annotator3_final.tsv', delimiter='\t',encoding='latin1')
df4 = pd.read_csv('Annotator4_final.tsv', delimiter='\t',encoding='latin1')
df5 = pd.read_csv('Annotator5_final.tsv', delimiter='\t',encoding='latin1')

# Merge the dataframes on the 'Text' column
merged_df = pd.merge(df1, df2, on=['sentence','Language'], suffixes=('_1', '_2'))
```

```
merged_df = pd.merge(merged_df, df3, on=['sentence','Language'], suffixes=('', '_
merged_df = pd.merge(merged_df, df4, on=['sentence','Language'], suffixes=('', '_
merged_df = pd.merge(merged_df, df5, on=['sentence','Language'], suffixes=('', '_

#print(merged_df.info())

merged_df = merged_df.rename(columns={"label_name": "label_name_3"})

# Calculate raw agreement counts for each pair of labelers
raw_counts = pd.DataFrame(columns=['df1', 'df2', 'count'])
for i in range(1, 5):
    for j in range(i+1, 6):
        count = sum((merged_df[f'label_name_{i}'] == merged_df[f'label_name_{j}']
        raw_counts = raw_counts.append({'df1': f'df{i}', 'df2': f'df{j}', 'count'

from sklearn.metrics import cohen_kappa_score
# Calculate Cohen kappa agreement scores for each pair of labelers
kappa_scores = pd.DataFrame(columns=['df1', 'df2', 'kappa'])
for i in range(1, 5):
    for j in range(i+1, 6):
        kappa = cohen_kappa_score(merged_df[f'label_name_{i}'], merged_df[f'label_
        kappa_scores = kappa_scores.append({'df1': f'df{i}', 'df2': f'df{j}', 'ka


# Print the raw agreement counts and Cohen kappa agreement scores for each pair o
print(f'Raw agreement: {raw_counts}')

print(f"Cohen's Kappa: {kappa_scores:}")
```

```
  raw_counts = raw_counts.append({'df1': f'df{i}', 'df2': f'df{j}', 'count': c
<ipython-input-3-781d30e2d086>:29: FutureWarning: The frame.append method is c
  raw_counts = raw_counts.append({'df1': f'df{i}', 'df2': f'df{j}', 'count': c
<ipython-input-3-781d30e2d086>:29: FutureWarning: The frame.append method is c
  raw_counts = raw_counts.append({'df1': f'df{i}', 'df2': f'df{j}', 'count': c
<ipython-input-3-781d30e2d086>:29: FutureWarning: The frame.append method is c
  raw_counts = raw_counts.append({'df1': f'df{i}', 'df2': f'df{j}', 'count': c
<ipython-input-3-781d30e2d086>:29: FutureWarning: The frame.append method is c
  raw_counts = raw_counts.append({'df1': f'df{i}', 'df2': f'df{j}', 'count': c
<ipython-input-3-781d30e2d086>:29: FutureWarning: The frame.append method is c
  raw_counts = raw_counts.append({'df1': f'df{i}', 'df2': f'df{j}', 'count': c
<ipython-input-3-781d30e2d086>:29: FutureWarning: The frame.append method is c
  raw_counts = raw_counts.append({'df1': f'df{i}', 'df2': f'df{j}', 'count': c
<ipython-input-3-781d30e2d086>:29: FutureWarning: The frame.append method is c
  raw_counts = raw_counts.append({'df1': f'df{i}', 'df2': f'df{j}', 'count': c
```

```
Raw agreement:    df1  df2      count
0  df1  df2   4.597701
1  df1  df3  36.206897
2  df1  df4  23.563218
3  df1  df5  23.563218
4  df2  df3   4.022989
5  df2  df4   5.172414
6  df2  df5   5.172414
7  df3  df4  26.436782
8  df3  df5  26.436782
9  df4  df5  81.034483
Cohen's Kappa:    df1  df2     kappa
0  df1  df2  0.010483
1  df1  df3  0.295315
2  df1  df4  0.159726
3  df1  df5  0.159726
4  df2  df3  0.007039
5  df2  df4  0.008256
6  df2  df5  0.008256
7  df3  df4  0.209989
8  df3  df5  0.209989
9  df4  df5  0.793150
<ipython-input-3-781d30e2d086>:37: FutureWarning: The frame.append method is (
  kappa_scores = kappa_scores.append({'df1': f'df{i}', 'df2': f'df{j}', 'kappa
<ipython-input-3-781d30e2d086>:37: FutureWarning: The frame.append method is (
  kappa_scores = kappa_scores.append({'df1': f'df{i}', 'df2': f'df{j}', 'kappa
<ipython-input-3-781d30e2d086>:37: FutureWarning: The frame.append method is (
  kappa_scores = kappa_scores.append({'df1': f'df{i}', 'df2': f'df{j}', 'kappa
<ipython-input-3-781d30e2d086>:37: FutureWarning: The frame.append method is (
  kappa_scores = kappa_scores.append({'df1': f'df{i}', 'df2': f'df{j}', 'kappa
<ipython-input-3-781d30e2d086>:37: FutureWarning: The frame.append method is (
  kappa_scores = kappa_scores.append({'df1': f'df{i}', 'df2': f'df{j}', 'kappa
<ipython-input-3-781d30e2d086>:37: FutureWarning: The frame.append method is (
  kappa_scores = kappa_scores.append({'df1': f'df{i}', 'df2': f'df{j}', 'kappa
<ipython-input-3-781d30e2d086>:37: FutureWarning: The frame.append method is (
  kappa_scores = kappa_scores.append({'df1': f'df{i}', 'df2': f'df{j}', 'kappa
<ipython-input-3-781d30e2d086>:37: FutureWarning: The frame.append method is (
  kappa_scores = kappa_scores.append({'df1': f'df{i}', 'df2': f'df{j}', 'kappa
<ipython-input-3-781d30e2d086>:37: FutureWarning: The frame.append method is (
  kappa_scores = kappa_scores.append({'df1': f'df{i}', 'df2': f'df{j}', 'kappa
<ipython-input-3-781d30e2d086>:37: FutureWarning: The frame.append method is (
  kappa_scores = kappa_scores.append({'df1': f'df{i}', 'df2': f'df{j}', 'kappa
```

RAW AGREEMENT:

COHEN'S KAPPA:

Part 2: Model Training and Testing

Now we'll move onto fine-tuning pretrained language models specifically on your dataset. This part of the homework is meant to be an introduction to the HuggingFace library, and it contains code that will potentially be useful for your final projects. Since we're dealing with large models, the first step is to change to a GPU runtime.

Adding a hardware accelerator

Please go to the menu and add a GPU as follows:

`Edit > Notebook Settings > Hardware accelerator > (GPU)`

Run the following cell to confirm that the GPU is detected.

