In [ ]:

```python
# This preliminary notebook tests BERT's classifaction capabilities on collaboration
# codes from student classroom discussions. The codes fall into the following
# classes: Non-argumentative, New Idea, Extension, Agreement, Challenge

# We will start off with a pretrained BERT model, train it on the new data,
# and evaluate its classification performance.



import numpy as np
import pandas as pd
import random


import os
import torch
from torch import nn
from torch.utils.data import DataLoader, Dataset
from transformers import BertTokenizer, BertModel, AdamW, get_linear_schedule_with_warmup
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report


from torch.utils.data import TensorDataset
from transformers import BertForSequenceClassification

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

from tqdm.notebook import tqdm
```

Intro and reading data

Reading in the student_discussion_data.csv file, which corresponds to the Nights1 data, our goal is to implement the Bert model to classify the discussions into the appropriate collaboration code.

Collaboration Code: This is the Collaboration label and is empty for teacher turns and for student turns it is one of the following 5 labels: 'Non': Non-argumentative 'N': New idea 'E': Extension 'A': Agree 'C': Challenge

So, we will first drop all columns excluding the Talk and Collaboration Code, as none of the other columns provides us with any useful info for classification.

In [ ]:

```python
data = pd.read_csv('student_discussion_data.csv')

data
```

Out[ ]:

	Disc.id	order	Sp.id	Talk	Time	Collaboration.Code	Turn.of.Reference	Claim	Evidence	Warrant	claim.segment	evidence.segment	warra
0	T1.5.DT_2022.1.Night.59	59	St_3_WM	I kind want to say that he should, although th...	03:33	N	T1.5.DT_2022.1.Night.59	True	True	False	I kind want to say that he should, although th...	since you can consider whenever the Russian in...	
1	T1.5.DT_2022.1.Night.60	60	St_3_WM	But he really did not know about the Russians....	04:03	E	T1.5.DT_2022.1.Night.59	True	True	False	But he really did not know about the Russians....	Elie really should have felt a major guilt for...	
2	T1.5.DT_2022.1.Night.61	61	St_2_WF	Yeah.\n	04:40	Non	NaN	False	False	False	NaN	NaN	
3	T1.5.DT_2022.1.Night.62	62	St_??_??	Sorry, go ahead.\n	04:41	Non	NaN	False	False	False	NaN	NaN	
4	T1.5.DT_2022.1.Night.63	63	St_2_WF	Adding onto that question, I said that there i...	04:43	E	T1.5.DT_2022.1.Night.60	True	True	True	Adding onto that question, I said that there i...	because he just wonders to himself, "Why did I...	But a bac thre
...	...	...	...	...	...	...	...	...	...	...	...	...	
83	T1.5.DT_2022.1.Night.155	155	St_22_WF	Adding to that point, I think this memoir did ...	43:56	E	T1.5.DT_2022.1.Night.154	True	True	False	Adding to that point, I think this memoir did ...	because what we've learned about in the past, ...	
84	T1.5.DT_2022.1.Night.156	156	St_4_WM	Yeah, I think even the fact that he names othe...	44:20	E	T1.5.DT_2022.1.Night.155	True	True	True	Yeah, I think even the fact that he names othe...	So, you just know them by their minority.	Tl becau like,
85	T1.5.DT_2022.1.Night.157	157	St_4_WM	I think there were even some people that we di...	45:07	E	T1.5.DT_2022.1.Night.156	True	True	False	I think there were even some people that we di...	I really appreciated that just from the standp...	
86	T1.5.DT_2022.1.Night.158	158	St_23_BF	I agree with that point because we even knew [...	45:35	A	T1.5.DT_2022.1.Night.157	True	False	False	I agree with that point because we even knew [...	NaN	
87	T1.5.DT_2022.1.Night.161	161	St_23_BF	We know about the people that helped Hitler, I...	45:49	E	T1.5.DT_2022.1.Night.158	True	False	False	We know about the people that helped Hitler, I...	NaN	

88 rows × 13 columns

In [ ]:

```python
data.drop(columns=['Disc.id', 'order', 'Sp.id', 'Time', 'Turn.of.Reference',
                   'Claim', 'Evidence', 'Warrant', 'claim.segment',
                   'evidence.segment', 'warrant.segment'],
          inplace=True)

data.rename(columns={"Collaboration.Code": "collabCode"},
            inplace=True)

data
```

