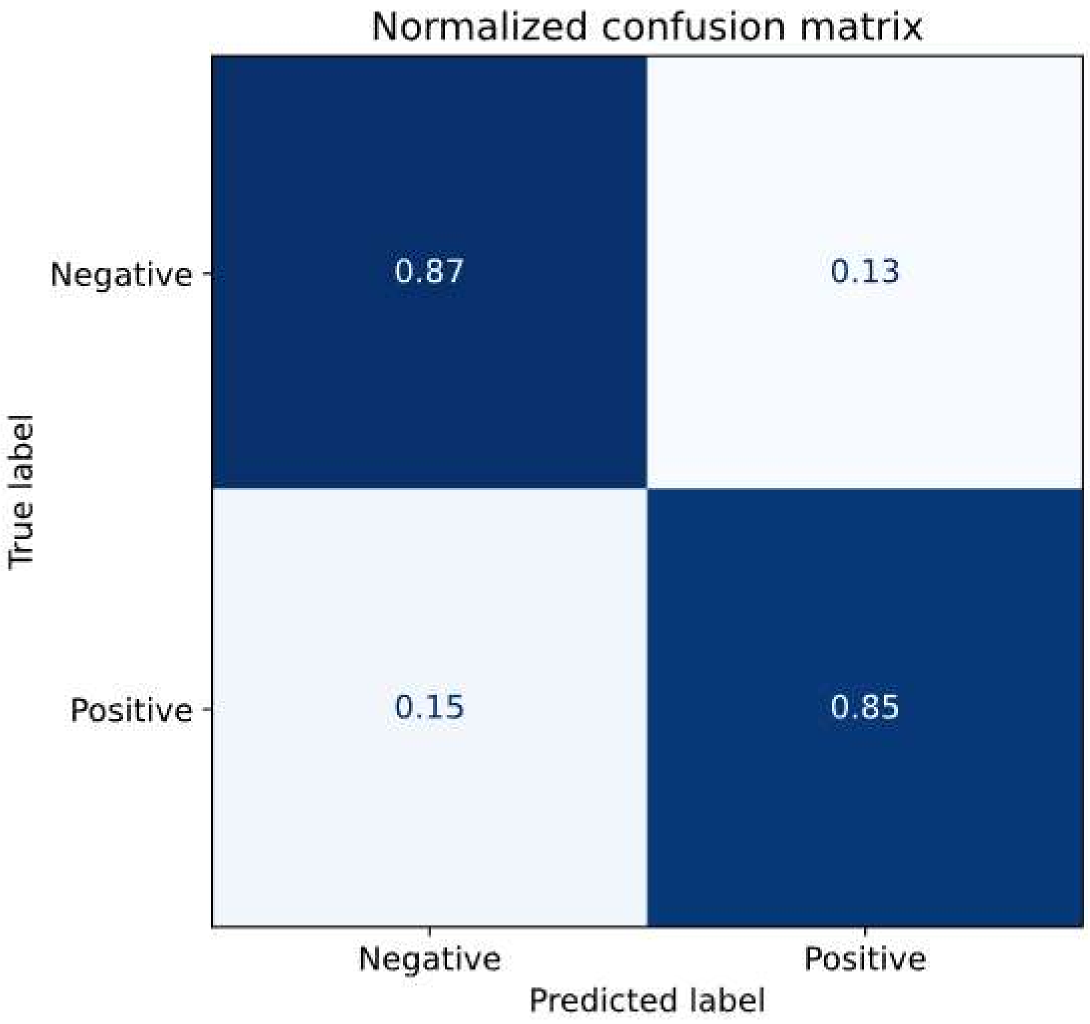
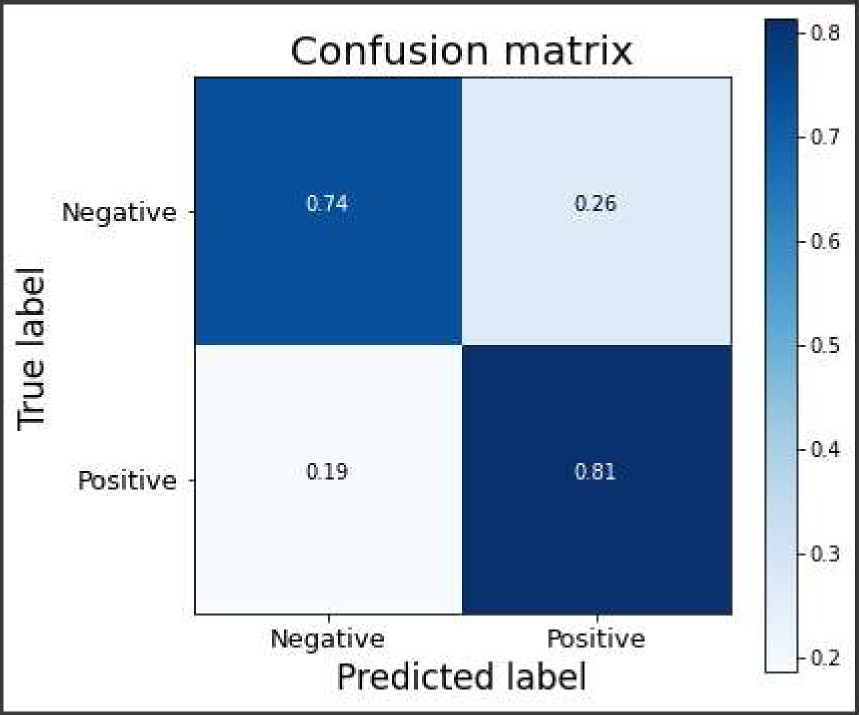
In this lab I explore the difference in results between LSTM sentiment analysis and transformer sentiment analysis. This exercise was a lot more difficult, mostly because the huggingface api was quite challenging to work with. After two epochs of training we were getting good results.



