```
import numpy as np
import pandas as pd
import time
import datetime
import gc
import random
from nltk.corpus import stopwords
import re
```

```
import torch
import torch.nn as nn
from torch.utils.data import TensorDataset, DataLoader, RandomSampler, SequentialSampler,random_split
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
```

```
!pip install transformers
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: transformers in /usr/local/lib/python3.8/dist-packages (4.26.0)
Requirement already satisfied: huggingface-hub<1.0,>=0.11.0 in /usr/local/lib/python3.8/dist-packages (from transformers) (0.12.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.8/dist-packages (from transformers) (21.3)
Requirement already satisfied: tokenizers!=0.11.3,<0.14,>=0.11.1 in /usr/local/lib/python3.8/dist-packages (from transformers) (0.13.2)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.8/dist-packages (from transformers) (4.64.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.8/dist-packages (from transformers) (3.9.0)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.8/dist-packages (from transformers) (2022.6.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.8/dist-packages (from transformers) (6.0)
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.8/dist-packages (from transformers) (1.21.6)
Requirement already satisfied: requests in /usr/local/lib/python3.8/dist-packages (from transformers) (2.25.1)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.8/dist-packages (from huggingface-hub<1.0,>=0.11.0-
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.8/dist-packages (from packaging>=20.0->transformers)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.8/dist-packages (from requests->transformers) (2.10)
Requirement already satisfied: urllib3<1.27,>=1.21.1 in /usr/local/lib/python3.8/dist-packages (from requests->transformers) (1.24.3)
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.8/dist-packages (from requests->transformers) (4.0.0)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.8/dist-packages (from requests->transformers) (2022.12.7)
```

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

```
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
device
```

```
device(type='cpu')
```

```
df = pd.read_csv("/content/train_all_tasks.csv")
df.head()
```

