

▼ Installing Dependencies

```
!pip install transformers
```



Looking in indexes: <https://pypi.org/simple>, <https://us-python.pkg.dev/colab-wheels/public/simple/>

Collecting transformers

Downloading transformers-4.20.1-py3-none-any.whl (4.4 MB)

|██| 4.4 MB 31.1 MB/s

Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.7/dist-packages (from transformers) (4.64.0)

Collecting pyyaml>=5.1

Downloading PyYAML-6.0-cp37-cp37m-manylinux_2_5_x86_64.manylinux1_x86_64.manylinux_2_12_x86_64.manylinux2010_x86_64.whl (596 kB)

|██| 596 kB 58.0 MB/s

Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.7/dist-packages (from transformers) (2022.6.0)

Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.7/dist-packages (from transformers) (1.21.6)

Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.7/dist-packages (from transformers) (21.3)

Requirement already satisfied: requests in /usr/local/lib/python3.7/dist-packages (from transformers) (2.23.0)

Collecting tokenizers!=0.11.3,<0.13,>=0.11.1

Downloading tokenizers-0.12.1-cp37-cp37m-manylinux_2_12_x86_64.manylinux2010_x86_64.whl (6.6 MB)

|██| 6.6 MB 59.9 MB/s

Requirement already satisfied: importlib-metadata in /usr/local/lib/python3.7/dist-packages (from transformers) (4.12.0)

Requirement already satisfied: filelock in /usr/local/lib/python3.7/dist-packages (from transformers) (3.7.1)

Collecting huggingface-hub<1.0,>=0.1.0

Downloading huggingface_hub-0.8.1-py3-none-any.whl (101 kB)

|██| 101 kB 13.9 MB/s

Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.7/dist-packages (from huggingface-hub) (4.5.0)

Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dist-packages (from packaging>=20.0) (3.0.9)

Requirement already satisfied: zipp>=0.5 in /usr/local/lib/python3.7/dist-packages (from importlib-metadata->transformers) (3.7.0)

Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/dist-packages (from requests->transformers) (3.4)

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) (1.26.13)

Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.7/dist-packages (from requests->transformers) (2022.9.24)

Installing collected packages: pyyaml, tokenizers, 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-0.8.1 pyyaml-6.0 tokenizers-0.12.1 transformers-4.20.1

```
import transformers
from transformers import BertModel, BertTokenizer, AdamW, get_linear_schedule_with_warmup
import torch
import gc
gc.collect()
torch.cuda.empty_cache()

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

%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")
```

```
device
```

```
device(type='cuda', index=0)
```

▼ Read the Data

```
df = pd.read_csv("/content/Pelabelan1.csv")
df.head()
```

	category	review content	lemma	rating	Subjectivity	Polarity	TextBlob	Vader Sentiment	vaderSentiment
0	Headsets	This gaming headset ticks all the boxes # look...	['game', 'headset', 'tick', 'box', 'look', 'gr...	5	0.583333	0.305556	Positive	0.8720	Positive
1	Headsets	Easy setup, rated for 6 hours battery but mine...	['easy', 'setup', 'rat', 'hours', 'battery', '...	3	0.546289	0.203782	Positive	0.9657	Positive
		I originally bought	I originally 'bu						

```
df.shape
```

```
(44756, 9)
```

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 44756 entries, 0 to 44755
Data columns (total 9 columns):
#   Column          Non-Null Count  Dtype
---  -
0   category        44756 non-null  object
1   review content  44756 non-null  object
2   lemma           44756 non-null  object
3   rating          44756 non-null  int64
4   Subjectivity    44756 non-null  float64
```

```
5  Polarity          44756 non-null  float64
6  TextBlob          44756 non-null  object
7  Vader Sentiment   44756 non-null  float64
8  vaderSentiment    44756 non-null  object
dtypes: float64(3), int64(1), object(5)
memory usage: 3.1+ MB
```

```
sns.countplot(df.rating)
plt.xlabel('review score');
```

/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword argument: 'FutureWarning'

We have a balanced dataset with respect to reviews (as we scraped it in a balanced way)