Out[ ]:

	Talk	collabCode
0	I kind want to say that he should, although th...	N
1	But he really did not know about the Russians....	E
2	Yeah.\n	Non
3	Sorry, go ahead.\n	Non
4	Adding onto that question, I said that there i...	E
...	...	...
83	Adding to that point, I think this memoir did ...	E
84	Yeah, I think even the fact that he names othe...	E
85	I think there were even some people that we di...	E
86	I agree with that point because we even knew [...	A
87	We know about the people that helped Hitler, I...	E

88 rows × 2 columns

Collaboration codes

Below are the collaboration codes:

'Non': Non-argumentative 'N': New idea 'E': Extension 'A': Agree 'C': Challenge

We will map 0: Non, 1:N, 2:E, 3:A, and 4:C. This will be changed in the dataframe below.

In [ ]:

```python
#codes = {'Non': 0, 'N': 1, 'E': 2, 'A': 3, 'C': 4}
#data['label'] = data['Collaboration.Code'].map(codes)
#data

possible_labels = data.collabCode.unique()

label_dict = {}
for index, possible_label in enumerate(possible_labels):
    label_dict[possible_label] = index

print(label_dict)


data['label'] = data.collabCode.replace(label_dict)

print(data)
```

```
{'N': 0, 'E': 1, 'Non': 2, 'C': 3, 'A': 4}
                                                 Talk collabCode  label
0   I kind want to say that he should, although th...          N      0
1   But he really did not know about the Russians....          E      1
2                                             Yeah.\n        Non      2
3                                  Sorry, go ahead.\n        Non      2
4   Adding onto that question, I said that there i...          E      1
..                                                ...        ...    ...
83  Adding to that point, I think this memoir did ...          E      1
84  Yeah, I think even the fact that he names othe...          E      1
85  I think there were even some people that we di...          E      1
86  I agree with that point because we even knew [...          A      4
87  We know about the people that helped Hitler, I...          E      1

[88 rows x 3 columns]
```

Below, we've listed the count of each collaboration code. As we can see, "Extension" appeared the most at 36 times, followed by "New idea" at 20 times, and so on. "Agree" appeared at the lowest frequency of 2 times.

We also see that we've correctly mapped each alphabetical collaboration code to its respective numerical value.

In [ ]:

```python
print(data['collabCode'].value_counts())
print(data['label'].value_counts())
```

```
collabCode
E      36
N      20
Non    16
C      14
A       2
Name: count, dtype: int64
label
1    36
0    20
2    16
3    14
4     2
Name: count, dtype: int64
```

Train test split

Since the classes are imbalanced, we must do the train/test split in a stratified manner, meaning we will preserve the approximate proportion of each class in the splits.

As we see below, there are 2 training cases of the "A" code and 0 test cases of that code. This is because there are only 2 occurences of the "A" collab code in the whole dataset.

In [ ]:

```python
X_train, X_val, y_train, y_val = train_test_split(data.index.values,
                                                  data.label.values,
                                                  test_size=0.15,
                                                  random_state=42,
                                                  stratify=data.label.values)

data['data_type'] = ['not_set']*data.shape[0]

data.loc[X_train, 'data_type'] = 'train'
data.loc[X_val, 'data_type'] = 'val'

data.groupby(['collabCode', 'label', 'data_type']).count()
```

Out[ ]:

collabCode	label	data_type	Talk
A	**4**	**train**	2
C	**3**	**train**	12
		val	2
E	**1**	**train**	30
		val	6
N	**0**	**train**	17
		val	3
Non	**2**	**train**	13
		val	3

Tokenization

Now we will tokenize the Talk column. This means the text of each talk will be split into tokens, which are a numeric form of words.

The code below creates a Bert tokenizer, which is based on WordPiece, a subword tokenization algorithm. We will also instantiate a pre-trained Bert model configuration to encode the data. To convert all the Talks from text to encoded form, we use a function called batch_encode_plus.

