

# AI Sentiment Analysis with PyTorch and Hugging Face Transformers

## Prerequisites and learning goals

Selecting transcript lines in this section will navigate to timestamp in the video

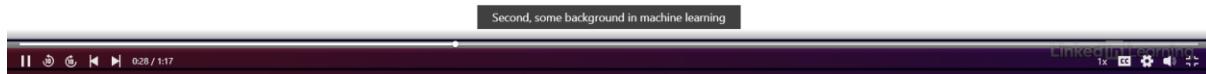
- [Instructor] Before we get into the details of sentiment analysis, let's go over the prerequisites and learning goals for this course. First, a basic understanding of Python programming will be helpful. We'll explain the code line by line in later chapters, so being familiar with Python syntax will make it easier to follow along. Second, some background in machine learning and neural networks is preferred, though not required. We'll introduce key concepts throughout the course. The goal of this course is to build a sentiment analysis model with PyTorch and Hugging Face Transformers to identify sentiment in text. By the end, you'll gain practical experience and confidently apply the skills you've learned to your own NLP projects. In the overview section of the course, you'll find an exercise files folder. Be sure to download it so you can follow along with the coding sessions. We'll guide you on how to use them later. Get ready to learn, code, and create.

A screenshot of a video player interface. At the top, there is a navigation bar with 'Contents' and the title 'AI Sentiment Analysis with PyTorch and Hugging Face Transformers'. Below the title, the subtitle 'Prerequisites and learning goals' is visible. The main content area features a large, bold, purple heading 'Prerequisites and Learning Goals'. A small, horizontal, multi-colored bar (pink, purple, yellow) is positioned below the heading. At the bottom of the screen, there is a dark footer bar containing various video control icons (play, pause, volume, etc.) and a timestamp '00:05 / 1:17'.



## Prerequisites

- Basic understanding of Python programming
- Background in machine learning and neural networks (optional)



## Learning Goals

- Build a sentiment analysis model with PyTorch and Hugging Face Transformers
- Gain practical experience and confidently apply the skills to your own NLP projects



What is sentiment analysis?

Selecting transcript lines in this section will navigate to timestamp in the video

- [Narrator] Imagine scrolling through social media or reading Amazon reviews. People are constantly sharing their thoughts and sentiments. Let's start by defining sentiment analysis. Sentiment analysis is a natural language processing task that identifies the emotional tone behind a piece

of text. It determines whether a sentiment is positive, negative, or neutral. More advanced models can even recognize complex emotions like joy, anger, or sadness. Imagine reading a movie review that says, "This movie exceeded my expectations. Engaging story and stunning visuals. A must watch." Sentiment analysis would classify this as positive. On the other hand, a review like, "This was a total waste of time. The plot was dull and the acting was terrible," would be categorized as negative. This ability to interpret text-based emotions is incredibly valuable. For example, a movie studio could analyze thousands of reviews in seconds to gauge audience reactions. Up next, we'll take a look at some real world applications of sentiment analysis and see how it brings data to life.

## What Is Sentiment Analysis?



Let's start by defining sentiment analysis.

LinkedIn Learning

## Sentiment Analysis

A natural language processing task that identifies the emotional tone behind a piece of text such as positive, negative, neutral, joy, anger, or sadness

that identifies the emotional tone behind a piece of text.

LinkedIn Learning

[Positive]

“This movie exceeded my expectations! Engaging story and stunning visuals—a must-watch!”

[Negative]

“This was a total waste of time. The plot was dull, and the acting was terrible.”

\*This movie exceeded my expectations.

LinkedIn Learning

Popular applications of sentiment analysis

Selecting transcript lines in this section will navigate to timestamp in the video

- [Narrator] Have you ever left a review for a product or service? Businesses use sentiment analysis to process countless reviews like yours, gaining insights into customer satisfaction. Here are some real world applications in

sentiment analysis. One major application is customer feedback analysis. Companies apply sentiment analysis to reviews, surveys, and feedback to better understand customer opinions and make adjustments accordingly. This helps improve customer loyalty. Sentiment analysis is also widely used in social media marketing. Brands monitor public sentiment about their campaigns and products by analyzing tweets, posts, and comments. For example, they can quickly determine whether a movie is receiving more compliments or criticism after its release. In politics, it is used for public opinion analysis to understand how people perceive certain policies, debates, or candidates. Another growing area is the financial market where investors assess the sentiment of news articles or social media posts to predict financial market trends. As you can see, sentiment analysis is a powerful tool with diverse applications. In this course, we'll focus on analyzing the sentiment of movie reviews, but the skills you learn can be applied to many other fields.

## Popular Application of Sentiment analysis



- Customer feedback analysis
- Social media marketing
- Public opinion analysis
- Financial market analysis

The screenshot shows a LinkedIn Learning quiz interface. At the top, a purple bar displays the text "where investors assess the sentiment of news articles". Below it, the LinkedIn Learning logo is on the right. The main area shows a user has answered 2 of 2 questions correctly. Question 1 asks about common applications of sentiment analysis, with "customer feedback analysis" marked as correct. Question 2 asks what sentiment analysis primarily analyzes, with "the emotional tone of text" marked as correct.

## Overview of Hugging Face Transformers

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In this video, we'll introduce Hugging Face Transformers. Hugging Face is an AI company that provides an open source platform for natural language processing. It has revolutionized NLP by offering pre-trained models that can be easily fine tuned for tasks like sentiment analysis, text generation, machine translation, and more. What are

transformers? Transformers are deep learning models designed to handle sequential data, like text, using the mechanism called attention. Unlike traditional NLP models that focus on limited context, transformers understand each word in the context of the entire sentence. Hugging Face has made these powerful models accessible to everyone with user-friendly libraries that save you from the complexities of building models. In this course, we'll use Hugging Face Transformers library and one of its pre-trained models, called DistilBERT, for our sentiment analysis project.

## Overview of Hugging Face Transformers



we'll introduce Hugging Face Transformers.

LinkedIn Learning

## Hugging Face

- Providing pre-trained models for NLP tasks
- Sentiment analysis
- Text generation
- Translation
- And more...

# Transformers

- Deep learning models designed to handle sequential data, like text
- Attention mechanism allows them to understand each word in the context of the entire sentence

using the mechanism called attention.

LinkedIn Learning

In this course, we'll use the Hugging Face Transformers library and its pre-trained models like DistilBERT, which is optimized for tasks like sentiment analysis.

and one of its pre-trained models, called  
DistilBERT.

LinkedIn Learning

## Introduction to PyTorch

Selecting transcript lines in this section will navigate to timestamp in the video

- [Narrator] Now let's talk about PyTorch. PyTorch is a powerful, open source, deep learning framework used in research and production environments. It delivers great performance while being easy to use. In this course, we use PyTorch to build and train our neural network. It handles all the underlying

computations for us. Here's the exciting part. PyTorch integrates seamlessly with Hugging Face Transformers Library. Hugging Face provides pre-trained models like DistilBERT for sentiment analysis. PyTorch serves as the backend, managing data flow and optimization. Hugging Face Transformers Library uses PyTorch for model training, inference, and fine tuning. In short, PyTorch is the engine and Hugging Face is the toolkit that provides pre-trained models. In the next video, we'll be setting up Hugging Face Transformers, PyTorch, and other libraries for the project. Let's dive into the code.

## Introduction to PyTorch



- [Narrator] Now let's talk about PyTorch.

LinkedIn Learning

## What Is PyTorch?

- PyTorch is a powerful open-source deep learning framework
- Widely used in both research and production environments
- Great performance and easy to use

It delivers great performance while being easy to use.

LinkedIn Learning

## In This Course . . .

- We'll use PyTorch to build and train our neural network, and handle all the underlying computations
- PyTorch integrates seamlessly with the Hugging Face Transformers library

with Hugging Face Transformers Library.

LinkedIn Learning

# Why Hugging Face Transformers and PyTorch?

- Hugging Face provides pre-trained models like DistilBERT
- PyTorch serves as the backend, managing data flow and optimization
- The Hugging Face Transformers library uses PyTorch for model training, fine-tuning, and inference
- PyTorch is the engine, Hugging Face is the toolkit that provides pre-trained models

that provides pre-trained models.

LinkedIn Learning

Let's dive into the code!

