Assignment 04 - Sentiment Analysis

We have divided the assignment into 3 steps:

1. Data Preparation (similar to what we did last time)
2. Model Training
3. Article Classification
4. Analysis

Stage 1 - Data Preparation

For preparing the data we have done something similar to last time (assignment 3) but needed to change in order to take the different requirements (5 class and not 3), so what we did is we wrote 4 different dictionary of words *pro-israel*, *pro-palestine*, *anti-israel*, *anti-palestine*

```
pro_palestine_words = [
    "resistance",
    "rights",
    "humanitarian",
    "peaceful",
    "legitimate",
    "protesters",
    "activists",
    "demonstrations",
    "supporters",
...
]

...
```

and we also changed the `extract_sentence` function from last time

```
# Classification logic
if is_pro_israeli and not (
    is_pro_palestinian or is_anti_israeli or is_anti_palestinian
    ):
    extracted.append((doc_id, sentence, "pro-israeli"))
elif is_pro_palestinian and not (
    is_pro_israeli or is_anti_israeli or is_anti_palestinian
    ):
    extracted.append((doc_id, sentence, "pro-palestinian"))
elif is_anti_israeli and not (
    is_pro_israeli or is_pro_palestinian or is_anti_palestinian
    ):
    extracted.append((doc_id, sentence, "anti-israeli"))
elif is_anti_palestinian and not (
    is_pro_israeli or is_pro_palestinian or is_anti_israeli
    ):
    extracted.append((doc_id, sentence, "anti-palestinian"))
elif not any(
    [
    is_pro_israeli,
    is_pro_palestinian,
    is_anti_israeli,
    is_anti_palestinian,
    ]
    ):
    extracted.append((doc_id, sentence, "neutral"))
```

The idea is still the same as last time but with more logical ands and ors.

This gave us a first file with 40k sentence but the problem was the distribution of each class so we did an analysis to see the distribution and saw: (the file is `analysis.py`)

```
Sentiment Class Distribution:
------------------------------
neutral: 38421
pro-palestinian: 2186
pro-israeli: 2002
anti-israeli: 1657
anti-palestinian: 1222

Percentages:
------------------------------
neutral: 84.46%
pro-palestinian: 4.81%
pro-israeli: 4.40%
anti-israeli: 3.64%
anti-palestinian: 2.69%
```

So since there was so much of the `neutral` class I wrote a script to take a random subset from this class to balance out the distribution + added `int` labels to the data frame and got this

```
Balanced Dataset Distribution:
------------------------------
pro-palestinian: 2186
pro-israeli: 2002
anti-israeli: 1657
neutral: 1500
anti-palestinian: 1222

Percentages:
------------------------------
```

```
pro-palestinian: 25.52%
pro-israeli: 23.37%
anti-israeli: 19.34%
neutral: 17.51%
anti-palestinian: 14.26%
```

Which is better.

Stage 2- Model Training

For the training of this model we used transfer learning from a pre-trained base **BERT** model.

The first step was to take a subset of 500 random instances from each class (to make the training easier) and then we used the `transformers` library to load the pre-trained model and tokenizer

We also spilt the data 85% - 15% for training and validation.

The following code is the `SentimentClassifier` class that we used to train the model

```
# Create model class

class SentimentClassifier(nn.Module):
    def __init__(self, n_classes=5):
        super().__init__()
        self.bert = DistilBertModel.from_pretrained('distilbert-base-uncased')
        self.drop = nn.Dropout(0.3)
        self.fc = nn.Linear(self.bert.config.hidden_size, n_classes)

    def forward(self, input_ids, attention_mask):
        output = self.bert(
        input_ids=input_ids,
        attention_mask=attention_mask
        )
        output = self.drop(output[0][:, 0, :])
        return self.fc(output)
```

The following code is the `train_model` function that we used to train the model

```
# Training loop
def train_model():
    model.train()
    total_loss = 0
    for batch in tqdm(train_loader, desc='Training'):
        optimizer.zero_grad()
        input_ids = batch['input_ids'].to(device)
        attention_mask = batch['attention_mask'].to(device)
        labels = batch['label'].to(device)
        outputs = model(input_ids=input_ids, attention_mask=attention_mask)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        total_loss += loss.item()
    return total_loss / len(train_loader)
```

