**Practical Natural Language Processing**
**Instructor: Rasika Bhalerao**
**Assignment 7**
**Due November 23** (at the beginning of class at 2pm)

Please submit your solutions as a pdf using NYU Classes. For this assignment, it is recommended to use Google Colab. You can print your solutions as a pdf and submit them the same way you would with a Jupyter notebook. (Please don't submit them as a link to the notebook in Google drive; the notebook can change unintentionally, and there is also the risk of permissions blocking us.)

Create a new notebook in Google Colab by going to https://colab.research.google.com/ and clicking on "New Notebook." Add a GPU by going to Edit → Notebook settings → Hardware accelerator → GPU. If resources are low at the time you try this, you may need to continue writing your code without a GPU and then come back for the GPU to run it later. We will write the code such that it will automatically detect if there is a GPU or not, and then choose to run on the GPU or a CPU. (The GPU is faster.)

In this assignment, we will fine tune BERT to perform text classification. We will do this by taking a BERT model pretrained by Huggingface, adding a layer to the end, and training the modified model to do text classification.

Note: For help with the functions used here, I recommend going directly to Huggingface's documentation, including these two pages: https://huggingface.co/transformers/training.html and https://huggingface.co/transformers/custom_datasets.html, rather than a separate tutorial. While most tutorials work, some include fatal flaws such as training on the test set.

Huggingface uses Pytorch (and Tensorflow, but we will use Pytorch). Import it like this:

```
import torch
```

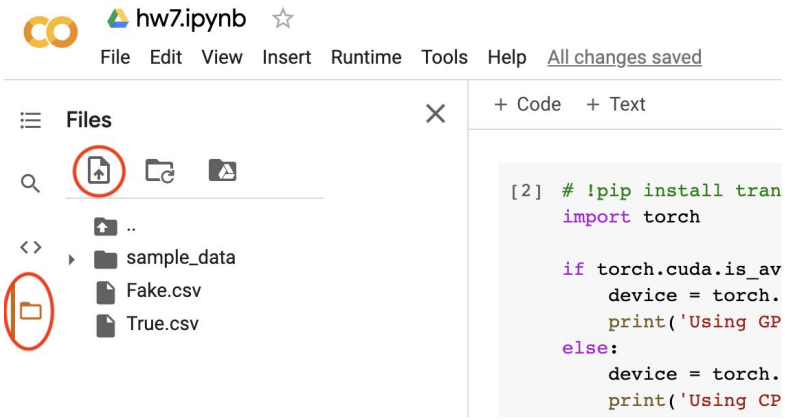
Let's detect if there is a GPU or not, and store the device in a variable called `device` so that we can use it to run things later.

```
if torch.cuda.is_available():
    device = torch.device("cuda")
    print('Using GPU ', torch.cuda.get_device_name(0))
else:
    device = torch.device("cpu")
    print('Using CPU')
```

We will use the transformers package from Huggingface.

```
!pip install transformers
```

We will use the same dataset as from assignments 2 and 6. Put the dataset in session storage:




You can then import pandas and manipulate the data the same way as in Jupyter notebook.

We will use the tokenizer and pretrained BERT model from Huggingface:

```
from transformers import BertTokenizer, BertForSequenceClassification

tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = BertForSequenceClassification.from_pretrained('bert-base-uncased', return_dict=True)
```

They are “uncased,” so they will ignore case. The warning about uninitialized weights is expected.

Make sure to put the model on the correct `device`, and also put it in training mode:

```
model.to(device)
model.train()
```

You can test out the tokenizer with this (modified) example from Huggingface:

```
text_batch = ["I love Pixar.",
              "I don't care for Pixar.",
              "This is such a super duper long sentence with so many words you can barely understand it oh my gosh"]
encoding = tokenizer(text_batch, return_tensors='pt', padding=True, truncation=True)
input_ids = encoding['input_ids'].to(device)
attention_mask = encoding['attention_mask'].to(device)
```

Look at the contents of `input_ids` and `attention_mask`. You can see that shorter sentences have been padded with 0s at the end, and all tensors are the length of the longest sentence. BERT has a maximum length of 512 tokens. For our task, we will be using entire documents instead of sentences, and documents longer than 512 will be truncated.

To make sure all tensors are on the correct device, use `.to(device)` for every tensor. Pytorch will complain if you try computation with one tensor on a CPU and another on a GPU. Keep this in mind for your tensor of labels (1 or 0 for real news or fake news).

We can now pass the encoded tokens to the language model.

```
outputs = model(input_ids, attention_mask=attention_mask)
```

`outputs` should be an object with a field for `logits`, the outputted logits. `outputs.logits` is a tensor with a row for each sentence in the input. Since we are doing binary classification, each row has two elements that add up to 1: the probabilities for outputting 0 and 1.

We will use cross entropy loss. In the example below, `input` and `target` are tensors. Each row of `input` represents the output probabilities for a sentence (`outputs.logits`). `target` should be a tensor with the label for each sentence, 0 or 1.

```
from torch.nn import functional as F
loss = F.cross_entropy(input, target)
```

The resulting `loss` variable should contain a tensor with a single value and a gradient.

To update the weights during training, we will use an optimizer. A popular one is AdamW. We need to pass references to the language model parameters and a learning rate.

```
from transformers import AdamW
optimizer = AdamW(model.parameters(), lr=1e-5)
```

Pytorch is nice because it can automatically do the backwards pass for us. To calculate the gradients, call:

```
loss.backward()
```

Then call `optimizer.step()` to update the weights in the language model.

The assignment is split into “parts” below, but they should all be answered at once in the same code. There is no need for separate code snippets for the different parts. Please do include comments to explain your code.

1. Most would not consider our dataset small, but we will practice what to do if we don’t have much data. You may use Scikit-learn’s KFold for this. Perform KFold cross validation to finetune the language model to classify each document as real or fake news. Use it to tune the hyperparameter for learning rate. You will need to first split the data into a training set and test set. Then, do the KFold cross validation on the training set. Use K = 5 folds. For each fold, four fifths of the training dataset is the “training set” and the remaining fifth is the “dev set” on which to evaluate the model trained during that fold. Rotate so that in each fold, a different fifth has a chance to be the “dev set.” Pick three different values for the learning rate. Test each value of the learning rate hyperparameter on each fold. Take the learning rate that did the best on any fold (lowest average loss per document) and claim it as your ultimate value for learning rate.
2. Each time you train the model, you need to choose a number of epochs. Pick a maximum number of epochs such as 20, and implement early stopping: if the loss does not decrease for 4 epochs in a row, stop training.
3. Use mini-batches with a batch size of 32. It may help to use the built-in DataLoader (`from torch.utils.data import DataLoader`).
4. Evaluate your overall model on the test set that you set aside in step 1. Train your model on the whole training set using your ultimate learning rate hyperparameter value. On the test set, report the precision, recall, and F1 score.

If you did this correctly, questions 1-3 should result in loops nested within each other, processing the training set. The test set should only be touched after the loops, for question 4.