Comparing this to the confusion matrix of the LSTM results, the transformer approach does significantly better, reducing misclassification by about 50%.



Classification Scores

```
[ ]  print(classification_report(list(test_data.sentiment), y_pred_
```

```
              precision    recall  f1-score   support

    Negative       0.80      0.74      0.77     80313
    Positive       0.75      0.81      0.78     79687

    accuracy                           0.78    160000
   macro avg       0.78      0.78      0.78    160000
weighted avg       0.78      0.78      0.78    160000
```

I was not able to do the excersises were we extract the last hidden state but I assume it wouldn't be better than the lstm approach.

Importing Dependencies

We shall start by importing all the neccessary libraries for preprocessing.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
import re

%matplotlib inline
```

Dataset Preprocessing

In this notebook, I am using **Sentiment-140** from Kaggle. It contains a labels data of 1.6 Million Tweets and I find it a good amount of data to train our model.

```
df = pd.read_csv('/content/training.1600000.processed.noemoticon.csv',
                 encoding='latin', header=None)
df.columns = ['sentiment', 'id', 'date', 'query', 'user_id', 'text']
df = df.drop(['id', 'date', 'query', 'user_id'], axis=1)
df.head()
```

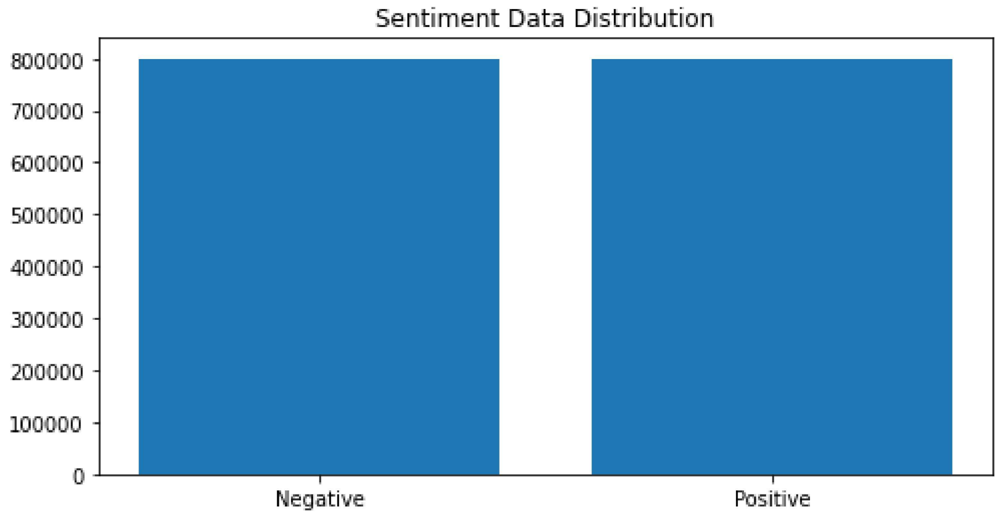
	sentiment	text
0	0	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	0	is upset that he can't update his Facebook by ...
2	0	@Kenichan I dived many times for the ball. Man...
3	0	my whole body feels itchy and like its on fire
4	0	@nationwideclass no, it's not behaving at all....

```
lab_to_sentiment = {0:"Negative", 4:"Positive"}
def label_decoder(label):
  return lab_to_sentiment[label]
df.sentiment = df.sentiment.apply(lambda x: label_decoder(x))
df.head()
```

	sentiment	text
0	Negative	@switchfoot http://twitpic.com/2y1zl - Awww, t...
1	Negative	is upset that he can't update his Facebook by ...
2	Negative	@Kenichan I dived many times for the ball. Man...
3	Negative	my whole body feels itchy and like its on fire
4	Negative	@nationwideclass no, it's not behaving at all....