we'll be setting up Hugging Face Transformers, PyTorch,

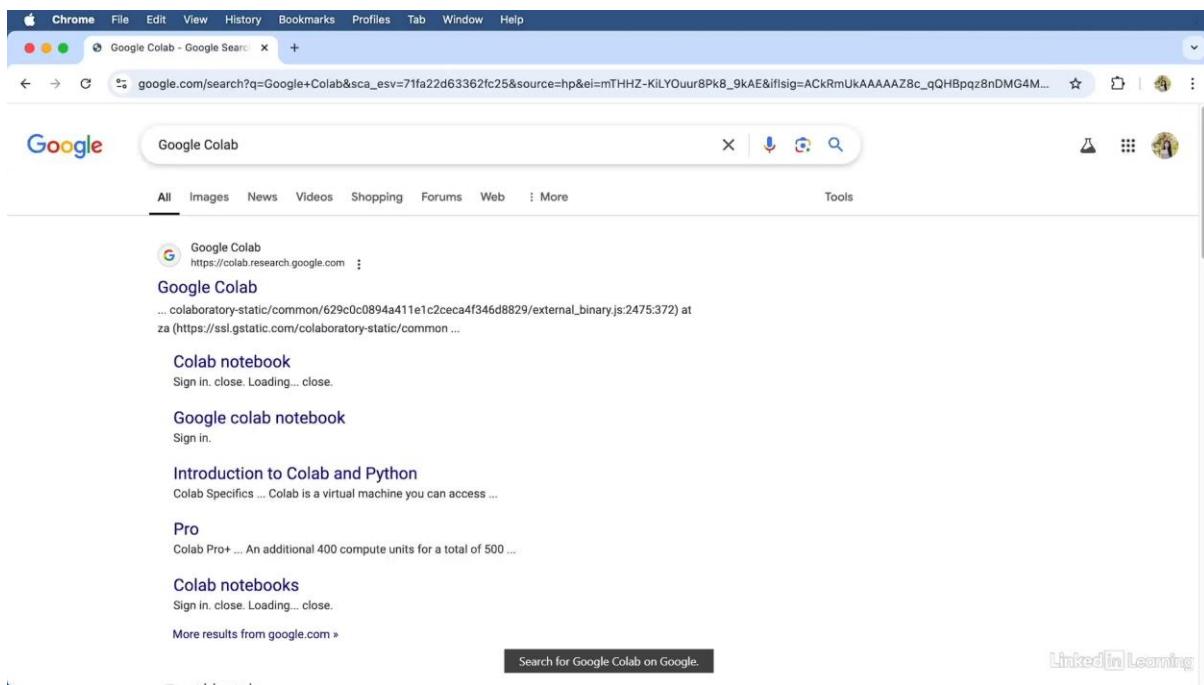
LinkedIn Learning

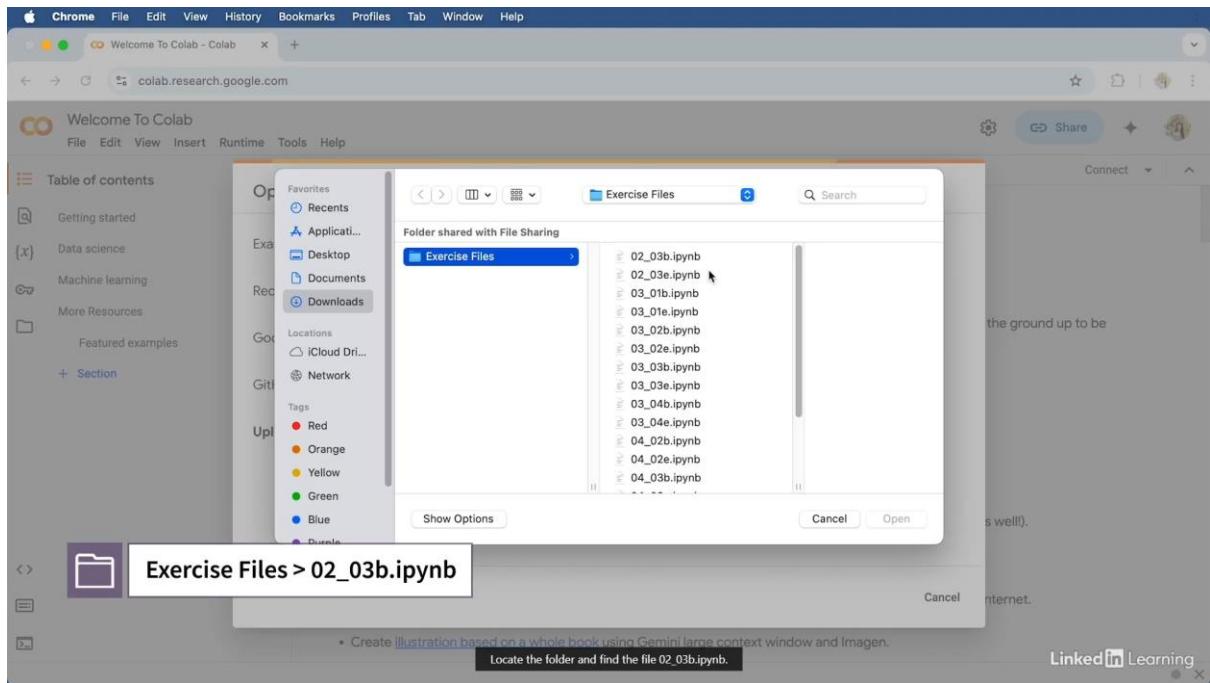
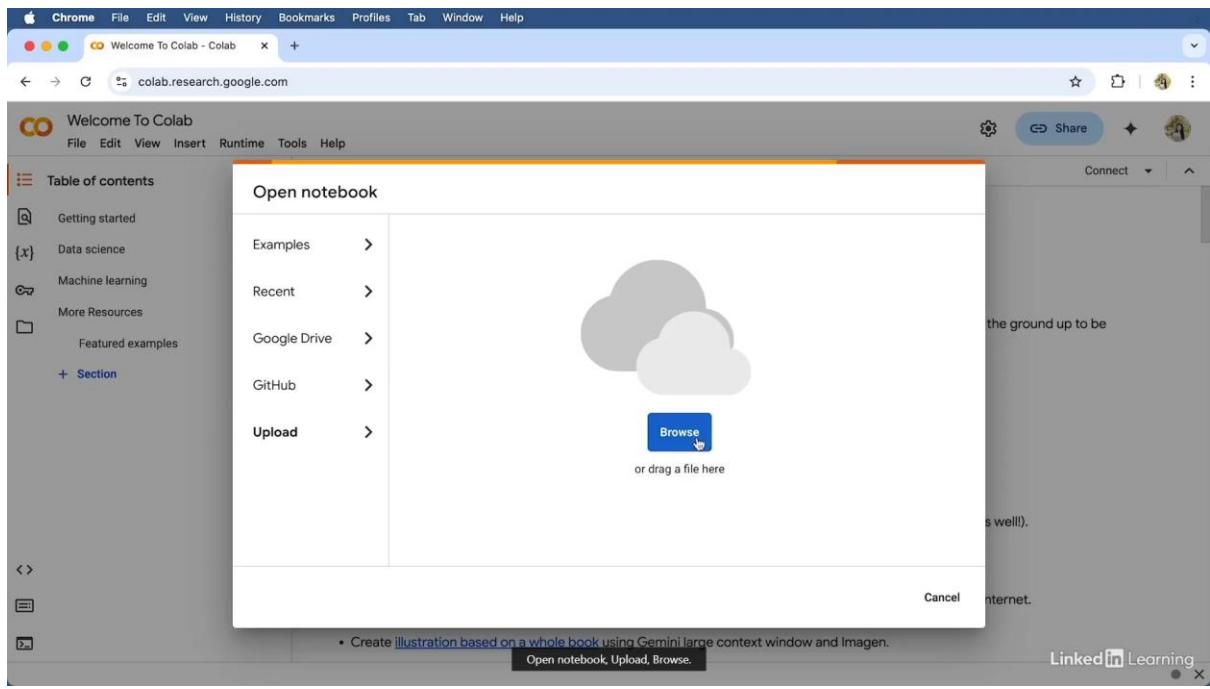
Setting up the environment on Google Colab

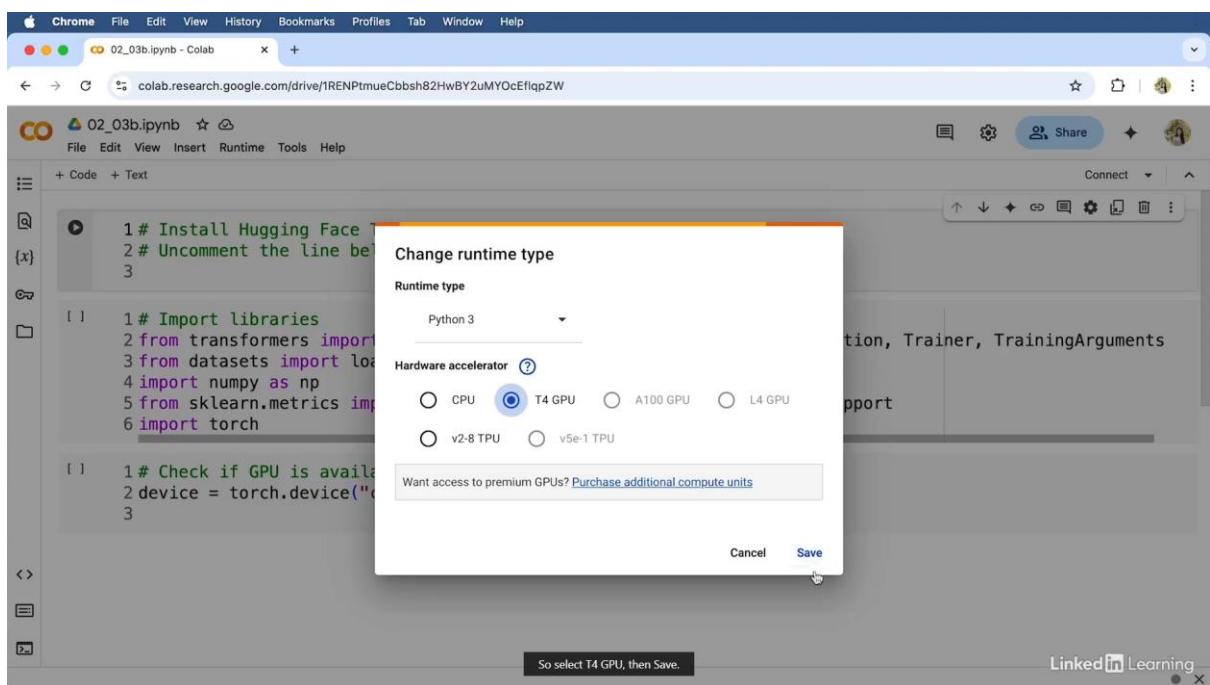
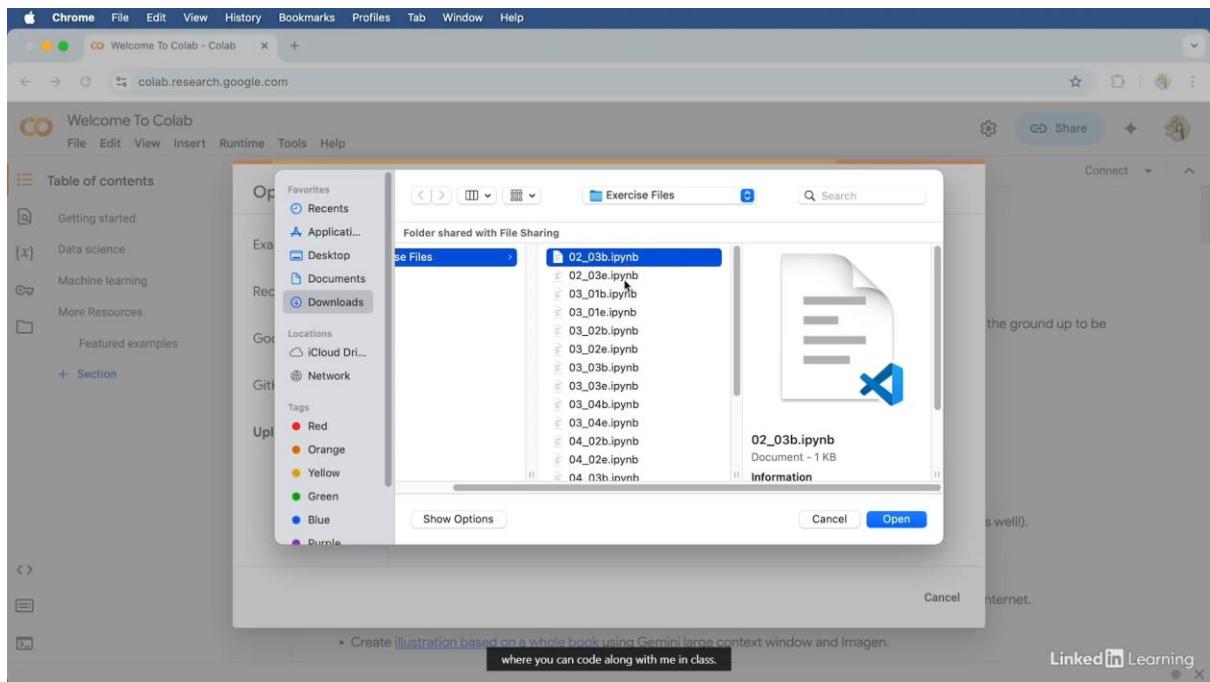
Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] We will be coding on a free cloud-based platform called Google Colab. It lets you run Python code without installing anything on your PC. But first, please download the Exercise Files folder from the Overview section of

this course. Then make sure you're logged into your Google account. Search for Google Colab on Google. Click the first link, Cancel, and go to File, Open notebook, Upload, Browse. Locate the folder and find the file 02\_03b.ipynb. The letter b in the file name indicates it is the beginning state of the file where you can code along with me in class. The files with letter e are the completed code for your reference after the class. In later videos, you'll find instructions on which Exercise File to use. Now let's open the file for this video. Before we start coding, let's click on Runtime, Change runtime type. We can see the current hardware accelerator is CPU, but Google Colab offers free GPU support, which is much stronger. So select T4 GPU, then Save. Now let's install three libraries. In the first cell, type the following code and hit Run button right here to install Hugging Face Transformers, datasets, and PyTorch libraries. You can also press Shift and Enter to run a code faster. Next, we'll import libraries for dataset loading, tokenization, model building, evaluation matrix, and so on. We will give them more introduction later in this course. Finally, let's confirm that we have access to GPU. If the name of the device is cuda, then you're all set. FYI, every time you run a cell, the code will be automatically saved. Great. With the environment ready, we can now proceed to load and pre-process the dataset.







The screenshot shows two parallel interfaces. On the left is Google Colab (version 02\_03b.ipynb) displaying Python code for importing libraries and checking GPU availability. The output shows successful installations of various CUDA components. On the right is a LinkedIn Learning quiz titled 'AI Sentiment Analysis with PyTorch and Hugging Face Transformers'. The quiz consists of three questions about PyTorch, Hugging Face Transformers, and the purpose of installing transformers/datasets in Google Colab. The user has answered all three questions correctly.

```

1 # Import libraries
2 from transformers import AutoTokenizer, AutoModelForSequenceClassification, Trainer, TrainingArguments
3 from datasets import load_dataset
4 import numpy as np
5 from sklearn.metrics import accuracy_score, precision_recall_fscore_support
6 import torch

# Check if GPU is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Using device: cuda
1 Start coding or generate with AI.

```

FYI, every time you run a cell, ✓ completed at 9:15AM

You answered 3 of 3 questions correctly.

Question 1 of 3  
Why do we use PyTorch in this course?

- It is a deep learning framework for training models.  
Correct  
PyTorch powers model training and inference.
- It provides pre-labeled datasets.
- It builds user interfaces for applications.

Question 2 of 3  
What is an advantage of Hugging Face Transformers?

- building hardware-specific neural networks
- access to pre-trained models for NLP tasks  
Correct  
Hugging Face provides pre-trained models.
- visualizing model performance metrics

Question 3 of 3  
What is the purpose of installing transformers and datasets in Google Colab?

- to compile Python code
- to create a user interface for the Colab environment
- to access pre-trained models and datasets  
Correct  
These libraries provide tools and pre-trained models for tasks like sentiment analysis.

