```
import pandas as pd

df = pd.read_csv('bbc-text.csv')
df.head(2)
```

```
   category                                               text
0      tech  tv future in the hands of viewers with home th...
1  business  worldcom boss  left books alone  former worldc...
```

```
!pip install transformers

from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained('bert-base-cased')

example_text = 'I will watch Memento tonight'
bert_input = tokenizer(example_text,padding='max_length', max_length =
10,
                       truncation=True, return_tensors="pt")


print(bert_input['input_ids'])
print(bert_input['token_type_ids'])
print(bert_input['attention_mask'])
```

{"model_id":"461e3ee88d624acfba1e99481c974238","version_major":2,"vers
ion_minor":0}

{"model_id":"374eae7ba2fa4426a8341f90386a24fb","version_major":2,"vers
ion_minor":0}

{"model_id":"824db6a5a7df4754aba5798dc33430ce","version_major":2,"vers
ion_minor":0}

```
tensor([[  101,   146,  1209,  2824,  2508, 26173,  3568,   102,
0,     0]])
tensor([[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]])
tensor([[1, 1, 1, 1, 1, 1, 1, 1, 0, 0]])

example_text = tokenizer.decode(bert_input.input_ids[0])

print(example_text)
```

```
[CLS] I will watch Memento tonight [SEP] [PAD] [PAD]
```

#Data Class

```
import torch
import numpy as np
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained('bert-base-cased')
```

```
labels = {'business':0,
          'entertainment':1,
          'sport':2,
          'tech':3,
          'politics':4
          }

class Dataset(torch.utils.data.Dataset):

    def __init__(self, df):

        self.labels = [labels[label] for label in df['category']]
        self.texts = [tokenizer(text,
                                padding='max_length', max_length = 512,
truncation=True,
                                 return_tensors="pt") for text in
df['text']]

    def classes(self):
        return self.labels

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

    def get_batch_labels(self, idx):
        # Fetch a batch of labels
        return np.array(self.labels[idx])

    def get_batch_texts(self, idx):
        # Fetch a batch of inputs
        return self.texts[idx]

    def __getitem__(self, idx):

        batch_texts = self.get_batch_texts(idx)
        batch_y = self.get_batch_labels(idx)

        return batch_texts, batch_y

np.random.seed(112)
df_train, df_val, df_test = np.split(df.sample(frac=1,
random_state=42),
                                     [int(.8*len(df)),
int(.9*len(df))])

print(len(df_train),len(df_val), len(df_test))
```

```
1780 222 223
```

#Model building

```
from torch import nn
from transformers import BertModel

class BertClassifier(nn.Module):

    def __init__(self, dropout=0.5):

        super(BertClassifier, self).__init__()

        self.bert = BertModel.from_pretrained('bert-base-cased')
        self.dropout = nn.Dropout(dropout)
        self.linear = nn.Linear(768, 5)
        self.relu = nn.ReLU()

    def forward(self, input_id, mask):

        _, pooled_output = self.bert(input_ids= input_id,
attention_mask=mask,return_dict=False)
        dropout_output = self.dropout(pooled_output)
        linear_output = self.linear(dropout_output)
        final_layer = self.relu(linear_output)

        return final_layer
```

#Training loop

```
from torch.optim import Adam
from tqdm import tqdm

def train(model, train_data, val_data, learning_rate, epochs):

    train, val = Dataset(train_data), Dataset(val_data)

    train_dataloader = torch.utils.data.DataLoader(train,
batch_size=2, shuffle=True)
    val_dataloader = torch.utils.data.DataLoader(val, batch_size=2)

    use_cuda = torch.cuda.is_available()
    device = torch.device("cuda" if use_cuda else "cpu")

    criterion = nn.CrossEntropyLoss()
    optimizer = Adam(model.parameters(), lr= learning_rate)

    if use_cuda:

            model = model.cuda()
            criterion = criterion.cuda()

    for epoch_num in range(epochs):
```

```
            total_acc_train = 0
            total_loss_train = 0

            for train_input, train_label in tqdm(train_dataloader):

                train_label = train_label.to(device)
                mask = train_input['attention_mask'].to(device)
                input_id =
train_input['input_ids'].squeeze(1).to(device)

                output = model(input_id, mask)

                batch_loss = criterion(output, train_label.long())
                total_loss_train += batch_loss.item()

                acc = (output.argmax(dim=1) ==
train_label).sum().item()
                total_acc_train += acc

                model.zero_grad()
                batch_loss.backward()
                optimizer.step()

            total_acc_val = 0
            total_loss_val = 0

            with torch.no_grad():

                for val_input, val_label in val_dataloader:

                    val_label = val_label.to(device)
                    mask = val_input['attention_mask'].to(device)
                    input_id =
val_input['input_ids'].squeeze(1).to(device)

                    output = model(input_id, mask)

                    batch_loss = criterion(output, val_label.long())
                    total_loss_val += batch_loss.item()

                    acc = (output.argmax(dim=1) ==
val_label).sum().item()
                    total_acc_val += acc

            print(
                f'Epochs: {epoch_num + 1} | Train Loss:
{total_loss_train / len(train_data): .3f} \
                | Train Accuracy: {total_acc_train /
```

```
len(train_data): .3f} \
                | Val Loss: {total_loss_val / len(val_data): .3f} \
                | Val Accuracy: {total_acc_val / len(val_data): .3f}')

EPOCHS = 5
model = BertClassifier()
LR = 1e-6

train(model, df_train, df_val, LR, EPOCHS)
```

```
Some weights of the model checkpoint at bert-base-cased were not used
when initializing BertModel:
['cls.predictions.transform.LayerNorm.weight',
'cls.predictions.decoder.weight', 'cls.seq_relationship.bias',
'cls.predictions.transform.LayerNorm.bias',
'cls.predictions.transform.dense.bias',
'cls.predictions.transform.dense.weight', 'cls.predictions.bias',
'cls.seq_relationship.weight']
- This IS expected if you are initializing BertModel from the
checkpoint of a model trained on another task or with another
architecture (e.g. initializing a BertForSequenceClassification model
from a BertForPreTraining model).
- This IS NOT expected if you are initializing BertModel from the
checkpoint of a model that you expect to be exactly identical
(initializing a BertForSequenceClassification model from a
BertForSequenceClassification model).
100%|██████████| 890/890 [03:16<00:00,  4.53it/s]

Epochs: 1 | Train Loss:  0.722                 | Train Accuracy:
0.411                 | Val Loss:  0.586                 | Val
Accuracy:  0.658

100%|██████████| 890/890 [03:14<00:00,  4.57it/s]

Epochs: 2 | Train Loss:  0.398                 | Train Accuracy:
0.813                 | Val Loss:  0.227                 | Val
Accuracy:  0.955

100%|██████████| 890/890 [03:14<00:00,  4.58it/s]

Epochs: 3 | Train Loss:  0.174                 | Train Accuracy:
0.975                 | Val Loss:  0.117                 | Val
Accuracy:  0.982

100%|██████████| 890/890 [03:13<00:00,  4.59it/s]

Epochs: 4 | Train Loss:  0.090                 | Train Accuracy:
0.992                 | Val Loss:  0.072                 | Val
Accuracy:  0.995

100%|██████████| 890/890 [03:13<00:00,  4.59it/s]
```

```
Epochs: 5 | Train Loss:  0.055                 | Train Accuracy:
0.995                  | Val Loss:  0.046                  | Val
Accuracy:  0.986
```