```
val_count = df.sentiment.value_counts()

plt.figure(figsize=(8,4))
plt.bar(val_count.index, val_count.values)
plt.title("Sentiment Data Distribution")
```

```
Text(0.5, 1.0, 'Sentiment Data Distribution')
```



```
import random
random_idx_list = [random.randint(1,len(df.text)) for i in range(10)] # creates random indexe
df.loc[random_idx_list,:].head(10) # Returns the rows with the index and display it
```

	sentiment	text
769068	Negative	@grigs Did you happen to get a recording of th...
964918	Positive	The Norwegian tune (Winner of the 2009 Eurovi...
1235153	Positive	tweeps! good morning
1192199	Positive	@makenai Haha. Yes. Because you went camping. ...
871737	Positive	@Torae LOL hahaha.. i just wanted to make sure...
284093	Negative	Wish I could be in SF tonight...Brian Greenber...
254697	Negative	@facunditas Nope , but will do in next few we...
298069	Negative	Thank you for making Sims 3 Mac-friendly! But ...
1409970	Positive	@Bella1876432 Yeah i know rl i havnt fallen ei...
1111052	Positive	hi michelle! because youre a stalker and you g...

Text Preprocessing

Tweet texts often consists of other user mentions, hyperlink texts, emoticons and punctuations. In order to use them for learning using a Language Model. We cannot permit those texts for training a model. So we have to clean the text data using various preprocessing and cleansing methods.

We could check for stop words, lemeitization, and special characters. For initial modeling I will just clean special characters and add Lemmatization later.

An important change made was that I kept all contractions as a single word, for example: can't --> can't, couldn't --> couldnt. This is important because the way it was done before we would turn can't into can and "t", two tokens needing to be processed sequentially which may or may not be helpful.

```
from nltk.stem import SnowballStemmer

stemmer = SnowballStemmer('english')
text_cleaning_re = "@\S+|https?:\S+|http?:\S|[^A-Za-z0-9]+"

def preprocess(text, stem=False):
  text = re.sub("(?<=\w)[‘`’'](?=\w)", '', text)
  text = re.sub(text_cleaning_re, ' ', str(text).lower()).strip()
  tokens = []
  for token in text.split():
    if stem:
      tokens.append(stemmer.stem(token))
    else:
      tokens.append(token)
  return " ".join(tokens)
```

```
df.text = df.text.apply(lambda x: preprocess(x))
df = df[['text', 'sentiment']]
df["text"] = df["text"].astype(str)
```

```
df
```

	text	sentiment
0	awww thats a bummer you shoulda got david carr...	Negative
1	is upset that he cant update his facebook by t...	Negative

```
sentiment_to_lab = {"Negative":0, "Positive":1}
def label_encoder(label):
  return sentiment_to_lab[label]
df.sentiment = df.sentiment.apply(lambda x: label_encoder(x))
df = df.dropna()
df.head()
```

	text	sentiment
0	awww thats a bummer you shoulda got david carr...	0
1	is upset that he cant update his facebook by t...	0
2	i dived many times for the ball managed to sav...	0
3	my whole body feels itchy and like its on fire	0
4	no its not behaving at all im mad why am i her...	0

```
train_data, test_data = train_test_split(df, test_size = 0.1,
                                         random_state = 42) # Splits Dataset into Training ar
train_data, val_data = train_test_split(train_data, test_size = 0.20,
                                         random_state = 42) # Splits Dataset into Training ar
print("Train Data size:", len(train_data))
print("Val Data size", len(val_data))
print("Test Data size", len(test_data))
```

```
     Train Data size: 1152000
     Val Data size 288000
     Test Data size 160000
```

```
train_data.head(10)
```

	text	sentiment
978125	and losanjalis dont forget the flashlight by b...	1
607634	overslept this morning but not going to quit w...	0
5518	oscar is getting ready to be neutered	0

```
test_data.dropna().to_csv('/content/test.csv', header=False, sep=';')
val_data.dropna().to_csv('/content/val.csv', header=False, sep=';')
train_data.dropna().to_csv('/content/train.csv', header=False, sep=';')
```

563229	so little time so much to do	0

```
#Uncomment and run this cell if you're on Colab or Kaggle
!git clone https://github.com/nlp-with-transformers/notebooks.git
%cd notebooks
from install import *
install_requirements(is_chapter2=True)
from utils import *
setup_chapter()
```

```
Cloning into 'notebooks'...
remote: Enumerating objects: 451, done.
remote: Counting objects: 100% (100/100), done.
remote: Compressing objects: 100% (24/24), done.
remote: Total 451 (delta 81), reused 78 (delta 76), pack-reused 351
Receiving objects: 100% (451/451), 28.47 MiB | 33.01 MiB/s, done.
Resolving deltas: 100% (209/209), done.
/content/notebooks
⏳ Installing base requirements ...
✅ Base requirements installed!
⏳ Installing Git LFS ...
✅ Git LFS installed!
Using transformers v4.13.0
Using datasets v1.16.1
```

```
from datasets import load_dataset
dataset = load_dataset("csv", data_files = {"train":"/content/train.csv",
                                            "val":"/content/val.csv",
                                            "test":"/content/test.csv",
                                            },
                      names=["text", "label"],
                       sep=';')
```

```
Downloading and preparing dataset csv/default to /root/.cache/huggingface/datasets
100%                                          3/3 [00:00<00:00, 120.83it/s]

100%                                          3/3 [00:00<00:00, 69.89it/s]
Dataset csv downloaded and prepared to /root/.cache/huggingface/datasets/csv/defau
100%                                          3/3 [00:00<00:00, 87.19it/s]
```

