BioBERT Fine Tuning

Class 0 ("DLT")   → Recall = 0.94
Class 1 ("Non-serious") → Recall = 1.00
Class 2 ("Serious") → Recall = 0.82

```
! pip install transformers
```

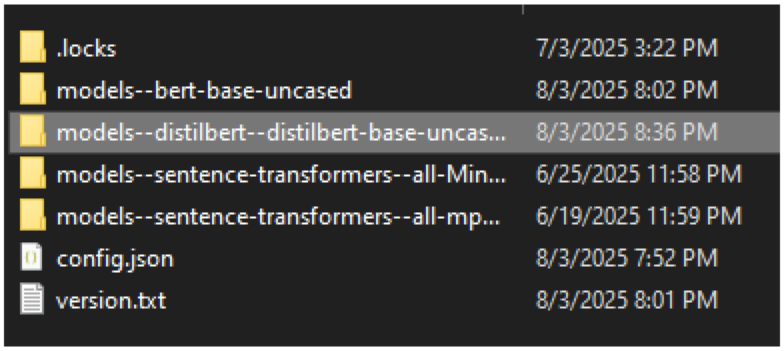
- BERT adds `CLS` at the beginning and `SEP` at end of the sentence.

```
from transformers import pipeline
```

- Pipeline helps you to call pretrained model

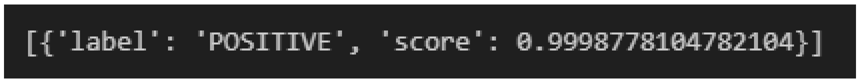
```
from transformers import pipeline
classifier = pipeline('sentiment-analysis')
```

- This will download `models--distilbert--distilbert-base-uncased-finetuned-sst-2-english` **model by default (255 MB)**



```
.locks                                       7/3/2025 3:22 PM
models--bert-base-uncased                    8/3/2025 8:02 PM
models--distilbert--distilbert-base-uncas...  8/3/2025 8:36 PM
models--sentence-transformers--all-Min...     6/25/2025 11:58 PM
models--sentence-transformers--all-mp...      6/19/2025 11:59 PM
config.json                                  8/3/2025 7:52 PM
version.txt                                  8/3/2025 8:01 PM
```

```
classifier("The movie is awesome")
```

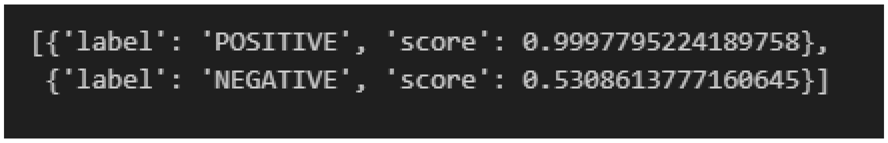


```
[{'label': 'POSITIVE', 'score': 0.9998778104782104}]
```

Multiple test:

- Provide list

```
classifier(["We are very happy to show you the 🤗 Transformers library.",
         "We hope you don't hate it."])
```



```
[{'label': 'POSITIVE', 'score': 0.9997795224189758},
 {'label': 'NEGATIVE', 'score': 0.5308613777160645}]
```

Fine Tuning using BioBERT (`dmis-lab/biobert-base-cased-v1.1`)