## Loading dataset

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] The dataset we are going to use in this course is the `imdb_dataset`. It includes tens of thousands of movie reviews with their sentiment labels. We load the `imdb_dataset` with Hugging Face datasets library. The function is called `load_dataset`, which is already imported. (keyboard clicking) Once we load the dataset, we can print the dataset structure and a sample data from training set. The `imdb_dataset` is organized in a dataset dictionary. There are three subdatasets, the training set, test set, and unsupervised set. We can ignore the third one for now. The features show that each set contains movie reviews and their sentiment labels. The training set is used to train our model, While the test set is used to evaluate model performance. The training and test sets each contain 25,000 samples. For the sample data point, we can see that each data item contains a movie review and a sentiment label where zero represents negative and one represents positive.

The screenshot shows a Google Colab notebook titled "03\_01b.ipynb". The code cell at the top imports datasets, numpy, sklearn.metrics, and torch. It then checks if a GPU is available and prints the device type. The output shows "Using device: cuda". Below this, another cell is loading the dataset and viewing its structure. The status bar indicates "It includes tens of thousands of movie reviews" and "0s completed at 9:15AM". A "LinkedIn Learning" watermark is visible in the bottom right corner.

```
from datasets import load_dataset
import numpy as np
from sklearn.metrics import accuracy_score, precision_recall_fscore_support
import torch

# Check if GPU is available
device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device}")

# Loading dataset
# View dataset structure
print(imdb_dataset)
# View a sample data point
print(imdb_dataset['train'][10])
```

The screenshot shows two instances of a Google Colab notebook titled "03\_01b.ipynb".

**Top Notebook (Screenshot 1):**

```

1 from datasets import load_dataset
2 import numpy as np
3 from sklearn.metrics import accuracy_score, precision_recall_fscore_support
4 import torch
5
6 # Check if GPU is available
7 device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
8 print(f"Using device: {device}")
9
10 # Loading dataset
11 imdb_dataset = load_dataset("imdb")
12 # View dataset structure
13 print(imdb_dataset)
14 # View a sample data point
15 print(imdb_dataset['train'][10])

```

**Bottom Notebook (Screenshot 2):**

```

1 You will be able to reuse this secret in all of your notebooks.
2 Please note that authentication is recommended but still optional to access public models or datasets.
3 warnings.warn(
4 README.md: 100% [██████████] 7.81k/7.81k [00:00<00:00, 700kB/s]
5 train-00000-of-00001.parquet: 100% [██████████] 21.0M/21.0M [00:00<00:00, 64.8MB/s]
6 test-00000-of-00001.parquet: 100% [██████████] 20.5M/20.5M [00:00<00:00, 183MB/s]
7 unsupervised-00000-of-00001.parquet: 100% [██████████] 42.0M/42.0M [00:00<00:00, 197MB/s]
8 Generating train split: 100% [██████████] 25000/25000 [00:00<00:00, 98437.40 examples/s]
9 Generating test split: 100% [██████████] 25000/25000 [00:00<00:00, 97046.82 examples/s]
10 Generating unsupervised split: 100% [██████████] 50000/50000 [00:00<00:00, 157280.20 examples/s]
11 DatasetDict({
12     train: Dataset({
13         features: ['text', 'label'],
14         num_rows: 25000
15     })
16     test: Dataset({
17         features: ['text', 'label'],
18         num_rows: 25000
19     })
20     unsupervised: Dataset({
21         features: ['text', 'label'],
22         num_rows: 50000
23     })
24 })
25 {'text': 'It was great to see some of my favorite stars of 30 years ago, including John Wayne, Ben Gazzara and Audrey Hepburn. They looked quite wonderful.'}

```

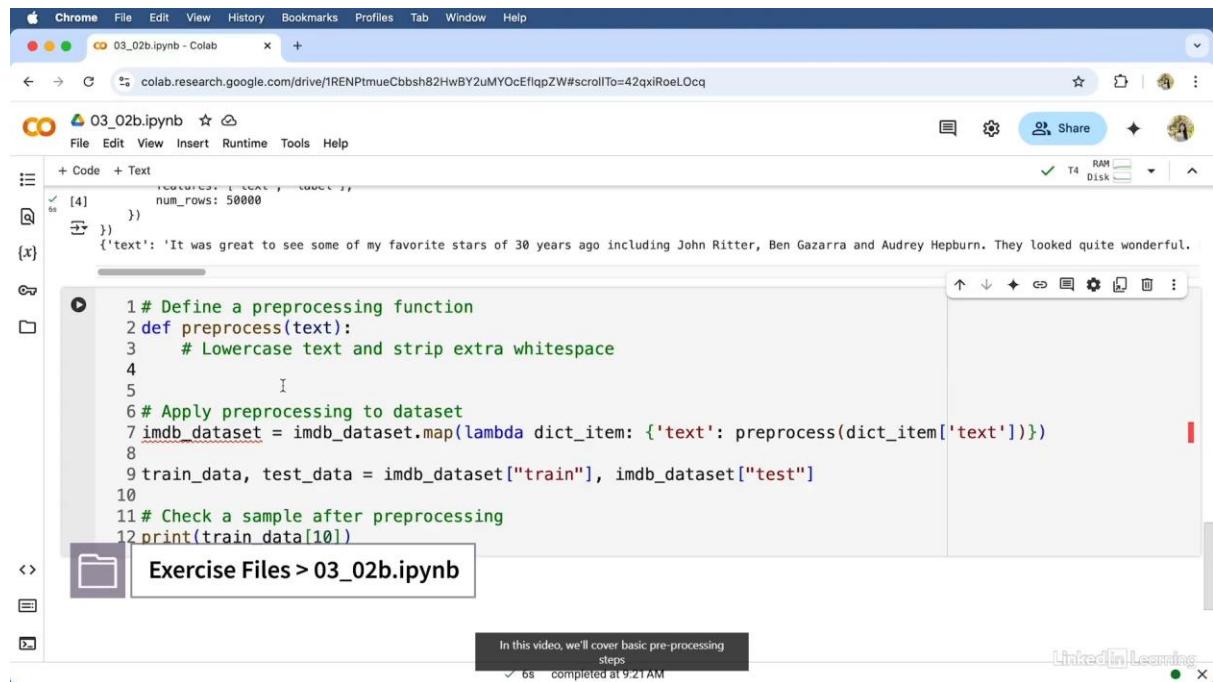
A tooltip at the bottom of the second screenshot indicates: "The imdb dataset is organized in a dataset dictionary." The status bar shows "6s completed at 9:21 AM".

## Preprocessing text data

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Data pre-processing is a crucial step in natural language processing. In this video, we'll cover basic pre-processing steps like converting text to lowercase and removing extra white space. We first define

a pre-processing function. This function will take a piece of text, lowercase it, and remove any leading or trailing white space. The lower function ensures the model treats words with the same spelling, but different capitalizations as identical. The strip function removes extra white space that could affect the tokenization process. The dataset's library from Hugging Face provides a map function that applies the pre-processing function to every data item in a dataset. We also split the pre-processed dataset into training and test sets. Finally, we'll verify if our pre-processing function is doing its job. Let's print a training data sample. We can see the text is properly lowercased and extra white space has been removed. Excellent.



```
# Define a preprocessing function
def preprocess(text):
    # Lowercase text and strip extra whitespace
    text = text.lower().strip()
    return text

# Apply preprocessing to dataset
imdb_dataset = imdb_dataset.map(lambda dict_item: {'text': preprocess(dict_item['text'])})
train_data, test_data = imdb_dataset["train"], imdb_dataset["test"]

# Check a sample after preprocessing
print(train_data[10])
```

In this video, we'll cover basic pre-processing steps  
6s completed at 9:27 AM

LinkedIn Learning

```

03_02b.ipynb - Colab
File Edit View History Bookmarks Profiles Tab Window Help
colab.research.google.com/drive/1RENPtmeCbbsh82HwBY2uMYOcEfIqpZW#scrollTo=42qxiRoeLocq
File Edit Insert Runtime Tools Help
+ Code + Text
[4]     num_rows: 50000
[4] })
{'text': "It was great to see some of my favorite stars of 30 years ago including John Ritter, Ben Gazzara and Audrey Hepburn. They looked quite wonderful."}
{x}
1 # Define a preprocessing function
2 def preprocess(text):
3     # Lowercase text and strip extra whitespace
4     return text.lower().strip()
5
6 # Apply preprocessing to dataset
7 imdb_dataset = imdb_dataset.map(lambda dict_item: {'text': preprocess(dict_item['text'])})
8
9 train_data, test_data = imdb_dataset["train"], imdb_dataset["test"]
10
11 # Check a sample after preprocessing
12 print(train_data[10])

```

to every data item in a dataset.

6s completed at 9:21 AM

```

03_02b.ipynb - Colab
File Edit View History Bookmarks Profiles Tab Window Help
colab.research.google.com/drive/1RENPtmeCbbsh82HwBY2uMYOcEfIqpZW#scrollTo=VdW0TzKANMHZ
File Edit Insert Runtime Tools Help
+ Code + Text
3     # Lowercase text and strip extra whitespace
4     return text.lower().strip()
5
6 # Apply preprocessing to dataset
7 imdb_dataset = imdb_dataset.map(lambda dict_item: {'text': preprocess(dict_item['text'])})
8
9 train_data, test_data = imdb_dataset["train"], imdb_dataset["test"]
10
11 # Check a sample after preprocessing
12 print(train_data[10])

```

Map: 100% [██████████] 25000/25000 [00:01<00:00, 20721.77 examples/s]

Map: 100% [██████████] 25000/25000 [00:01<00:00, 21482.31 examples/s]

Map: 100% [██████████] 50000/50000 [00:02<00:00, 21274.38 examples/s]

{'text': "it was great to see some of my favorite stars of 30 years ago including john ritter, ben gazzara and audrey hepburn. they looked quite wonderful."}

1 Start coding or generate with AI.

Let's print a training data sample.

5s completed at 9:29 AM

## Tokenization

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Tokenization is the process of breaking down larger pieces of text, like sentences or paragraphs into smaller units called tokens. For example, the sentence "I love movies," is split into tokens, I, love, and

movies. To create a tokenizer, we use AutoTokenizer class from Hugging Face. It initializes the tokenizer with the distilbert model name distilbert-base-uncased. Now, let's define a tokenization function with the tokenizer. (keyboard clicking) The tokenizer here processes each item in the dataset, the parameters for padding, truncation, and max\_length, make sure all sequences are of the same length, and are less than 128 tokens. These steps are necessary because our model expects a fixed-size input. Now, we apply the tokenized function to both the training and test datasets using the map function from datasets library. This tokenizes the data efficiently in batches. The dataset is now transformed and ready for model training. In the next video, we'll introduce padding and truncation with an example.

The screenshot shows a Google Colab notebook titled "03\_03b.ipynb". The code cell contains Python code for initializing a tokenizer and defining a tokenization function. The code is as follows:

```
1 # Initialize tokenizer
2 model_name = "distilbert-base-uncased"
3 tokenizer = AutoTokenizer.from_pretrained(model_name)
4
5 # Define tokenization function
6 def tokenize(dict_items):
7
8
9 # Apply tokenization
10 tokenized_train_data = train_data.map(tokenize, batched=True)
11 tokenized_test_data = test_data.map(tokenize, batched=True)
```

The status bar at the bottom indicates "like sentences or paragraphs" and "5s completed at 9:29 AM".

```

1 # Initialize tokenizer
2 model_name = "distilbert-base-uncased"
3 tokenizer = AutoTokenizer.from_pretrained(model_name)
4
5 # Define tokenization function
6 def tokenize(dict_items):
7     return tokenizer(dict_items["text"], padding="max_length", truncation=True, max_length=128)
8
9 # Apply tokenization
10 tokenized_train_data = train_data.map(tokenize, batched=True)
11 tokenized_test_data = test_data.map(tokenize, batched=True)

```

The tokenizer here processes each item in the dataset.

5s completed at 9:29 AM

1 \$start coding or generate with AI.

The dataset is now transformed and ready for model training.

53s completed at 9:38 AM

## Padding and truncation

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] You may have noticed text data varies in length. Some sentences are short, while others are much longer. Models like DistilBERT usually require inputs of the same length, which is where padding and

truncation come into play. Padding ensures all sequences have the same length by adding extra tokens, often zeros, to shorter sentences. On the other hand, truncation shortens longer sequences to a specified length. Both steps standardize the input size for the model. The tokenizer we used earlier automatically handles padding and truncation. To demonstrate how this works, we'll apply them to another sample from the dataset. First, we tokenize the sample text, while specifying padding and truncation parameters. We set the truncation to true and padding to max length. Then we set the maximum input size to 512 tokens. Now let's print the tokenized sample. The input IDs here are the numerical representation of the tokenized sequence. We also print the length of the sample to confirm it satisfies the maximum length requirement. It looks like padding and truncation are working as expected.