```
dataset.set_format(type="pandas")
```

```
df = dataset['train'][:]
df = df.iloc[:, :-1]
df.head()
```

	text	label
0	and losanjalis dont forget the flashlight by b...	1
1	overslept this morning but not going to quit w...	0
2	oscar is getting ready to be neutered	0
3	to all the momoams colleagues great event hope...	1
4	got loadsa fucking bites all over me bet its f...	0

```
print(dataset["train"]["text"][404])
print(dataset["train"]["text"][405])
```

```
huhuhuhu they left already i wish we did more than chat and play
None
```

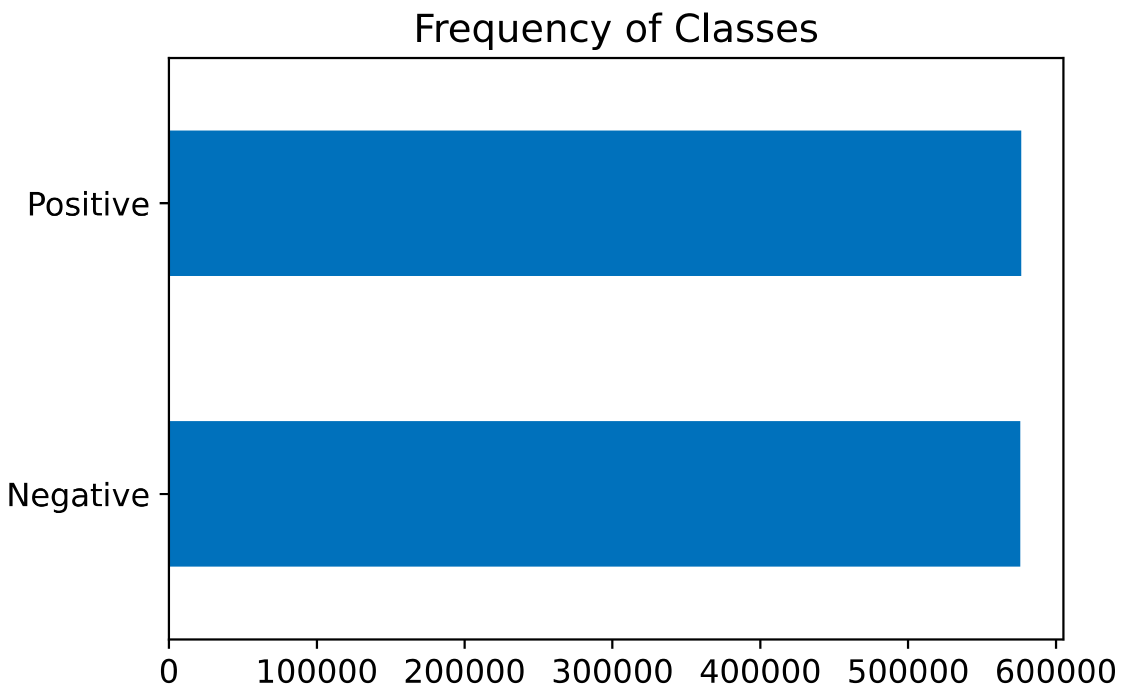
```
dataset['train'] = dataset['train'].filter(lambda x: x["text"] != None)
dataset['val'] = dataset['val'].filter(lambda x: x["text"] != None)
dataset['test'] = dataset['test'].filter(lambda x: x["text"] != None)
'''
test = dataset.copy()
test['train'] = test['train'].filter(lambda x: x["text"] != None)
'''
```

```
100%                                            1152/1152 [00:26<00:00, 44.89ba/s]

100%                                            288/288 [00:06<00:00, 45.20ba/s]

100%                                            160/160 [00:03<00:00, 44.38ba/s]
'\ntest = dataset.copy()\ntest[\'train\'] = test[\'train\'].filter(lambda x: x["t
ext"] != None)\n'
```

```
lab_to_sentiment = {0:"Negative", 1:"Positive"}
def label_decoder(label):
  return lab_to_sentiment[label]
df["label_name"] = df.label.apply(lambda x: label_decoder(x))
df.head()
```

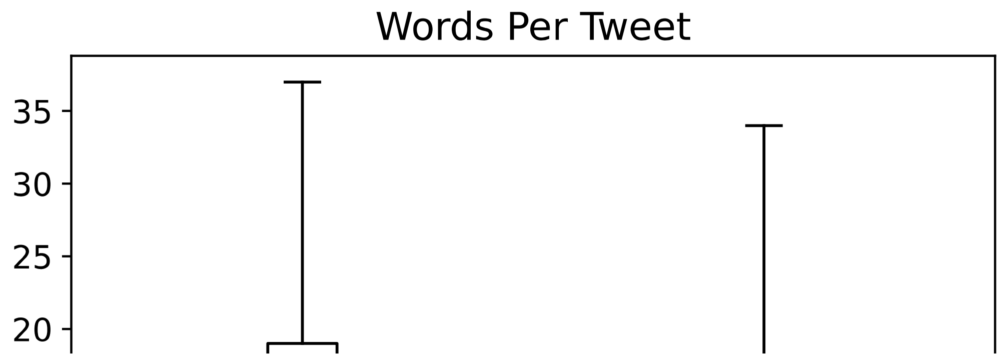
	text	label	label_name
0	and losanjalis dont forget the flashlight by b...	1	Positive
1	overslept this morning but not going to quit w...	0	Negative
2	oscar is getting ready to be neutered	0	Negative
3	to all the momoams colleagues great event hope...	1	Positive

