# Question Answering with Huggingface Transformers

This notebook uses the shared drive: QA_Model_Drive

This shared drive has been shared with the professor

## Install Huggingface Transformers

```
# Contributions: Luke and Katie
#install huggingface transformers

!pip install transformers
```

```
# Contributions Luke and Katie

# Import os for file access
import os

# Import skleanr's train_test_split for splitting train and test data
from sklearn.model_selection import train_test_split

#Import Huggingface Transformers' DistilBert sequence classified, Trainer, and Training Args for simplified training
from transformers import DistilBertTokenizerFast
bertTokenizer = DistilBertTokenizerFast.from_pretrained('distilbert-base-uncased')

#Import Huggingface Transformers' DistilBert sequence classified, Trainer, and Training Args for simplified training
from transformers import DistilBertForSequenceClassification, Trainer, TrainingArguments

#Import PyTorch for preparing the data as PoeShakespeareDataset objects
import torch

#Import sklearn's accuracy score metrics for computing accuracy
from sklearn.metrics import accuracy_score
```

```
# Contributions: Luke and Katie

# Import our data from google drive

from google.colab import drive
drive.mount('/content/drive')
```

## Preparing the data

```
# Conrtibutions: Katie and Luke

# Get list of tests for both authors using paths

!cd ../
poe_path = 'drive/Shareddrives/QA_Model_Drive/QA_Model_Txt_Data/Poe/'
```

```
poe_path = 'drive/Shareddrives/QA_Model_Drive/QA_Model_Txt_Data/Poe/'
shakespeare_path = 'drive/Shareddrives/QA_Model_Drive/QA_Model_Txt_Data/Shakespeare/'

poe_text_names = os.listdir(poe_path)
shakespeare_text_names = os.listdir(shakespeare_path)
```

## Text Processing

```
# Contributions: Katie and Luke

# Create lists[text][sentence]

poe_texts = []
shakespeare_texts = []

# poe_texts[#texts][#sentences in texts[i]]

for poe_text_idx in range(len(poe_text_names)):
  p_file = open(poe_path + poe_text_names[poe_text_idx], 'r')
  poe_texts = poe_texts + p_file.readlines()

# shakespeare_texts[#texts][#sentences in texts[i]]

for shakespeare_text_idx in range(len(shakespeare_text_names)):
  s_file = open(shakespeare_path + shakespeare_text_names[shakespeare_text_idx], 'r')
  shakespeare_texts = shakespeare_texts + s_file.readlines()


# Contributions: Luke

# Run this block ONLY ONCE
# Get 1d list of Poe sentences and 1d list of Shakespeare sentences
one_col_poes = []
one_col_shakespeare = []
poe_labels = []
shakespeare_labels = []

for text in range(len(poe_texts)):
  clean_line = poe_texts[text].rstrip('\n')
  split_lines_period = clean_line.split('.')

  # split on '?'
  split_lines_question = []
  for x in split_lines_period:
    split_lines_question = split_lines_question + x.split('?')

  # split on '!'
  split_lines_complete = []
  for x in split_lines_question:
    split_lines_complete = split_lines_complete + x.split('!')

  if len(clean_line) > 0:
    no_garbage_rows = [x for x in split_lines_complete if len(x) is not 0]
    one_col_poes = one_col_poes + no_garbage_rows
    for i in range(len(no_garbage_rows)):
      poe_labels.append(0)
poe_texts = one_col_poes



for text in range(len(shakespeare_texts)):
  clean_line = shakespeare_texts[text].rstrip('\n')
```

```
  clean_line = shakespeare_texts[text].rstrip('\n')
  split_lines = clean_line.split('.')
  if len(clean_line) > 0:
    no_garbage_rows = [x for x in split_lines if len(x) is not 0]
    one_col_shakespeare = one_col_shakespeare + no_garbage_rows
    for i in range(len(no_garbage_rows)):
      shakespeare_labels.append(1)
shakespeare_texts = one_col_shakespeare


# Contributions: Luke

print(len(poe_texts), len(poe_labels), len(shakespeare_texts), len(shakespeare_labels))

max = 0

for x in poe_texts:
  if len(x) > max:
    max = len(x)

max
```

```
2523 2523 3955 3955
77
```

