Import Libraries

```
!pip install transformers
```

```
Collecting transformers
  Downloading transformers-4.34.1-py3-none-any.whl (7.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 7.7/7.7 MB 20.5 MB/s eta 0:00:00
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from transformers) (3.12.4)
Collecting huggingface-hub<1.0,>=0.16.4 (from transformers)
  Downloading huggingface_hub-0.18.0-py3-none-any.whl (301 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 302.0/302.0 kB 31.4 MB/s eta 0:00:00
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (1.23.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/dist-packages (from transformers) (23.2)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.10/dist-packages (from transformers) (6.0.1)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.10/dist-packages (from transformers) (2023.6.3)
Requirement already satisfied: requests in /usr/local/lib/python3.10/dist-packages (from transformers) (2.31.0)
Collecting tokenizers<0.15,>=0.14 (from transformers)
  Downloading tokenizers-0.14.1-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (3.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 3.8/3.8 MB 52.6 MB/s eta 0:00:00
Collecting safetensors>=0.3.1 (from transformers)
  Downloading safetensors-0.4.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.3/1.3 MB 53.3 MB/s eta 0:00:00
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.10/dist-packages (from transformers) (4.66.1)
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transformers) (202
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.10/dist-packages (from huggingface-hub<1.0,>=0.16.4->transfor
Collecting huggingface-hub<1.0,>=0.16.4 (from transformers)
  Downloading huggingface_hub-0.17.3-py3-none-any.whl (295 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 295.0/295.0 kB 39.0 MB/s eta 0:00:00
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.3.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2.0.6)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests->transformers) (2023.7.22)
Installing collected packages: safetensors, huggingface-hub, tokenizers, transformers
Successfully installed huggingface-hub-0.17.3 safetensors-0.4.0 tokenizers-0.14.1 transformers-4.34.1
```

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import cross_val_score
import torch
import transformers as ppb
import warnings
```

```
warnings.filterwarnings('ignore')
import torch
from torch.utils.data import TensorDataset, DataLoader
```

Loading dataset

```
import pandas as pd

#data = pd.read_csv('/content/drive/MyDrive/Roman Urdu DataSet.csv')
data = pd.read_csv('/content/Roman Urdu DataSet.csv')

# Assuming your dataset has two columns: 'text' for the text data and 'label' for the sentiment labels
# Display the first few rows to verify it loaded correctly
data.head(10)

data = data.dropna()
```

Removing 2nd column

```
data.drop(["Unnamed: 2"],
          axis=1,
          inplace=True)

data.head(10)
```

	Sai kha ya her kisi kay bus ki bat nhi hai lakin main ki hal kal bi Aj aur aj bi sirf Aus say bus	Positive
13636	movie abi b baki h	Neutral
13652	Hahahahahaha bilkul sahi	Neutral
14217	tjhe ase mar na chahti hun tjhe nae tu achi b...	Negative
14809	Yr tym pta chali kb ata raat m?	Positive
17160	Kya khatab g ledy type ka sahafi la k betha diya	Negative
19498	kabhi bhai ki bhi aesi pic lele :P	Neutral
19779	Jahil awam ko jahil leader ki hi zroorat hai, ...	Negative

Loading text data and their corresponding labels

```
# Define column names
column_names = ['text', 'label']

# Assign the column names to the DataFrame
data.columns = column_names


sentences = data['text'].tolist()
labels = data['label'].tolist()
print(labels)
print(sentences)
```

```
['Neutral', 'Neutral', 'Negative', 'Positive', 'Negative', 'Neutral', 'Negative']
['movie abi b baki h ', 'Hahahahahaha bilkul sahi ', 'tjhe ase mar na chahti hun  tjhe nae tu achi bachi hy', 'Yr tym pta chali kb ata raat m?',
```

BERT Model

```
# Want BERT instead of distilBERT? Uncomment the following line:
model_class, tokenizer_class, pretrained_weights = (ppb.BertModel, ppb.BertTokenizer, 'bert-base-uncased')

# Load pretrained model/tokenizer
tokenizer = tokenizer_class.from_pretrained(pretrained_weights)
model = model_class.from_pretrained(pretrained_weights)
```

BERT pretrain

```
tokenized = [tokenizer.encode(sentence, add_special_tokens=True) for sentence in sentences]
max_len = max(map(len, tokenized))
padded = np.array([i + [0]*(max_len-len(i)) for i in tokenized])
attention_mask = np.where(padded != 0, 1, 0)
```

Converting Labels and Creating PyTorch Dataset