```
import torch
torch.cuda.empty_cache()

# Confirm that the GPU is detected

assert torch.cuda.is_available()

# Get the GPU device name.
device_name = torch.cuda.get_device_name()
n_gpu = torch.cuda.device_count()
print(f"Found device: {device_name}, n_gpu: {n_gpu}")
device = torch.device("cuda")
```

```
Found device: Tesla T4, n_gpu: 1
```

Installing Hugging Face's Transformers library

We will use Hugging Face's Transformers (https://github.com/huggingface/transformers), an open-source library that provides general-purpose architectures for natural language understanding and generation with a collection of various pretrained models made by the NLP community. This library will allow us to easily use pretrained models like `BERT` and perform experiments on top of them. We can use these models to solve downstream target tasks, such as text classification, question answering, and sequence labeling.

Run the following cell to install Hugging Face's Transformers library and download a sample data file called seed.tsv that contains 250 sentences in English, annotated with their frame.

```
!pip install transformers
!pip install -U -q PyDrive
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
Collecting transformers
  Downloading transformers-4.27.1-py3-none-any.whl (6.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.7/6.7 MB 55.6 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.9/dist-pac
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.9/dis
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.9/d
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-packa
Requirement already satisfied: filelock in /usr/local/lib/python3.9/dist-packa
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.9/dist-pa
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.9/dist-pa
Collecting huggingface-hub<1.0,>=0.11.0
  Downloading huggingface_hub-0.13.2-py3-none-any.whl (199 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 199.2/199.2 KB 26.5 MB/s eta 0:00:
Collecting tokenizers!=0.11.3,<0.14,>=0.11.1
  Downloading tokenizers-0.13.2-cp39-cp39-manylinux_2_17_x86_64.manylinux2014_
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.6/7.6 MB 48.5 MB/s eta 0:00:00
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/py
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.9/
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/pyt
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-p
Installing collected packages: tokenizers, huggingface-hub, transformers
Successfully installed huggingface-hub-0.13.2 tokenizers-0.13.2 transformers-4
```

The cell below imports some helper functions we wrote to demonstrate the task on the sample seed dataset.

```
from helpers import tokenize_and_format, flat_accuracy
```

Part 1: Data Prep and Model Specifications

Upload your data using the file explorer to the left. We have provided a function below to tokenize and format your data as BERT requires. Make sure that your tsv file, titled final_data.tsv, has one column "sentence" and another column "labels_ID" containing integers/float.

If you run the cell below without modifications, it will run on the seed.tsv example data we have provided. It imports some helper functions we wrote to demonstrate the task on the sample dataset. You should first run all of the following cells with seed.tsv just to see how everything works. Then, once you understand the whole preprocessing / fine-tuning process, change the tsv in the below cell to your final_data.tsv file, add any extra preprocessing code you wish, and then run the cells again on your own data.

```
from helpers import tokenize_and_format, flat_accuracy
import pandas as pd
import numpy as np

df = pd.read_csv('Data_Translated.tsv',delimiter='\t',encoding='latin1')
#df = pd.read_csv('Data_First_Seed.tsv',delimiter='\t',encoding='latin1')
#df = pd.read_csv('seed.tsv')
df = df.drop_duplicates(subset=['sentence'])

#df = df.sample(n = 300)

df = df.sample(frac=1).reset_index(drop=True)

print(df.info())

# Count the frequency of each language
label_counts = df['label_name'].value_counts()

# Print the language counts
print(label_counts)
```

```
texts = df.sentence.values
labels = df.label_ID.values

### tokenize_and_format() is a helper function provided in helpers.py ###
input_ids, attention_masks = tokenize_and_format(texts)

label_list = []
for l in labels:
  label_array = np.zeros(len(set(labels)))
  label_array[int(l)-1] = 1
  label_list.append(label_array)

# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(np.array(label_list))

# Print sentence 0, now as a list of IDs.
print('Original: ', texts[0])
print('Token IDs:', input_ids[0])
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 316 entries, 0 to 315
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   sentence    316 non-null    object
 1   label_name  316 non-null    object
 2   Language    289 non-null    object
 3   label_ID    316 non-null    float64
dtypes: float64(1), object(3)
memory usage: 10.0+ KB
None
Cultural Identity                               27
Quality of Life                                 25
Economic                                        25
Capacity and Resources                          24
Public Sentiment                                23
Crime and Punishment                            23
Political                                       21
Fairness and Equality                           20
Legality, Constitutionality, Jurisdiction       20
Policy Prescription and Evaluation              20
Morality                                        20
Health and Safety                               19
External Regulation and Reputation              18
Security and Defense                            17
Other                                           14
Name: label_name, dtype: int64
```

Downloading (…)solve/main/vocab.txt: 100% 232k/232k [00:00<00:00, 2.11MB/s]

Downloading (…)okenizer_config.json: 100% 28.0/28.0 [00:00<00:00, 415B/s]

Downloading (…)lve/main/config.json: 100% 570/570 [00:00<00:00, 12.3kB/s]

```
Original:  Immigration regulations are located in Title 8 (Aliens and National
Token IDs: tensor([  101,  7521,  7040,  2024,  2284,  1999,  2516,  1022,  1(
         1998, 10662,  1007,  1997,  1996,  3642,  1997,  2976,  7040,  1010,
         2029, 14788,  2007,  2516,  1022,  1006, 12114,  1998, 10662,  1007,
         1997,  1996,  2142,  2163,  3642,  1012,   102,     0,     0,     0,
```

Create train/test/validation splits

Here we split your dataset into 3 parts: a training set, a validation set, and a testing set. Each item in your dataset will be a 3-tuple containing an input_id tensor, an attention_mask tensor, and a label tensor.