```
df["label_name"].value_counts(ascending=True).plot.barh()
plt.title("Frequency of Classes")
plt.show()
```



```
df.loc[:, "Words Per Tweet"] = df["text"].str.split().str.len()
df.boxplot("Words Per Tweet", by="label_name", grid=False, showfliers=False,
           color="black")
plt.suptitle("")
plt.xlabel("")
plt.show()
```

```
/usr/local/lib/python3.7/dist-packages/matplotlib/cbook/__init__.py:1376: VisibleD
  X = np.atleast_1d(X.T if isinstance(X, np.ndarray) else np.asarray(X))
```



From the plot we see that for each emotion, most tweets are around 15 words long and the longest tweets are well below DistilBERT's maximum context size. Texts that are longer than a model's context size need to be truncated, which can lead to a loss in performance if the truncated text contains crucial information; in this case, it looks like that won't be an issue.

Let's now figure out how we can convert these raw texts into a format suitable for Transformers! While we're at it, let's also reset the output format of our dataset since we don't need the DataFrame format anymore:

```
dataset.reset_format()
```

Subword Tokenization

The basic idea behind subword tokenization is to combine the best aspects of character and word tokenization. On the one hand, we want to split rare words into smaller units to allow the model to deal with complex words and misspellings. On the other hand, we want to keep frequent words as unique entities so that we can keep the length of our inputs to a manageable size. The main distinguishing feature of subword tokenization (as well as word tokenization) is that it is learned from the pretraining corpus using a mix of statistical rules and algorithms.

There are several subword tokenization algorithms that are commonly used in NLP, but let's start with WordPiece,footnote:[M. Schuster and K. Nakajima, "Japanese and Korean Voice Search," 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (2012): 5149–5152, https://doi.org/10.1109/ICASSP.2012.6289079.] which is used by the BERT and DistilBERT tokenizers. The easiest way to understand how WordPiece works is to see it in action. image:images/logo.png[hf,13,13] Transformers provides a convenient AutoTokenizer class that allows you to quickly load the tokenizer associated with a pretrained model—we just call its from_pretrained() method, providing the ID of a model on the Hub or a local file path. Let's start by loading the tokenizer for DistilBERT:

```
from transformers import AutoTokenizer

model_ckpt = "distilbert-base-uncased"
tokenizer = AutoTokenizer.from_pretrained(model_ckpt)
```

```
Downloading: 100%                          28.0/28.0 [00:00<00:00, 1.03kB/s]

Downloading: 100%                          483/483 [00:00<00:00, 18.8kB/s]

Downloading:                               226k/226k [00:00<00:00,
100%                                       3.08MB/s]
```

Let's examine how this tokenizer works by feeding it our simple "Tokenizing text is a core task of NLP." example text:

```
encoded_text = tokenizer("Tokenizing text is a core task of NLP")
print(encoded_text)
```

```
{'input_ids': [101, 19204, 6026, 3793, 2003, 1037, 4563, 4708, 1997, 17953,
2361, 102], 'attention_mask': [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]}
```

```
tokens = tokenizer.convert_ids_to_tokens(encoded_text.input_ids)
print(tokens)
```

```
['[CLS]', 'token', '##izing', 'text', 'is', 'a', 'core', 'task', 'of', 'nl',
'##p', '[SEP]']
```

```
print(tokenizer.convert_tokens_to_string(tokens))
```

```
[CLS] tokenizing text is a core task of nlp [SEP]
```

```
tokenizer.vocab_size
```

```
30522
```

```
tokenizer.model_max_length
```

```
512
```

```
tokenizer.model_input_names
```

```
['input_ids', 'attention_mask']
```

Tokenizing the Whole Dataset

To tokenize the whole corpus, we'll use the map() method of our DatasetDict object. We'll encounter this method many times throughout this book, as it provides a convenient way to apply a processing function to each element in a dataset. As we'll soon see, the map() method can also be used to create new rows and columns.

To get started, the first thing we need is a processing function to tokenize our examples with:

```
def tokenize(batch):
    if batch["text"] != None:
      return tokenizer(batch["text"], padding=True, truncation=True)
```

```
print(tokenize(dataset["train"][:2]))
```

```
{'input_ids': [[101, 1998, 3050, 2319, 28948, 2015, 2123, 2102, 5293, 1996,
15257, 2011, 2793, 7540, 7347, 1998, 2562, 2115, 3526, 3042, 13003, 2058, 2305,
102, 0, 0, 0, 0, 0, 0, 0, 0], [101, 15849, 2571, 13876, 2023, 2851, 2021, 2025,
2183, 2000, 8046, 2007, 2026, 17786, 2000, 2131, 1999, 2488, 4338, 8186, 1998,
6259, 2135, 1015, 5199, 1018, 1021, 1022, 1051, 6300, 2050, 102]],
'attention_mask': [[1, 1, 1, 1, 1, 1, 1, 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], [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1]]}
```