```
# Define a mapping from label strings to numerical values
label_mapping = {'Positive': 0, 'Negative': 1, 'Neutral': 2}

# Convert labels to numerical values
numerical_labels = [label_mapping[label] for label in labels]

# Convert the data to PyTorch tensors
input_ids = torch.tensor(padded)
attention_mask = torch.tensor(attention_mask)
labels = torch.tensor(numerical_labels)  # Use the converted numerical labels

# Create a TensorDataset
dataset = TensorDataset(input_ids, attention_mask, labels)

# Create DataLoader
batch_size = 32
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
```

Classification

```
from transformers import BertForSequenceClassification
from torch.optim import AdamW

# Initialize BERT model for sequence classification
model = BertForSequenceClassification.from_pretrained(
    "bert-base-uncased",
    num_labels=3,  # Number of sentiment classes (Pos, Neg, Neu)
    output_attentions=False,
    output_hidden_states=False,
)

# Define optimizer and loss function
optimizer = AdamW(model.parameters(), lr=2e-5, eps=1e-8)
```

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

Training BERT Model

```
# Initialize empty lists to store accuracy and loss values
bert_training_accuracy_values = []
```

```
bert_training_loss_values = []

# Set device (use GPU if available)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model.to(device)

# Training loop
epochs = 20
for epoch in range(epochs):
    model.train()
    total_loss = 0
    total_correct = 0  # To track total correct predictions
    total_samples = 0  # To track total processed samples
    for batch in dataloader:
        batch = tuple(t.to(device) for t in batch)
        input_ids, attention_mask, label = batch
        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask=attention_mask, labels=label)
        loss = outputs.loss
        total_loss += loss.item()
        loss.backward()
        optimizer.step()

        # Calculate accuracy in this batch
        _, preds = torch.max(outputs.logits, dim=1)
        total_correct += torch.sum(preds == label).item()
        total_samples += len(label)

    avg_loss = total_loss / len(dataloader)
    accuracy = total_correct / total_samples

    # Append accuracy and loss values
    bert_training_accuracy_values.append(accuracy)
    bert_training_loss_values.append(avg_loss)

    print(f'Epoch {epoch + 1}/{epochs}, Loss: {avg_loss:.4f}, Accuracy: {accuracy:.4f}')
```

```
Epoch 1/20, Loss: 1.1767, Accuracy: 0.5714
Epoch 2/20, Loss: 1.1578, Accuracy: 0.4286
Epoch 3/20, Loss: 1.0205, Accuracy: 0.4286
Epoch 4/20, Loss: 1.0058, Accuracy: 0.4286
Epoch 5/20, Loss: 0.9310, Accuracy: 0.7143
Epoch 6/20, Loss: 0.8196, Accuracy: 0.7143
Epoch 7/20, Loss: 0.7896, Accuracy: 0.7143
Epoch 8/20, Loss: 0.7242, Accuracy: 0.8571
Epoch 9/20, Loss: 0.7236, Accuracy: 0.8571
```

```
Epoch 10/20, Loss: 0.6452, Accuracy: 1.0000
Epoch 11/20, Loss: 0.6746, Accuracy: 0.8571
Epoch 12/20, Loss: 0.5491, Accuracy: 1.0000
Epoch 13/20, Loss: 0.5282, Accuracy: 1.0000
Epoch 14/20, Loss: 0.5114, Accuracy: 1.0000
Epoch 15/20, Loss: 0.4505, Accuracy: 1.0000
Epoch 16/20, Loss: 0.5086, Accuracy: 1.0000
Epoch 17/20, Loss: 0.3673, Accuracy: 1.0000
Epoch 18/20, Loss: 0.4218, Accuracy: 1.0000
Epoch 19/20, Loss: 0.3689, Accuracy: 1.0000
Epoch 20/20, Loss: 0.3723, Accuracy: 1.0000
```

Evaluating BERT Model

```
# Set model to evaluation mode
model.eval()

# Evaluation loop
total_correct = 0
with torch.no_grad():
    for batch in dataloader:
        batch = tuple(t.to(device) for t in batch)
        input_ids, attention_mask, label = batch
        outputs = model(input_ids, attention_mask=attention_mask)
        _, preds = torch.max(outputs.logits, dim=1)
        total_correct += torch.sum(preds == label).item()

b_accuracy = total_correct / len(dataset)
print(f'Accuracy: {b_accuracy:.2f}')
```