	rewire_id	text	label_sexist	label_category	label_vector
0	sexism2022_english-7358	Damn, this writing was pretty chaotic	not sexist	none	none
1	sexism2022_english-2367	Yeah, and apparently a bunch of misogynistic v...	not sexist	none	none
2	sexism2022_english-3073	How the FUCK is this woman still an MP!!!???	not sexist	none	none
3	sexism2022_english-14895	Understand. Know you're right. At same time I ...	not sexist	none	none
4	sexism2022_english-	Surprized they didn't	not sexist	none	none

```
df['label_vector'].value_counts()
```

```
none                                                    10602
2.1 descriptive attacks                                   717
2.2 aggressive and emotive attacks                        673
3.1 casual use of gendered slurs, profanities, and insults  637
3.2 immutable gender differences and gender stereotypes   417
```

```
4.2 supporting systemic discrimination against women as a group      258
1.2 incitement and encouragement of harm                             254
2.3 dehumanising attacks & overt sexual objectification              200
4.1 supporting mistreatment of individual women                       75
3.3 backhanded gendered compliments                                   64
1.1 threats of harm                                                   56
3.4 condescending explanations or unwelcome advice                    47
Name: label_vector, dtype: int64
```

```
# dropping the null values from df['label_category']

df = df[df['label_vector']!='none']
```

```
df['label_vector'].value_counts()
```

```
2.1 descriptive attacks                                    717
2.2 aggressive and emotive attacks                         673
3.1 casual use of gendered slurs, profanities, and insults 637
3.2 immutable gender differences and gender stereotypes    417
4.2 supporting systemic discrimination against women as a group 258
1.2 incitement and encouragement of harm                   254
2.3 dehumanising attacks & overt sexual objectification    200
4.1 supporting mistreatment of individual women             75
3.3 backhanded gendered compliments                         64
1.1 threats of harm                                         56
3.4 condescending explanations or unwelcome advice          47
Name: label_vector, dtype: int64
```

```
!pip install nltk
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: nltk in /usr/local/lib/python3.8/dist-packages (3.7)
Requirement already satisfied: joblib in /usr/local/lib/python3.8/dist-packages (from nltk) (1.2.0)
Requirement already satisfied: click in /usr/local/lib/python3.8/dist-packages (from nltk) (7.1.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.8/dist-packages (from nltk) (4.64.1)
Requirement already satisfied: regex>=2021.8.3 in /usr/local/lib/python3.8/dist-packages (from nltk) (2022.6.2)
```

```
import nltk
nltk.download('stopwords')
```

```
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
True
```

```
sw = stopwords.words('english')

def clean_text(text):

    text = text.lower()

    text = re.sub(r"[^a-zA-Z?.!,¿]+", " ", text) # replacing everything with space except (a-z, A-Z, ".", "?", "!", ",")

    text = re.sub(r"http\S+", "",text) #Removing URLs
    #text = re.sub(r"http", "",text)

    html=re.compile(r'<.*?>')

    text = html.sub(r'',text) #Removing html tags

    punctuations = '@#!?+&*[]-%.:/();$=><|{}^' + "'`" + '_'
    for p in punctuations:
        text = text.replace(p,'') #Removing punctuations

    text = [word.lower() for word in text.split() if word.lower() not in sw]

    text = " ".join(text) #removing stopwords

    emoji_pattern = re.compile("["
                           u"\U0001F600-\U0001F64F"  # emoticons
                           u"\U0001F300-\U0001F5FF"  # symbols & pictographs
                           u"\U0001F680-\U0001F6FF"  # transport & map symbols
                           u"\U0001F1E0-\U0001F1FF"  # flags (iOS)
                           u"\U00002702-\U000027B0"
                           u"\U000024C2-\U0001F251"
                           "]+", flags=re.UNICODE)
```

```
    text = emoji_pattern.sub(r'', text) #Removing emojis

    return text
```

```
df['text'] = df['text'].apply(lambda x: clean_text(x))
```

```
tweets = df.text.values
labels = df.label_vector.values
```

```
set(list(labels))
```

```
{'1.1 threats of harm',
 '1.2 incitement and encouragement of harm',
 '2.1 descriptive attacks',
 '2.2 aggressive and emotive attacks',
 '2.3 dehumanising attacks & overt sexual objectification',
 '3.1 casual use of gendered slurs, profanities, and insults',
 '3.2 immutable gender differences and gender stereotypes',
 '3.3 backhanded gendered compliments',
 '3.4 condescending explanations or unwelcome advice',
 '4.1 supporting mistreatment of individual women',
 '4.2 supporting systemic discrimination against women as a group'}
```

```
from sklearn.preprocessing import LabelEncoder
```

```
le=LabelEncoder()
```

```
labels = le.fit_transform(labels)
```

```
labels
```

```
array([ 4,  2,  4, ..., 10,  3,  2])
```

```
# Load the BERT tokenizer
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased', do_lower_case=True)
```

```
print(' Original: ', tweets[0])

# Print the sentence split into tokens.
print('Tokenized: ', tokenizer.tokenize(tweets[0]))

# Print the sentence mapped to token ids.
print('Token IDs: ', tokenizer.convert_tokens_to_ids(tokenizer.tokenize(tweets[0])))
```

```
 Original:  inside wallet cash, bank cards, credit cards, debit cards inside matters femoids
Tokenized:  ['inside', 'wallet', 'cash', ',', 'bank', 'cards', ',', 'credit', 'cards', ',', 'de', '##bit', 'cards', 'inside', 'matters'
Token IDs:  [2503, 15882, 5356, 1010, 2924, 5329, 1010, 4923, 5329, 1010, 2139, 16313, 5329, 2503, 5609, 10768, 5302, 9821]
```

```
max_len = 0

# For every sentence...
for sent in tweets:

    # Tokenize the text and add `[CLS]` and `[SEP]` tokens.
    input_ids = tokenizer.encode(sent, add_special_tokens=True)

    # Update the maximum sentence length.
    max_len = max(max_len, len(input_ids))

print('Max sentence length: ', max_len)
```

```
 Max sentence length:  52
```

```
input_ids = []
attention_masks = []

# For every tweet...
for tweet in tweets:
```

```
    # `encode_plus` will:
    #   (1) Tokenize the sentence.
    #   (2) Prepend the `[CLS]` token to the start.
    #   (3) Append the `[SEP]` token to the end.
    #   (4) Map tokens to their IDs.
    #   (5) Pad or truncate the sentence to `max_length`
    #   (6) Create attention masks for [PAD] tokens.
    encoded_dict = tokenizer.encode_plus(
                        tweet,                      # Sentence to encode.
                        add_special_tokens = True, # Add '[CLS]' and '[SEP]'
                        max_length = max_len,           # Pad & truncate all sentences.
                        pad_to_max_length = True,
                        return_attention_mask = True,   # Construct attn. masks.
                        return_tensors = 'pt',     # Return pytorch tensors.
                   )

    # Add the encoded sentence to the list.
    input_ids.append(encoded_dict['input_ids'])

    # And its attention mask (simply differentiates padding from non-padding).
    attention_masks.append(encoded_dict['attention_mask'])

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

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