```
55000 |
def group_sentiment(rating):
    #rating = int(rating)
    if rating <= 2:
        return 0          # Negative sentiment
    elif rating == 3:
        return 1          # Neutral Sentiment
    else:
        return 2          # positive Sentiment

df['TextBlob'] = df.rating.apply(group_sentiment)
```

```
15000 |
class_names = ['negative', 'neutral', 'positive']
```

```
10000 |
ax = sns.countplot(df.TextBlob)
plt.xlabel('review sentiment')
ax.set_xticklabels(class_names);
```

```
/usr/local/lib/python3.7/dist-packages/seaborn/_decorators.py:43: FutureWarning: Pass the following variable as a keyword argument: FutureWarning
```



▼ Data Preprocessing

Machine Learning models don't work with raw text. We need to convert text to numbers (of some sort). BERT requires even more attention. Here are the requirements:

- Add special tokens to separate sentences and do classification
- Pass sequences of constant length (introduce padding)
- Create array of 0s (pad token) and 1s (real token) called attention mask

The Transformers library provides a wide variety of Transformer models (including BERT). It also includes prebuilt tokenizers that solves most of the load required for pre-processing

```
PRE_TRAINED_MODEL_NAME = 'bert-base-cased'
```

Let's load a pre-trained BertTokenizer:

```
tokenizer = BertTokenizer.from_pretrained(PRE_TRAINED_MODEL_NAME)
```

Downloading: 100%

208k/208k [00:00<00:00, 3.95MB/s]

Downloading: 100%

29.0/29.0 [00:00<00:00, 931B/s]

Downloading: 100%

570/570 [00:00<00:00, 17.2kB/s]

The requirements of special tokens which indicate separation between sentences is taken care by the tokenizer provided by the huggingface. BERT was trained for question and answering task but we are using it to train with a sequence of sentences as input. We require sequences of equal length which is done by padding tokens of zero meaning without loss of generality to the end of sentences.

We also need to pass attention mask which is basically passing a value of 1 to all the tokens which have meaning and 0 to padding tokens.

- ▶ We'll use this simple text to understand the tokenization process:

[] ↳ 3 cells hidden

- ▶ Special Tokens

[SEP] - marker for ending of a sentence

[] ↳ 7 cells hidden

- ▶ All of the above work can be done using the `encode_plus()` method

[] ↳ 5 cells hidden

- ▶ Choosing Sequence Length

[]

[] ↳ 11 cells hidden

▼ Splitting into train and validation sets

```
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, df_val.shape, df_test.shape
```

```
((31329, 9), (6713, 9), (6714, 9))
```

- ▶ Create data loaders for to feed as input to our model. The below function does that.

[] ↳ 5 cells hidden

▼ Sentiment Classification with BERT and Hugging Face

There are a lot of helpers that make using BERT easy with the Transformers library. Depending on the task we might use "BertForSequenceClassification", "BertForQuestionAnswering" or something else.

We'll use the basic BertModel and build our sentiment classifier on top of it. Let's load the model:

```
bert_model = BertModel.from_pretrained(PRE_TRAINED_MODEL_NAME)
```


Downloading: 100%

416M/416M [00:07<00:00, 58.0MB/s]

Some weights of the model checkpoint at bert-base-uncased were not used when initializing BertModel: ['cls', 'pooler_output']

```
model_test = bert_model(  
    input_ids=encoding_test['input_ids'],  
    attention_mask=encoding_test['attention_mask']  
)  
model_test.keys()  
  
dict_keys(['last_hidden_state', 'pooler_output'])
```

The "last_hidden_state" is a sequence of hidden states of the last layer of the model. Obtaining the "pooled_output" is done by applying the BertPooler which basically applies the tanh function to pool all the outputs.

```
last_hidden_state=model_test['last_hidden_state']  
pooled_output=model_test['pooler_output']
```

```
last_hidden_state.shape
```

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

We have the hidden state for each of our 32 tokens (the length of our example sequence) and 768 is the number of hidden units in the feedforward-networks. We can verify that by checking the config:

```
bert_model.config.hidden_size
```

```
768
```

We can think of the pooled_output as a summary of the content, according to BERT. Let's look at the shape of the output:

```
pooled_output.shape
```

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

We can use all this knowledge to create a sentiment classifier that uses the BERT model:

```
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) ## For regularization with dropout probability 0.3.
        self.out = nn.Linear(self.bert.config.hidden_size, n_classes) ## append an Output fully connected layer representing the

    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)
```

The classifier delegates most of the work to the BertModel. We use a dropout layer for some regularization and a fully-connected layer for our output. We're returning the raw output of the last layer since that is required for the cross-entropy loss function in PyTorch to work.

This should work like any other PyTorch model. Create an instance and move it to the GPU:

```
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.seq_relationship']

- This IS expected if you are initializing BertModel from the checkpoint of a model trained on another task or with another architecture.
- This IS NOT expected if you are initializing BertModel from the checkpoint of a model that you expect to be exactly identical to.