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** AI Sentiment Analysis with PyTorch and Hugging Face Transformers  
Padding and truncation  
03\_04b.ipynb - Colab
- Toolbar:** Includes back, forward, search, and file operations.
- Code Cell:** [6] contains the following Python code:

```
1 # For illustration purpose only
2 sample_text = imdb_dataset['train'][10]['text']
3 # Example configuration for padding and truncation
4 tokenized_sample = tokenizer(
5     sample_text,
6     # Cut off sequences longer than model's max input size
7
8     # Pad to model's max input size
9
10    # Maximum token length for DistilBERT
11    max_length=512
12 )
```
- Output Cell:** [6] displays the output of the code:

```
13
14 Exercise Files > 03_04b.ipynb ized output
15
16 print(len(tokenized_sample['input_ids']))
```

while others are much longer.
- Runtime Status:** RAM T4 Disk
- Bottom Bar:** Includes navigation icons, a progress bar (006 / 142), and a status message (53s completed at 9:38 AM).

Chrome File Edit View History Bookmarks Profiles Tab Window Help

03\_04b.ipynb - Colab

colab.research.google.com/drive/1RENPtmeCbbsh82HwBY2uMYOcEfIqpZW#scrollTo=87Po94df07Aa

03\_04b.ipynb Saving...

File Edit Insert Runtime Tools Help

+ Code + Text

[6] tokenzier.json: 100% 488K/488K [UUUU<UUUU] 2.85MB/s

53s Map: 100% 25000/25000 [00:29<00:00, 1171.71 examples/s]

{x} Map: 100%

```
def __call__(text: Union[TextInput, PreTokenizedInput, List[TextInput], List[PreTokenizedInput]] = None, sample_text: imdb_dataset.TextPair = None, text_pair: Optional[Union[TextInput, PreTokenizedInput, List[TextInput], List[PreTokenizedInput]]] = None, tokenized_sample: tokenizer.PreTokenizedInput = None, truncation: bool = False, padding: str = "max_length", max_length: int = 512):
```

1 # For illustration purposes only

2 sample\_text = imdb\_dataset['train'][10]['text']

3 # Example configuration for padding and truncation

4 tokenized\_sample = tokenizer(

5 sample\_text,

6 # Cut off sequences longer than model's max input size

7 truncation=True,

8 # Pad to model's max input size

9 padding="max\_length",

10 # Maximum token length for DistilBERT

11 max\_length=512)

12 )

13

14 # View padded and truncated tokenized output

15 print(tokenized\_sample)

16 print(len(tokenized\_sample['input\_ids']))

Then we set the maximum input size

53s completed at 9:38 AM

LinkedIn Learning

Chrome File Edit View History Bookmarks Profiles Tab Window Help

03\_04b.ipynb - Colab

colab.research.google.com/drive/1RENPtmeCbbsh82HwBY2uMYOcEfIqpZW#scrollTo=87Po94df07Aa

03\_04b.ipynb Saving...

File Edit Insert Runtime Tools Help

+ Code + Text

[6] tokenzier.json: 100% 25000/25000 [00:29<00:00, 1018.61 examples/s]

53s Map: 100% 25000/25000 [00:21<00:00, 1018.61 examples/s]

{x} Map: 100%

```
1 # For illustration purpose only
```

2 sample\_text = imdb\_dataset['train'][10]['text']

3 # Example configuration for padding and truncation

4 tokenized\_sample = tokenizer(

5 sample\_text,

6 # Cut off sequences longer than model's max input size

7 truncation=True,

8 # Pad to model's max input size

9 padding="max\_length",

10 # Maximum token length for DistilBERT

11 max\_length=512)

12 )

13

14 # View padded and truncated tokenized output

15 print(tokenized\_sample)

16 print(len(tokenized\_sample['input\_ids']))

{'input\_ids': [101, 2009, 2001, 2307, 2000, 2156, 2070, 1997, 2026, 5440, 3340, 1997, 2382, 2086, 3283, 2164, 2198, 23168, 1010, 3841, 14474, 11335, 1998, 1512]}

Now let's print the tokenized sample.

53s completed at 9:44 AM

LinkedIn Learning

The screenshot shows a LinkedIn Learning quiz interface. At the top, there's a navigation bar with 'Learning' and various user icons. Below it, a header bar indicates the course is 'AI Sentiment Analysis with PyTorch and Hugging Face Transformers' and the section is 'Chapter Quiz'. A sidebar on the left has icons for home, contents, and search. The main area displays four questions:

- Question 1 of 4**: Why is the IMDB dataset used in this course?
  - It is a labeled dataset ideal for sentiment analysis.  
Correct  
IMDb reviews are labeled, which are great for sentiment analysis tasks.
  - It provides real-time data.
  - It contains multiple languages.
- Question 2 of 4**: Why is padding used in tokenization?
  - to improve the model's accuracy
  - to make all input sequences the same length  
Correct  
Padding standardizes input size.
  - to shorten the text for faster processing
- Question 3 of 4**: What does tokenization do?
  - summarizes long paragraphs
  - converts text into tokens for model input  
Correct  
Tokenization transforms text into tokens.
  - translates sentences into another language
- Question 4 of 4**: What is the purpose of data preprocessing in NLP?
  - to train the model on raw text
  - to translate text into another language
  - to clean text and prepare it for tokenization  
Correct  
Preprocessing ensures text is ready for modeling.

## What is DistilBERT?

Selecting transcript lines in this section will navigate to timestamp in the video

- [Narrator] Imagine a model that's nearly as powerful as BERT, but smaller and faster. That's exactly what DistilBERT offers. In this video, we'll explore what makes DistilBERT unique and why it's ideal for our project. But first, what is BERT? BERT stands for bi-directional encoder representations from transformers. It processes text by understanding the context of words in both directions. For example, in a sentence, "The bank by the river," BERT recognizes that bank refers to a river bank, not a financial institution by analyzing the surrounding context. While BERT is very powerful, it can be computationally expensive and slow for real world applications. DistilBERT is 40% smaller and 60% faster while keeping most of BERT's performance. It balances model power and efficiency, even when you're processing large

scale data. DistilBERT involves a process called knowledge distillation. Here's how it works. A larger model called the teacher model, in this case BERT, trains a smaller model, known as the student model. The student learns to mimic the teacher's predictions and retain most of its capabilities. In short, DistilBERT provides almost all the benefits of BERT while being faster and lighter.

## What Is DistilBERT?



- [Narrator] Imagine a model that's nearly as powerful

LinkedIn Learning

## What Is BERT?

- BERT = bidirectional encoder representations from transformers
- Processes text by understanding the context of words in both directions
- Highly accurate but computationally expensive

it can be computationally expensive and slow

LinkedIn Learning

## Why DistilBERT?

- Optimized for speed and efficiency
- 40% smaller and 60% faster than BERT
- Retains most of BERT's performance

It balances model power and efficiency.

LinkedIn Learning

## How Does DistilBERT Work?

- Built using **knowledge distillation**
- A “teacher model” (BERT) trains a smaller “student model” (DistilBERT)
- The result: a compact model that mimics BERT’s strengths

In short, DistilBERT provides almost all the benefits

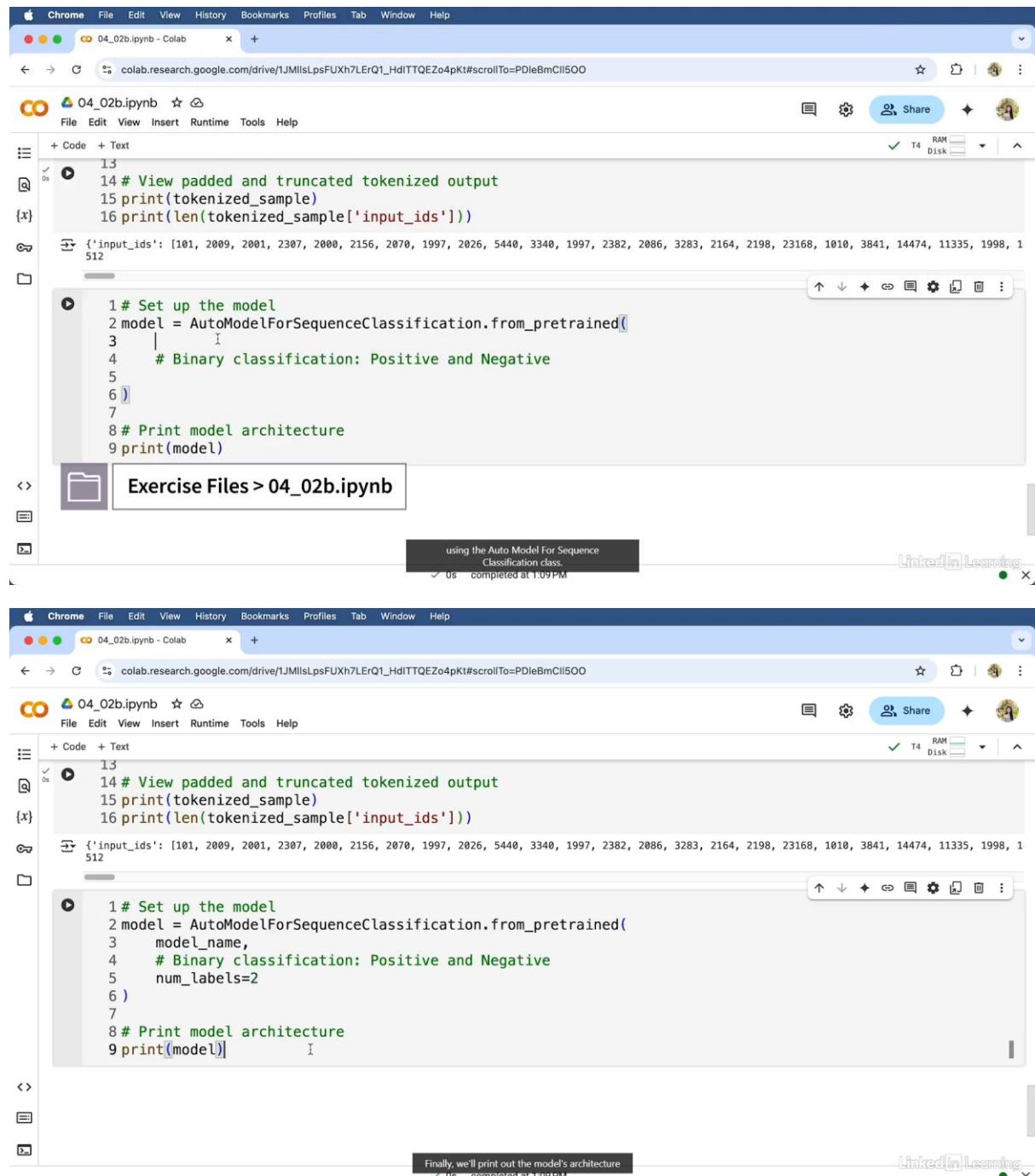
LinkedIn Learning

Setting up the model

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In this video we'll set up the SteelBERT using the Auto Model For Sequence Classification class. The best part is we don't have to build the model from scratch. This pre-trained model has done all the heavy lifting for

us. Since sentiment analysis in our case is a binary classification problem, we pass in model name and set the number of labels to two, one for positive and one for negative. Finally, we'll print out the model's architecture to see the individual layers and components. This helps us understand how the model processes text and makes predictions.



The screenshot shows two instances of Google Colab in a browser window. Both instances are running the same Jupyter notebook, "04\_02b.ipynb".