```
dataset["train"]
```

```
Dataset({
    features: ['text', 'label', '__index_level_0__'],
    num_rows: 1149527
})
```

```
dataset_enc = dataset.map(tokenize, batched=True, batch_size=True)
```

```
100%                                    1149527/1149527 [12:34<00:00, 1532.00ba/s]

100%                                    287403/287403 [03:08<00:00, 1527.04ba/s]

100%                                    159624/159624 [01:44<00:00, 1513.34ba/s]
```

```
print(dataset_enc["train"].column_names)
```

```
['__index_level_0__', 'attention_mask', 'input_ids', 'label', 'text']
```

Here we can see the result of padding: the first element of input_ids is shorter than the second, so zeros have been added to that element to make them the same length. These zeros have a corresponding [PAD] token in the vocabulary, and the set of special tokens also includes the [CLS] and [SEP] tokens that we encountered earlier:

Transformers as Feature Extractors

Using a transformer as a feature extractor is fairly simple. As shown in <>, we freeze the body's weights during training and use the hidden states as features for the classifier. The advantage of this approach is that we can quickly train a small or shallow model. Such a model could be a neural classification layer or a method that does not rely on gradients, such as a random forest. This method is especially convenient if GPUs are unavailable, since the hidden states only need to be precomputed once.

Using pretrained models

We will use another convenient auto class from image:images/logo.png[hf,13,13] Transformers called AutoModel. Similar to the AutoTokenizer class, AutoModel has a from_pretrained() method to load the weights of a pretrained model. Let's use this method to load the DistilBERT checkpoint:

```
from transformers import AutoModel
import torch
import torch.nn.functional as F

model_ckpt = "distilbert-base-uncased"
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
model = AutoModel.from_pretrained(model_ckpt).to(device)
```

```
     Downloading:                                                256M/256M [00:05<00:00,
     100%                                                        62.9MB/s]
```

```
text = "this is a test"
inputs = tokenizer(text, return_tensors="pt")
inputs
```

```
     {'input_ids': tensor([[ 101, 2023, 2003, 1037, 3231,  102]]), 'attention_mask':
     tensor([[1, 1, 1, 1, 1, 1]])}
```

```
print(f"Input tensor shape: {inputs['input_ids'].size()}")
```

```
     Input tensor shape: torch.Size([1, 6])
```

```
inputs = {k:v.to(device) for k,v in inputs.items()}
with torch.no_grad():
    outputs = model(**inputs)
print(outputs)
```

```
     BaseModelOutput(last_hidden_state=tensor([[[-0.1565, -0.1862,  0.0528,  ...,
     -0.1188,  0.0662,  0.5470],
              [-0.3575, -0.6484, -0.0618,  ..., -0.3040,  0.3508,  0.5221],
              [-0.2772, -0.4459,  0.1818,  ..., -0.0948, -0.0076,  0.9958],
              [-0.2841, -0.3917,  0.3753,  ..., -0.2151, -0.1173,  1.0526],
              [ 0.2661, -0.5094, -0.3180,  ..., -0.4203,  0.0144, -0.2149],
              [ 0.9441,  0.0112, -0.4714,  ...,  0.1439, -0.7288, -0.1619]]],
            device='cuda:0'), hidden_states=None, attentions=None)
```

```
outputs.last_hidden_state.size()
```

```
     torch.Size([1, 6, 768])
```

```
outputs.last_hidden_state[:,0].size()
```

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

Looking at the hidden state tensor, we see that it has the shape [batch_size, n_tokens, hidden_dim]. In other words, a 768-dimensional vector is returned for each of the 6 input tokens. For classification tasks, it is common practice to just use the hidden state associated with the [CLS] token as the input feature. Since this token appears at the start of each sequence, we can extract it by simply indexing into outputs.last_hidden_state as follows:

```
dataset_enc.set_format("torch", columns=["input_ids", "attention_mask", "label"])
```

```
print(dataset_enc)
```

```
     DatasetDict({
         train: Dataset({
             features: ['__index_level_0__', 'attention_mask', 'input_ids', 'label',
     'text'],
             num_rows: 1149527
         })
         val: Dataset({
             features: ['__index_level_0__', 'attention_mask', 'input_ids', 'label',
     'text'],
             num_rows: 287403
         })
         test: Dataset({
             features: ['__index_level_0__', 'attention_mask', 'input_ids', 'label',
     'text'],
             num_rows: 159624
         })
     })
```

```
def extract_hidden_states(batch):
    # Place model inputs on the GPU
    inputs = {k:v.to(device) for k,v in batch.items() if k in tokenizer.model_input_names}
    # Extract last hidden states
    with torch.no_grad():
        last_hidden_state = model(**inputs).last_hidden_state
    # Return vector for [CLS] token
    return {"hidden_state": last_hidden_state[:,0].cpu().numpy()}
```

