_sentiment_analysis_assigment

July 10, 2023

[ ]:
```
!pip install transformers
```

```
Collecting transformers
  Downloading transformers-4.11.3-py3-none-any.whl (2.9 MB)
     |                  | 2.9 MB 4.1 MB/s
Requirement already satisfied: tqdm>=4.27 in
/usr/local/lib/python3.7/dist-packages (from transformers) (4.62.3)
Requirement already satisfied: importlib-metadata in
/usr/local/lib/python3.7/dist-packages (from transformers) (4.8.1)
Collecting huggingface-hub>=0.0.17
  Downloading huggingface_hub-0.0.19-py3-none-any.whl (56 kB)
     |                  | 56 kB 5.0 MB/s
Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-
packages (from transformers) (2.23.0)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-
packages (from transformers) (21.0)
Collecting tokenizers<0.11,>=0.10.1
  Downloading tokenizers-0.10.3-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_6
4.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (3.3 MB)
     |                  | 3.3 MB 35.4 MB/s
Requirement already satisfied: regex!=2019.12.17 in
/usr/local/lib/python3.7/dist-packages (from transformers) (2019.12.20)
Collecting pyyaml>=5.1
  Downloading PyYAML-6.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manyl
inux_2_12_x86_64.manylinux2010_x86_64.whl (596 kB)
     |                  | 596 kB 41.6 MB/s
Requirement already satisfied: numpy>=1.17 in
/usr/local/lib/python3.7/dist-packages (from transformers) (1.19.5)
Collecting sacremoses
  Downloading sacremoses-0.0.46-py3-none-any.whl (895 kB)
     |                  | 895 kB 48.7 MB/s
Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-
packages (from transformers) (3.3.0)
Requirement already satisfied: typing-extensions in
/usr/local/lib/python3.7/dist-packages (from huggingface-
hub>=0.0.17->transformers) (3.7.4.3)
Requirement already satisfied: pyparsing>=2.0.2 in
/usr/local/lib/python3.7/dist-packages (from packaging>=20.0->transformers)
```

```
(2.4.7)
Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-
packages (from importlib-metadata->transformers) (3.6.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-
packages (from requests->transformers) (2.10)
Requirement already satisfied: chardet<4,>=3.0.2 in
/usr/local/lib/python3.7/dist-packages (from requests->transformers) (3.0.4)
Requirement already satisfied: urllib3!=1.25.0,!=1.25.1,<1.26,>=1.21.1 in
/usr/local/lib/python3.7/dist-packages (from requests->transformers) (1.24.3)
Requirement already satisfied: certifi>=2017.4.17 in
/usr/local/lib/python3.7/dist-packages (from requests->transformers) (2021.5.30)
Requirement already satisfied: joblib in /usr/local/lib/python3.7/dist-packages
(from sacremoses->transformers) (1.0.1)
Requirement already satisfied: six in /usr/local/lib/python3.7/dist-packages
(from sacremoses->transformers) (1.15.0)
Requirement already satisfied: click in /usr/local/lib/python3.7/dist-packages
(from sacremoses->transformers) (7.1.2)
Installing collected packages: pyyaml, tokenizers, sacremoses, huggingface-hub,
transformers
  Attempting uninstall: pyyaml
    Found existing installation: PyYAML 3.13
    Uninstalling PyYAML-3.13:
      Successfully uninstalled PyYAML-3.13
Successfully installed huggingface-hub-0.0.19 pyyaml-6.0 sacremoses-0.0.46
tokenizers-0.10.3 transformers-4.11.3
```

[ ]:

```python
import transformers
from transformers import BertModel, BertTokenizer, AdamW, get_linear_schedule_with_warmup
import torch

import numpy as np
import pandas as pd
import seaborn as sns
from pylab import rcParams
import matplotlib.pyplot as plt
from matplotlib import rc
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix, classification_report
from collections import defaultdict
from textwrap import wrap

from torch import nn, optim
from torch.utils.data import Dataset, DataLoader
import torch.nn.functional as F

import warnings
```

```
warnings.filterwarnings('ignore')
```

[ ]:
```
%matplotlib inline
%config InlineBackend.figure_format='retina'
```

[ ]:
```
sns.set(style='whitegrid', palette='muted', font_scale=1.2)
HAPPY_COLORS_PALETTE = ["#01BEFE", "#FFDD00", "#FF7D00", "#FF006D", "#ADFF02",␣
 ↪"#8F00FF"]
sns.set_palette(sns.color_palette(HAPPY_COLORS_PALETTE))
rcParams['figure.figsize'] = 12, 8
RANDOM_SEED = 42
np.random.seed(RANDOM_SEED)
torch.manual_seed(RANDOM_SEED)
# device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")


# If there's a GPU available...
if torch.cuda.is_available():

    # Tell PyTorch to use the GPU.
    device = torch.device("cuda")

    print('There are %d GPU(s) available.' % torch.cuda.device_count())

    print('We will use the GPU:', torch.cuda.get_device_name(0))

# If not...
else:
    print('No GPU available, using the CPU instead.')
    device = torch.device("cpu")
```

```
There are 1 GPU(s) available.
We will use the GPU: Tesla K80
```

[ ]:
```
!gdown --id 1S6qMioqPJjyBLpLVz4gmRTnJHnjitnuV
!gdown --id 1zdmewp7ayS4js4VtrJEHzAheSW-5NBZv
```

```
Downloading…
From: https://drive.google.com/uc?id=1S6qMioqPJjyBLpLVz4gmRTnJHnjitnuV
To: /content/apps.csv
100% 134k/134k [00:00<00:00, 46.2MB/s]
Downloading…
From: https://drive.google.com/uc?id=1zdmewp7ayS4js4VtrJEHzAheSW-5NBZv
To: /content/reviews.csv
100% 7.17M/7.17M [00:00<00:00, 22.9MB/s]
```