**Top Notebook (Initial State):**

- Code cell 13:

```
13
14 # View padded and truncated tokenized output
15 print(tokenized_sample)
16 print(len(tokenized_sample['input_ids']))
```

- Output cell 15:{'input\_ids': [101, 2009, 2001, 2307, 2000, 2156, 2070, 1997, 2026, 5440, 3340, 1997, 2382, 2086, 3283, 2164, 2198, 23168, 1010, 3841, 14474, 11335, 1998, 512]

**Bottom Notebook (After Model Setup):**

- Code cell 13:

```
13
14 # View padded and truncated tokenized output
15 print(tokenized_sample)
16 print(len(tokenized_sample['input_ids']))
```

- Output cell 15:{'input\_ids': [101, 2009, 2001, 2307, 2000, 2156, 2070, 1997, 2026, 5440, 3340, 1997, 2382, 2086, 3283, 2164, 2198, 23168, 1010, 3841, 14474, 11335, 1998, 512]
- Code cell 16:

```
1 # Set up the model
2 model = AutoModelForSequenceClassification.from_pretrained(
3     ...
4     # Binary classification: Positive and Negative
5
6 )
7
8 # Print model architecture
9 print(model)
```

- Output cell 16:

```
using the Auto Model For Sequence
Classification class.
✓ Us completed at 1:09 PM
```

**Bottom Notebook (After Model Print):**

  - Code cell 13:

```
13
14 # View padded and truncated tokenized output
15 print(tokenized_sample)
16 print(len(tokenized_sample['input_ids']))
```

  - Output cell 15:{'input\_ids': [101, 2009, 2001, 2307, 2000, 2156, 2070, 1997, 2026, 5440, 3340, 1997, 2382, 2086, 3283, 2164, 2198, 23168, 1010, 3841, 14474, 11335, 1998, 512]
  - Code cell 16:

```
1 # Set up the model
2 model = AutoModelForSequenceClassification.from_pretrained(
3     model_name,
4     # Binary classification: Positive and Negative
5     num_labels=2
6 )
7
8 # Print model architecture
9 print(model)
```

  - Output cell 16:

```
Finally, we'll print out the model's architecture
✓ Us completed at 1:09 PM
```

```

  (word_embeddings): Embedding(30522, 768, padding_10x=0)
  (position_embeddings): Embedding(512, 768)
  (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
  (dropout): Dropout(p=0.1, inplace=False)
)
)
(x):
  (transformer): Transformer(
    (layer): ModuleList(
      (0-5): 6 x TransformerBlock(
        (attention): DistilBertSdpAttention(
          (dropout): Dropout(p=0.1, inplace=False)
          (q_lin): Linear(in_features=768, out_features=768, bias=True)
          (k_lin): Linear(in_features=768, out_features=768, bias=True)
          (v_lin): Linear(in_features=768, out_features=768, bias=True)
          (out_lin): Linear(in_features=768, out_features=768, bias=True)
        )
        (sa_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
        (ffn):
          (dropout): Dropout(p=0.1, inplace=False)
          (lin1): Linear(in_features=768, out_features=3072, bias=True)
          (lin2): Linear(in_features=3072, out_features=768, bias=True)
          (activation): GELUActivation()
        )
        (output_layer_norm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      )
    )
  )
  (pre_classifier): Linear(in_features=768, out_features=768, bias=True)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
  (dropout): Dropout(p=0.2, inplace=False)
)
)

```

This helps us understand how the model processes text

## Configuring training parameters

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] We'll use the training arguments class from Hugging Face Transformers library. First, we specify an output directory to save model checkpoints. This helps track different training stages, so we can load the best model later. The evaluation strategy is set to epoch. This means the model will be evaluated at the end of each training epoch. The learning rate controls how much the model updates its parameters during training. We use a learning rate of 5e-5 which is a common choice for fine tuning pre-trained models. (keyboard clicking) We set the batch size for both training and evaluation to 16 and specify three training epochs. (keyboard clicking) Three, is a good starting point and we can adjust it based on the model's performance during training. To prevent over-fitting, we set weight decay to 0.01. This regularization technique prevents large weights from affecting model performance. We also log after every 10 steps and save the model after each epoch. Finally, we set the model to load its best version after training. Let's print the training arguments to confirm they're setup correctly. Everything looks good.

Chrome AI Sentiment Analysis with PyTorch and Hugging Face Transformers Tab Window Help

Contents 04\_03b.ipynb - Colab x +

colab.research.google.com/drive/1wElARbvVZskow9lwktLZFe-c8ZUtnDhB#scrollTo=Un-IPgW\_1P3g

04\_03b.ipynb Saving...

File Edit View Insert Runtime Tools Help

+ Code + Text

```
[8] (classifier): Linear(in_features=768, out_features=2, bias=True)
(dropout): Dropout(p=0.2, inplace=False)
```

{x}

```
1 training_args = TrainingArguments(
2     # Output directory for model checkpoints
3     output_dir='./results',
4     # Evaluate the model at the end of each epoch
5     eval_strategy="epoch",
6     # Learning rate
7     |
8     # Batch size for training
9     per_device_train_batch_size=16,
10    # Batch size for evaluation
11    per_device_eval_batch_size=16,
12    # Number of training epochs
13
14    # Directory for storing logs
15    logging_dir="./logs",
16    # Log after every 10 steps
```

Exercise Files > 04\_03b.ipynb

First, we specify an output directory

0s completed at 1:22PM

Apple Chrome File Edit View History Bookmarks Profiles Tab Window Help

04\_03b.ipynb - Colab x +

colab.research.google.com/drive/1wElARbvVZskow9lwktLZFe-c8ZUtnDhB#scrollTo=Un-IPgW\_1P3g

04\_03b.ipynb Saving...

File Edit View Insert Runtime Tools Help

+ Code + Text

```
[8] (classifier): Linear(in_features=768, out_features=2, bias=True)
(dropout): Dropout(p=0.2, inplace=False)
```

{x}

```
1 training_args = TrainingArguments(
2     # Output directory for model checkpoints
3     output_dir='./results',
4     # Evaluate the model at the end of each epoch
5     eval_strategy="epoch",
6     # Learning rate
7     learning_rate=5e-5,
8     # Batch size for training
9     per_device_train_batch_size=16,
10    # Batch size for evaluation
11    per_device_eval_batch_size=16,
12    # Number of training epochs
13
14    # Weight decay for regularization
15
16    # Directory for storing logs
17    logging_dir="./logs",
18    # Log after every 10 steps
```

and evaluation to 16 and specify three training epochs.

✓ 0s completed at 1:22PM

LinkedIn Learning

```

# BATCH SIZE FOR TRAINING
9 per_device_train_batch_size=16,
10 # Batch size for evaluation
11 per_device_eval_batch_size=16,
12 # Number of training epochs
13 num_train_epochs=3,
14 # Weight decay for regularization
15 weight_decay=0.01,
16 # Directory for storing checkpoints
17 logging_dir='./log' overwrite_output_dir: bool = False,
18 # Log after every 10 do_train: bool = False, do_eval:
19 logging_steps=10, bool = False, do_predict: bool =
20 # Save model after 100 False, eval_strategy:
21 save_strategy='epoch' IntervalStrategy | str = "no",
22 # Load the best model prediction_loss_only: bool = False,
23 load_best_model_at per_device_train_batch_size: int =
24 8, per_device_eval_batch_size: int
25 = 8, per_gpu_train_batch_size: int
26 print(training_args)

```

from affecting model performance.  
✓ 0s completed at 1:22PM

```

key_scope=None,
remove_unused_columns=True,
report_to=['tensorboard', 'wandb'],
restore_callback_states_from_checkpoint=False,
resume_from_checkpoint=None,
run_name='./results',
save_on_each_node=False,
save_only_model=False,
save_safetensors=True,
save_steps=500,
save_strategy=SaveStrategy.EPOCH,
save_total_limit=None,
seed=42,
skip_memory_metrics=True,
split_batches=None,
tf32=None,
torch_compile=False,
torch_compile_backend=None,
torch_compile_mode=None,
torch_empty_cache_steps=None,
torchdynamo=None,
tpu_metrics_debug=False,
tpu_num_cores=None,
use_cpu=False,
use_ipex=False,
use_legacy_prediction_loop=False,
use_liger_kernel=False,
use_mps_device=False,
warmup_ratio=0.0,
warmup_steps=0,
weight_decay=0.01,
)

```

✓ 0s completed at 1:26 PM

## Training the model

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Before training the model, we define a function called `compute_metrics` to calculate accuracy, precision, recall, and F1 score. I'll dive deeper into these metrics later, but for now, just know that this function

evaluates model performance on both training and test data sets. Next, we initialize the trainer object. It simplifies the training and evaluation process. We pass in the model, training arguments, data sets, and the compute\_metrics function. (keyboard clicking) Finally, we call the train method on the trainer to start model training. You will be prompted to enter an API key. Simply follow the provided link. Sign in with your Google account, copy the key, (keyboard clicking) and paste it here. As training proceeds, the model computes and logs evaluation metrics at each step. You'll also see the estimated time remaining, so be patient. Once training is complete, we use trainer.evaluate to print a summary of the evaluation results. We'll analyze those results in the next chapter.

The screenshot shows a Google Colab notebook titled "04\_04b.ipynb". The code cell contains Python code for defining evaluation metrics and initializing a Trainer object. A tooltip provides information about the precision\_recall\_fscore\_support function. The Colab interface includes a sidebar with file navigation, a status bar showing RAM and Disk usage, and a LinkedIn Learning logo.

```
1 # Define evaluation metrics
2 def compute_metrics(pred):
3     labels = pred.label_ids
4     preds = np.argmax(pred.predictions, axis=1)
5     accuracy = accuracy_score(labels, preds)
6     precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average="binary")
7     return {"accuracy": accuracy, "precision": precision, "recall": recall, "f1": f1}
8
9 # Initialize the Trainer
10 trainer = Trainer([
11     ],
12     train_dataset=tokenized_train_data,
13     eval_dataset=tokenized_test_data,
14     )
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
279
280
281
282
283
284
285
286
287
288
289
289
290
291
292
293
294
295
296
297
298
299
299
300
301
302
303
304
305
306
307
308
309
309
310
311
312
313
314
315
316
317
318
319
319
320
321
322
323
324
325
326
327
328
329
329
330
331
332
333
334
335
336
337
338
339
339
340
341
342
343
344
345
346
347
348
349
349
350
351
352
353
354
355
356
357
358
359
359
360
361
362
363
364
365
366
367
368
369
369
370
371
372
373
374
375
376
377
378
379
379
380
381
382
383
384
385
386
387
388
389
389
390
391
392
393
394
395
396
397
398
399
399
400
401
402
403
404
405
406
407
408
409
409
410
411
412
413
414
415
416
417
418
419
419
420
421
422
423
424
425
426
427
428
429
429
430
431
432
433
434
435
436
437
438
439
439
440
441
442
443
444
445
446
447
448
449
449
450
451
452
453
454
455
456
457
458
459
459
460
461
462
463
464
465
466
467
468
469
469
470
471
472
473
474
475
476
477
478
479
479
480
481
482
483
484
485
486
487
488
489
489
490
491
492
493
494
495
496
497
498
499
499
500
501
502
503
504
505
506
507
508
509
509
510
511
512
513
514
515
516
517
518
519
519
520
521
522
523
524
525
526
527
528
529
529
530
531
532
533
534
535
536
537
538
539
539
540
541
542
543
544
545
546
547
548
549
549
550
551
552
553
554
555
556
557
558
559
559
560
561
562
563
564
565
566
567
568
569
569
570
571
572
573
574
575
576
577
578
579
579
580
581
582
583
584
585
586
587
588
589
589
590
591
592
593
594
595
596
597
598
599
599
600
601
602
603
604
605
606
607
608
609
609
610
611
612
613
614
615
616
617
618
619
619
620
621
622
623
624
625
626
627
628
629
629
630
631
632
633
634
635
636
637
638
639
639
640
641
642
643
644
645
646
647
648
649
649
650
651
652
653
654
```

Chrome File Edit View History Bookmarks Profiles Tab Window Help

04\_04b.ipynb - Colab colab.research.google.com/drive/tIQkbt20GSJVyJMqEMNmSqtHfJDIMTTG#scrollTo=G9xaf1MhMKsc

File Edit Insert Runtime Tools Help

+ Code + Text

```
1 # Define evaluation metrics
2 def compute_metrics(pred):
3     labels = pred.label_ids
4     preds = np.argmax(pred.predictions, axis=1)
5     accuracy = accuracy_score(labels, preds)
6     precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average="binary")
7     return {"accuracy": accuracy, "precision": precision, "recall": recall, "f1": f1}
8
9 # Initialize the Trainer
10 trainer = Trainer(
11     model=model,
12     args=training_args,
13     train_dataset=tokenized_train_data,
14     eval_dataset=tokenized_test_data,
15     compute_metrics=compute_metrics,
16 )
17
18 # Train the model
19 |
```

[ ] 1 # Evaluate the model

Finally, we call the train method on the trainer  
✓ 0s completed at 1:32PM

LinkedIn Learning

Chrome File Edit View History Bookmarks Profiles Tab Window Help

04\_04b.ipynb - Colab colab.research.google.com/drive/tIQkbt20GSJVyJMqEMNmSqtHfJDIMTTG#scrollTo=20g1RFhOPP3p

File Edit View Insert Runtime Tools Help

+ Code + Text

```
7     return {"accuracy": accuracy, "precision": precision, "recall": recall, "f1": f1}
8
9 # Initialize the Trainer
10 trainer = Trainer(
11     model=model,
12     args=training_args,
13     train_dataset=tokenized_train_data,
14     eval_dataset=tokenized_test_data,
15     compute_metrics=compute_metrics,
16 )
17
18 # Train the model
19 trainer.train()
```

... wandb: WARNING: The `run\_name` is currently set to the same value as `TrainingArguments.output\_dir`. If this was not intended, please specify a different run  
wandb: Using wandb-core as the SDK backend. Please refer to <https://wandb.me/wandb-core> for more information.  
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: <https://wandb.me/wandb-server>)  
wandb: You can find your API key in your browser here: <https://wandb.ai/authorize>  
wandb: Paste an API key from your profile and hit enter:

[ ] 1 # Evaluate the model

You will be prompted to enter an API key.