In [ ]:

```python
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased',
                                          do_lower_case=True)


encoded_data_train = tokenizer.batch_encode_plus(
    data[data.data_type=='train'].Talk.values,
    add_special_tokens=True,
    return_attention_mask=True,
    pad_to_max_length=True,
    max_length=256,
    truncation=True,
    return_tensors='pt'
)



encoded_data_val = tokenizer.batch_encode_plus(
    data[data.data_type=='val'].Talk.values,
    add_special_tokens=True,
    return_attention_mask=True,
    pad_to_max_length=True,
    max_length=256,
    truncation=True,
    return_tensors='pt'
)



input_ids_train = encoded_data_train['input_ids']
attention_masks_train = encoded_data_train['attention_mask']
labels_train = torch.tensor(data[data.data_type=='train'].label.values)

input_ids_val = encoded_data_val['input_ids']
attention_masks_val = encoded_data_val['attention_mask']
labels_val = torch.tensor(data[data.data_type=='val'].label.values)

dataset_train = TensorDataset(input_ids_train, attention_masks_train, labels_train)
dataset_val = TensorDataset(input_ids_val, attention_masks_val, labels_val)
```

```
/usr/local/lib/python3.10/dist-packages/huggingface_hub/utils/_token.py:89: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in you
r Google Colab and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
```

```
/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:2760: FutureWarning: The `pad_to_max_length` argument is depreca
ted and will be removed in a future version, use `padding=True` or `padding='longest'` to pad to the longest sequence in the batch, or use `padd
ing='max_length'` to pad to a max length. In this case, you can give a specific length with `max_length` (e.g. `max_length=45`) or leave max_len
gth to None to pad to the maximal input size of the model (e.g. 512 for Bert).
  warnings.warn(
```

Bert pre-trained model

In the function below, bert-base-uncased is a smaller pre-trained model. We also input num_labels=5, as that is the number of output labels.

We are not interested in output_attentions or output_hidden_states.

In [ ]:

```python
model = BertForSequenceClassification.from_pretrained("bert-base-uncased",
                                                      num_labels=5,
                                                      output_attentions=False,
                                                      output_hidden_states=False)
```

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

Dataloader

DataLoader combines a dataset and a sampler, and provides an iterable over the given dataset.

We use RandomSampler for training and SequentialSampler for validation.

In [ ]:

```python
batch_size = 3

dataloader_train = DataLoader(dataset_train,
                              sampler=RandomSampler(dataset_train),
                              batch_size=batch_size)

dataloader_validation = DataLoader(dataset_val,
                                   sampler=SequentialSampler(dataset_val),
                                   batch_size=batch_size)
```

Optimizer and scheduler

To construct an optimizer, we pass an iterable containing the parameters to optimize. Then, we can specify optimizer-specific options such as the learning rate, epsilon, etc.

We will also create a schedule with a learning rate that decreases linearly from the initial learning rate set in the optimizer to 0, after a warmup period during which it increases linearly from 0 to the initial learning rate set in the optimizer.

I will use 5 epochs for now.

In [ ]:

```python
optimizer = AdamW(model.parameters(),
                  lr=1e-5,
                  eps=1e-8)

epochs = 5

scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps=0,
                                            num_training_steps=len(dataloader_train)*epochs)
```

```
/usr/local/lib/python3.10/dist-packages/transformers/optimization.py:591: FutureWarning: This implementation of AdamW is deprecated and will be
removed in a future version. Use the PyTorch implementation torch.optim.AdamW instead, or set `no_deprecation_warning=True` to disable this warn
ing
  warnings.warn(
```

Performance

We will use F1 score and class accuracy to evaluate performance.

In [ ]:

```python
from sklearn.metrics import f1_score

def f1_score_func(preds, labels):
    preds_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()
    return f1_score(labels_flat, preds_flat, average='weighted')

def accuracy_per_class(preds, labels):
    label_dict_inverse = {v: k for k, v in label_dict.items()}

    preds_flat = np.argmax(preds, axis=1).flatten()
    labels_flat = labels.flatten()

    for label in np.unique(labels_flat):
        y_preds = preds_flat[labels_flat==label]
        y_true = labels_flat[labels_flat==label]
        print(f'Class: {label_dict_inverse[label]}')
        print(f'Accuracy: {len(y_preds[y_preds==label])}/{len(y_true)}\n')
```

Training the model

We will train the model, setting a seed of 17, and we will use the GPU if it is available. otherwise, we'll train on CPU.