#Evaluation

```
def evaluate(model, test_data):

    test = Dataset(test_data)

    test_dataloader = torch.utils.data.DataLoader(test, batch_size=2)

    use_cuda = torch.cuda.is_available()
    device = torch.device("cuda" if use_cuda else "cpu")

    if use_cuda:

        model = model.cuda()

    total_acc_test = 0
    with torch.no_grad():

        for test_input, test_label in test_dataloader:

              test_label = test_label.to(device)
              mask = test_input['attention_mask'].to(device)
              input_id = test_input['input_ids'].squeeze(1).to(device)

              output = model(input_id, mask)

              acc = (output.argmax(dim=1) == test_label).sum().item()
              total_acc_test += acc

    print(f'Test Accuracy: {total_acc_test / len(test_data): .3f}')

evaluate(model, df_test)
```

```
Test Accuracy:  0.996
```

#Sample Prediction

```
test = df['text'][0]
test
```

```
{"type":"string"}
```

```
encoding = tokenizer.encode_plus(
  test,
  add_special_tokens=True,
  max_length=512,
  return_token_type_ids=False,
```

```
  padding=True,
  truncation=True,
  return_attention_mask=True,
  return_tensors='pt',
)

encoding.keys()

dict_keys(['input_ids', 'attention_mask'])

use_cuda = torch.cuda.is_available()
device = torch.device("cuda" if use_cuda else "cpu")

mask = encoding['attention_mask'].to(device)
input_id = encoding['input_ids'].squeeze(1).to(device)

output = model(input_id,mask)

prediction_value = torch.argmax(output, axis = 1).cpu().numpy()[0]

prediction_value

3

#Prediction function

def prediction(model,test):


  encoding = tokenizer.encode_plus(
  test,
  add_special_tokens=True,
  max_length=512,
  return_token_type_ids=False,
  padding=True,
  truncation=True,
  return_attention_mask=True,
  return_tensors='pt',
)

  use_cuda = torch.cuda.is_available()
  device = torch.device("cuda" if use_cuda else "cpu")

  mask = encoding['attention_mask'].to(device)
  input_id = encoding['input_ids'].squeeze(1).to(device)

  output = model(input_id,mask)

  prediction_value = torch.argmax(output, axis = 1).cpu().numpy()[0]

  labels = {'business':0,
```

```
            'entertainment':1,
            'sport':2,
            'tech':3,
            'politics':4
            }

  key_list = list(labels.keys())
  val_list = list(labels.values())

  position = val_list.index(prediction_value)

  print(f'Model Prediction: {key_list[position]}')
```

###Sample prediction to check prediction function

```
df.head(10)
```

```
        category                                               text
0           tech  tv future in the hands of viewers with home th...
1       business  worldcom boss  left books alone  former worldc...
2          sport  tigers wary of farrell  gamble  leicester say ...
3          sport  yeading face newcastle in fa cup premiership s...
4  entertainment  ocean s twelve raids box office ocean s twelve...
5       politics  howard hits back at mongrel jibe michael howar...
6       politics  blair prepares to name poll date tony blair is...
7          sport  henman hopes ended in dubai third seed tim hen...
8          sport  wilkinson fit to face edinburgh england captai...
9  entertainment  last star wars  not for children  the sixth an...
```

```
test = df['text'][9]
test
```

```
{"type":"string"}
```

```
prediction(model,test)
```

```
Model Prediction: entertainment
```

From above we can see that the model prediction is accurate