```
total = len(df)

num_train = int(total * .8)
num_val = int(total * .1)
num_test = total - num_train - num_val

# make lists of 3-tuples (already shuffled the dataframe in cell above)

train_set = [(input_ids[i], attention_masks[i], labels[i]) for i in range(num_trair
val_set = [(input_ids[i], attention_masks[i], labels[i]) for i in range(num_train,
test_set = [(input_ids[i], attention_masks[i], labels[i]) for i in range(num_val +

train_text = [texts[i] for i in range(num_train)]
val_text = [texts[i] for i in range(num_train, num_val+num_train)]
test_text = [texts[i] for i in range(num_val + num_train, total)]

print(len(train_text))
print(len(val_text))
print(len(test_text))
```

```
252
31
33
```

Here we choose the model we want to finetune from https://huggingface.co/transformers/pretrained_models.html. Because the task requires us to label sentences, we wil be using BertForSequenceClassification below. You may see a warning that states that `some weights of the model checkpoint at [model name] were not used when initializing. . .` This warning is expected and means that you should fine-tune your pre-trained model before using it on your downstream task. See [here]() for more info.

```
from transformers import BertForSequenceClassification, AdamW, BertConfig
```

```
from transformers import BertForSequenceClassification, AdamW, BertConfig

model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased", # Use the 12-layer English BERT model, with an uncased voc
    num_labels = 15, # The number of output labels.
    output_attentions = False, # Whether the model returns attentions weights.
    output_hidden_states = False, # Whether the model returns all hidden-states.
)

# Tell pytorch to run this model on the GPU.
model.cuda()
```

Downloading pytorch_model.bin: 100% 440M/440M [00:03<00:00, 135MB/s]

```
Some weights of the model checkpoint at bert-base-uncased were not used when i
- This IS expected if you are initializing BertForSequenceClassification from
- This IS NOT expected if you are initializing BertForSequenceClassification f
Some weights of BertForSequenceClassification were not initialized from the mo
You should probably TRAIN this model on a down-stream task to be able to use i
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): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (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=True)
              (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=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (1): BertLayer(
            (attention): BertAttention(
              (self): BertSelfAttention(
                (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=True)
                (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=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (2): BertLayer(
            (attention): BertAttention(
              (self): BertSelfAttention(
                (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=True)
                (dropout): Dropout(p=0.1, inplace=False)
              )
            )
            (intermediate): BertIntermediate(
```

```
            (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=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (3): BertLayer(
            (attention): BertAttention(
              (self): BertSelfAttention(
                (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=True)
                (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=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (4): BertLayer(
            (attention): BertAttention(
              (self): BertSelfAttention(
                (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=True)
                (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=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (5): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (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=True)
              (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=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (6): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (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=True)
```

```
              (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=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (7): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (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=True)
              (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=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (8): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (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,
```

```
              (LayerNorm): LayerNorm((768,), eps=1e-12,
elementwise_affine=True)
              (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=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (9): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (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=True)
              (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=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (10): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (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=True)
              (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=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
        (11): BertLayer(
          (attention): BertAttention(
            (self): BertSelfAttention(
              (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=True)
```

ACTION REQUIRED

Define your fine-tuning hyperparameters in the cell below (we have randomly picked some values to start with). We want you to experiment with different configurations to find the one that works best (i.e., highest accuracy) on your validation set. Feel free to also change pretrained models to others available in the HuggingFace library (you'll have to modify the cell above to do this). You might find papers on BERT fine-tuning stability (e.g., [Mosbach et al., ICLR 2021]) to be of interest.

```
batch_size = 50
optimizer = AdamW(model.parameters(),lr=2e-5, eps=1e-8) #with default values of lea
epochs = 50
```

```
/usr/local/lib/python3.9/dist-packages/transformers/optimization.py:391: Futu
  warnings.warn(
```

Fine-tune your model

Here we provide code for fine-tuning your model, monitoring the loss, and checking your validation accuracy. Rerun both of the below cells when you change your hyperparameters above.

```
# function to get validation accuracy
def get_validation_performance(val_set):
    # Put the model in evaluation mode
    model.eval()

    # Tracking variables
    total_eval_accuracy = 0
    total_eval_loss = 0

    num_batches = int(len(val_set)/batch_size) + 1

    total_correct = 0

    for i in range(num_batches):

      end_index = min(batch_size * (i+1), len(val_set))

      batch = val_set[i*batch_size:end_index]

      if len(batch) == 0: continue

      input_id_tensors = torch.stack([data[0] for data in batch])
      input_mask_tensors = torch.stack([data[1] for data in batch])
      label_tensors = torch.stack([data[2] for data in batch])

      # Move tensors to the GPU
      b_input_ids = input_id_tensors.to(device)
      b_input_mask = input_mask_tensors.to(device)
```

```
        b_labels = label_tensors.to(device)

        # Tell pytorch not to bother with constructing the compute graph during
        # the forward pass, since this is only needed for backprop (training).
        with torch.no_grad():

          # Forward pass, calculate logit predictions.
          outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_ma
          loss = outputs.loss
          logits = outputs.logits

          # Accumulate the validation loss.
          total_eval_loss += loss.item()

          # Move logits and labels to CPU
          logits = (logits).detach().cpu().numpy()
          label_ids = b_labels.to('cpu').numpy()