[ ]:
```
df = pd.read_csv("reviews.csv")
df.head()
```

[ ]:
```
          userName  …      appId
0    Andrew Thomas  …  com.anydo
1     Craig Haines  …  com.anydo
2    steven adkins  …  com.anydo
3  Lars Panzerbjørn  …  com.anydo
4    Scott Prewitt  …  com.anydo

[5 rows x 11 columns]
```

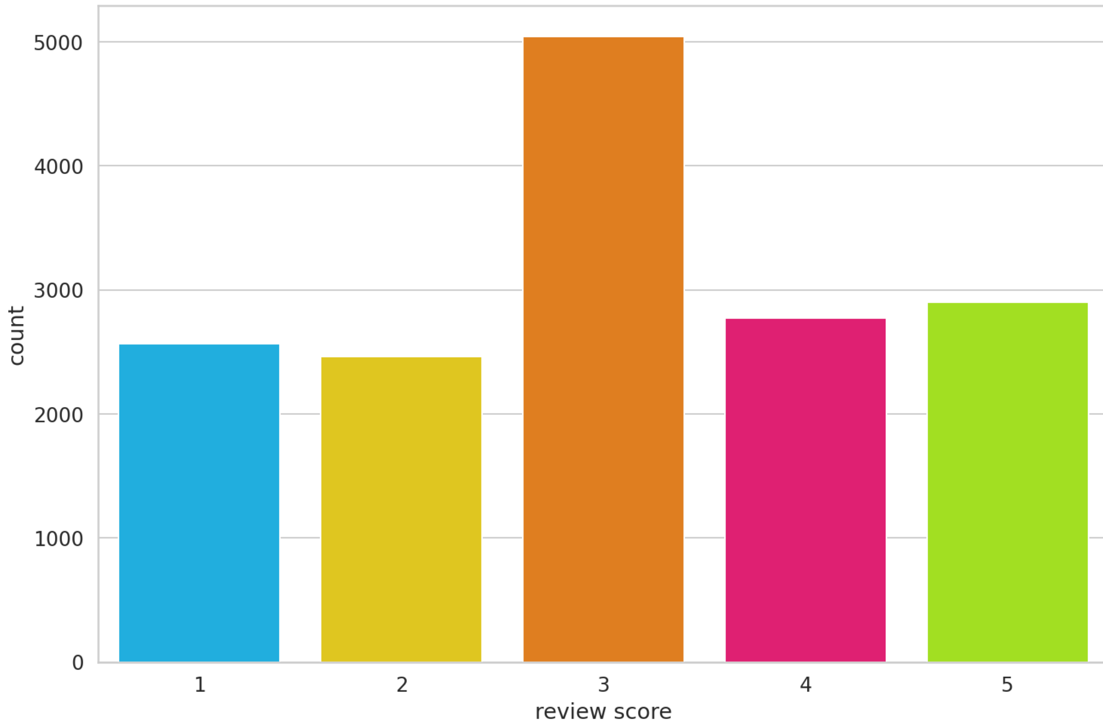
[ ]:
```
df.shape
```

[ ]: (15746, 11)

[ ]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 15746 entries, 0 to 15745
Data columns (total 11 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   userName              15746 non-null  object
 1   userImage             15746 non-null  object
 2   content               15746 non-null  object
 3   score                 15746 non-null  int64
 4   thumbsUpCount         15746 non-null  int64
 5   reviewCreatedVersion  13533 non-null  object
 6   at                    15746 non-null  object
 7   replyContent          7367 non-null   object
 8   repliedAt             7367 non-null   object
 9   sortOrder             15746 non-null  object
 10  appId                 15746 non-null  object
dtypes: int64(2), object(9)
memory usage: 1.3+ MB
```

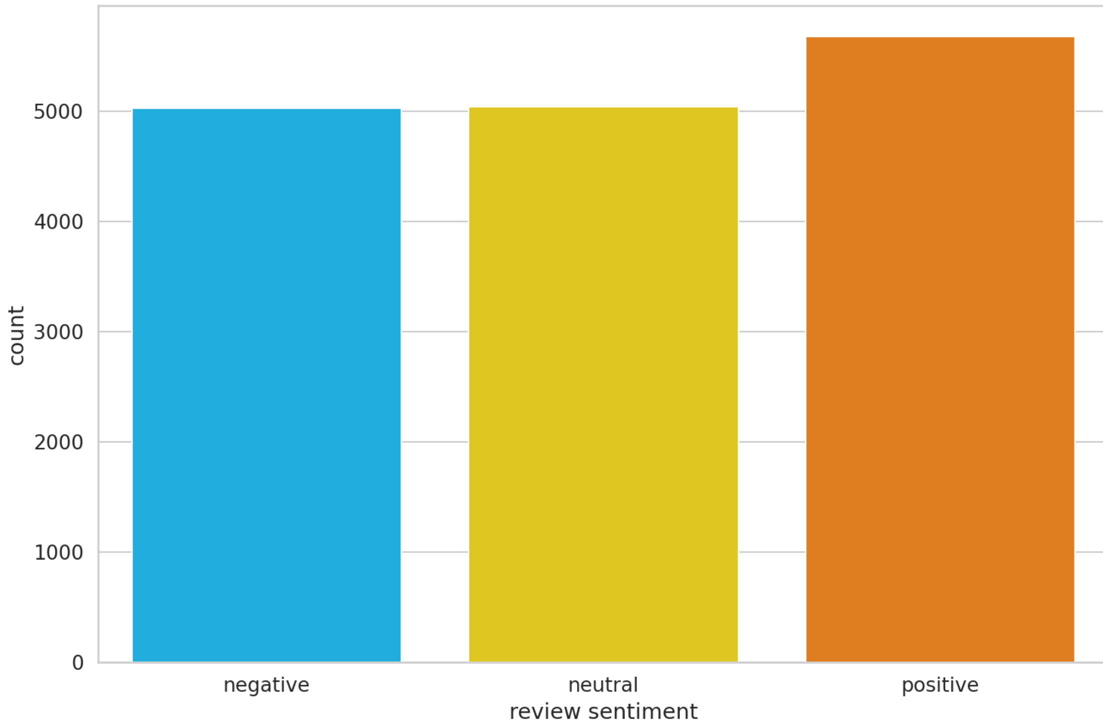
[ ]:
```
sns.countplot(df.score)
plt.xlabel('review score');
```