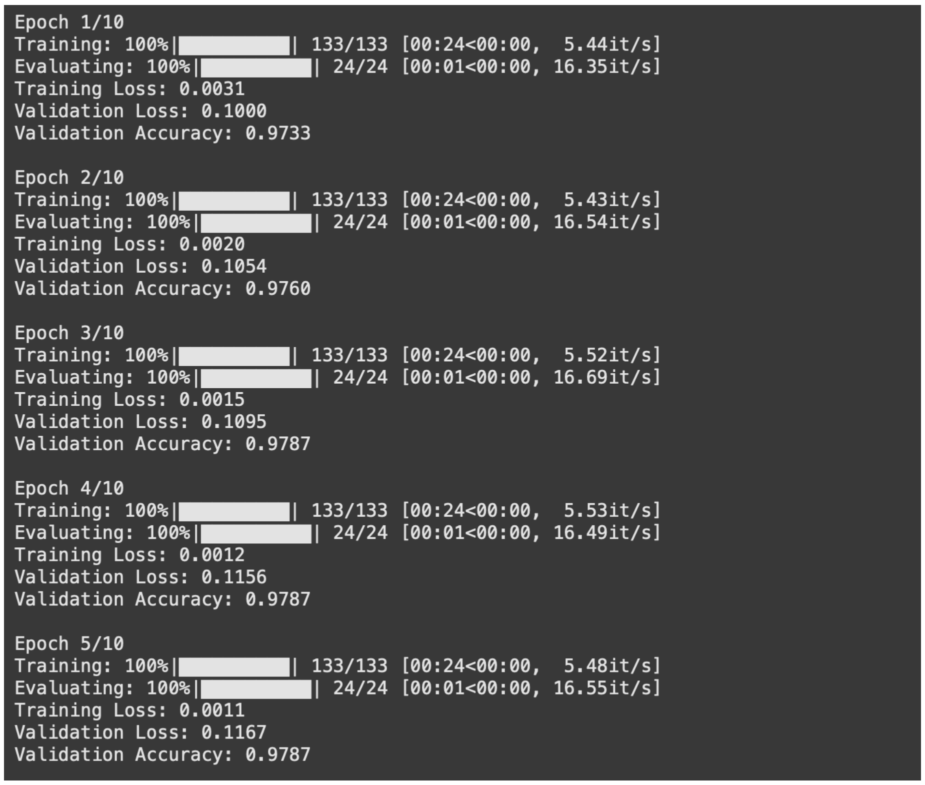
The following code is the `evaluate_model` function that we used to evaluate the model

```
# Evaluation loop
def evaluate_model():
    model.eval()
    total_loss = 0
    all_predictions = []
    all_labels = []
    with torch.no_grad():
        for batch in tqdm(val_loader, desc='Evaluating'):
            input_ids = batch['input_ids'].to(device)
            attention_mask = batch['attention_mask'].to(device)
            labels = batch['label'].to(device)
            outputs = model(input_ids=input_ids, attention_mask=attention_mask)
            loss = criterion(outputs, labels)
            _, predictions = torch.max(outputs, dim=1)
            total_loss += loss.item()
            all_predictions.extend(predictions.cpu().numpy())
            all_labels.extend(labels.cpu().numpy())
    accuracy = np.mean(np.array(all_predictions) == np.array(all_labels))
    return total_loss / len(val_loader), accuracy
```

The following code is the training loop

```
# Training
for epoch in range(n_epochs):
    print(f'\nEpoch {epoch + 1}/{n_epochs}')
    train_loss = train_model()
    val_loss, val_accuracy = evaluate_model()
    print(f'Training Loss: {train_loss:.4f}')
    print(f'Validation Loss: {val_loss:.4f}')
    print(f'Validation Accuracy: {val_accuracy:.4f}')
```