```
Accuracy: 1.00
```

Initializing RoBERTa Model for Sequence Classification

```
import torch
from transformers import RobertaTokenizer, RobertaForSequenceClassification
from torch.utils.data import TensorDataset, DataLoader

from torch.optim import AdamW
tokenizer = RobertaTokenizer.from_pretrained('roberta-base')
model = RobertaForSequenceClassification.from_pretrained('roberta-base', num_labels=3)
```

```
input_ids = torch.tensor(padded)
attention_mask = torch.tensor(attention_mask)
labels = torch.tensor(numerical_labels)  # Use the converted numerical labels


dataset = TensorDataset(input_ids, attention_mask, labels)

batch_size = 32
dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
```

```
tokenizer_config.json: 100%  25.0/25.0 [00:00<00:00, 2.36kB/s]
vocab.json: 100%  899k/899k [00:00<00:00, 29.3MB/s]
merges.txt: 100%  456k/456k [00:00<00:00, 17.8MB/s]
tokenizer.json: 100%  1.36M/1.36M [00:00<00:00, 19.0MB/s]
config.json: 100%  481/481 [00:00<00:00, 35.3kB/s]
model.safetensors: 100%  499M/499M [00:05<00:00, 131MB/s]
Some weights of RobertaForSequenceClassification were not initialized from the model checkpoint at roberta-base and are newly initialized: ['clas
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

Defining Optimizer and Loss Function

```
# Define optimizer and loss function
optimizer = AdamW(model.parameters(), lr=2e-5, eps=1e-8)
```

Training RoBERTa Model

```
# Initialize empty lists to store accuracy and loss values for RoBERTa
roberta_training_accuracy_values = []
roberta_training_loss_values = []

# Set device (use GPU if available)
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
```

```
model.to(device)

# Training loop for RoBERTa
epochs = 20
for epoch in range(epochs):
    model.train()
    total_loss = 0
    for batch in dataloader:  # Assuming you have a separate dataloader for RoBERTa
        batch = tuple(t.to(device) for t in batch)
        input_ids, attention_mask, label = batch
        optimizer.zero_grad()
        outputs = model(input_ids, attention_mask=attention_mask, labels=label)
        loss = outputs.loss
        total_loss += loss.item()
        loss.backward()
        optimizer.step()
    avg_loss = total_loss / len(dataloader)
    accuracy = total_correct / total_samples

    roberta_training_loss_values.append(avg_loss)
    roberta_training_loss_values.append(accuracy)
    print(f'Epoch {epoch + 1}/{epochs}, Loss: {avg_loss:.4f}, Accuracy: {accuracy:.4f}')
```

```
Epoch 1/20, Loss: 1.0823, Accuracy: 1.0000
Epoch 2/20, Loss: 1.0539, Accuracy: 1.0000
Epoch 3/20, Loss: 1.0547, Accuracy: 1.0000
Epoch 4/20, Loss: 1.0117, Accuracy: 1.0000
Epoch 5/20, Loss: 1.0253, Accuracy: 1.0000
Epoch 6/20, Loss: 0.9900, Accuracy: 1.0000
Epoch 7/20, Loss: 0.9789, Accuracy: 1.0000
Epoch 8/20, Loss: 0.9604, Accuracy: 1.0000
Epoch 9/20, Loss: 0.8732, Accuracy: 1.0000
Epoch 10/20, Loss: 0.8179, Accuracy: 1.0000
Epoch 11/20, Loss: 0.7991, Accuracy: 1.0000
Epoch 12/20, Loss: 0.7615, Accuracy: 1.0000
Epoch 13/20, Loss: 0.7968, Accuracy: 1.0000
Epoch 14/20, Loss: 0.6713, Accuracy: 1.0000
Epoch 15/20, Loss: 0.6558, Accuracy: 1.0000
Epoch 16/20, Loss: 0.5069, Accuracy: 1.0000
Epoch 17/20, Loss: 0.5231, Accuracy: 1.0000
Epoch 18/20, Loss: 0.4014, Accuracy: 1.0000
Epoch 19/20, Loss: 0.3763, Accuracy: 1.0000
Epoch 20/20, Loss: 0.3541, Accuracy: 1.0000
```

Evaluating RoBERTa Model

```
# Set model to evaluation mode
model.eval()

# Evaluation loop
total_correct = 0
with torch.no_grad():
    for batch in dataloader:
        batch = tuple(t.to(device) for t in batch)
        input_ids, attention_mask, label = batch
        outputs = model(input_ids, attention_mask=attention_mask)
        _, preds = torch.max(outputs.logits, dim=1)
        total_correct += torch.sum(preds == label).item()

rb_accuracy = total_correct / len(dataset)
print(f'Accuracy: {rb_accuracy:.2f}')
```