We'll move the example batch of our training data created above using dataloader to the GPU:

```

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([8, 160])
torch.Size([8, 160])

```

To get the predicted probabilities from our trained model, we'll apply the softmax function to the output obtained from the output layer:

```

F.softmax(model(input_ids, attention_mask), dim=1)

tensor([[0.2454, 0.3799, 0.3747],
        [0.2231, 0.3302, 0.4467],
        [0.3508, 0.2543, 0.3949],
        [0.2469, 0.3208, 0.4323],
        [0.5969, 0.1803, 0.2228],
        [0.2482, 0.3598, 0.3920],
        [0.2840, 0.3073, 0.4086],
        [0.3782, 0.2312, 0.3906]], device='cuda:0', grad_fn=<SoftmaxBackward0>)

```

▼ Training the model

To reproduce the training procedure from the BERT paper, we'll use the AdamW optimizer provided by Hugging Face. It corrects weight decay, so it's similar to the original paper. We'll also use a linear scheduler with no warmup steps:

```

EPOCHS = 10

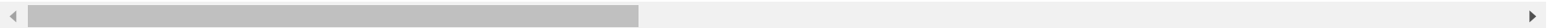
optimizer = AdamW(model.parameters(), lr=2e-5, correct_bias=False)
total_steps = len(train_data_loader) * EPOCHS      # Number of batches * Epochs (Required for the scheduler.)

```

```
scheduler = get_linear_schedule_with_warmup(
    optimizer,
    num_warmup_steps=0,      # Recommended in the BERT paper.
    num_training_steps=total_steps
)
```

```
loss_fn = nn.CrossEntropyLoss().to(device)
```

```
/usr/local/lib/python3.7/dist-packages/transformers/optimization.py:310: FutureWarning: This implementation of AdamW is
FutureWarning,
```



The BERT authors have some recommendations for fine-tuning:

- Batch size: 16, 32
- Learning rate (Adam): 5e-5, 3e-5, 2e-5
- Number of epochs: 2, 3, 4

Except for the number of epochs recommendation We'll stick with the rest. Increasing the batch size reduces the training time significantly, but gives lower accuracy.

▼ Helper function for training our model for one epoch:

```
def train_epoch(
    model,
    data_loader,
    loss_fn,
    optimizer,
    device,
    scheduler,
    n_examples
):
    model = model.train()    # To make sure that the dropout and normalization is enabled during the training.
```

```

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
    )

    max_prob, preds = torch.max(outputs, dim=1)    # Returns 2 tensors, one with max_probability and another with the respect
    loss = loss_fn(outputs, targets)

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

    loss.backward()    # Back_Propagation
    nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0) # Recommended by the BERT paper to clip the gradients to avoid
    optimizer.step()
    scheduler.step()
    optimizer.zero_grad()

return correct_predictions.double() / n_examples, np.mean(losses)    # Return the mean loss and the ratio of correct predictions

```

Training the model is similar to training a deep neural network, except for two things. The scheduler gets called every time a batch is fed to the model. We're avoiding exploding gradients by clipping the gradients of the model using `clip_grad_norm`

▼ Helper function to evaluate the model on a given data loader:

```

def eval_model(model, data_loader, loss_fn, device, n_examples):
    model = model.eval()    # To make sure that the dropout and normalization is disabled during the training.