```
dataset_hidden = dataset_enc.map(extract_hidden_states, batched=True)
dataset_hidden['train'].column_names
```

```
import numpy as np

X_train = np.array(dataset_hidden["train"]["hidden_state"])
X_valid = np.array(dataset_hidden["val"]["hidden_state"])
y_train·=·np.array(dataset_hidden["train"]["label"])
y_valid·=·np.array(dataset_hidden["val"]["label"])
X_train.shape, X_valid.shape
```

```
from umap import UMAP
from sklearn.preprocessing import MinMaxScaler

# Scale features to [0,1] range
X_scaled = MinMaxScaler().fit_transform(X_train)
# Initialize and fit UMAP
mapper = UMAP(n_components=2, metric="cosine").fit(X_scaled)
# Create a DataFrame of 2D embeddings
df_emb = pd.DataFrame(mapper.embedding_, columns=["X", "Y"])
df_emb["label"] = y_train
df_emb.head()
```

```
fig, axes = plt.subplots(2, 3, figsize=(7,5))
axes = axes.flatten()
cmaps = ["Greys", "Blues", "Oranges", "Reds", "Purples", "Greens"]
labels = dataset["train"].features["label"].names

for i, (label, cmap) in enumerate(zip(labels, cmaps)):
    df_emb_sub = df_emb.query(f"label == {i}")
    axes[i].hexbin(df_emb_sub["X"], df_emb_sub["Y"], cmap=cmap,
                   gridsize=20, linewidths=(0,))
    axes[i].set_title(label)
    axes[i].set_xticks([]), axes[i].set_yticks([])

plt.tight_layout()
plt.show()
```

```
# We increase `max_iter` to guarantee convergence
from sklearn.linear_model import LogisticRegression
```

```
lr_clf = LogisticRegression(max_iter=3000)
lr_clf.fit(X_train, y_train)

lr_clf.score(X_valid, y_valid)
```

```
from sklearn.dummy import DummyClassifier

dummy_clf = DummyClassifier(strategy="most_frequent")
dummy_clf.fit(X_train, y_train)
dummy_clf.score(X_valid, y_valid)
```

```
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix

def plot_confusion_matrix(y_preds, y_true, labels):
    cm = confusion_matrix(y_true, y_preds, normalize="true")
    fig, ax = plt.subplots(figsize=(6, 6))
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
    disp.plot(cmap="Blues", values_format=".2f", ax=ax, colorbar=False)
    plt.title("Normalized confusion matrix")
    plt.show()

y_preds = lr_clf.predict(X_valid)
plot_confusion_matrix(y_preds, y_valid, labels)
```

Fine-Tuning Transformers

Let's now explore what it takes to fine-tune a transformer end-to-end. With the fine-tuning approach we do not use the hidden states as fixed features, but instead train them as shown in <>. This requires the classification head to be differentiable, which is why this method usually uses a neural network for classification.

Training the hidden states that serve as inputs to the classification model will help us avoid the problem of working with data that may not be well suited for the classification task. Instead, the initial hidden states adapt during training to decrease the model loss and thus increase its performance.

We'll be using the Trainer API from image:images/logo.png[hf,13,13] Transformers to simplify the training loop. Let's look at the ingredients we need to set one up!

```
from transformers import AutoModelForSequenceClassification

num_labels = 2
model = (AutoModelForSequenceClassification
         .from_pretrained(model_ckpt, num_labels=num_labels)
         .to(device))
```

```
from sklearn.metrics import accuracy_score, f1_score

def compute_metrics(pred):
    labels = pred.label_ids
    preds = pred.predictions.argmax(-1)
    f1 = f1_score(labels, preds, average="weighted")
    acc = accuracy_score(labels, preds)
    return {"accuracy": acc, "f1": f1}
```

```
from huggingface_hub import notebook_login

notebook_login()
```

```
     Login successful
     Your token has been saved to /root/.huggingface/token
```

```
from transformers import Trainer, TrainingArguments

batch_size = 64
logging_steps = len(dataset_enc["train"]) // batch_size
model_name = f"{model_ckpt}-finetuned-emotion-saul-test"
training_args = TrainingArguments(output_dir=model_name,
                                  num_train_epochs=2,
                                  learning_rate=2e-5,
                                  per_device_train_batch_size=batch_size,
                                  per_device_eval_batch_size=batch_size,
                                  weight_decay=0.01,
                                  evaluation_strategy="epoch",
                                  disable_tqdm=False,
                                  logging_steps=logging_steps,
                                  push_to_hub=True,
                                  log_level="error")
```

```
from transformers import Trainer

trainer = Trainer(model=model, args=training_args,
                  compute_metrics=compute_metrics,
                  train_dataset=dataset_enc["train"],
                  eval_dataset=dataset_enc["val"],
                  tokenizer=tokenizer)
trainer.train();
```

```
Cloning https://huggingface.co/Saulr/distilbert-base-uncased-finetuned-emotion-sau
WARNING:huggingface_hub.repository:Cloning https://huggingface.co/Saulr/distilbert
```

[35924/35924 55:34, Epoch 2/2]