          # Calculate the number of correctly labeled examples in batch
          pred_flat = np.argmax(logits, axis=1).flatten()
          labels_flat = np.argmax(label_ids, axis=1).flatten()

          num_correct = np.sum(pred_flat == labels_flat)
          total_correct += num_correct

    # Report the final accuracy for this validation run.
    print("Num of correct predictions =", total_correct)
    avg_val_accuracy = total_correct / len(val_set)
    return avg_val_accuracy
```

```
import random

# training loop

# For each epoch...
for epoch_i in range(0, epochs):
    # Perform one full pass over the training set.

    print("")
    print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs))
    print('Training...')
```

```
    # Reset the total loss for this epoch.
    total_train_loss = 0

    # Put the model into training mode.
    model.train()

    # For each batch of training data...
    num_batches = int(len(train_set)/batch_size) + 1

    for i in range(num_batches):
      end_index = min(batch_size * (i+1), len(train_set))

      batch = train_set[i*batch_size:end_index]

      if len(batch) == 0: continue

      input_id_tensors = torch.stack([data[0] for data in batch])
      input_mask_tensors = torch.stack([data[1] for data in batch])
      label_tensors = torch.stack([data[2] for data in batch])

      # Move tensors to the GPU
      b_input_ids = input_id_tensors.to(device)
      b_input_mask = input_mask_tensors.to(device)
      b_labels = label_tensors.to(device)

      # Perform a forward pass (evaluate the model on this training batch).
      outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask
      loss = outputs.loss
      logits = outputs.logits

      total_train_loss += loss.item()

      # Clear the previously calculated gradient
      model.zero_grad()

      # Perform a backward pass to calculate the gradients.
      loss.backward()

      # Update parameters and take a step using the computed gradient.
      optimizer.step()

    # ========================================
    #               Validation
    # ========================================
```

```
    # After the completion of each training epoch, measure our performance on
    # our validation set. Implement this function in the cell above.
    print(f"Total loss: {total_train_loss}")
    val_acc = get_validation_performance(val_set)
    print(f"Validation accuracy: {val_acc}")

print("")
print("Training complete!")
```

```
Total loss: 1.062415417291224
Num of correct predictions = 9
Validation accuracy: 0.2903225806451613

======== Epoch 42 / 50 ========
Training...
Total loss: 1.0412422889918087
Num of correct predictions = 9
Validation accuracy: 0.2903225806451613

======== Epoch 43 / 50 ========
Training...
Total loss: 1.0198591057707866
Num of correct predictions = 9
Validation accuracy: 0.2903225806451613

======== Epoch 44 / 50 ========
Training...
Total loss: 1.00330280688374
Num of correct predictions = 8
Validation accuracy: 0.25806451612903225

======== Epoch 45 / 50 ========
Training...
Total loss: 0.9818803386129439
Num of correct predictions = 9
Validation accuracy: 0.2903225806451613

======== Epoch 46 / 50 ========
Training...
Total loss: 0.9556927941913407
Num of correct predictions = 10
Validation accuracy: 0.3225806451612903

======== Epoch 47 / 50 ========
Training...
Total loss: 0.9457336377302805
Num of correct predictions = 10
Validation accuracy: 0.3225806451612903
```

```
======== Epoch 48 / 50 ========
Training...
Total loss: 0.9239283576508363
Num of correct predictions = 8
Validation accuracy: 0.25806451612903225

======== Epoch 49 / 50 ========
Training...
Total loss: 0.9069636873913307
Num of correct predictions = 11
Validation accuracy: 0.3548387096774194

======== Epoch 50 / 50 ========
Training...
Total loss: 0.895257378473878
Num of correct predictions = 10
Validation accuracy: 0.3225806451612903

Training complete!
```

Evaluate your model on the test set

After you're satisfied with your hyperparameters (i.e., you're unable to achieve higher validation accuracy by modifying them further), it's time to evaluate your model on the test set! Run the below cell to compute test set accuracy.

```
get_validation_performance(test_set)
```

```
Num of correct predictions = 15
0.45454545454545453
```

```
def get_misclassified(val_set):
    # Put the model in evaluation mode
    model.eval()

    # Tracking variables
    total_eval_accuracy = 0
    total_eval_loss = 0

    num_batches = int(len(val_set)/batch_size) + 1

    total_correct = 0
```

```
    misclassified_indices = []

    for i in range(num_batches):

        end_index = min(batch_size * (i+1), len(val_set))

        batch = val_set[i*batch_size:end_index]

        if len(batch) == 0: continue

        input_id_tensors = torch.stack([data[0] for data in batch])
        input_mask_tensors = torch.stack([data[1] for data in batch])
        label_tensors = torch.stack([data[2] for data in batch])

        # Move tensors to the GPU
        b_input_ids = input_id_tensors.to(device)
        b_input_mask = input_mask_tensors.to(device)
        b_labels = label_tensors.to(device)

        # Tell pytorch not to bother with constructing the compute graph during
        # the forward pass, since this is only needed for backprop (training).
        with torch.no_grad():

            # Forward pass, calculate logit predictions.
            outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_inpu
            loss = outputs.loss
            logits = outputs.logits

            # Accumulate the validation loss.
            total_eval_loss += loss.item()

            # Move logits and labels to CPU
            logits = (logits).detach().cpu().numpy()
            label_ids = b_labels.to('cpu').numpy()

            # Calculate the number of correctly labeled examples in batch
            pred_flat = np.argmax(logits, axis=1).flatten()
            labels_flat = np.argmax(label_ids, axis=1).flatten()

            num_correct = np.sum(pred_flat == labels_flat)
            total_correct += num_correct

            # Append misclassified indices to list
            for j in range(len(batch)):
                if pred_flat[j] != labels_flat[j]:
```

```
                    misclassified_indices.append(j + i*batch_size)