📌**Official model name:**

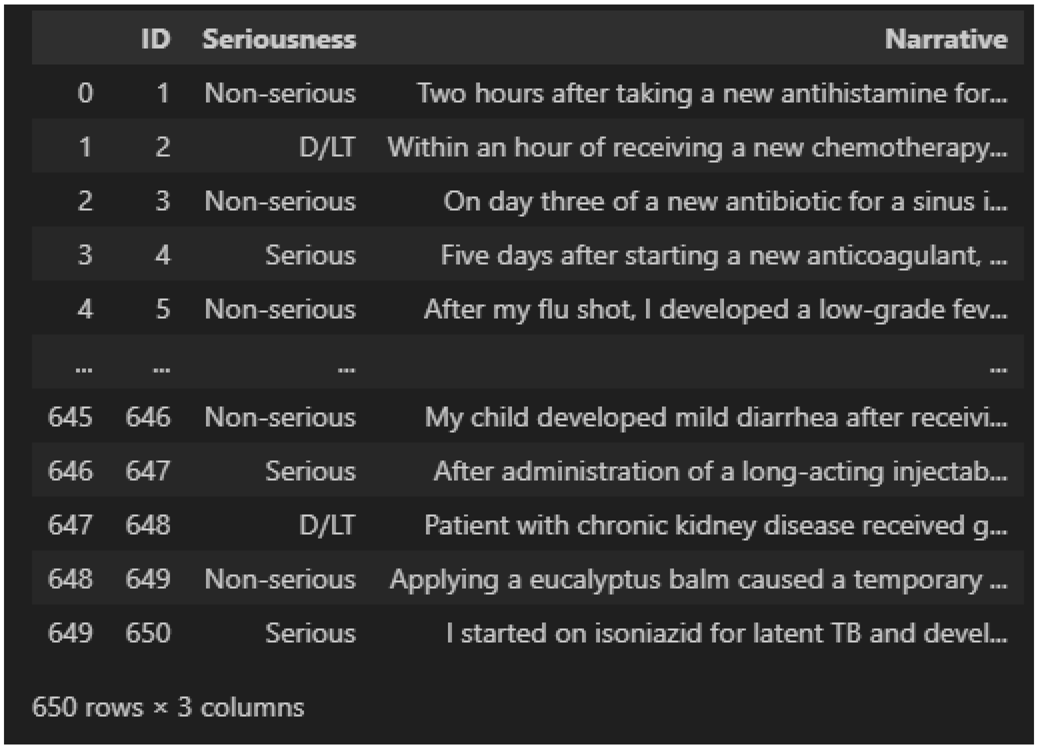
```
dmis-lab/biobert-base-cased-v1.1
```

Steps:

1. Call pretrained model
2. Tokenize using BioBERT
3. Convert the encodings into dataset objects

📦 Load and Prepare Your CSV Data

```
import pandas as pd

# Load your CSV file
df = pd.read_csv("PV Classification.csv")
df
```

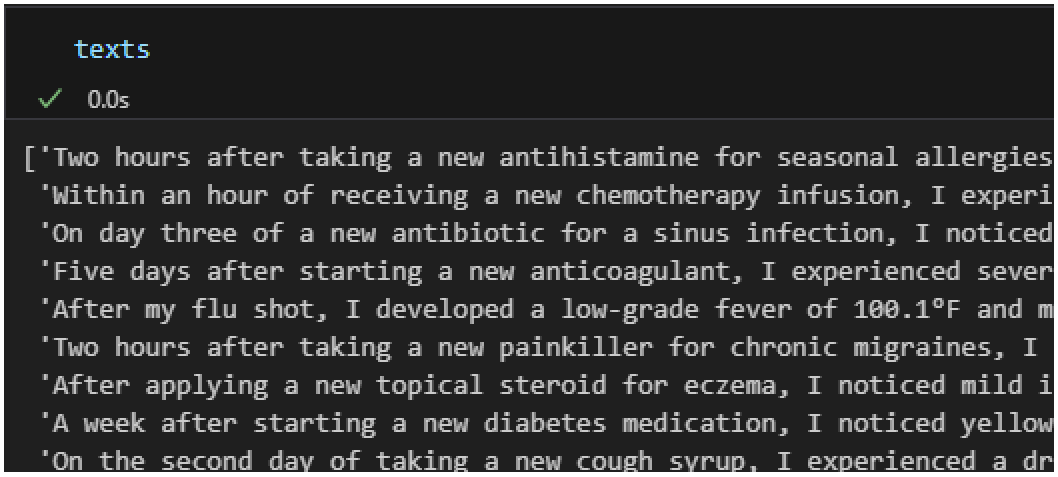


	ID	Seriousness	Narrative
0	1	Non-serious	Two hours after taking a new antihistamine for...
1	2	D/LT	Within an hour of receiving a new chemotherapy...
2	3	Non-serious	On day three of a new antibiotic for a sinus i...
3	4	Serious	Five days after starting a new anticoagulant, ...
4	5	Non-serious	After my flu shot, I developed a low-grade fev...
...	...	...	...
645	646	Non-serious	My child developed mild diarrhea after receivi...
646	647	Serious	After administration of a long-acting injectab...
647	648	D/LT	Patient with chronic kidney disease received g...
648	649	Non-serious	Applying a eucalyptus balm caused a temporary ...
649	650	Serious	I started on isoniazid for latent TB and devel...