```
[ ]: def to_sentiment(rating):
         rating = int(rating)
         if rating <= 2:
             return 0
         elif rating == 3:
             return 1
         else:
             return 2

     df['sentiment'] = df.score.apply(to_sentiment)
```

```
[ ]: class_names = ['negative', 'neutral', 'positive']
```

```
[ ]: ax = sns.countplot(df.sentiment)
     plt.xlabel('review sentiment')
     ax.set_xticklabels(class_names);
```




```
[ ]: PRE_TRAINED_MODEL_NAME = 'bert-base-cased'
```

```
[ ]: tokenizer = BertTokenizer.from_pretrained(PRE_TRAINED_MODEL_NAME)
```

```
Downloading:   0%|          | 0.00/208k [00:00<?, ?B/s]

Downloading:   0%|          | 0.00/29.0 [00:00<?, ?B/s]

Downloading:   0%|          | 0.00/426k [00:00<?, ?B/s]

Downloading:   0%|          | 0.00/570 [00:00<?, ?B/s]
```

```
[ ]: sample_txt = 'When was I last outside? I am stuck at home for 2 weeks.'
```

```
[ ]: tokens = tokenizer.tokenize(sample_txt)
token_ids = tokenizer.convert_tokens_to_ids(tokens)

print(f' Sentence: {sample_txt}')
print(f'   Tokens: {tokens}')
print(f'Token IDs: {token_ids}')
```

```
 Sentence: When was I last outside? I am stuck at home for 2 weeks.
   Tokens: ['When', 'was', 'I', 'last', 'outside', '?', 'I', 'am', 'stuck',
'at', 'home', 'for', '2', 'weeks', '.']
Token IDs: [1332, 1108, 146, 1314, 1796, 136, 146, 1821, 5342, 1120, 1313, 1111,
123, 2277, 119]
```

[ ]:
```
tokenizer.sep_token, tokenizer.sep_token_id
```

[ ]: ('[SEP]', 102)

[ ]:
```
tokenizer.cls_token, tokenizer.cls_token_id
```

[ ]: ('[CLS]', 101)

[ ]:
```
tokenizer.pad_token, tokenizer.pad_token_id
```

[ ]: ('[PAD]', 0)

[ ]:
```
tokenizer.unk_token, tokenizer.unk_token_id
```

[ ]: ('[UNK]', 100)

[ ]:
```
encoding = tokenizer.encode_plus(
  sample_txt,
  max_length=32,
  add_special_tokens=True, # Add '[CLS]' and '[SEP]'
  return_token_type_ids=False,
  pad_to_max_length=True,
  return_attention_mask=True,
  return_tensors='pt',  # Return PyTorch tensors
)

encoding.keys()
```

```
Truncation was not explicitly activated but `max_length` is provided a specific
value, please use `truncation=True` to explicitly truncate examples to max
length. Defaulting to 'longest_first' truncation strategy. If you encode pairs
of sequences (GLUE-style) with the tokenizer you can select this strategy more
precisely by providing a specific strategy to `truncation`.
```

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

[ ]:
```
print(len(encoding['input_ids'][0]))
encoding['input_ids'][0]
```

32

[ ]:
```
tensor([ 101, 1332, 1108,  146, 1314, 1796,  136,  146, 1821, 5342, 1120, 1313,
        1111,  123, 2277,  119,  102,    0,    0,    0,    0,    0,    0,    0,
           0,    0,    0,    0,    0,    0,    0,    0])
```

[ ]:
```
print(len(encoding['attention_mask'][0]))
encoding['attention_mask']
```