    # Report the final accuracy for this validation run.
    print("Num of correct predictions =", total_correct)
    avg_val_accuracy = total_correct / len(val_set)

    # Return list of misclassified indices
    return avg_val_accuracy, misclassified_indices
```

Question 2.2 (10 points):

Finally, perform an *error analysis* on your model. This is good practice for your final project. Write some code in the below code cell to print out the text of up to five test set examples that your model gets **wrong**. If your model gets more than five test examples wrong, randomly choose five of them to analyze. If your model gets fewer than five examples wrong, please design five test examples that fool your model (i.e., *adversarial examples*). Then, in the following text cell, perform a qualitative analysis of these examples. See if you can figure out any reasons for errors that you observe, or if you have any informed guesses (e.g., common linguistic properties of these particular examples). Does this analysis suggest any possible future steps to improve your classifier?

DESCRIBE YOUR QUALITATIVE ANALYSIS OF THE ABOVE EXAMPLES IN YOUR REPORT

```
## YOUR ERROR ANALYSIS CODE HERE
## print out up to 5 test set examples (or adversarial examples) that your model

acc, mis_val = get_misclassified(val_set)

#print(mis_val[:5])

for i in mis_val[:5]:
  print(val_text[i])


acc, mis_test = get_misclassified(test_set)

#print(mis_test[:5])

for i in mis_test[:5]:
  print(test_text[i])
```

```
Num of correct predictions = 10
By the late 20th and early 21st centuries, the perspectives of one or more of
Facing a surge of migrants at the US-Mexico border and on the heels of a cris
You might be wondering if itâs enough to look at the officer and say, âCa
Among groups who feel strongly that same-sex marriage is problematic, there is
The ethical considerations surrounding immigration include questions about the
Num of correct predictions = 15
Bidenâs steps to undo Trump-era policies have included reducing immigration
Same-sex marriage is seen as a key issue of fairness and equality for LGBTQ+
State and local leaders in particular need to advance a bottom-up framework fo
An official familiar with a draft of the budget plan described details of the
Supporters of same-sex marriage argue that it is a moral imperative to recogn
```

```
from helpers import tokenize_and_format, flat_accuracy
import pandas as pd
import numpy as np

df = pd.read_csv('Output_Test.tsv',delimiter='\t',encoding='latin1')
#df = pd.read_csv('seed.tsv')
#df = df.drop_duplicates(subset=['sentence'])

#df = df.sample(n = 300)

#df = df.sample(frac=1).reset_index(drop=True)
```

```
print(df.info())

# Count the frequency of each language
#label_counts = df['label_name'].value_counts()

# Print the language counts
#print(label_counts)

texts = df.sentence.values
#labels = df.label_ID.values

### tokenize_and_format() is a helper function provided in helpers.py ###
input_ids, attention_masks = tokenize_and_format(texts)

# label_list = []
# for l in labels:
#   label_array = np.zeros(len(set(labels)))
#   label_array[int(l)-1] = 1
#   label_list.append(label_array)

# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
#labels = torch.tensor(np.array(label_list))

# Print sentence 0, now as a list of IDs.
print('Original: ', texts[0])
print('Token IDs:', input_ids[0])
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 38100 entries, 0 to 38099
Data columns (total 5 columns):
 #   Column           Non-Null Count  Dtype 
---  ------           --------------  ----- 
 0   id               38100 non-null  int64 
 1   sentence         38100 non-null  object
 2   predicted_label  38100 non-null  object
 3   label_ID         38100 non-null  object
 4   language         38100 non-null  object
dtypes: int64(1), object(4)
memory usage: 1.5+ MB
None
Original:  à°□à°¤ 15 à°¸à°□à°µà°¤à±□à°¸à°°à°¾à°²à±□à°□à°¾ à°□à°°à±□à°²à°¿à°□à°
Token IDs: tensor([  101,  1037,  7737,  2050,  7737, 29647,  2321,   100,  1
         2050,  7737,  7737,  2050, 29657,  2050,  7737, 10701,  2050,  7737,
         1094,  1037,  7737,  2050,  7737,  2050, 29657,  2050,  7737,  2050,
         7737, 29651,  2050, 29657,  1037,  7737,  2050, 29657,  2050,  7737,
         2050,  7737,  2050, 29657,  1037,  7737, 29655,  2050, 29657,  2050,
         7737,  7737,  2050, 29657,  2050,  7737,  1067,  1037, 29657,   100,
         1067,  1037,  7737,   102])
```

```
total = len(df)
print(total)

# num_train = int(total * .8)
# num_val = int(total * .1)
# num_test = total - num_train - num_val

# make lists of 3-tuples (already shuffled the dataframe in cell above)

#train_set = [(input_ids[i], attention_masks[i], labels[i]) for i in range(num_trai
#val_set = [(input_ids[i], attention_masks[i], labels[i]) for i in range(num_train,
test_set = [(input_ids[i], attention_masks[i]) for i in range(total)]

# train_text = [texts[i] for i in range(num_train)]
# val_text = [texts[i] for i in range(num_train, num_val+num_train)]
test_text = [texts[i] for i in range(total)]

# print(len(train_text))
# print(len(val_text))
print(len(test_text))
```

```
38100
38100
```

```
# function to get validation accuracy
def get_validation_performance1(val_set):
    # Put the model in evaluation mode
    model.eval()
    output_label = []

    # Tracking variables
    total_eval_accuracy = 0
    total_eval_loss = 0

    num_batches = int(len(val_set)/batch_size) + 1

    total_correct = 0

    for i in range(num_batches):

      end_index = min(batch_size * (i+1), len(val_set))

      batch = val_set[i*batch_size:end_index]
```

```
        if len(batch) == 0: continue

        input_id_tensors = torch.stack([data[0] for data in batch])
        input_mask_tensors = torch.stack([data[1] for data in batch])
        #label_tensors = torch.stack([data[2] for data in batch])