```
preds_output = trainer.predict(dataset_enc["test"])
```

[2495/2495 01:31]

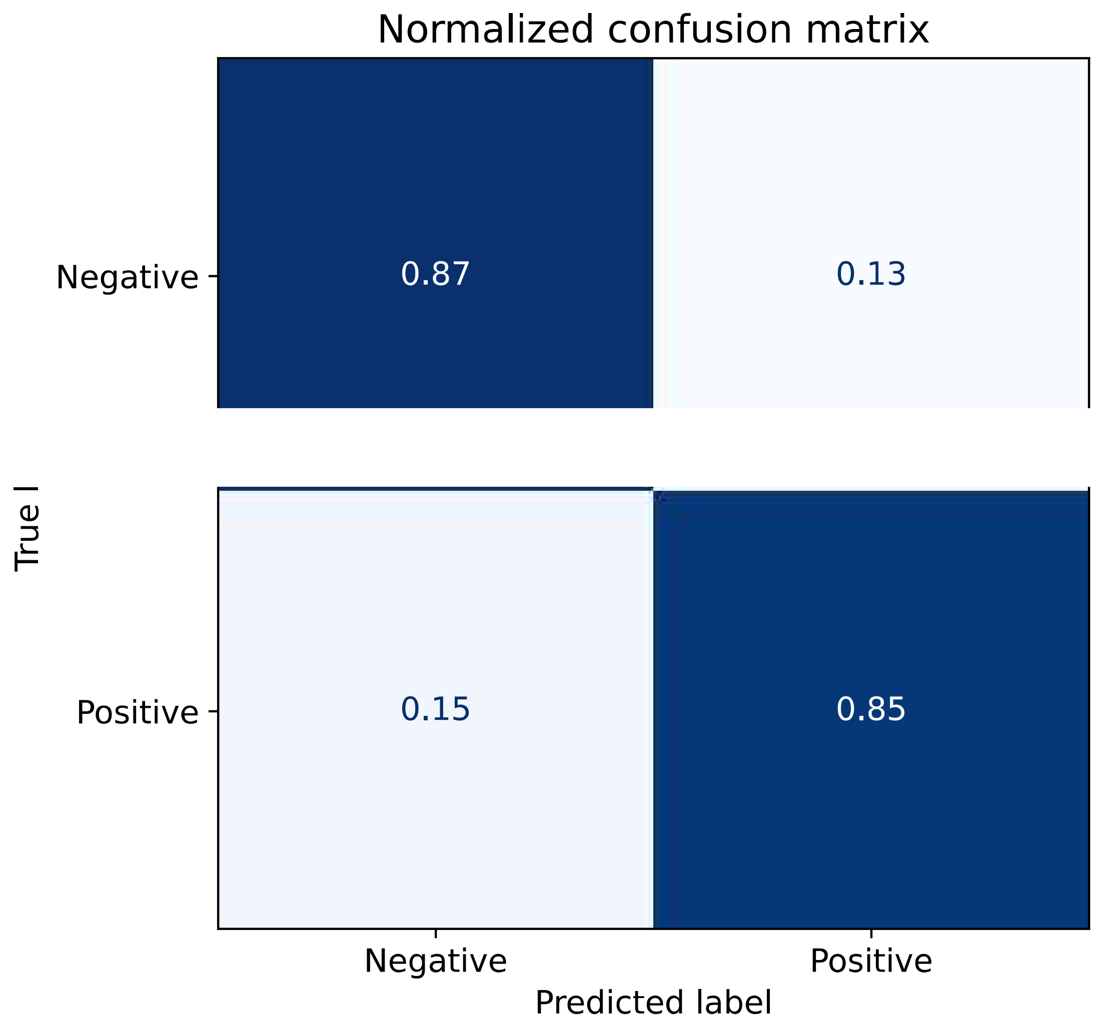
```
        2          0.304700              0.329923    0.858248   0.858223
```

```
preds_output.metrics
```

```
{'test_loss': 0.33048686385154724,
 'test_accuracy': 0.8579474264521626,
 'test_f1': 0.8579289102423674,
 'test_runtime': 91.3006,
 'test_samples_per_second': 1748.335,
 'test_steps_per_second': 27.327}
```

```
y_preds = np.argmax(preds_output.predictions, axis=1)
y_truth = np.array(dataset["test"]["label"])
labels = ["Negative", "Positive"]
```

```
from sklearn.metrics import ConfusionMatrixDisplay, confusion_matrix

def plot_confusion_matrix(y_preds, y_true, labels):
    cm = confusion_matrix(y_true, y_preds, normalize="true")
    fig, ax = plt.subplots(figsize=(6, 6))
    disp = ConfusionMatrixDisplay(confusion_matrix=cm, display_labels=labels)
    disp.plot(cmap="Blues", values_format=".2f", ax=ax, colorbar=False)
    plt.title("Normalized confusion matrix")
    plt.show()

plot_confusion_matrix(y_preds, y_truth, labels)
```



Colab paid products - Cancel contracts here

✓ 0s completed at 10:24 PM