```
32
```

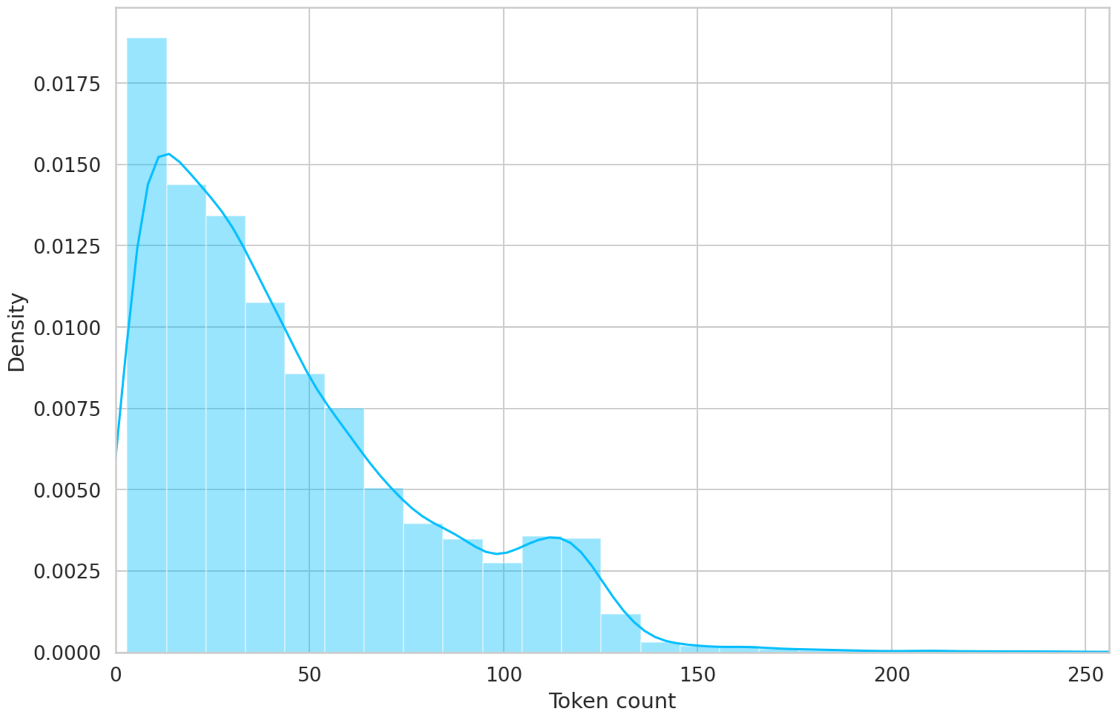
```
[ ]: tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0,
              0, 0, 0, 0, 0, 0, 0, 0]])
```

```
[ ]: tokenizer.convert_ids_to_tokens(encoding['input_ids'][0])
```

```
[ ]: ['[CLS]',
      'When',
      'was',
      'I',
      'last',
      'outside',
      '?',
      'I',
      'am',
      'stuck',
      'at',
      'home',
      'for',
      '2',
      'weeks',
      '.',
      '[SEP]',
      '[PAD]',
      '[PAD]',
      '[PAD]',
      '[PAD]',
      '[PAD]',
      '[PAD]',
      '[PAD]',
      '[PAD]',
      '[PAD]',
      '[PAD]',
      '[PAD]',
      '[PAD]',
      '[PAD]',
      '[PAD]',
      '[PAD]']
```

```
[ ]: token_lens = []

     for content in df.content:
         tokens = tokenizer.encode(content, max_length=512)
         token_lens.append(len(tokens))
```

```
[ ]: sns.distplot(token_lens)
     plt.xlim([0, 256]);
     plt.xlabel('Token count');
```




```
[ ]: MAX_LEN = 160
```

```
[ ]: class GPReviewDataset(Dataset):
         def __init__(self, reviews, targets, tokenizer, max_len):
             self.reviews = reviews
             self.targets = targets
             self.tokenizer = tokenizer
             self.max_len = max_len

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

         def __getitem__(self, item):
             review = str(self.reviews[item])
             target = self.targets[item]

             encoding = self.tokenizer.encode_plus(
               review,
               add_special_tokens=True,
               max_length=self.max_len,
```

```
            return_token_type_ids=False,
            pad_to_max_length=True,
            return_attention_mask=True,
            return_tensors='pt',
          )

          return {
            'review_text': review,
            'input_ids': encoding['input_ids'].flatten(),
            'attention_mask': encoding['attention_mask'].flatten(),
            'targets': torch.tensor(target, dtype=torch.long)
          }
```

[ ]:
```
df_train, df_test = train_test_split(df, test_size=0.3,␣
 ↪random_state=RANDOM_SEED)

df_val, df_test = train_test_split(df_test, test_size=0.5,␣
 ↪random_state=RANDOM_SEED)
```

[ ]:
```
df_train.shape
```

[ ]: (11022, 12)

[ ]:
```
df_val.shape
```

[ ]: (2362, 12)

[ ]:
```
df_test.shape
```

[ ]: (2362, 12)

[ ]:
```
def create_data_loader(df, tokenizer, max_len, batch_size):
    ds = GPReviewDataset(
        reviews=df.content.to_numpy(),
        targets=df.sentiment.to_numpy(),
        tokenizer=tokenizer,
        max_len=max_len
    )

    return DataLoader(
        ds,
        batch_size=batch_size,
        num_workers=2
    )
```

[ ]:
```
BATCH_SIZE = 16
```

```python
train_data_loader = create_data_loader(df_train, tokenizer, MAX_LEN, BATCH_SIZE)
val_data_loader = create_data_loader(df_val, tokenizer, MAX_LEN, BATCH_SIZE)
test_data_loader = create_data_loader(df_test, tokenizer, MAX_LEN, BATCH_SIZE)
```

[ ]:
```python
data = next(iter(train_data_loader))
data.keys()
```

[ ]:
```
dict_keys(['review_text', 'input_ids', 'attention_mask', 'targets'])
```

[ ]:
```python
print(data['input_ids'].shape)
print(data['attention_mask'].shape)
print(data['targets'].shape)
```

```
torch.Size([16, 160])
torch.Size([16, 160])
torch.Size([16])
```

[ ]:
```python
bert_model = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME)
```