```
Truncation was not explicitly activated but `max_length` is provided a specific value, please use `truncation=True` to explicitly trunc
/usr/local/lib/python3.8/dist-packages/transformers/tokenization_utils_base.py:2339: FutureWarning: The `pad_to_max_length` argument is
  warnings.warn(
Original:  inside wallet cash, bank cards, credit cards, debit cards inside matters femoids
Token IDs: tensor([  101,  2503, 15882,  5356,  1010,  2924,  5329,  1010,  4923,  5329,
         1010,  2139, 16313,  5329,  2503,  5609, 10768,  5302,  9821,   102,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0])
```

```
# Combine the training inputs into a TensorDataset.
dataset = TensorDataset(input_ids, attention_masks, labels)

# Create a 90-10 train-validation split.

# Calculate the number of samples to include in each set.
train_size = int(0.8 * len(dataset))
#val_size = int(0.2 * len(dataset))
val_size = len(dataset)  - train_size

# Divide the dataset by randomly selecting samples.
train_dataset, val_dataset = random_split(dataset, [train_size, val_size])

print('{:>5,} training samples'.format(train_size))
print('{:>5,} validation samples'.format(val_size))
```

```
2,718 training samples
  680 validation samples
```

```
# The DataLoader needs to know our batch size for training, so we specify it
# here. For fine-tuning BERT on a specific task, the authors recommend a batch
# size of 16 or 32.
batch_size = 32

# Create the DataLoaders for our training and validation sets.
# We'll take training samples in random order.
train_dataloader = DataLoader(
            train_dataset,  # The training samples.
            sampler = RandomSampler(train_dataset), # Select batches randomly
            batch_size = batch_size # Trains with this batch size.
        )
```

```
# For validation the order doesn't matter, so we'll just read them sequentially.
validation_dataloader = DataLoader(
            val_dataset, # The validation samples.
            sampler = SequentialSampler(val_dataset), # Pull out batches sequentially.
            batch_size = batch_size # Evaluate with this batch size.
        )
```

```
# Load BertForSequenceClassification, the pretrained BERT model with a single
# linear classification layer on top.
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased", # Use the 12-layer BERT model, with an uncased vocab.
    num_labels = 11, # The number of output labels--2 for binary classification.
                  # You can increase this for multi-class tasks.
    output_attentions = False, # Whether the model returns attentions weights.
    output_hidden_states = False, # Whether the model returns all hidden-states.
)

# if device == "cuda:0":
# # Tell pytorch to run this model on the GPU.
#     model = model.cuda()
model = model.to(device)
```

```
Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertForSequenceClassification: ['cls.seq_rela
- This IS expected if you are initializing BertForSequenceClassification from the checkpoint of a model trained on another task or with
- This IS NOT expected if you are initializing BertForSequenceClassification from the checkpoint of a model that you expect to be exact
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initial
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
optimizer = AdamW(model.parameters(),
                  lr = 2e-5, # args.learning_rate - default is 5e-5, our notebook had 2e-5
                  eps = 1e-8 # args.adam_epsilon  - default is 1e-8.
                )
```

```
eWarning: This implementation of AdamW is deprecated and will be removed in a future version. Use the PyTorch implementation torch.optim
```

```
# Number of training epochs. The BERT authors recommend between 2 and 4.
# We chose to run for 4, but we'll see later that this may be over-fitting the
# training data.
epochs = 10

# Total number of training steps is [number of batches] x [number of epochs].
# (Note that this is not the same as the number of training samples).
total_steps = len(train_dataloader) * epochs

# Create the learning rate scheduler.
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps = 0, # Default value in run_glue.py
                                            num_training_steps = total_steps)
```

```
# Function to calculate the accuracy of our predictions vs labels
def flat_accuracy(preds, labels):
    pred_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    return np.sum(pred_flat == labels_flat) / len(labels_flat)
```

```
def format_time(elapsed):
    '''
    Takes a time in seconds and returns a string hh:mm:ss
    '''
    # Round to the nearest second.
    elapsed_rounded = int(round((elapsed)))
    # Format as hh:mm:ss
    return str(datetime.timedelta(seconds=elapsed_rounded))
```

```
seed_val = 42
random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)
training_stats = []
```

```
training_stats = []