```

```

losses = []
correct_predictions = 0

with torch.no_grad():          # Back propogation is not required. Torch would perform faster.
    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
        )
        max_prob, 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)

```

Using these two helper functions, we can write our training loop. We'll also store the training history:

```

%%time

history = defaultdict(list)          # Similar to Keras library saves history
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} 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} 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:
    torch.save(model.state_dict(), 'best_model_state.bin')
    best_accuracy = val_acc

```

```

/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2307: FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future version. Please use the `padding` argument instead.
FutureWarning,
Epoch 1/10
-----
Train loss 0.2500950274506347 accuracy 0.9387149286603467
Val   loss 0.19400512281205184 accuracy 0.9474154625353791

Epoch 2/10
-----
Train loss 0.1879689592559977 accuracy 0.9539723578792813

```

Val loss 0.27577801976821364 accuracy 0.9484582154029495

Epoch 3/10

Train loss 0.15263224061496886 accuracy 0.9635162309681127

Val loss 0.2956123263478644 accuracy 0.9468196037539104

Epoch 4/10

Train loss 0.1229035550237561 accuracy 0.9723897985891666

Val loss 0.32717882809996635 accuracy 0.9387755102040817

Epoch 5/10

Train loss 0.09349571792390769 accuracy 0.9805611414344537

Val loss 0.32558039127489746 accuracy 0.9399672277670192

Epoch 6/10

Train loss 0.06958505261750292 accuracy 0.9865619713364615

Val loss 0.3569465411736365 accuracy 0.9402651571577536

Epoch 7/10

Train loss 0.04786165051572958 accuracy 0.9910945130709566

Val loss 0.41341221163188707 accuracy 0.9432444510650976

Epoch 8/10

Train loss 0.035464768701965195 accuracy 0.9938076542500559

Val loss 0.4729398164600709 accuracy 0.9305824519588858

Epoch 9/10

Train loss 0.02388020522153619 accuracy 0.9959781671933353

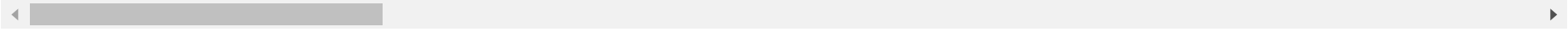
Val loss 0.47767839577625293 accuracy 0.939073439594816

Epoch 10/10

Train loss 0.019289403616402342 accuracy 0.9968719078170386

Val loss 0.46808376536826557 accuracy 0.9410099806345896


```
CPU times: user 1h 59min 25s, sys: 46min 52s, total: 2h 46min 18s
Wall time: 2h 47min 9s
```



Note that we're storing the state of the best model, indicated by the highest validation accuracy.

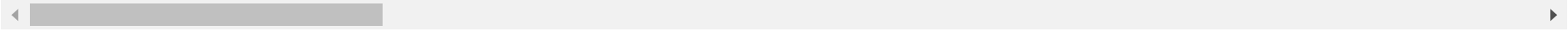
▼ Evaluation

So how good is our model on predicting sentiment? Let's start by calculating the accuracy on the test data:

```
test_acc, _ = eval_model(
    model,
    test_data_loader,
    loss_fn,
    device,
    len(df_test)
)

test_acc.item()
```

```
/usr/local/lib/python3.7/dist-packages/transformers/tokenization_utils_base.py:2307: FutureWarning: The `pad_to_max_length` argument is deprecated and will be removed in a future version. Please use the `padding` argument instead.
FutureWarning,
0.9413166517724159
```



The accuracy is about 1% lower on the test set. Our model seems to generalize well.

We'll define a helper function to get the predictions from our model:

```
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

```

This is similar to the evaluation function, except that we're storing the text of the reviews and the predicted probabilities (by applying the softmax on the model outputs):

```

y_review_texts, y_pred, y_pred_probs, y_test = get_predictions(
    model,
    test_data_loader
)

```

Let's have a look at the classification report

```
print(classification_report(y_test, y_pred, target_names=class_names))
```