```
Downloading:   0%|          | 0.00/416M [00:00<?, ?B/s]
```

```
Some weights of the model checkpoint at bert-base-cased were not used when
initializing BertModel: ['cls.predictions.transform.LayerNorm.weight',
'cls.seq_relationship.bias', 'cls.seq_relationship.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.predictions.bias', 'cls.predictions.transform.dense.weight',
'cls.predictions.transform.LayerNorm.bias']
- 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).
```

[ ]:
```python
last_hidden_state, pooled_output = bert_model(
    input_ids=encoding['input_ids'], attention_mask=encoding['attention_mask'],␣
 ↪return_dict = False)
```

[ ]:
```python
# last_hidden_state=dict['last_hidden_state']
# pooled_output=dict['pooler_output']

last_hidden_state.shape
```

[ ]:
```
torch.Size([1, 32, 768])
```

[ ]:
```python
bert_model.config.hidden_size
```

[ ]:
```
768
```

[ ]:
```
pooled_output.shape
```

[ ]:
```
torch.Size([1, 768])
```

[ ]:
```
class SentimentClassifier(nn.Module):
    def __init__(self, n_classes):
        super(SentimentClassifier, self).__init__()
        self.bert = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME)
        self.drop = nn.Dropout(p=0.3)
        self.out = nn.Linear(self.bert.config.hidden_size, n_classes)

#     def forward(self, input_ids, attention_mask):
#         _, pooled_output = self.bert(input_ids=input_ids,␣
↪attention_mask=attention_mask)
#         output = self.drop(pooled_output)

#         return self.out(output)


    def forward(self, input_ids, attention_mask):
        returned = self.bert(input_ids=input_ids, attention_mask=attention_mask)

        pooled_output = returned["pooler_output"]
        output = self.drop(pooled_output)

        return self.out(output)
```

[ ]:
```
model = SentimentClassifier(len(class_names))
model = model.to(device)
```

```
Some weights of the model checkpoint at bert-base-cased were not used when
initializing BertModel: ['cls.predictions.transform.LayerNorm.weight',
'cls.seq_relationship.bias', 'cls.seq_relationship.weight',
'cls.predictions.transform.dense.bias', 'cls.predictions.decoder.weight',
'cls.predictions.bias', 'cls.predictions.transform.dense.weight',
'cls.predictions.transform.LayerNorm.bias']
- 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).
```

[ ]:
```
input_ids = data['input_ids'].to(device)
attention_mask = data['attention_mask'].to(device)
print(input_ids.shape) # batch size x seq length
print(attention_mask.shape) # batch size x seq length
```

```
torch.Size([16, 160])
torch.Size([16, 160])
```

[ ]:
```
F.softmax(model(input_ids, attention_mask), dim=1)
```

[ ]:
```
tensor([[0.2236, 0.4834, 0.2931],
        [0.2244, 0.3195, 0.4561],
        [0.3162, 0.2818, 0.4020],
        [0.2029, 0.4641, 0.3330],
        [0.5511, 0.2338, 0.2151],
        [0.2239, 0.4568, 0.3194],
        [0.2738, 0.3206, 0.4055],
        [0.4069, 0.1912, 0.4020],
        [0.4032, 0.1856, 0.4113],
        [0.3382, 0.1997, 0.4621],
        [0.3512, 0.2262, 0.4226],
        [0.3372, 0.2153, 0.4474],
        [0.2272, 0.3774, 0.3954],
        [0.2966, 0.3315, 0.3719],
        [0.3520, 0.2277, 0.4203],
        [0.1511, 0.3517, 0.4973]], device='cuda:0', grad_fn=<SoftmaxBackward>)
```

[ ]:
```
EPOCHS = 10
optimizer = AdamW(model.parameters(), lr=2e-5, correct_bias=False)
total_steps = len(train_data_loader) * EPOCHS
scheduler = get_linear_schedule_with_warmup(
  optimizer,
  num_warmup_steps=0,
  num_training_steps=total_steps
)
loss_fn = nn.CrossEntropyLoss().to(device)
```

[ ]:
```
def train_epoch(model, data_loader, loss_fn, optimizer, device, scheduler,␣
 ↪n_examples):
    model = model.train()

    losses = []
    correct_predictions = 0

    for d in data_loader:
        input_ids = d["input_ids"].to(device)
        attention_mask = d["attention_mask"].to(device)
        targets = d["targets"].to(device)

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

```
        )

        _, preds = torch.max(outputs, dim=1)
        loss = loss_fn(outputs, targets)

        correct_predictions += torch.sum(preds == targets)
        losses.append(loss.item())

        loss.backward()
        nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
        optimizer.step()
        scheduler.step()
        optimizer.zero_grad()

    return correct_predictions.double() / n_examples, np.mean(losses)
```

[ ]:
```
def eval_model(model, data_loader, loss_fn, device, n_examples):
    model = model.eval()

    losses = []
    correct_predictions = 0

    with torch.no_grad():
        for d in data_loader:
            input_ids = d["input_ids"].to(device)
            attention_mask = d["attention_mask"].to(device)
            targets = d["targets"].to(device)

            outputs = model(
            input_ids=input_ids,
            attention_mask=attention_mask
            )
            _, preds = torch.max(outputs, dim=1)

            loss = loss_fn(outputs, targets)

            correct_predictions += torch.sum(preds == targets)
            losses.append(loss.item())

    return correct_predictions.double() / n_examples, np.mean(losses)
```

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

```
Mounted at /content/drive
```