After 5 epochs, we get a

Training loss = 0.93, Validation loss = 1.198, and F1 Score = 0.482.

In [ ]:

```python
# Training

seed_val = 17
random.seed(seed_val)
np.random.seed(seed_val)
torch.manual_seed(seed_val)
torch.cuda.manual_seed_all(seed_val)


# Check if GPU is available and set device accordingly
if torch.cuda.is_available():
    device = torch.device("cuda")
    torch.cuda.manual_seed_all(seed_val)
    print('Training on GPU.')
else:
    device = torch.device("cpu")
    print('No GPU available, training on CPU.')




def evaluate(dataloader_val):

    model.eval()

    loss_val_total = 0
    predictions, true_vals = [], []

    for batch in dataloader_val:

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

        inputs = {'input_ids':      batch[0],
                  'attention_mask': batch[1],
                  'labels':         batch[2],
                 }

        with torch.no_grad():
            outputs = model(**inputs)

        loss = outputs[0]
        logits = outputs[1]
        loss_val_total += loss.item()

        logits = logits.detach().cpu().numpy()
        label_ids = inputs['labels'].cpu().numpy()
        predictions.append(logits)
        true_vals.append(label_ids)

    loss_val_avg = loss_val_total/len(dataloader_val)

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

    return loss_val_avg, predictions, true_vals

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

    model.train()

    loss_train_total = 0

    progress_bar = tqdm(dataloader_train, desc='Epoch {:1d}'.format(epoch), leave=False, disable=False)
    for batch in progress_bar:

        model.zero_grad()

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

        inputs = {'input_ids':      batch[0],
                  'attention_mask': batch[1],
                  'labels':         batch[2],
                 }

        outputs = model(**inputs)

        loss = outputs[0]
        loss_train_total += loss.item()
        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

        optimizer.step()
        scheduler.step()

        progress_bar.set_postfix({'training_loss': '{:.3f}'.format(loss.item()/len(batch))})


    # Create the directory if it doesn't exist
    os.makedirs('data_volume', exist_ok=True)

    torch.save(model.state_dict(), f'data_volume/finetuned_BERT_epoch_{epoch}.model')

    tqdm.write(f'\nEpoch {epoch}')

    loss_train_avg = loss_train_total/len(dataloader_train)
    tqdm.write(f'Training loss: {loss_train_avg}')

    val_loss, predictions, true_vals = evaluate(dataloader_validation)
    val_f1 = f1_score_func(predictions, true_vals)
    tqdm.write(f'Validation loss: {val_loss}')
    tqdm.write(f'F1 Score (Weighted): {val_f1}')
```

```
No GPU available, training on CPU.


Epoch 1
Training loss: 1.1045088028907777
Validation loss: 1.253140354156494
F1 Score (Weighted): 0.5168067226890756

Epoch 2
Training loss: 1.0811616468429566
Validation loss: 1.2385986685752868
F1 Score (Weighted): 0.5306122448979592

Epoch 3
Training loss: 1.0410297322273254
Validation loss: 1.1971393704414368
F1 Score (Weighted): 0.5168067226890756

Epoch 4
Training loss: 0.9591530156135559
Validation loss: 1.1984485387802124
F1 Score (Weighted): 0.48214285714285715

Epoch 5
Training loss: 0.9277505803108216
Validation loss: 1.1984485387802124
F1 Score (Weighted): 0.48214285714285715
```

Loading and evaluating the model

We display the class accuracies below.

In [ ]:

```python
model = BertForSequenceClassification.from_pretrained("bert-base-uncased",
                                                      num_labels=len(label_dict),
                                                      output_attentions=False,
                                                      output_hidden_states=False)

model.to(device)

model.load_state_dict(torch.load('data_volume/finetuned_BERT_epoch_1.model', map_location=torch.device('cpu')))

_, predictions, true_vals = evaluate(dataloader_validation)
accuracy_per_class(predictions, true_vals)
```

```
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-base-uncased and are newly initialized: ['c
lassifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
Class: N
Accuracy: 0/3

Class: E
Accuracy: 6/6

Class: Non
Accuracy: 3/3

Class: C
Accuracy: 0/2
```