650 rows × 3 columns

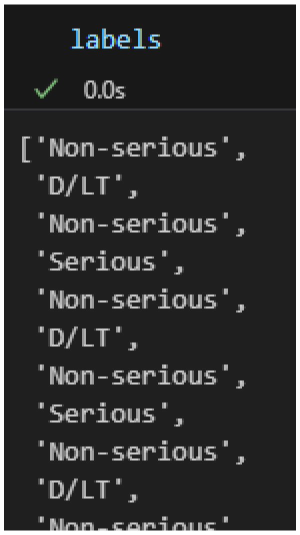
Convert into list

```
# Only keep the text and labels
texts = df["Narrative"].tolist()
labels = df["Seriousness"].tolist()
```



```
texts
✓ 0.0s
```

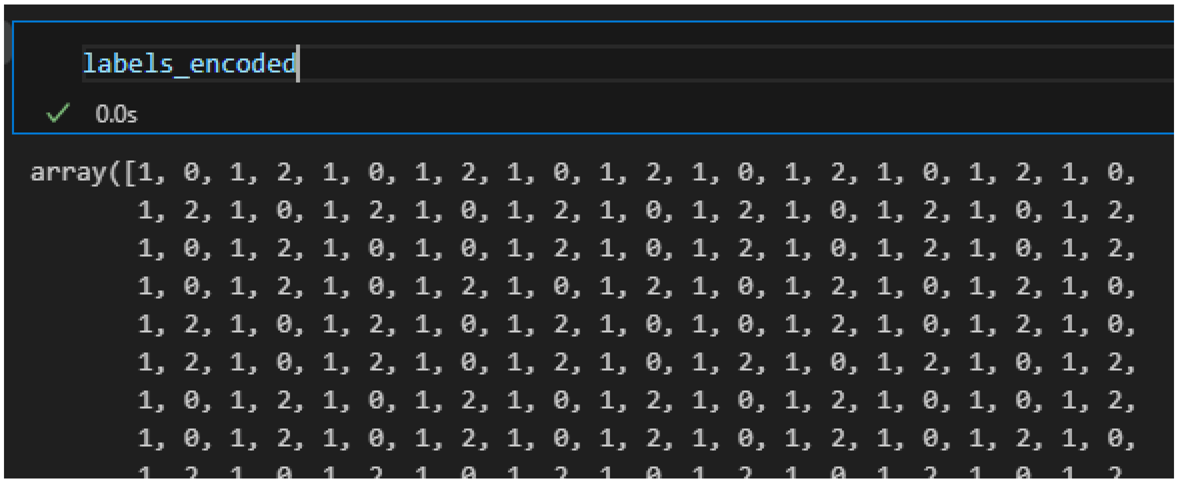
```
['Two hours after taking a new antihistamine for seasonal allergies
 'Within an hour of receiving a new chemotherapy infusion, I experi
 'On day three of a new antibiotic for a sinus infection, I noticed
 'Five days after starting a new anticoagulant, I experienced sever
 'After my flu shot, I developed a low-grade fever of 100.1°F and m
 'Two hours after taking a new painkiller for chronic migraines, I
 'After applying a new topical steroid for eczema, I noticed mild i
 'A week after starting a new diabetes medication, I noticed yellow
 'On the second day of taking a new cough syrup, I experienced a dr
```



```
labels
✓ 0.0s

['Non-serious',
 'D/LT',
 'Non-serious',
 'Serious',
 'Non-serious',
 'D/LT',
 'Non-serious',
 'Serious',
 'Non-serious',
 'D/LT',
 'Non-serious',
```

🏷️ Convert Labels to Numbers

```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(labels)
```



```
labels_encoded
✓ 0.0s

array([1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0,
       1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2,
       1, 0, 1, 2, 1, 0, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2,
       1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0,
       1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0,
       1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2,
       1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 0, 1, 2,
       1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0,
       1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2, 1, 0, 1, 2
```

`LabelEncoder` converts text labels like `"Non-serious"`, `"D/LT"` to integers like `0`, `1`, `2`.