# Measure the total training time for the whole run.
total_t0 = time.time()

# For each epoch...
for epoch_i in range(0, epochs):

    # ========================================
    #               Training
    # ========================================
    # Perform one full pass over the training set.
    print("")
    print('======== Epoch {:} / {:} ========'.format(epoch_i + 1, epochs))
    print('Training...')
    # Measure how long the training epoch takes.
    t0 = time.time()
    total_train_loss = 0
    model.train()
    for step, batch in enumerate(train_dataloader):
        # Unpack this training batch from our dataloader.
        #
        # As we unpack the batch, we'll also copy each tensor to the device using the
        # `to` method.
        #
        # `batch` contains three pytorch tensors:
        #   [0]: input ids
        #   [1]: attention masks
        #   [2]: labels
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].to(device)
        optimizer.zero_grad()
        output = model(b_input_ids,
                            token_type_ids=None,
                            attention_mask=b_input_mask,
                            labels=b_labels)
        loss = output.loss
        total_train_loss += loss.item()
        # Perform a backward pass to calculate the gradients.
        loss.backward()
        # Clip the norm of the gradients to 1.0.
        # This is to help prevent the "exploding gradients" problem.
        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)
        # Update parameters and take a step using the computed gradient.
        # The optimizer dictates the "update rule"--how the parameters are
        # modified based on their gradients, the learning rate, etc.
        optimizer.step()
        # Update the learning rate.
        scheduler.step()

    # Calculate the average loss over all of the batches.
    avg_train_loss = total_train_loss / len(train_dataloader)

    # Measure how long this epoch took.
    training_time = format_time(time.time() - t0)
    print("")
    print("  Average training loss: {0:.2f}".format(avg_train_loss))
    print("  Training epcoh took: {:}".format(training_time))
    # ========================================
    #               Validation
    # ========================================
    # After the completion of each training epoch, measure our performance on
    # our validation set.
    print("")
    print("Running Validation...")
    t0 = time.time()
    # Put the model in evaluation mode--the dropout layers behave differently
    # during evaluation.
    model.eval()
    # Tracking variables
    total_eval_accuracy = 0
    best_eval_accuracy = 0
    total_eval_loss = 0
    nb_eval_steps = 0
    # Evaluate data for one epoch
    for batch in validation_dataloader:
        b_input_ids = batch[0].to(device)
```

```
        b_input_mask = batch[1].to(device)
        b_labels = batch[2].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():
            output= model(b_input_ids,
                                token_type_ids=None,
                                attention_mask=b_input_mask,
                                labels=b_labels)
        loss = output.loss
        total_eval_loss += loss.item()
        # Move logits and labels to CPU if we are using GPU
        logits = output.logits
        logits = logits.detach().cpu().numpy()
        label_ids = b_labels.to('cpu').numpy()
        # Calculate the accuracy for this batch of test sentences, and
        # accumulate it over all batches.
        total_eval_accuracy += flat_accuracy(logits, label_ids)
    # Report the final accuracy for this validation run.
    avg_val_accuracy = total_eval_accuracy / len(validation_dataloader)
    print("  Accuracy: {0:.2f}".format(avg_val_accuracy))
    # Calculate the average loss over all of the batches.
    avg_val_loss = total_eval_loss / len(validation_dataloader)
    # Measure how long the validation run took.
    validation_time = format_time(time.time() - t0)
    if avg_val_accuracy > best_eval_accuracy:
        torch.save(model, 'bert_model')
        best_eval_accuracy = avg_val_accuracy
    #print("  Validation Loss: {0:.2f}".format(avg_val_loss))
    #print("  Validation took: {:}".format(validation_time))
    # Record all statistics from this epoch.
    training_stats.append(
        {
            'epoch': epoch_i + 1,
            'Training Loss': avg_train_loss,
            'Valid. Loss': avg_val_loss,
            'Valid. Accur.': avg_val_accuracy,
            'Training Time': training_time,
            'Validation Time': validation_time
        }
    )
print("")
print("Training complete!")

print("Total training took {:} (h:mm:ss)".format(format_time(time.time()-total_t0)))
```

```
Running Validation...
  Accuracy: 0.46

======== Epoch 9 / 10 ========
Training...

  Average training loss: 0.66
  Training epcoh took: 0:20:30

Running Validation...
  Accuracy: 0.46

======== Epoch 10 / 10 ========
Training...