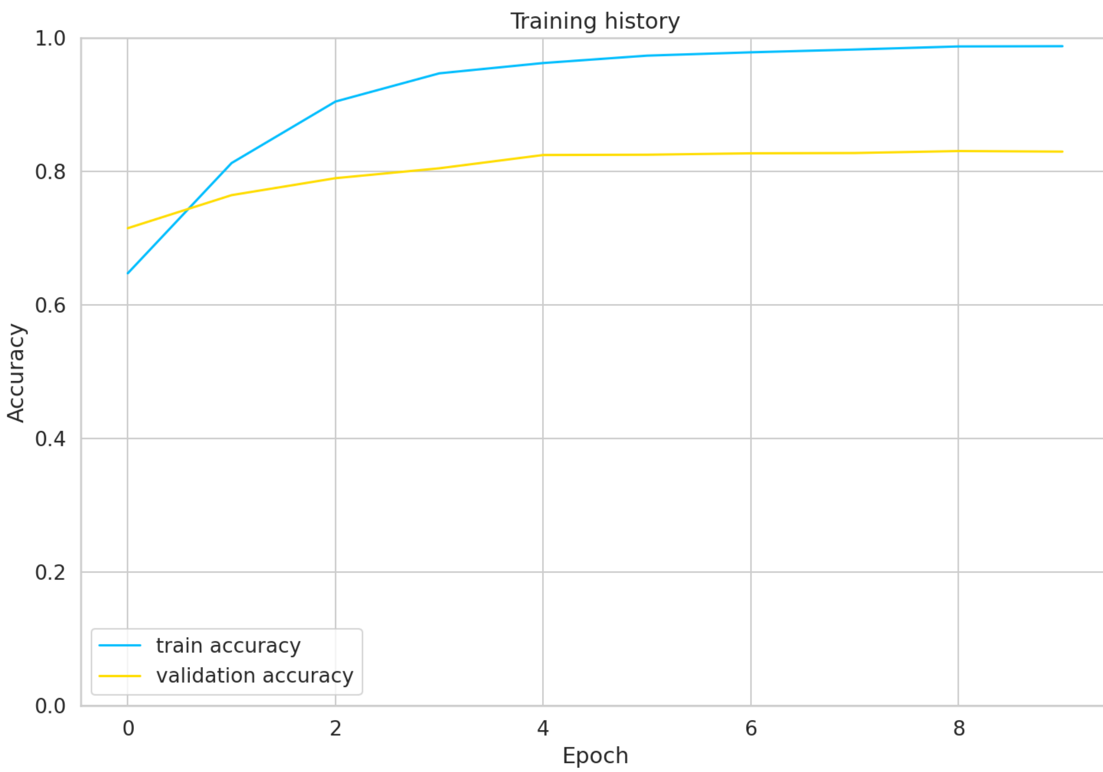
[ ]:
```
%%time

history = defaultdict(list)
best_accuracy = 0

for epoch in range(EPOCHS):
  print(f'Epoch {epoch + 1}/{EPOCHS}')
  print('-' * 10)

  train_acc, train_loss = train_epoch(
      model,
      train_data_loader,
      loss_fn,
      optimizer,
      device,
      scheduler,
      len(df_train)
  )

  print(f'Train loss {train_loss} --- Train accuracy {train_acc}')

  val_acc, val_loss = eval_model(
      model,
      val_data_loader,
      loss_fn,
      device,
      len(df_val)
  )

  print(f'Val loss {val_loss} --- Val accuracy {val_acc}')
  print()

  history['train_acc'].append(train_acc)
  history['train_loss'].append(train_loss)
  history['val_acc'].append(val_acc)
  history['val_loss'].append(val_loss)

  if val_acc > best_accuracy:
    model_save_name = 'best_model_state.pt'
    path = F"/content/drive/My Drive/{model_save_name}"
    torch.save(model.state_dict(), path)

    # torch.save(model.state_dict(), 'best_model_state.bin')
    best_accuracy = val_acc
```

```
Epoch 1/10
----------
```

```
Train loss 0.7674604126607731 --- Train accuracy 0.6470694973688985
Val loss 0.6753895640171863 --- Val accuracy 0.7146486028789162

Epoch 2/10
----------
Train loss 0.4834810435577297 --- Train accuracy 0.8122845218653602
Val loss 0.67174454239776 --- Val accuracy 0.7641828958509738

Epoch 3/10
----------
Train loss 0.28917152965806303 --- Train accuracy 0.9043730720377426
Val loss 0.8130055506638175 --- Val accuracy 0.7895850973751058

Epoch 4/10
----------
Train loss 0.18364281476041402 --- Train accuracy 0.9466521502449646
Val loss 0.9616580203354888 --- Val accuracy 0.8044030482641829

Epoch 5/10
----------
Train loss 0.13613030183980923 --- Train accuracy 0.9620758483033932
Val loss 0.9913723825082196 --- Val accuracy 0.8243014394580863

Epoch 6/10
----------
Train loss 0.09584020892865598 --- Train accuracy 0.9731446198512067
Val loss 1.0787120965762518 --- Val accuracy 0.8247248094834886

Epoch 7/10
----------
Train loss 0.07124643715384724 --- Train accuracy 0.9781346398112866
Val loss 1.1752637787159639 --- Val accuracy 0.8268416596104995

Epoch 8/10
----------
Train loss 0.061142171285499064 --- Train accuracy 0.982308111050626
Val loss 1.2530784735851763 --- Val accuracy 0.8272650296359018

Epoch 9/10
----------
Train loss 0.04363285115154944 --- Train accuracy 0.9869352204681546
Val loss 1.278646860217221 --- Val accuracy 0.8302286198137172

Epoch 10/10
----------
Train loss 0.04183529664257373 --- Train accuracy 0.9872981310107058
Val loss 1.2865800096325395 --- Val accuracy 0.8293818797629128
```

```
CPU times: user 1h 49min 22s, sys: 1min, total: 1h 50min 23s
Wall time: 1h 50min 40s
```