Executing (34s) <cell ...> tr... > \_inner\_trainin... > on\_train... > call\_e... > on\_train... > se... > l... > maybe... > \_jo... > prompt\_a... > \_prompt\_a... > prompt\_a... > pro... > prompt... > hidden\_prom... > getp... > \_input\_re... > sel...

LinkedIn Learning

Chrome File Edit View History Bookmarks Profiles Tab Window Help

04\_04b.ipynb - Colab

colab.research.google.com/drive/tIQkbt20GSJVyJMqEMNmSqtHfJDIMTTG#scrollTo=20g1RFhOPP3p

04\_04b.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text

```
7     return {"accuracy": accuracy, "precision": precision, "recall": recall, "f1": f1}
8
9 # Initialize the Trainer
10 trainer = Trainer(
11     model=model,
12     args=training_args,
13     train_dataset=tokenized_train_data,
14     eval_dataset=tokenized_test_data,
15     compute_metrics=compute_metrics,
16 )
17
18 # Train the model
19 trainer.train()
```

... wandb: WARNING The 'run\_name' is currently set to the same value as 'TrainingArguments.output\_dir'. If this was not intended, please specify a different run  
wandb: Using wandb-core as the SDK backend. Please refer to <https://wandb.me/wandb-core> for more information.  
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: <https://wandb.me/wandb-server>)  
wandb: You can find your API key in your browser here: <https://wandb.ai/authorize>  
wandb: Paste an API key from your profile and hit enter:

1 # Evaluate the model

Simply follow the provided link.

LinkedIn Learning

https://wandb.ai/authorize

Chrome File Edit View History Bookmarks Profiles Tab Window Help

04\_04b.ipynb - Colab

wandb.auth0.com/login?state=hKFo2SAwSHhwZTlpVEFZSC0zNfIbkVqRjBNC3BRNWNNt0ppRqFupWxvZ2luo3RpZNkgY3lxMzhSTno0dERuRUlpUzJ4ZkVHdDZBb1Q1...

Sign in with Auth0

Weights & Biases

Log in Sign up

Sign in with GitHub

Sign in with Google

Sign in with Microsoft

or

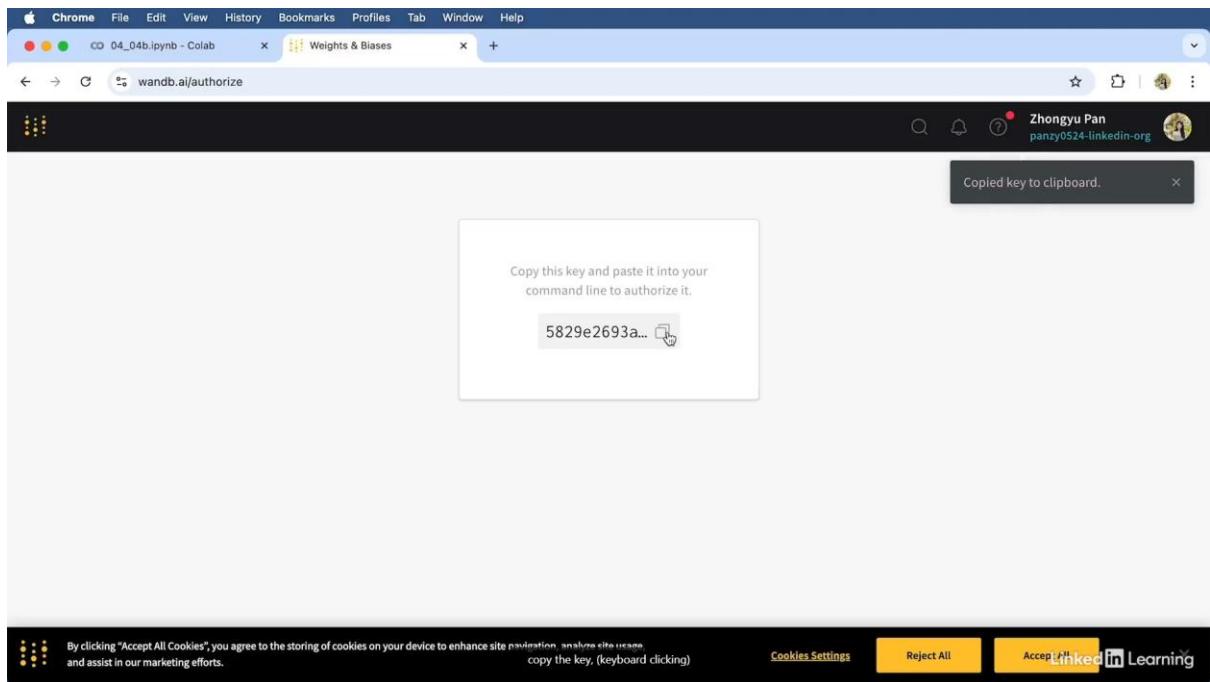
name@work-email.com

your password

Don't remember your password?

Sign in with your Google account.

LinkedIn Learning



The screenshot shows a Google Colab notebook titled "04\_04b.ipynb". The code cell contains the following Python code:

```
7     return {"accuracy": accuracy, "precision": precision, "recall": recall, "f1": f1}
8
9 # Initialize the Trainer
10 trainer = Trainer(
11     model=model,
12     args=training_args,
13     train_dataset=tokenized_train_data,
14     eval_dataset=tokenized_test_data,
15     compute_metrics=compute_metrics,
16 )
17
18 # Train the model
19 trainer.train()
```

Below the code, there is a series of Wandb logs:

```
... wandb: WARNING The 'run_name' is currently set to the same value as 'TrainingArguments.output_dir'. If this was not intended, please specify a different run
wandb: Using wandb-core as the SDK backend. Please refer to https://wandb.me/wandb-core for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter:
*****
```

The notebook interface includes standard Colab controls like "Share", "Run", and "Cell" dropdown menus. A "Linkedin Learning" watermark is visible in the bottom right corner.

04\_04b.ipynb - Colab    Weights & Biases

File Edit View Insert Runtime Tools Help

+ Code + Text

19m 19 trainer.train()

```
... wandb: WARNING The 'run_name' is currently set to the same value as `TrainingArguments.output_dir`. If this was not intended, please specify a different run
wandb: Using wandb-core as the SDK backend. Please refer to https://wandb.me/wandb-core for more information.
wandb: Logging into wandb.ai. (Learn how to deploy a W&B server locally: https://wandb.me/wandb-server)
wandb: You can find your API key in your browser here: https://wandb.ai/authorize
wandb: Paste an API key from your profile and hit enter: .....
wandb: WARNING If you're specifying your api key in code, ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or running 'wandb login' from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: panzy0524 (panzy0524-linkedin) to https://api.wandb.ai. Use 'wandb login --relogin' to force relogin
Tracking run with wandb version 0.19.7
Run data is saved locally in /content/wandb/run-20250303_213648-vppudlcw
Syncing run /results to Weights & Biases (docs)
View project at https://wandb.ai/panzy0524-linkedin/huggingface
View run at https://wandb.ai/panzy0524-linkedin/huggingface/runs/vppudlcw
[144/4689 00:23 <12:41, 5.97 it/s, Epoch 0.09/3]
```

Epoch Training Loss Validation Loss

```
[1] 1 # Evaluate the model
2
3 print("Evaluation results:", eval_result)
```

You'll also see the estimated time remaining.  
Executing (1m 35s) <cell line: 0> train() >\_inner\_training\_loop()

04\_04b.ipynb - Colab    Weights & Biases

File Edit View Insert Runtime Tools Help

+ Code + Text

19m [10] wandb: WARNING If you're specifying your api key in code, ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB\_API\_KEY environment variable, or running 'wandb login' from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: panzy0524 (panzy0524-linkedin) to <https://api.wandb.ai>. Use 'wandb login --relogin' to force relogin
Tracking run with wandb version 0.19.7
Run data is saved locally in /content/wandb/run-20250303\_213648-vppudlcw
Syncing run [/results](#) to Weights & Biases (docs)
View project at <https://wandb.ai/panzy0524-linkedin/huggingface>
View run at <https://wandb.ai/panzy0524-linkedin/huggingface/runs/vppudlcw>
[4689/4689 18:46, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.309200	0.332507	0.857760	0.913003	0.790880	0.847565
2	0.260800	0.354657	0.874440	0.874650	0.874160	0.874405
3	0.210700	0.521265	0.877160	0.874137	0.881200	0.877654

```
TrainOutput(global_step=4689, training_loss=0.2213985287296566, metrics={'train_runtime': 1193.0442, 'train_samples_per_second': 62.864, 'train_steps_per_second': 3.93, 'total_flos': 2483763724800000.0, 'train_loss': 0.2213985287296566, 'epoch': 3.0})
```

Once training is complete.

19m 58s completed at 1:55PM

Chrome File Edit View History Bookmarks Profiles Tab Window Help

04\_04b.ipynb - Colab Weights & Biases

colab.research.google.com/drive/tIQkbt20GSJVyJMcEMNmSqtHfJDIMTTG#scrollTo=20g1RFhOPP3p

**04\_04b.ipynb**

File Edit View Insert Runtime Tools Help

+ Code + Text

```
wandb: Paste an API key from your profile and hit enter: .....
wandb: WARNING If you're specifying your api key in code, ensure this code is not shared publicly.
wandb: WARNING Consider setting the WANDB_API_KEY environment variable, or running `wandb login` from the command line.
wandb: No netrc file found, creating one.
wandb: Appending key for api.wandb.ai to your netrc file: /root/.netrc
wandb: Currently logged in as: panzy@524 (panzy@524-linkedin) to https://api.wandb.ai. Use `wandb login --relogin` to force relogin
Tracking run with wandb version 0.19.7
Run data is saved locally in /content/wandb/run-20250303_213648-vppudlcw
Syncing run /results to Weights & Biases (docs)
View project at https://wandb.ai/panzy0524-linkedin/huggingface
View run at https://wandb.ai/panzy0524-linkedin/huggingface/runs/vppudlcw [4689/4689 18:46, Epoch 3/3]
```

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.309200	0.332507	0.857760	0.913003	0.790880	0.847565
2	0.260800	0.354657	0.874440	0.874650	0.874160	0.874405
3	0.210700	0.521265	0.877160	0.874137	0.881200	0.877654

```
TrainOutput(global_step=4689, training_loss=0.2213985287296566, metrics={'train_runtime': 1193.0442, 'train_samples_per_second': 62.864, 'train_steps_per_second': 3.93, 'total_flos': 2483763724800000.0, 'train_loss': 0.2213985287296566, 'epoch': 3.0})
```

```
1 # Evaluate the model
2 eval_result = trainer.evaluate()
3 print("Evaluation results:", eval_result)
```

19m 58s completed at 1:55PM

LinkedIn Learning

AI Sentiment Analysis with PyTorch and Hugging Face Transformers

04\_04b.ipynb - Colab Weights & Biases

colab.research.google.com/drive/tIQkbt20GSJVyJMcEMNmSqtHfJDIMTTG#scrollTo=Urr8KXNxAsdT

**04\_04b.ipynb**

File Edit View Insert Runtime Tools Help

+ Code + Text

```
Tracking run with wandb version 0.19.7
Run data is saved locally in /content/wandb/run-20250303_213648-vppudlcw
Syncing run /results to Weights & Biases (docs)
View project at https://wandb.ai/panzy0524-linkedin/huggingface
View run at https://wandb.ai/panzy0524-linkedin/huggingface/runs/vppudlcw [4689/4689 18:46, Epoch 3/3]
```

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.309200	0.332507	0.857760	0.913003	0.790880	0.847565
2	0.260800	0.354657	0.874440	0.874650	0.874160	0.874405
3	0.210700	0.521265	0.877160	0.874137	0.881200	0.877654

```
TrainOutput(global_step=4689, training_loss=0.2213985287296566, metrics={'train_runtime': 1193.0442, 'train_samples_per_second': 62.864, 'train_steps_per_second': 3.93, 'total_flos': 2483763724800000.0, 'train_loss': 0.2213985287296566, 'epoch': 3.0})
```

```
1 # Evaluate the model
2 eval_result = trainer.evaluate()
3 print("Evaluation results:", eval_result)
```

[1548/1563 01:29 < 00:00, 17.25 it/s]

1 Start coding or generate with AI.

201/204 cutting (1m 28s) <cell line: 0> > evaluated() > evaluation\_loop() > pad\_across\_processes() > wrapper() > pad\_across\_processes() > recursively\_apply() > \_pad\_across\_processes()

The screenshot shows a LinkedIn Learning quiz interface. At the top, there's a navigation bar with icons for search, profile, and language (EN). Below it, a header bar indicates the user has 633 lessons and is viewing the 'Chapter Quiz' for the 'AI Sentiment Analysis with PyTorch and Hugging Face Transformers' course. The main area displays four questions with their respective answers and feedback.