  Average training loss: 0.61
  Training epcoh took: 0:20:25

Running Validation...
  Accuracy: 0.47

Training complete!
Total training took 3:41:14 (h:mm:ss)
```

```
model = torch.load('bert_model')
```

```
df_test = pd.read_csv('/content/test_task_c_entries.csv')
df_test['text'] = df_test['text'].apply(lambda x:clean_text(x))
test_tweets = df_test['text'].values
```

```
test_input_ids = []
test_attention_masks = []
for tweet in test_tweets:
    encoded_dict = tokenizer.encode_plus(
                        tweet,
                        add_special_tokens = True,
                        max_length = max_len,
                        pad_to_max_length = True,
                        return_attention_mask = True,
                        return_tensors = 'pt',
                   )
    test_input_ids.append(encoded_dict['input_ids'])
    test_attention_masks.append(encoded_dict['attention_mask'])
test_input_ids = torch.cat(test_input_ids, dim=0)
test_attention_masks = torch.cat(test_attention_masks, dim=0)
```

```
/usr/local/lib/python3.8/dist-packages/transformers/tokenization_utils_base.py:2339: FutureWarning: The `pad_to_max_length` argument is
  warnings.warn(
```

```
test_dataset = TensorDataset(test_input_ids, test_attention_masks)
test_dataloader = DataLoader(
            test_dataset, # The validation samples.
            sampler = SequentialSampler(test_dataset), # Pull out batches sequentially.
            batch_size = batch_size # Evaluate with this batch size.
        )
```

```
predictions = []
for batch in test_dataloader:
        b_input_ids = batch[0].to(device)
        b_input_mask = batch[1].to(device)
        with torch.no_grad():
            output= model(b_input_ids,
                                token_type_ids=None,
                                attention_mask=b_input_mask)
            logits = output.logits
            pred_flat = np.argmax(logits, axis=1).flatten()

            predictions.extend(list(pred_flat))
```

```
predictions=le.inverse_transform(predictions)
```

```
df_output = pd.DataFrame()
df_output['rewire_id'] = df_test['rewire_id']
df_output['label_pred'] =predictions
df_output.to_csv('test_submission1.csv',index=False)
```

```
df1 = pd.read_csv('/content/test_submission1.csv')
```

```
df1.head()
```

	rewire_id	label_pred
0	sexism2022_english-10731	2.1 descriptive attacks
1	sexism2022_english-7356	1.2 incitement and encouragement of harm
2	sexism2022_english-13064	1.2 incitement and encouragement of harm
3	sexism2022_english-17039	3.2 immutable gender differences and gender st...
4	sexism2022_english-14482	2.1 descriptive attacks

```
label_pred = le.inverse_transform(labels)
```

```
label_pred
```

```
array(['2.3 dehumanising attacks & overt sexual objectification',
       '2.1 descriptive attacks',
       '2.3 dehumanising attacks & overt sexual objectification', ...,
       '4.2 supporting systemic discrimination against women as a group',
       '2.2 aggressive and emotive attacks', '2.1 descriptive attacks'],
      dtype=object)
```

```
df1.head()
```

	rewire_id	label_pred
0	sexism2022_english-739	2
1	sexism2022_english-10787	2
2	sexism2022_english-18547	2
3	sexism2022_english-6425	1
4	sexism2022_english-10001	3

```
df1['label_pred'].value_counts()
```

```
2     138
3     101
5      78
6      65
1      46
10     30
4      24
9       4
Name: label_pred, dtype: int64
```

```
df1['label_pred'].shape
```

```
(486,)
```

```
set(list(labels))
```

```
        tensor(3),
        tensor(3),
        tensor(1),
        tensor(1),
        tensor(4),
        tensor(5),
        tensor(2),
        tensor(6),
        tensor(3),
        tensor(7),
        tensor(3),
        tensor(3),
        tensor(5),
        tensor(10),
        tensor(3),
        tensor(6),
        tensor(2),
        tensor(1),
        tensor(1),
        tensor(5),
        tensor(2),
        tensor(2),
        tensor(6),
        tensor(6),
        tensor(5),
        tensor(0),
        tensor(2),
        tensor(5),
        tensor(3),
        tensor(5),
        tensor(3),
        tensor(2),
        tensor(5),
        tensor(2),
        tensor(2),
        tensor(3),
        tensor(3),
        tensor(7),
        tensor(3),
        tensor(6),
        tensor(5),
        tensor(2),
        tensor(3),
        tensor(2),
        tensor(5),
```