🤖 Load BioBERT Tokenizer

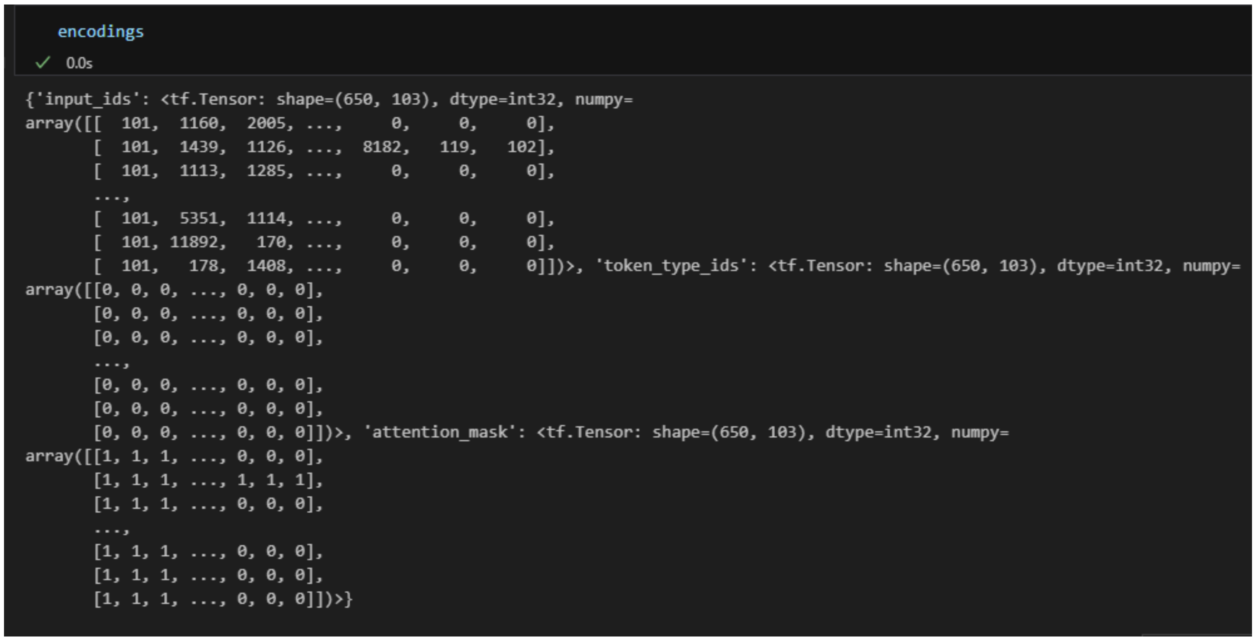
```
from transformers import AutoTokenizer

tokenizer = AutoTokenizer.from_pretrained("dmis-lab/biobert-base-cased-v1.1", cache_dir="hf_models")
```

This loads **BioBERT** tokenizer. It breaks the clinical sentence into special tokens that the model can understand.

🧹 Tokenize Text

```
# Convert text to input IDs and attention masks
encodings = tokenizer(texts, truncation=True, padding=True, max_length=256, return_tensors="tf")
```

- `truncation=True`: Cuts off long texts. (**Default Value**: `False` : truncation is **not applied** by default)
- `padding=True`: Ensures all sequences are of the same length. (**Default Value**: `False` (padding is **not applied** by default))
- `return_tensors="tf"`: Converts to TensorFlow tensors.



```
encodings
✓ 0.0s
```

```
{'input_ids': <tf.Tensor: shape=(650, 103), dtype=int32, numpy=
array([[  101,  1160,  2005, ...,     0,     0,     0],
       [  101,  1439,  1126, ...,  8182,   119,   102],
       [  101,  1113,  1285, ...,     0,     0,     0],
       ...,
       [  101,  5351,  1114, ...,     0,     0,     0],
       [  101, 11892,   170, ...,     0,     0,     0],
       [  101,   178,  1408, ...,     0,     0,     0]])>, 'token_type_ids': <tf.Tensor: shape=(650, 103), dtype=int32, numpy=
array([[0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       ...,
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]])>, 'attention_mask': <tf.Tensor: shape=(650, 103), dtype=int32, numpy=
array([[1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 1, 1, 1],
       [1, 1, 1, ..., 0, 0, 0],
       ...,
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0],
       [1, 1, 1, ..., 0, 0, 0]])>}
```

📊 Prepare Dataset for Training

Convert to dataset:

```
import tensorflow as tf
from tensorflow.data import Dataset

dataset = Dataset.from_tensor_slices(
    (dict(encodings), labels_encoded)
)
```

Shuffle and split:

```
train_size = int(0.8 * len(dataset))
train_dataset = dataset.take(train_size).shuffle(100).batch(8)
val_dataset = dataset.skip(train_size).batch(8)
```

```
int(0.8 * len(dataset))

Output: 520
```

`dataset.take(train_size)`: **take the first `train_size` samples** from the dataset.

`.shuffle(100)`: The argument `100` is the **buffer size** used for shuffling. It means that it will randomly shuffle the dataset using a buffer of 100 elements at a time.