- Question 1 of 4:** What is the purpose of the TrainingArguments class?
  - to configure hyperparameters and settings for training process  
Correct  
TrainingArguments sets parameters like learning rate, batch size, and epochs.
  - to define the model architecture for training
  - to directly compute evaluation metrics during training
- Question 2 of 4:** What is the purpose of training arguments?
  - to set parameters for model training  
Correct  
Training arguments set parameters like batch size and learning rate.
  - to preprocess the dataset
  - to evaluate the model's performance
- Question 3 of 4:** Why do we use num\_labels=2 for this task?
  - It defines a binary classification problem.  
Correct  
The model predicts positive or negative sentiment.
  - It processes multi-class sentiment analysis.
  - It improves model performance.
- Question 4 of 4:** Why do we choose DistilBERT for the project?
  - because it uses more computational resources than BERT for higher accuracy
  - because it is a smaller and faster version of BERT  
Correct  
DistilBERT is designed to be lightweight and efficient.
  - because it doesn't require tokenization of input text

## Introduction to evaluation metrics

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In this chapter, we'll look at some commonly used evaluation metrics. Evaluation metrics tell us how well a model performs. This allows us to compare different models. By analyzing these metrics, we can make informed decisions on improving our model. Let's go over four key metrics. First is accuracy. It represents the percentage of correctly predicted labels. Second is precision. It measures how many of the predicted positives are actually correct. Recall focuses on the percentage of actual positives that were correctly identified. Lastly, the F1 score balances precision and recall. It provides a more comprehensive measure of model performance. In sentiment analysis, our task is a binary classification. Identifying positive or

negative sentiment. These metrics are especially useful for evaluating how well our model distinguishes between the two classes.

## Introduction to Evaluation Metrics



we'll look at some commonly used evaluation metrics.

LinkedIn Learning

## What Evaluation Metrics Do

- Measure how well the model performs
- Help compare different models
- Provide insights into areas of improvement

we can make informed decisions on improving our model.

LinkedIn Learning

## Common Evaluation Metrics

- Accuracy: overall correctness of predictions
- Precision: correct positive predictions out of all positive predictions made
- Recall: correct positive predictions out of all actual positives
- F1 score: balance between precision and recall

Lastly, the F1 score balances precision and recall.

LinkedIn Learning

## Binary Classification Context

- Two classes: positive and negative
- Metrics tailored to evaluate sentiment analysis performance

These metrics are especially useful for evaluating

LinkedIn Learning

Calculating accuracy, precision, recall, and F1 score

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In this video, we'll take a closer look at compute metrics function from the last coding exercise. There won't be any new coding session in this video. The compute metrics function calculates four key evaluation metrics. Let's break it down step by step-by-step. First, we extract

label IDs from the predictions object into the variable called labels. These are the true sentiment labels from our dataset, which we'll compare against the model's predictions. Next, we calculate preds or model predictions by applying the argmax function, which selects the class with the highest prediction scores. Now, the function calculates the four key metrics we introduced in the last video. I encourage you to implement them yourself based on their definitions. Finally, the function returns these metrics as a dictionary.

```
1 # Define evaluation metrics
2 def compute_metrics(pred):
3     labels = pred.label_ids
4     preds = np.argmax(pred.predictions, axis=1)
5     accuracy = accuracy_score(labels, preds)
6     precision, recall, f1, _ = precision_recall_fscore_support(labels, preds, average="binary")
7     return {"accuracy": accuracy, "precision": precision, "recall": recall, "f1": f1}
8
9 # Initialize the Trainer
10 trainer = Trainer(
11     model=model,
12     args=training_args,
13     train_dataset=tokenized_train_data,
14     eval_dataset=tokenized_test_data,
15     compute_metrics=compute_metrics,
16 )
17
18 Exercise Files > 04_04b.ipynb
19 trainer.train()
```

wandb: WARNING: The 'run\_name' is currently set to the same as the compute metrics function from the last output\_dir'. If this was not intended, please specify a different run  
wandb: Using wandb-core as the SDK backend. Please refer to <https://wandb.com> for more information.

## Analyzing results

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In this video, we'll look at evaluation results. The table shows results for each of the three epochs, but we'll only focus on the overall evaluation results here. The overall results show the model performs well with an accuracy of 85.776%, along with relatively high precision recall and F1 score. The model also processes data efficiently with a round time of 91 seconds and a throughput of 276 samples per second. These matrix give us confidence in the model's ability to classify movie reviews as positive or negative.

04\_04b.ipynb

```

1 # Evaluate the model
2 eval_result = trainer.evaluate()
3 print("Evaluation results:", eval_result)

```

Evaluation results: {'eval\_loss': 0.3325071930885315, 'eval\_accuracy': 0.85776, 'eval\_precision': 0.913003247137052, 'eval\_recall': 0.79088, 'eval\_f1': 0.8}

**Exercise Files > 04\_04b.ipynb**

The table shows results for each of the three epochs.

LinkedIn Learning

You answered 3 of 3 questions correctly.

Question 1 of 3  
Which of the following is NOT an evaluation metric?

- Latency
- Accuracy
- Correct
- Accuracy is a valid evaluation metric.
- F1 Score

Question 2 of 3  
What does an F1 score of 0.88 indicate?

- The model has a good balance between precision and recall.  
**Correct**  
An F1 score of 0.88 suggests a strong balance and good model performance.
- The model is not performing well because the score is lower than 1.
- The model has very high recall but low precision.

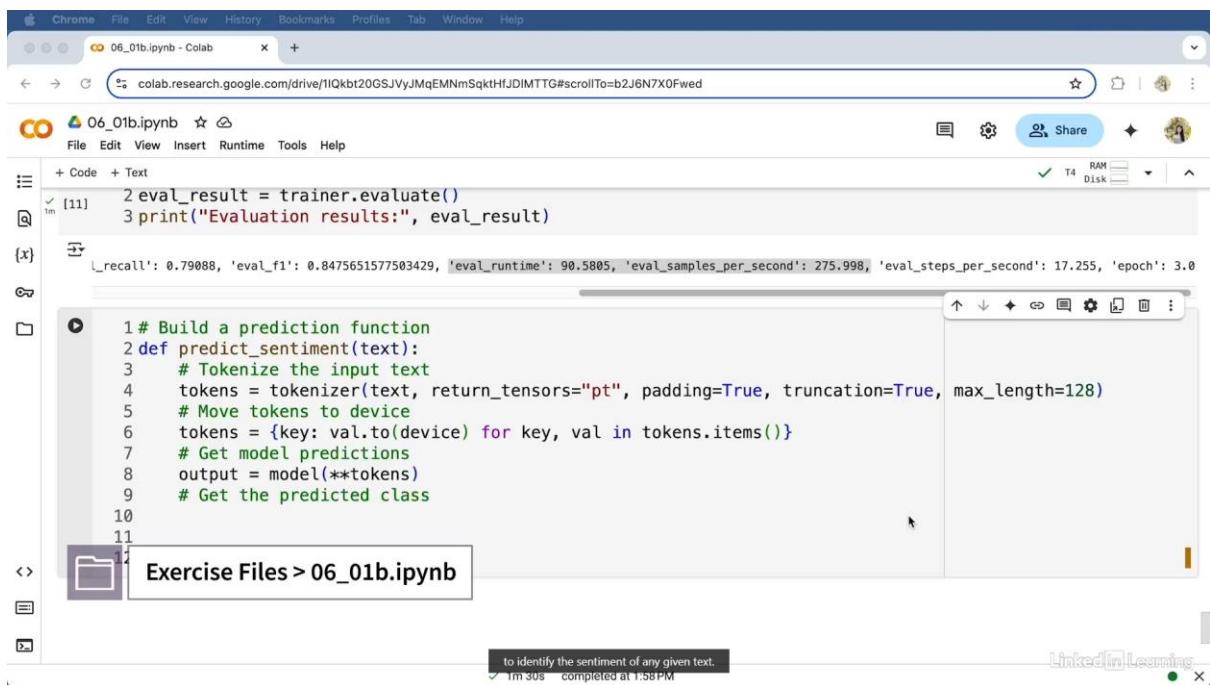
Question 3 of 3  
Which metric provides a balance between precision and recall in a classification task?

- Precision
- Accuracy
- F1 Score  
**Correct**  
F1 Score combines precision and recall to provide a single performance measure.

Let's build a prediction function

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] In this section, we'll create a prediction function to identify the sentiment of any given text. The function predict\_sentiment takes an input text, processes it, and returns whether the sentiment is positive or negative. The function tokenizes the input text, applies padding and truncation, and moves the tokens to the same device as the model. The model then processes them and produces raw prediction scores. Next, the function identifies the class with the highest score using torch.argmax. Finally, it converts the numerical class into a sentiment label, positive for class 1 and a negative for class 0. Then return the sentiment.



The screenshot shows a Google Colab notebook titled "06\_01b.ipynb". The code cell at the top contains the following Python code:

```
2 eval_result = trainer.evaluate()
3 print("Evaluation results:", eval_result)
```

The output cell below shows the evaluation results:

```
{x} {y}
  _recall': 0.79088, 'eval_f1': 0.8475651577503429, 'eval_runtime': 90.5805, 'eval_samples_per_second': 275.998, 'eval_steps_per_second': 17.255, 'epoch': 3.0
```

The main code block contains the following Python code:

```
1 # Build a prediction function
2 def predict_sentiment(text):
3     # Tokenize the input text
4     tokens = tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=128)
5     # Move tokens to device
6     tokens = {key: val.to(device) for key, val in tokens.items()}
7     # Get model predictions
8     output = model(**tokens)
9     # Get the predicted class
10
11
```

A tooltip at the bottom left of the code editor says "to identify the sentiment of any given text." A LinkedIn Learning watermark is visible in the bottom right corner.

```

 2 eval_result = trainer.evaluate()
 3 print("Evaluation results:", eval_result)

[x]   ↓_recall': 0.79088, 'eval_f1': 0.8475651577503429, 'eval_runtime': 90.5805, 'eval_samples_per_second': 275.998, 'eval_steps_per_second': 17.255, 'epoch': 3.0

[ ]  ↴
 1 # Build a prediction function
 2 def predict_sentiment(text):
 3     # Tokenize the input text
 4     tokens = tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=128)
 5     # Move tokens to device
 6     tokens = {key: val.to(device) for key, val in tokens.items()}
 7     # Get model predictions
 8     output = model(**tokens)
 9     # Get the predicted class
10     prediction = torch.argmax(output.logits, dim=1).item()
11
12     if prediction == 1:
13         sentiment = "Positive"
14     else:
15         sentiment = "Negative"
16
17     return sentiment