[ ]:
```python
plt.plot(history['train_acc'], label='train accuracy')
plt.plot(history['val_acc'], label='validation accuracy')

plt.title('Training history')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend()
plt.ylim([0, 1]);
```



[ ]:
```python
test_acc, _ = eval_model(
  model,
  test_data_loader,
  loss_fn,
  device,
  len(df_test)
)

test_acc.item()
```

[ ]: 0.8209144792548687

[ ]:
```python
def get_predictions(model, data_loader):
    model = model.eval()

    review_texts = []
    predictions = []
    prediction_probs = []
    real_values = []

    with torch.no_grad():
        for d in data_loader:
            texts = d["review_text"]
            input_ids = d["input_ids"].to(device)
            attention_mask = d["attention_mask"].to(device)
            targets = d["targets"].to(device)

            outputs = model(
            input_ids=input_ids,
            attention_mask=attention_mask
            )
            _, preds = torch.max(outputs, dim=1)

            probs = F.softmax(outputs, dim=1)

            review_texts.extend(texts)
            predictions.extend(preds)
            prediction_probs.extend(probs)
            real_values.extend(targets)

    predictions = torch.stack(predictions).cpu()
    prediction_probs = torch.stack(prediction_probs).cpu()
    real_values = torch.stack(real_values).cpu()
    return review_texts, predictions, prediction_probs, real_values
```

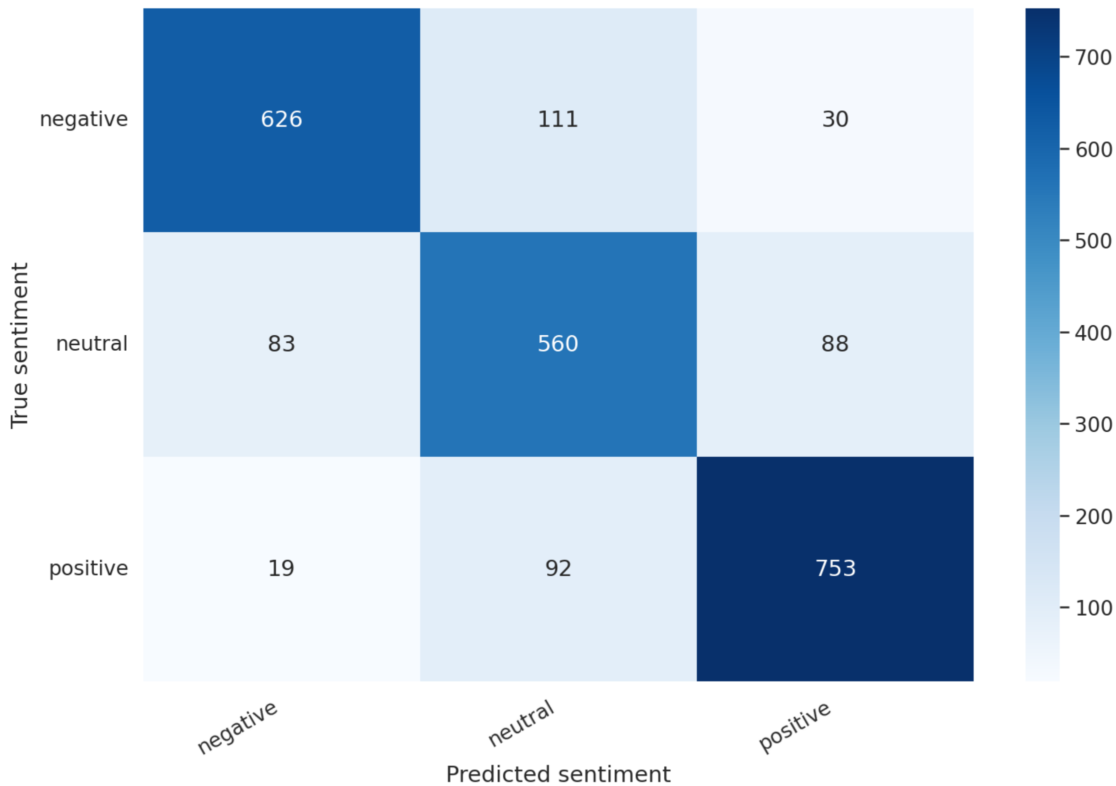
[ ]:
```python
y_review_texts, y_pred, y_pred_probs, y_test = get_predictions(
  model,
  test_data_loader
)
```

[ ]:
```python
print(classification_report(y_test, y_pred, target_names=class_names))
```

```
              precision    recall  f1-score   support

    negative       0.86      0.82      0.84       767
     neutral       0.73      0.77      0.75       731
    positive       0.86      0.87      0.87       864

    accuracy                           0.82      2362
```

```
   macro avg       0.82      0.82      0.82      2362
weighted avg       0.82      0.82      0.82      2362
```

[ ]:
```
def show_confusion_matrix(confusion_matrix):
    hmap = sns.heatmap(confusion_matrix, annot=True, fmt="d", cmap="Blues")
    hmap.yaxis.set_ticklabels(hmap.yaxis.get_ticklabels(), rotation=0,␣
  ↪ha='right')
    hmap.xaxis.set_ticklabels(hmap.xaxis.get_ticklabels(), rotation=30,␣
  ↪ha='right')
    plt.ylabel('True sentiment')
    plt.xlabel('Predicted sentiment');

cm = confusion_matrix(y_test, y_pred)
df_cm = pd.DataFrame(cm, index=class_names, columns=class_names)
show_confusion_matrix(df_cm)
```