The screenshot from the training process



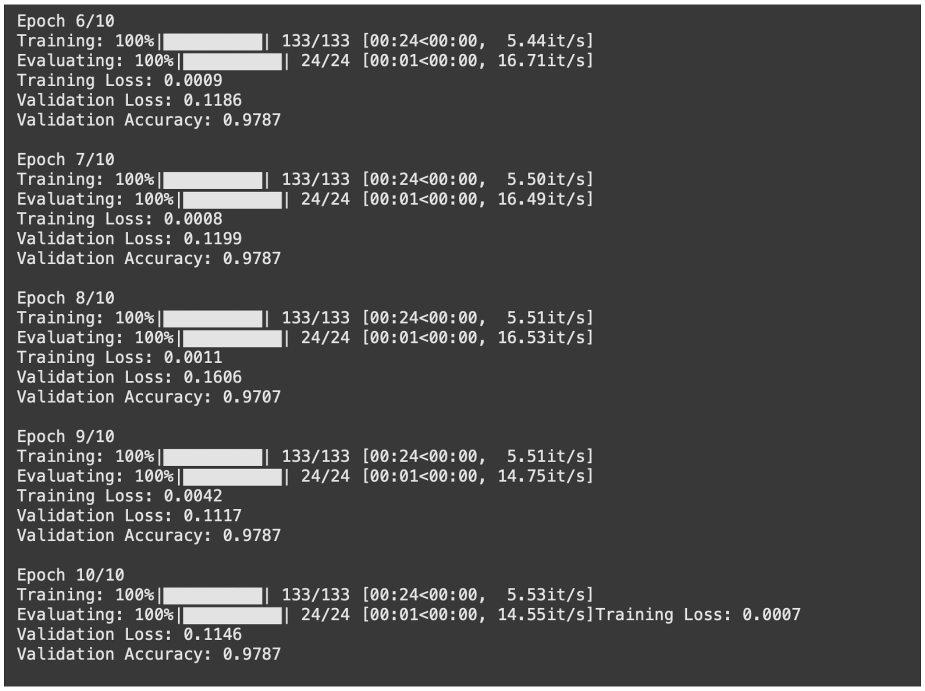
```
Epoch 1/10
Training: 100%|██████████| 133/133 [00:24<00:00,  5.44it/s]
Evaluating: 100%|██████████| 24/24 [00:01<00:00, 16.35it/s]
Training Loss: 0.0031
Validation Loss: 0.1000
Validation Accuracy: 0.9733

Epoch 2/10
Training: 100%|██████████| 133/133 [00:24<00:00,  5.43it/s]
Evaluating: 100%|██████████| 24/24 [00:01<00:00, 16.54it/s]
Training Loss: 0.0020
Validation Loss: 0.1054
Validation Accuracy: 0.9760

Epoch 3/10
Training: 100%|██████████| 133/133 [00:24<00:00,  5.52it/s]
Evaluating: 100%|██████████| 24/24 [00:01<00:00, 16.69it/s]
Training Loss: 0.0015
Validation Loss: 0.1095
Validation Accuracy: 0.9787

Epoch 4/10
Training: 100%|██████████| 133/133 [00:24<00:00,  5.53it/s]
Evaluating: 100%|██████████| 24/24 [00:01<00:00, 16.49it/s]
Training Loss: 0.0012
Validation Loss: 0.1156
Validation Accuracy: 0.9787

Epoch 5/10
Training: 100%|██████████| 133/133 [00:24<00:00,  5.48it/s]
Evaluating: 100%|██████████| 24/24 [00:01<00:00, 16.55it/s]
Training Loss: 0.0011
Validation Loss: 0.1167
Validation Accuracy: 0.9787
```



```
Epoch 6/10
Training: 100%|██████████| 133/133 [00:24<00:00,  5.44it/s]
Evaluating: 100%|██████████| 24/24 [00:01<00:00, 16.71it/s]
Training Loss: 0.0009
Validation Loss: 0.1186
Validation Accuracy: 0.9787

Epoch 7/10
Training: 100%|██████████| 133/133 [00:24<00:00,  5.50it/s]
Evaluating: 100%|██████████| 24/24 [00:01<00:00, 16.49it/s]
Training Loss: 0.0008
Validation Loss: 0.1199
Validation Accuracy: 0.9787

Epoch 8/10
Training: 100%|██████████| 133/133 [00:24<00:00,  5.51it/s]
Evaluating: 100%|██████████| 24/24 [00:01<00:00, 16.53it/s]
Training Loss: 0.0011
Validation Loss: 0.1606
Validation Accuracy: 0.9707

Epoch 9/10
Training: 100%|██████████| 133/133 [00:24<00:00,  5.51it/s]
Evaluating: 100%|██████████| 24/24 [00:01<00:00, 14.75it/s]
Training Loss: 0.0042
Validation Loss: 0.1117
Validation Accuracy: 0.9787

Epoch 10/10
Training: 100%|██████████| 133/133 [00:24<00:00,  5.53it/s]
Evaluating: 100%|██████████| 24/24 [00:01<00:00, 14.55it/s]Training Loss: 0.0007
Validation Loss: 0.1146
Validation Accuracy: 0.9787
```

As we can see after the final epoch the model has:

- Validation Loss: 0.1146
- Validation Accuracy: 0.9787

Meaning our model is accurate 97.87% of the time.