## Split Data

```
# Contributions: Luke and Katie

# Split train and test data and labels for encoding

poe_train_data, poe_test_data = train_test_split(poe_texts, test_size=0.25, random_state=7, shuffle=True)
poe_train_labels, poe_test_labels = train_test_split(poe_labels, test_size=0.25, random_state=7, shuffle=True)
shakespeare_train_data, shakespeare_test_data = train_test_split(shakespeare_texts, test_size=0.25, random_state=7, shuffle=True)
shakespeare_train_labels, shakespeare_test_labels = train_test_split(shakespeare_labels, test_size=0.25, random_state=7, shuffle=T

poe_shakespeare_train_labels = poe_train_labels + shakespeare_train_labels
poe_shakespeare_test_labels = poe_test_labels + shakespeare_test_labels

# Combining the poe and shakespeare data for encoding.

poe_shakespeare_train = poe_train_data + shakespeare_train_data
poe_shakespeare_test = poe_test_data + shakespeare_test_data


# Contributions: Luke and Katie

# Create validation set we can use for evaluation and tuning without tainting test set results.

train_texts, val_texts, train_labels, val_labels = train_test_split(poe_shakespeare_train, poe_shakespeare_train_labels, test_size
```

## Tokenize data

```
# Contributions: Luke and Katie

# Sources: https://huggingface.co/transformers/training.html
```

```
# Tokenize training, validation, and test data

train_encodings = bertTokenizer(train_texts, truncation=True, padding=True)
val_encodings = bertTokenizer(val_texts, truncation=True, padding=True)
test_encodings = bertTokenizer(poe_shakespeare_test, truncation=True, padding=True)

# Added these for more testing with the pipelines. Seems like we needed them separated.
#poe_encodings_for_pipeline_testing = bertTokenizer(poe_test_data, truncation=True, padding=True)
#shakespeare_encodings_for_pipeline_testing = bertTokenizer(shakespeare_test_data, truncation=True, padding=True)



# Contributions: Katie and Kyler

# Sources: https://huggingface.co/transformers/training.html

class PoeShakespeareDataset(torch.utils.data.Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {key: torch.tensor(val[idx]) for key, val in self.encodings.items()}
        item['labels'] = torch.tensor(self.labels[idx])
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = PoeShakespeareDataset(train_encodings, train_labels)
val_dataset = PoeShakespeareDataset(val_encodings, val_labels)
test_dataset = PoeShakespeareDataset(test_encodings, poe_shakespeare_test_labels)



# Contributions: Matthew

# Sources: https://www.thepythoncode.com/article/finetuning-bert-using-huggingface-transformers-python

def compute_metrics(pred):
  labels = pred.label_ids
  preds = pred.predictions.argmax(-1)
  # calculate accuracy using sklearn's function
  acc = accuracy_score(labels, preds)
  return {
      'accuracy': acc,
  }
```

# Training or Loading the Model

```
# Contributions: Matthew, Kyler and Ryan

# Sources: https://huggingface.co/transformers/training.html

# Load pretrained model if there is one. If not, train new model.

model = None

def checkForSavedModel():
  global model
  if len(os.listdir("/content/drive/Shareddrives/QA_Model_Drive/Saved_Models/")) == 0:
    print("Training new model...")
```

```
    training_args = TrainingArguments(
        output_dir='./results',          # output directory
        num_train_epochs=10,             # total number of training epochs
        per_device_train_batch_size=16,  # batch size per device during training
        per_device_eval_batch_size=64,   # batch size for evaluation
        warmup_steps=500,                # number of warmup steps for learning rate scheduler
        weight_decay=1.0,                # strength of weight decay (increased to 1.0 from 0.01 to generalize)
        logging_dir='./logs',            # directory for storing logs
        logging_steps=100,               # Using a larger value to reduce runtime/disk usage
        learning_rate=0.000001,          # decreased to 10^-6 from 5x10^-5 in order to generalize
        load_best_model_at_end=True,     # saves the best model once training is finished
        evaluation_strategy="steps",     # displays validation loss in the results table
    )

    model = DistilBertForSequenceClassification.from_pretrained("distilbert-base-uncased")

    trainer = Trainer(
        model=model,                         # the instantiated 🤗 Transformers model to be trained
        args=training_args,                  # training arguments, defined above
        train_dataset=train_dataset,         # training dataset
        eval_dataset=val_dataset,            # evaluation dataset
        compute_metrics=compute_metrics
    )

    trainer.train()

    model.save_pretrained("/content/drive/Shareddrives/QA_Model_Drive/Saved_Models")
  else:
    print("Loading Pre-Saved Model...")
    model = DistilBertForSequenceClassification.from_pretrained("/content/drive/Shareddrives/QA_Model_Drive/Saved_Models/")
    print("Loaded")
checkForSavedModel()
```

```
Loading Pre-Saved Model...
Loaded
```