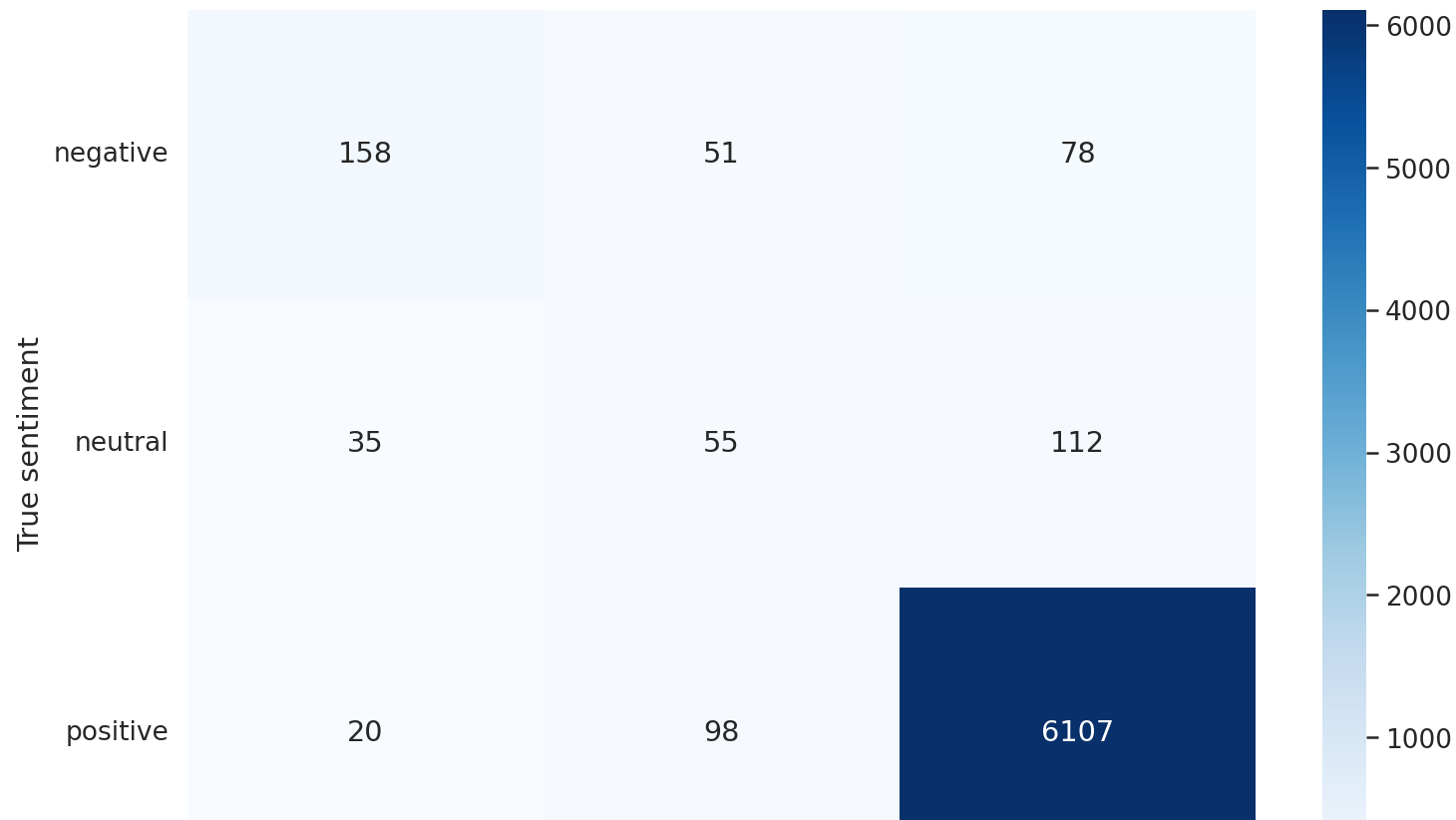
	precision	recall	f1-score	support
negative	0.74	0.55	0.63	287
neutral	0.27	0.27	0.27	202
positive	0.97	0.98	0.98	6225
accuracy			0.94	6714
macro avg	0.66	0.60	0.63	6714
weighted avg	0.94	0.94	0.94	6714

Looks like it is really hard to classify neutral (3 stars) reviews. And I can tell you from experience, looking at many reviews, those are hard to classify.

We'll continue with the confusion matrix:

```
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)
```



This confirms that our model is having difficulty classifying neutral reviews. It mistakes those for negative and positive at a roughly equal frequency.

That's a good overview of the performance of our model. But let's have a look at an example from our test data:

```
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]}')
```

Now we can look at the confidence of each sentiment of our model:

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

► Predicting on Raw Text

Let's use our model to predict the sentiment of some raw text:

```
[ ] ↳ 6 cells hidden
```