        # Move tensors to the GPU
        b_input_ids = input_id_tensors.to(device)
        b_input_mask = input_mask_tensors.to(device)
        #b_labels = label_tensors.to(device)

        # Tell pytorch not to bother with constructing the compute graph during
        # the forward pass, since this is only needed for backprop (training).
        with torch.no_grad():

          # Forward pass, calculate logit predictions.
          outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_ma
          loss = outputs.loss
          logits = outputs.logits

          # Accumulate the validation loss.
          #total_eval_loss += loss.item()

          # Move logits and labels to CPU
          logits = (logits).detach().cpu().numpy()
          #label_ids = b_labels.to('cpu').numpy()


          # Calculate the number of correctly labeled examples in batch
          pred_flat = np.argmax(logits, axis=1).flatten()
          output_label.append(pred_flat)
          #labels_flat = np.argmax(label_ids, axis=1).flatten()

    #     num_correct = np.sum(pred_flat == labels_flat)
    #     total_correct += num_correct

    # # Report the final accuracy for this validation run.
    # print("Num of correct predictions =", total_correct)
    # avg_val_accuracy = total_correct / len(val_set)
    # return avg_val_accuracy
    return output_label
```

```
output_lab = []
output_lab = get_validation_performance1(test_set)

output_final = []
for pred in output_lab:
  for j in range(len(pred)):
    output_final.append(pred[j])
```

```
print((output_final[16]))
```

```
14
```

```
df.loc[:, 'label_ID'] = output_final
```

```
label_map = {
    0.0: 'None',
    1.0: 'Economic',
    2.0: 'Capacity and Resources',
    3.0: 'Morality',
    4.0: 'Fairness and Equality',
    5.0: 'Legality, Constitutionality, Jurisdiction',
    6.0: 'Policy Prescription and Evaluation',
    7.0: 'Crime and Punishment',
    8.0: 'Security and Defense',
    9.0: 'Health and Safety',
    10.0: 'Quality of Life',
    11.0: 'Cultural Identity',
    12.0: 'Public Sentiment',
    13.0: 'Political',
    14.0: 'External Regulation and Reputation',
    15.0: 'Other'
}

df['predicted_label'] = df['label_ID'].map(label_map)

df.head()

df.to_csv("Final_Output1.tsv", sep='\t', index=False)
```

```
!pip install googletrans==3.1.0a0
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-w
Collecting googletrans==3.1.0a0
  Downloading googletrans-3.1.0a0.tar.gz (19 kB)
  Preparing metadata (setup.py) ... done
Collecting httpx==0.13.3
  Downloading httpx-0.13.3-py3-none-any.whl (55 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 55.1/55.1 KB 8.5 MB/s eta 0:00:(
Collecting rfc3986<2,>=1.3
  Downloading rfc3986-1.5.0-py2.py3-none-any.whl (31 kB)
Collecting httpcore==0.9.*
  Downloading httpcore-0.9.1-py3-none-any.whl (42 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 42.6/42.6 KB 6.2 MB/s eta 0:00:(
Requirement already satisfied: certifi in /usr/local/lib/python3.9/dist-packag
Collecting idna==2.*
  Downloading idna-2.10-py2.py3-none-any.whl (58 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 58.8/58.8 KB 10.2 MB/s eta 0:00:
Collecting sniffio
  Downloading sniffio-1.3.0-py3-none-any.whl (10 kB)
Collecting hstspreload
  Downloading hstspreload-2023.1.1-py3-none-any.whl (1.5 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.5/1.5 MB 66.5 MB/s eta 0:00:0(
Requirement already satisfied: chardet==3.* in /usr/lib/python3/dist-packages
Collecting h2==3.*
  Downloading h2-3.2.0-py2.py3-none-any.whl (65 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 65.0/65.0 KB 9.7 MB/s eta 0:00:(
Collecting h11<0.10,>=0.8
  Downloading h11-0.9.0-py2.py3-none-any.whl (53 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 53.6/53.6 KB 8.2 MB/s eta 0:00:(
Collecting hyperframe<6,>=5.2.0
  Downloading hyperframe-5.2.0-py2.py3-none-any.whl (12 kB)
Collecting hpack<4,>=3.0
  Downloading hpack-3.0.0-py2.py3-none-any.whl (38 kB)
Building wheels for collected packages: googletrans
  Building wheel for googletrans (setup.py) ... done
  Created wheel for googletrans: filename=googletrans-3.1.0a0-py3-none-any.whl
  Stored in directory: /root/.cache/pip/wheels/ae/e1/6c/5137bc3f35aa130deea715
Successfully built googletrans
Installing collected packages: rfc3986, hyperframe, hpack, h11, sniffio, idna,
  Attempting uninstall: idna
    Found existing installation: idna 3.4
    Uninstalling idna-3.4:
      Successfully uninstalled idna-3.4
Successfully installed googletrans-3.1.0a0 h11-0.9.0 h2-3.2.0 hpack-3.0.0 hsts
```

```
import googletrans
from tqdm import tqdm
```

```
from googletrans import Translator
translator = Translator()

# Define languages to translate to
languages = ['te','de','zh-CN','ne','tr','el','bn','it','ru','sw','hi']

# see available languages with the below
print(googletrans.LANGUAGES)
df1 = pd.read_csv('Data_First_Seed.tsv', delimiter='\t', encoding='latin1')

print(df1.info())
# df1 = df1[~df1['sentence'].isin(val_text)]
# df1 = df1[~df1['sentence'].isin(test_text)]
# df1.to_csv(f"en_tr.tsv", sep='\t', index=False)
# df1 = df1.sample(n=300).reset_index(drop=True)