# Testing the Model

```
# Contributions: Katie and Ryan

#https://huggingface.co/transformers/usage.html
#https://stackoverflow.com/questions/64914598/pytorch-runtimeerror-input-output-and-indices-must-be-on-the-current-device
# edgar allan poe = 0
# shakespeare = 1

model.eval()
model.to("cpu")
classes = [0, 1]

def testModel(sequence, label):
  lineTokenized = bertTokenizer.encode_plus(sequence, return_tensors="pt")

  classifiedLogits = model(**lineTokenized)[0]

  results = torch.softmax(classifiedLogits, dim=1).tolist()[0]


  #uncomment for verbose output

  #print("Should be {label} ({author})".format(label=label, author= "Poe" if label is 0 else "Shakespeare"))
```

```
  #for i in range(len(classes)):
      #print(f"{classes[i]}: {round(results[i] * 100)}%")
  if label==0:
    return round(results[0] * 100)
  else:
    return round(results[1] * 100)


# Contributions: Matthew

# Run tests on test sentences.
# Uncomment the print statements to see the individual results on each sentence

poe_average_correct = 0
shakes_average_correct = 0
poe_count = 0
shakes_count = 0

for poe_sequence in poe_test_data:
  #print(poe_sequence)
  if poe_count < 2:
    poe_count += 1
  poe_average_correct = (poe_average_correct + testModel(poe_sequence, 0)) / poe_count

for shakespeare_sequence in shakespeare_test_data:
  #print(shakespeare_sequence)
  if shakes_count < 2:
    shakes_count += 1
  shakes_average_correct = (shakes_average_correct + testModel(shakespeare_sequence, 1)) / shakes_count

print("\nPoe average: ", poe_average_correct, "%\nShakespeare average: ", shakes_average_correct, "%\n")

print(len(poe_test_data))
print(len(shakespeare_test_data))
```

```
Poe average:  54.64731653532235 %
Shakespeare average:  92.88797187670198 %

631
989
```

## UI For Using the Model

```
# Contributions to this block: Ryan and Katie

classes = [0, 1]

def classify(sentence):
  lineTokenized = bertTokenizer.encode_plus(sentence, return_tensors="pt")
  classifiedLogits = model(**lineTokenized)[0]
  results = torch.softmax(classifiedLogits, dim=1).tolist()[0]
  return results

def printAnswer(results):
  guess = "Poe" if (round(results[0] * 100)) > (round(results[1] * 100)) else "Shakespeare" #written by Katie
  print(f"I think {guess} wrote this sentence.")
  print(f"Poe: {round(results[0] * 100)}%")
  print(f"Shakespeare: {round(results[1] * 100)}%")
```

```
def printIntro():
  print("Hello, and welcome to...\n")
  printTitle()
  print("(Enter a blank question to exit)\n\n")

def printOutro():
  print("Thanks for using...\n")
  printTitle()

def printTitle():
  print("Who Said It?")
  print("Edgar Allen Poe vs Shakespeare!\n")

def main():
  printIntro()
  sentence = None
  while sentence != "":
    print("\nQuestion:")
    sentence = input("Who wrote the following sentence: ");
    if sentence != "":
      print("\nThinking...");
      results = classify(sentence);
      printAnswer(results)
    print("\n")
  printOutro()


# Contributions: Ryan

#make sure the model is in evaluation mode
model.eval()

#make sure the model is using the cpu for testing the model
model.to("cpu")


main()
```

```
Hello, and welcome to...

Who Said It?
Edgar Allen Poe vs Shakespeare!

(Enter a blank question to exit)



Question:
Who wrote the following sentence: How many memories of what radiant hours

Thinking...
I think Poe wrote this sentence.
Poe: 66%
Shakespeare: 34%



Question:
Who wrote the following sentence:     Jealous of catching, swiftly doth forsake him,

Thinking...
I think Shakespeare wrote this sentence.
```

```
Poe: 4%
Shakespeare: 96%



Question:
Who wrote the following sentence:


Thanks for using...

Who Said It?
Edgar Allen Poe vs Shakespeare!
```