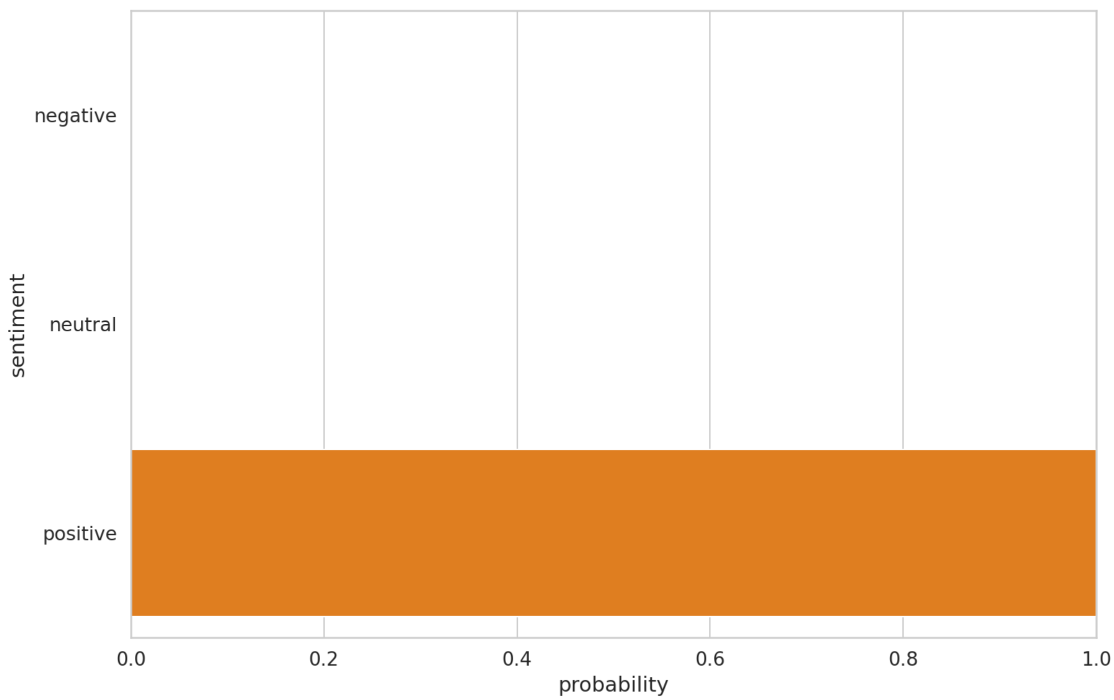
[ ]:
```
idx = 2

review_text = y_review_texts[idx]
true_sentiment = y_test[idx]
pred_df = pd.DataFrame({
```

```
  'class_names': class_names,
  'values': y_pred_probs[idx]
})
```

[ ]:
```
print("\n".join(wrap(review_text)))
print()
print(f'True sentiment: {class_names[true_sentiment]}')
```

```
I have tied a few calenders. This one is good for multiple accounts.
Easy to turn them off and on. Its quite clear on a phone screen

True sentiment: positive
```

[ ]:
```
sns.barplot(x='values', y='class_names', data=pred_df, orient='h')
plt.ylabel('sentiment')
plt.xlabel('probability')
plt.xlim([0, 1]);
```



[ ]:
```
review_text = "maybe i like you."
```

[ ]:
```
encoded_review = tokenizer.encode_plus(
  review_text,
  max_length=MAX_LEN,
  add_special_tokens=True,
```

```
  return_token_type_ids=False,
  pad_to_max_length=True,
  return_attention_mask=True,
  return_tensors='pt',
)
```

[ ]:
```
input_ids = encoded_review['input_ids'].to(device)
attention_mask = encoded_review['attention_mask'].to(device)

output = model(input_ids, attention_mask)
_, prediction = torch.max(output, dim=1)

print(f'Review text: {review_text}')
print(f'Sentiment  : {class_names[prediction]}')
```

```
Review text: maybe i like you.
Sentiment  : neutral
```

[ ]:
```
model_save_name = 'best_model_state.bin'
path = F"/content/drive/My Drive/{model_save_name}"
torch.save(model.state_dict(), path)
# model.load_state_dict(torch.load(path))
```

[ ]:
```
model_save_name = 'best_model_state.bin'
path = F"/content/drive/My Drive/{model_save_name}"
model.load_state_dict(torch.load(path))
```

[ ]: <All keys matched successfully>

[ ]:
```
sentiment_test = "maybe i like you."
```

[ ]:
```
encoded_review = tokenizer.encode_plus(
  sentiment_test,
  max_length=MAX_LEN,
  add_special_tokens=True,
  return_token_type_ids=False,
  pad_to_max_length=True,
  return_attention_mask=True,
  return_tensors='pt',
)
```

[ ]:
```
input_ids = encoded_review['input_ids'].to(device)
attention_mask = encoded_review['attention_mask'].to(device)

output = model(input_ids, attention_mask)
_, prediction = torch.max(output, dim=1)
```

```python
print(f'Review text: {sentiment_test}')
print(f'Sentiment  : {class_names[prediction]}')
```

```
Review text: maybe i like you.
Sentiment  : neutral
```

[ ]:

```python

```