```

Finally, it converts the numerical class

1m 30s completed at 1:58PM

```

 2 eval_result = trainer.evaluate()
 3 print("Evaluation results:", eval_result)

[x]   ↓_recall': 0.79088, 'eval_f1': 0.8475651577503429, 'eval_runtime': 90.5805, 'eval_samples_per_second': 275.998, 'eval_steps_per_second': 17.255, 'epoch': 3.0

[ ]  ↴
 1 # Build a prediction function
 2 def predict_sentiment(text):
 3     # Tokenize the input text
 4     tokens = tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=128)
 5     # Move tokens to device
 6     tokens = {key: val.to(device) for key, val in tokens.items()}
 7     # Get model predictions
 8     output = model(**tokens)
 9     # Get the predicted class
10     prediction = torch.argmax(output.logits, dim=1).item()
11     if prediction == 1:
12         sentiment = "Positive"
13     else:
14         sentiment = "Negative"
15
16     return sentiment

```

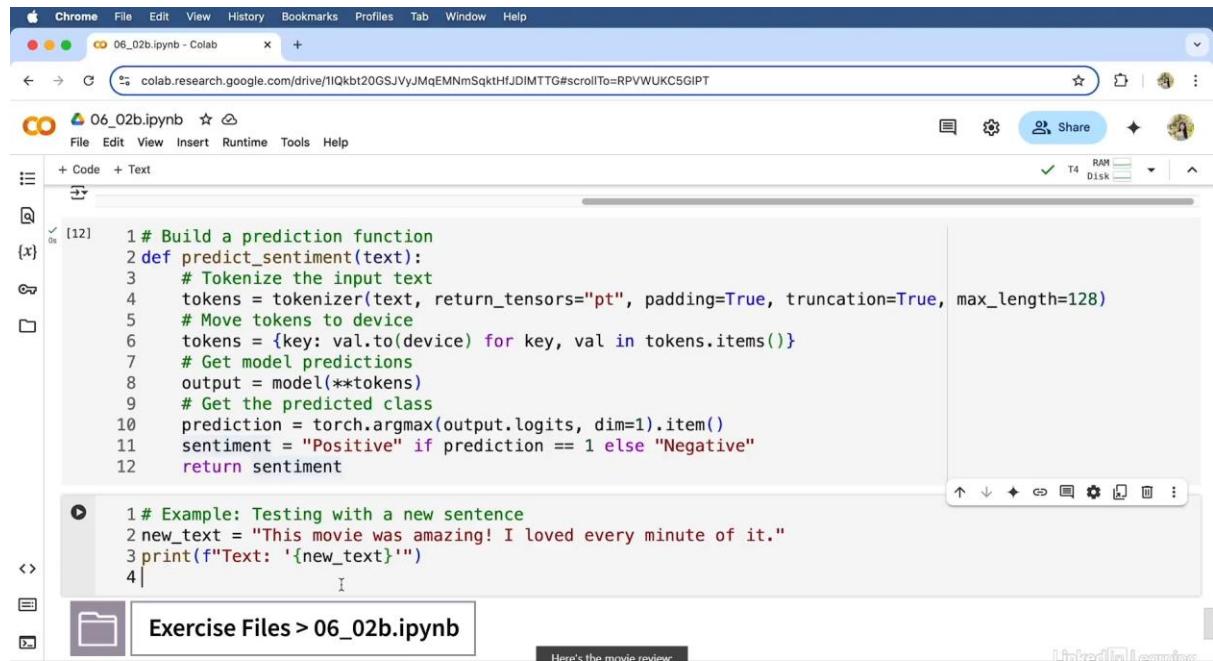
1m 30s completed at 1:58PM

## Test the model yourself

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] Let's test our model with a real example using print sentiment function. Here's the movie review: "This movie was amazing! I loved every minute of it." Clearly we expect the output label to be positive. We pass this

review to predict sentiment function, and the model will tell us if it's positive or negative. Let's print out the predicted sentiment. It is positive, awesome. Here comes the exciting part. Try testing it yourself with different examples and see how well the model captures the tone of movie reviews.



The screenshot shows a Google Colab notebook titled "06\_02b.ipynb". The code cell [12] contains a prediction function and an example test:

```
1 # Build a prediction function
2 def predict_sentiment(text):
3     # Tokenize the input text
4     tokens = tokenizer(text, return_tensors="pt", padding=True, truncation=True, max_length=128)
5     # Move tokens to device
6     tokens = {key: val.to(device) for key, val in tokens.items()}
7     # Get model predictions
8     output = model(**tokens)
9     # Get the predicted class
10    prediction = torch.argmax(output.logits, dim=1).item()
11    sentiment = "Positive" if prediction == 1 else "Negative"
12    return sentiment

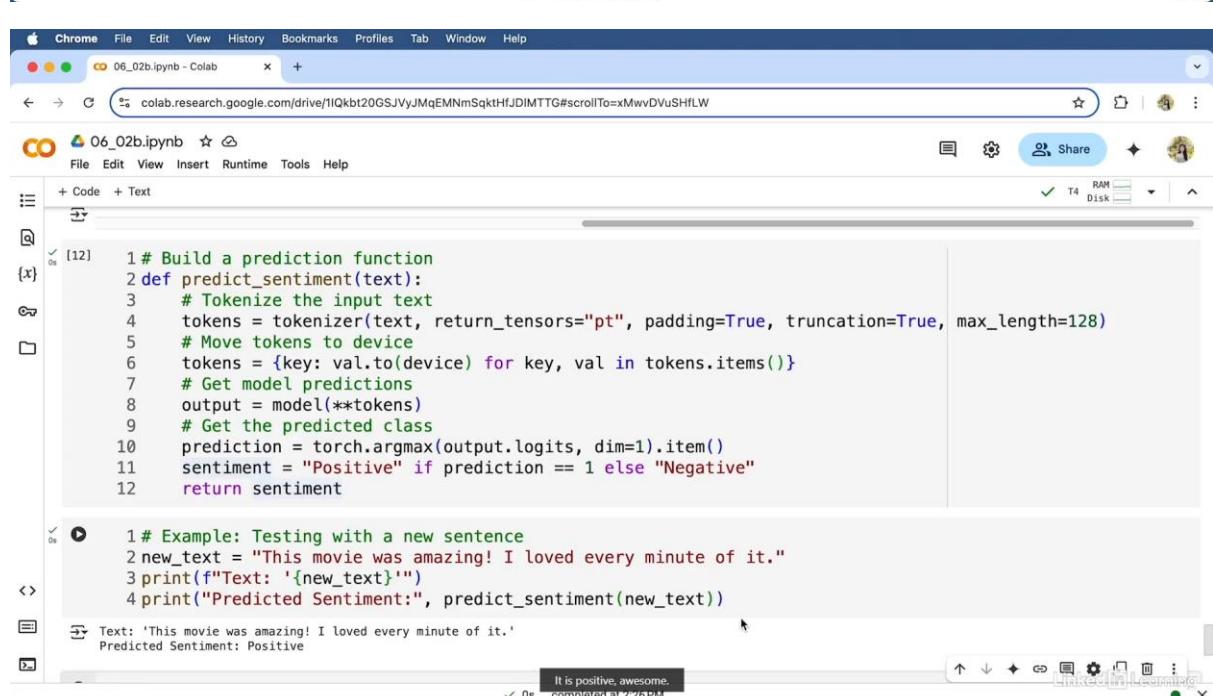
1 # Example: Testing with a new sentence
2 new_text = "This movie was amazing! I loved every minute of it."
3 print(f"Text: '{new_text}'")
4
```

The output cell shows the result of the print statement:

```
Text: 'This movie was amazing! I loved every minute of it.'
```

Below the code cell, a tooltip displays the predicted sentiment:

Here's the movie review:  
It is positive, awesome.  
0s completed at 2:26PM



The second screenshot shows the same notebook and code. The output cell now includes the predicted sentiment:

```
Text: 'This movie was amazing! I loved every minute of it.'
Predicted Sentiment: Positive
```

Below the output cell, a tooltip displays the predicted sentiment:

It is positive, awesome.  
0s completed at 2:26PM

## How to deal with edge cases

Selecting transcript lines in this section will navigate to timestamp in the video

- [Instructor] You might be wondering, what about edge cases? For example, consider sentences with mixed sentiments, like, "The movie was visually stunning, but the story was boring." These inputs contain both positive and negative elements, which can confuse the model. Another challenge is sarcasm, such as, "Oh, great, another masterpiece of boredom." Detecting sarcasm requires a deeper understanding of context. Handling edge cases often requires fine-tuning on specific datasets, preprocessing tricky inputs, or combining sentiment analysis with techniques like contextual embeddings or sentiment lexicons. I encourage you to explore these topics further to see how they help manage edge cases. When building applications, it's important to test thoroughly on a diverse dataset to identify potential weaknesses. And remember, no model is perfect, but understanding its limitations helps us improve.

## How to Deal with Edge Cases



what about edge cases?

LinkedIn Learning

## Edge Cases

- Mixed sentiments
- Sarcasm

"Oh, great, another masterpiece of boredom."

LinkedIn Learning

## How to Deal with Edge Cases

- Fine-tuning the model on specific data
- Preprocessing to handle tricky inputs
- Combining sentiment analysis with other techniques

or combining sentiment analysis with techniques

LinkedIn Learning

# Reminders

- Test the model thoroughly on a diverse dataset
- No model is perfect. Understanding its limitations helps us improve!

The screenshot shows a LinkedIn Learning quiz interface. At the top, a purple bar displays the text "And remember, no model is perfect," followed by the LinkedIn Learning logo. The main area shows a quiz titled "AI Sentiment Analysis with PyTorch and Hugging Face Transformers Chapter Quiz". A sidebar on the left includes icons for home, contents, search, and help. The quiz summary at the top indicates "You answered 3 of 3 questions correctly." Below are three questions:

- Question 1 of 3:** What is the main focus when dealing with edge cases?
  - improving the model's accuracy through fine-tuning
  - adding more layers to the neural network
  - handling text that might not fit into the categories of positive or negative sentiment  
Correct  
Edge cases often involve ambiguous inputs that may challenge the model's predictions.
- Question 2 of 3:** What does the code in this video do when testing with a new text input?
  - It predicts the sentiment of the next text.  
Correct  
The code passes the input text to the predict\_sentiment function and prints the predicted sentiment.
  - It trains the model with the new text input.
  - It shows training history of the model.
- Question 3 of 3:** What does predict\_sentiment function do?
  - to tokenize the text input
  - to classify the sentiment of a given text  
Correct  
The function uses the model to predict whether the sentiment of the input text is positive or negative.
  - to train the model on new data

At the bottom right of the interface are buttons for "Continue watching" and "Retake quiz". The top right corner shows user statistics: 633 lessons, a plus sign, and a refresh icon.

## Advanced NLP applications

Selecting transcript lines in this section will navigate to timestamp in the video

- [Narrator] Great job on completing the sentiment analysis project. Now let's take a break and check out some other exciting applications in NLP. Text summarization is a game changer for handling large amounts of information. Imagine reading a paper and using NLP to generate a short, concise summary. This is especially useful for news, research and any content where you need the key points without going through everything. Next is machine translation. Tools like Google Translate use NLP models to break down language barriers. These models enable real time translation, making it easier to communicate with people across the globe. Have you ever interacted with a chat bot? It is powered by question answering models. These models process questions and generate answers from customer support to retrieving information online. QA systems are great for delivering helpful and accurate responses.

## Advanced NLP Applications



## Text Summarization

- Automatically generating concise summaries of long documents
- Useful for news, research papers, and content curation

This is especially useful for news, research

LinkedIn Learning

## Machine Translation

- Translating text from one language to another using AI models
- Examples include Google Translate and other multilingual applications

These models enable real time translation,  
making it easier

LinkedIn Learning

## Question Answering (QA)

- Automatically answering questions based on a body of text
- Can be used in customer support, education, and search engines

QA systems are great for delivering helpful

LinkedIn Learning

Continue your AI sentiment analysis journey

Selecting transcript lines in this section will navigate to timestamp in the video

- Congratulations on completing the course. You now have a solid understanding and hands-on practice with sentiment analysis. This is just the beginning. I can't wait to see how you apply the skill you've learned to real-world projects. There's so much more to discover in the world of NLP. Stay curious and keep learning through online resources, research papers and NLP communities. And keep an eye out for more courses from me on advanced NLP applications. I hope you're excited to put your new skills into practice. Keep up the great work, and I look forward to seeing you soon.