# print(df1.info())

for lang in tqdm(languages):
  print(lang)
  # Iterate through each row and translate non-English sentences
  for index, row in tqdm(df1.iterrows(), total=len(df1)):
    translated = translator.translate(row['sentence'], src='auto', dest=lang).text
    df1.loc[index, 'sentence'] = translated
    df1.loc[index, 'Language'] = lang

  df1.to_csv(f"{lang}_tr.tsv", sep='\t', index=False)
```

```
{'af': 'afrikaans', 'sq': 'albanian', 'am': 'amharic', 'ar': 'arabic', 'hy':
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 320 entries, 0 to 319
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   sentence    320 non-null    object
 1   label_name  320 non-null    object
 2   Language    292 non-null    object
 3   label_ID    320 non-null    float64
dtypes: float64(1), object(3)
memory usage: 10.1+ KB
None
  0%|          | 0/11 [00:00<?, ?it/s]te

  0%|          | 0/320 [00:00<?, ?it/s]
```

```
  0%|          | 1/320 [00:00<00:32,  9.75it/s]
  1%|          | 2/320 [00:00<00:43,  7.35it/s]
  1%|▏         | 4/320 [00:00<00:35,  9.00it/s]
  2%|▏         | 5/320 [00:00<00:34,  9.11it/s]
  2%|▏         | 6/320 [00:00<00:34,  9.15it/s]
  2%|▏         | 7/320 [00:00<00:33,  9.27it/s]
  3%|▎         | 9/320 [00:00<00:32,  9.72it/s]
  3%|▎         | 11/320 [00:01<00:31,  9.86it/s]
  4%|▍         | 12/320 [00:01<00:31,  9.84it/s]
  4%|▍         | 14/320 [00:01<00:29, 10.44it/s]
  5%|▌         | 16/320 [00:01<00:29, 10.32it/s]
  6%|▌         | 18/320 [00:01<00:29, 10.37it/s]
  6%|▋         | 20/320 [00:02<00:28, 10.37it/s]
  7%|▋         | 22/320 [00:02<00:29, 10.26it/s]
  8%|▊         | 24/320 [00:02<00:29, 10.17it/s]
  8%|▊         | 26/320 [00:02<00:31,  9.31it/s]
  8%|▊         | 27/320 [00:02<00:31,  9.23it/s]
  9%|▉         | 29/320 [00:02<00:29,  9.78it/s]
  9%|▉         | 30/320 [00:03<00:29,  9.74it/s]
 10%|█         | 32/320 [00:03<00:28, 10.05it/s]
 11%|█         | 34/320 [00:03<00:29,  9.83it/s]
 11%|█         | 35/320 [00:03<00:29,  9.55it/s]
 11%|█▏        | 36/320 [00:03<00:29,  9.55it/s]
 12%|█▏        | 37/320 [00:03<00:30,  9.40it/s]
 12%|█▏        | 38/320 [00:03<00:30,  9.38it/s]
 12%|█▏        | 39/320 [00:04<00:30,  9.36it/s]
 12%|█▎        | 40/320 [00:04<00:30,  9.05it/s]
 13%|█▎        | 41/320 [00:04<00:36,  7.66it/s]
 13%|█▎        | 42/320 [00:04<00:34,  7.99it/s]
 13%|█▎        | 43/320 [00:04<00:35,  7.85it/s]
 14%|█▍        | 44/320 [00:04<00:32,  8.37it/s]
 14%|█▍        | 45/320 [00:04<00:31,  8.66it/s]
 15%|█▍        | 47/320 [00:04<00:29,  9.38it/s]
 15%|█▌        | 48/320 [00:05<00:30,  9.07it/s]
 15%|█▌        | 49/320 [00:05<00:30,  8.93it/s]
 16%|█▌        | 50/320 [00:05<00:30,  8.97it/s]
 16%|█▌        | 51/320 [00:05<00:31,  8.48it/s]
 16%|█▋        | 52/320 [00:05<00:30,  8.72it/s]
 17%|█▋        | 53/320 [00:05<00:29,  9.05it/s]
 17%|█▋        | 54/320 [00:05<00:29,  9.16it/s]
 17%|█▋        | 55/320 [00:05<00:29,  8.99it/s]
 18%|█▊        | 57/320 [00:06<00:27,  9.74it/s]
 18%|█▊        | 58/320 [00:06<00:28,  9.09it/s]
 19%|█▉        | 60/320 [00:06<00:26,  9.78it/s]
```

```
import os
import glob
import pandas as pd

# get all files containing 'aug' and ending with '.tsv'
file_list = glob.glob('*tr*.tsv')
print(file_list)
```

```
['te_tr.tsv', 'de_tr.tsv', 'zh-CN_tr.tsv', 'ne_tr.tsv', 'tr_tr.tsv', 'el_tr.ts
```

```
# initialize an empty dataframe
df_all = pd.DataFrame()

data = pd.read_csv('Data_First_Seed.tsv', delimiter='\t', encoding='latin1')
data['Language'] = 'en'
df_all = pd.concat([df_all, data], ignore_index=True)

# loop through all files and concatenate them
for file in file_list:
    data = pd.read_csv(file, delimiter='\t', encoding='latin1')
    df_all = pd.concat([df_all, data], ignore_index=True)

# save the concatenated dataframe to a new tsv file
df_all.to_csv('Data_Translated.tsv', sep='\t', index=False)
```

```
print(df_all.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3840 entries, 0 to 3839
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   sentence    3840 non-null   object
 1   label_name  3840 non-null   object
 2   Language    3840 non-null   object
 3   label_ID    3840 non-null   float64
dtypes: float64(1), object(3)
memory usage: 120.1+ KB
None
```

```
df = pd.read_csv('Data_Translated.tsv',delimiter='\t',encoding='latin1')
#df = pd.read_csv('seed.tsv')
df = df.drop_duplicates(subset=['sentence'])
```

```
df = df.sample(frac=1).reset_index(drop=True)

print(df.info())

# Count the frequency of each language
label_counts = df['label_name'].value_counts()

# Print the language counts
print(label_counts)

texts = df.sentence.values
labels = df.label_ID.values

### tokenize_and_format() is a helper function provided in helpers.py ###
input_ids, attention_masks = tokenize_and_format(texts)

label_list = []
for l in labels:
  label_array = np.zeros(len(set(labels)))
  label_array[int(l)-1] = 1
  label_list.append(label_array)