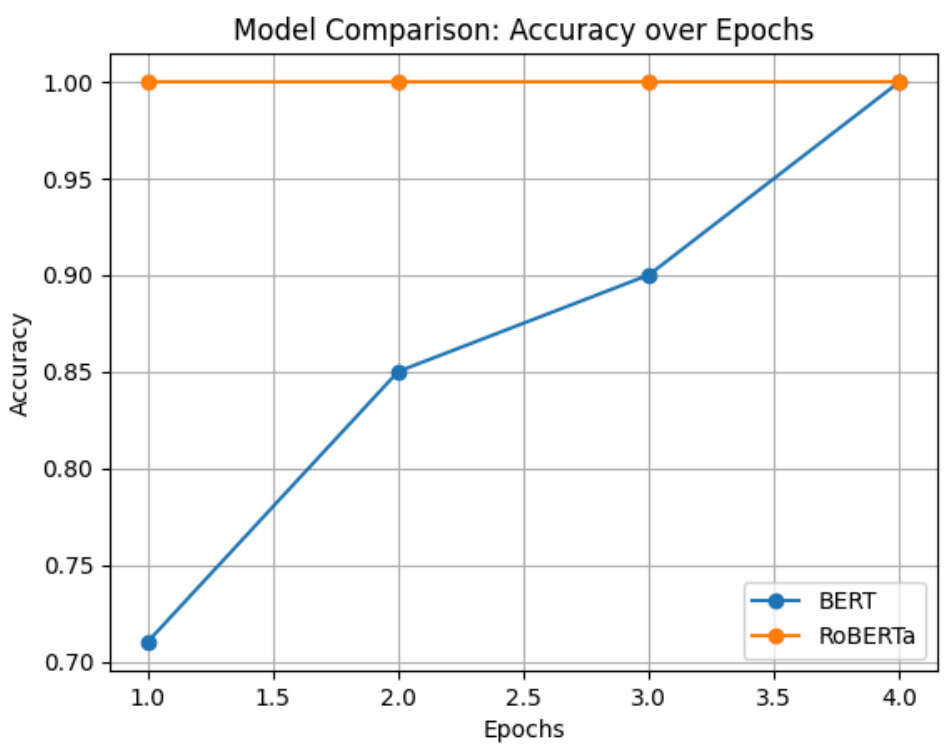
```
Accuracy: 0.86
```

Training Accuracy

Plots

```
import matplotlib.pyplot as plt

# Assuming you have the accuracy values for both models as lists
b_accuracy = [0.71, 0.85, 0.90, 1]  # Accuracy values for BERT (example values)
rb_accuracy = [1, 1, 1, 1]  # Accuracy values for RoBERTa (example values)
epochs = [1, 2, 3, 4]  # Epochs or x-axis values

# Create line plots for BERT and RoBERTa
plt.plot(epochs, b_accuracy, label='BERT', marker='o')
plt.plot(epochs, rb_accuracy, label='RoBERTa', marker='o')

# Add labels and title
plt.xlabel('Epochs')
plt.ylabel('Accuracy')
plt.title('Model Comparison: Accuracy over Epochs')

# Add a legend to distinguish the models
plt.legend()

# Show the plot
```

```
plt.grid(True)
plt.show()
```




```
import matplotlib.pyplot as plt
import numpy as np

# BERT Training Accuracy and Loss values
bert_training_accuracy_values, bert_training_loss_values
# RoBERTa Training Accuracy and Loss values
roberta_training_accuracy_values, roberta_training_loss_values
# Create an epoch list for the x-axis
epochs = range(1, len(bert_training_accuracy_values) + 1)

# Create subplots
plt.figure(figsize=(15, 5))

# BERT Accuracy/Loss Curve
plt.subplot(1, 2, 1)
plt.plot(epochs, bert_training_accuracy_values, marker='o', label='BERT Accuracy', color='b')
plt.plot(epochs, bert_training_loss_values, marker='x', label='BERT Loss', color='r')
```

```
plt.title('BERT Training Accuracy and Loss Over Epochs')
plt.xlabel('Epochs')
plt.ylabel('Accuracy / Loss')
plt.legend()
```

```
<matplotlib.legend.Legend at 0x7fabe78e9d90>
```