Stage 3 - Article Classification

First to classify the articles we needed to write a function that uses the model and give out a tuple of the probabilities vectors and the string representation of the class.

```python
def classify_sentence(sentence):
    # Tokenize the sentence
    inputs = tokenizer(sentence, return_tensors='pt', truncation=True,
     padding=True, max_length=512)
    inputs = {k: v.to(device) for k, v in inputs.items()}
    # Get the model's output
    with torch.no_grad():
        logits = model(inputs['input_ids'], inputs['attention_mask'])
    # Apply softmax to get probabilities
    probabilities = torch.nn.functional.softmax(logits, dim=1)
    # get the str of the calss based on the one-hot-encoded vector
    predicted_class = torch.argmax(probabilities, dim=1).item()
    # Create a one-hot encoded vector
    one_hot_vector = torch.zeros(probabilities.size(1))
    one_hot_vector[predicted_class] = 1
    map_class = {0: 'pro-israeli', 1: 'pro-palestinan',
        2: 'neutral', 3: 'anti-isreali', 4: 'anti-palestinian'}

    return probabilities, one_hot_to_class(one_hot_vector, map_class)
```

This function is taking a string sentence and returning the probabilities vector and the string representation of the class by passing the sentence into the model and getting back the last layer which is the logits and then applying softmax to get the probabilities and then getting the class by taking the argmax of the probabilities vector.

Then we needed to get the original article and separated them into sentences like so:

```python
# get the data from the github repository
aj_url = "https://github.com/dattali18/IR_Assignments/blob/main/Assignment.01/data/word/A_J_
bbc_url = "https://github.com/dattali18/IR_Assignments/blob/main/Assignment.01/data/word/BBC
jp_url = "https://github.com/dattali18/IR_Assignments/blob/main/Assignment.01/data/word/J_P_
nyt_url = "https://github.com/dattali18/IR_Assignments/blob/main/Assignment.01/data/word/NYT

import pandas as pd

# load the data
aj_df = pd.read_csv(aj_url)
bbc_df = pd.read_csv(bbc_url)
jp_df = pd.read_csv(jp_url)
nyt_df = pd.read_csv(nyt_url)


def clean_text(text):
    # Normalize all types of single and double
    # quotation marks to standard forms
    text = re.sub(r"[‘’`]", "'", text)
    # Convert all single quote variations to '
    text = re.sub(r"[""]", '"', text)
    # Convert all double quote variations to "

    # remove any and all special characters
    # since it will not be useful for our analysis
    text = re.sub(r"[^a-zA-Z0-9\s]", "", text)

    return text
```

```python
def extract_all_sentences(df):
    # this will return a dict with key the id of the article "aj_1" for example
    # and a list of all the sentences in the article
    all_sentences = []
    for index, row in df.iterrows():
        text = row["document"]
        # TODO - ask gpt for a smarter sentence extratctor
        sentences = re.split(r"[.!?]", text)
        sentences = [sentence for sentence in sentences if sentence != ""]
        # clean the sentences
        sentences = [clean_text(sentence) for sentence in sentences]

    # for all sentence in sentences add to df
    for sentence in sentences:
        all_sentences.append({"id": row["id"], "document": sentence})
    return all_sentences
```

Then we will use those function to extract all the sentences from each journal.

```python
aj_sentences = extract_all_sentences(aj_df)
bbc_sentences = extract_all_sentences(bbc_df)
jp_sentences = extract_all_sentences(jp_df)
nyt_sentences = extract_all_sentences(nyt_df)

aj_df = pd.DataFrame(aj_sentences)
bbc_df = pd.DataFrame(bbc_sentences)
jp_df = pd.DataFrame(jp_sentences)
nyt_df = pd.DataFrame(nyt_sentences)

df = pd.DataFrame(columns=["id", "document", "pro-israeli",
                           "pro-palestinan", "neutral", "anti-isreali",
                            "anti-palestinian", "majority_class"])

df = pd.concat([df, aj_df], ignore_index=True)
df = pd.concat([df, bbc_df], ignore_index=True)
df = pd.concat([df, jp_df], ignore_index=True)
df = pd.concat([df, nyt_df], ignore_index=True)



df[["pro-israeli", "pro-palestinan",
    "neutral", "anti-isreali", "anti-palestinian"]] = 0
df['majority_class'] = ''

df = df.rename(columns={"document": "sentence"})
```

Now that we have all we need, we will use all the function above to loop over all the sentences in every article and classify them and then get the majority class of the article.

```python
# loop through every items in df
for index, row in df.iterrows():
    sentence = row['sentence']
    probs, cls = classify_sentence(sentence)
    # put the values in the df
    df.at[index, 'majority_class'] = cls
    # unpack the values in probs (len 5) to the 5 classes of ["pro-israeli", "pro-palestinan

    # map index to key
    map_class = {0: 'pro-israeli', 1: 'pro-palestinan',
                 2: 'neutral', 3: 'anti-isreali', 4: 'anti-palestinian'}

    for i in range(5):
        df.at[index, map_class[i]] = probs[0][i].item()
        # print at interval of 100 indexs
        if index % 100 == 0:
            print(f"processing index {index}")
```

And finally, we will save the data frame to a csv file.

```python
df.to_csv("sentences_with_class.csv", index=False)
```

Now that we have all the sentences with their classes we can do the analysis.

Stage 4 - Analysis