# Convert the lists into tensors.
input_ids = torch.cat(input_ids, dim=0)
attention_masks = torch.cat(attention_masks, dim=0)
labels = torch.tensor(np.array(label_list))

# Print sentence 0, now as a list of IDs.
print('Original: ', texts[0])
print('Token IDs:', input_ids[0])
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 7012 entries, 0 to 7011
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   label_name  7012 non-null   object
 1   sentence    7012 non-null   object
 2   label_ID    7012 non-null   float64
 3   Language    7012 non-null   object
dtypes: float64(1), object(3)
memory usage: 219.2+ KB
None
Political                                    1714
Fairness and Equality                         882
Public Sentiment                              705
External Regulation and Reputation            679
Other                                         443
Economic                                      416
Morality                                      395
Capacity and Resources                        358
Crime and Punishment                          338
Quality of Life                               315
Legality, Constitutionality, Jurisdiction     255
Security and Defense                          211
Cultural Identity                             198
Health and Safety                              73
Policy Prescription and Evaluation             30
Name: label_name, dtype: int64
Original:  Nel 2016, sono stati commessi 307 crimini d'odio contro i musulman
Token IDs: tensor([  101, 11265,  2140,  2355,  1010,  2365,  2080, 28093,  2
         7834,  5332, 24559, 13675, 27605,  3490,  1040,  1005, 21045,  2080,
         9530, 13181,  1045, 14163, 23722, 20799,  1010,  2539,  1999, 24624,
         2050, 27904, 15544, 13102, 20082,  2632,  2325,  1010,  1041,  6335,
         2079,  9397,  3695, 14866, 13675, 27605,  3490,  1040,  1005, 21045,
         2080,  9530, 13181,  1045, 14163, 23722, 20799, 11265,  2140,  2355,
        15544, 13102, 20082,   102])
```

```
total = len(df)

num_train = int(total)


# make lists of 3-tuples (already shuffled the dataframe in cell above)

train_set = [(input_ids[i], attention_masks[i], labels[i]) for i in range(num_trair


train_text = [texts[i] for i in range(num_train)]


print(len(train_text))
```

```
7012
```

```
batch_size = 50
optimizer = AdamW(model.parameters(),lr=2e-6, eps=1e-8) #with default values of lea
epochs = 25
```

```
/usr/local/lib/python3.9/dist-packages/transformers/optimization.py:391: Futur
  warnings.warn(
```

```
import random

# training loop

# For each epoch...
for epoch_i in range(0, epochs):
    # Perform one full pass over the training set.

    print("")
    print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs))
    print('Training...')

    # Reset the total loss for this epoch.
    total_train_loss = 0

    # Put the model into training mode.
    model.train()

    # For each batch of training data...
```

```
    num_batches = int(len(train_set)/batch_size) + 1

    for i in range(num_batches):
      end_index = min(batch_size * (i+1), len(train_set))

      batch = train_set[i*batch_size:end_index]

      if len(batch) == 0: continue

      input_id_tensors = torch.stack([data[0] for data in batch])
      input_mask_tensors = torch.stack([data[1] for data in batch])
      label_tensors = torch.stack([data[2] for data in batch])

      # Move tensors to the GPU
      b_input_ids = input_id_tensors.to(device)
      b_input_mask = input_mask_tensors.to(device)
      b_labels = label_tensors.to(device)

      # Perform a forward pass (evaluate the model on this training batch).
      outputs = model(b_input_ids, token_type_ids=None, attention_mask=b_input_mask
      loss = outputs.loss
      logits = outputs.logits

      total_train_loss += loss.item()

      # Clear the previously calculated gradient
      model.zero_grad()

      # Perform a backward pass to calculate the gradients.
      loss.backward()

      # Update parameters and take a step using the computed gradient.
      optimizer.step()

    # ========================================
    #               Validation
    # ========================================
    # After the completion of each training epoch, measure our performance on
    # our validation set. Implement this function in the cell above.
    print(f"Total loss: {total_train_loss}")
    val_acc = get_validation_performance(val_set)
    print(f"Validation accuracy: {val_acc}")

print("")
print("Training complete!")
```

```
======== Epoch 1 / 25 ========
Training...
Total loss: 31.652747612509046
Num of correct predictions = 27
Validation accuracy: 0.2125984251968504

======== Epoch 2 / 25 ========
Training...
Total loss: 31.590699122981075
Num of correct predictions = 27
Validation accuracy: 0.2125984251968504

======== Epoch 3 / 25 ========
Training...
Total loss: 31.55457921603901
Num of correct predictions = 27
Validation accuracy: 0.2125984251968504

======== Epoch 4 / 25 ========
Training...
Total loss: 31.538235533048557
Num of correct predictions = 27
Validation accuracy: 0.2125984251968504

======== Epoch 5 / 25 ========
Training...
Total loss: 31.523047991908527
Num of correct predictions = 27
Validation accuracy: 0.2125984251968504

======== Epoch 6 / 25 ========
Training...
Total loss: 31.488271860489565
Num of correct predictions = 27
Validation accuracy: 0.2125984251968504

======== Epoch 7 / 25 ========
Training...
Total loss: 31.475427753208212
Num of correct predictions = 27
Validation accuracy: 0.2125984251968504

======== Epoch 8 / 25 ========
Training...
Total loss: 31.477269627198037
Num of correct predictions = 27
Validation accuracy: 0.2125984251968504
```

```
======== Epoch 9 / 25 ========
Training...
```

```
get_validation_performance(test_set)
```

Finished? Remember to upload the PDF file of this notebook, report and your three dataset files (annotator1.tsv, annotator2.tsv, and final_data.tsv) to Gradescope with the filename line formatted as **Firstname_Lastname_HW2**.