`dataset.batch(8)`

- **Purpose**: The `batch()` function groups samples into batches, which is crucial for efficient training.

- The argument `8` is the **batch size**, which means the dataset will be split into batches of 8 samples each.
- **Why batching?**: Training a model on the entire dataset at once might be very memory-intensive, especially for large datasets. Instead, you train your model on small chunks or **mini-batches** of data.
    - A batch size of 8 means the model will receive 8 samples at a time for training

🧠 Load BioBERT Model for Classification

```
from transformers import TFAutoModelForSequenceClassification

# Load the PyTorch weights and convert them to TensorFlow
model = TFAutoModelForSequenceClassification.from_pretrained(
    "dmis-lab/biobert-base-cased-v1.1",
    num_labels=3,
    from_pt=True,  # This tells it to load from PyTorch weights
    cache_dir="hf_models"
)
```

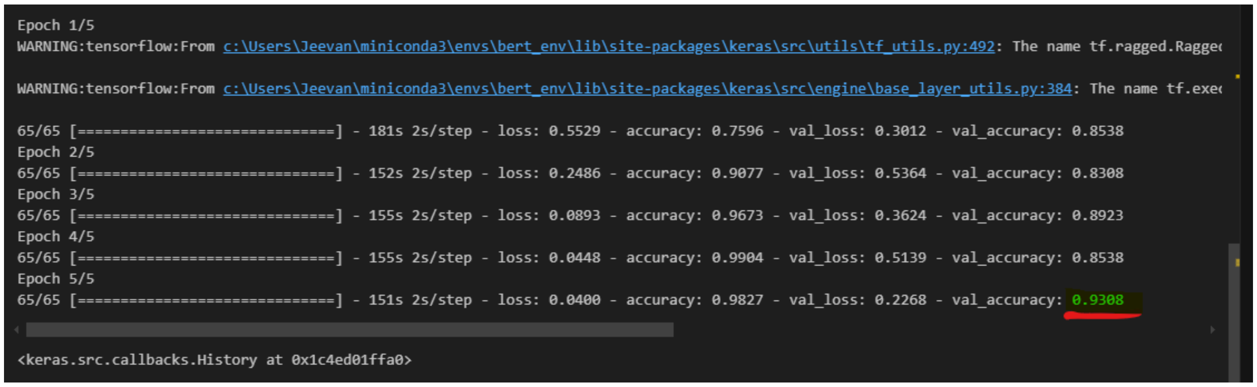
- `TFAutoModelForSequenceClassification`: Loads BERT with classification head on top.
- `num_labels=3`: Number of output categories.

⚙️ Compile the Model

```
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.losses import SparseCategoricalCrossentropy

optimizer = Adam(learning_rate=5e-5)
loss = SparseCategoricalCrossentropy(from_logits=True)

model.compile(optimizer=optimizer, loss=loss, metrics=["accuracy"])
```

🚀 Train the Model

```
model.fit(
    train_dataset,
    validation_data=val_dataset,
    epochs=5
)
```



```
Epoch 1/5
WARNING:tensorflow:From c:\Users\Jeevan\miniconda3\envs\bert_env\lib\site-packages\keras\src\utils\tf_utils.py:492: The name tf.ragged.Ragge

WARNING:tensorflow:From c:\Users\Jeevan\miniconda3\envs\bert_env\lib\site-packages\keras\src\engine\base_layer_utils.py:384: The name tf.exe

65/65 [==============================] - 181s 2s/step - loss: 0.5529 - accuracy: 0.7596 - val_loss: 0.3012 - val_accuracy: 0.8538
Epoch 2/5
65/65 [==============================] - 152s 2s/step - loss: 0.2486 - accuracy: 0.9077 - val_loss: 0.5364 - val_accuracy: 0.8308
Epoch 3/5
65/65 [==============================] - 155s 2s/step - loss: 0.0893 - accuracy: 0.9673 - val_loss: 0.3624 - val_accuracy: 0.8923
Epoch 4/5
65/65 [==============================] - 155s 2s/step - loss: 0.0448 - accuracy: 0.9904 - val_loss: 0.5139 - val_accuracy: 0.8538
Epoch 5/5
65/65 [==============================] - 151s 2s/step - loss: 0.0400 - accuracy: 0.9827 - val_loss: 0.2268 - val_accuracy: 0.9308

<keras.src.callbacks.History at 0x1c4ed01ffa0>
```

💡 **Took 13+ Mins**

Common question:

🔷 1. Was it necessary to convert `Narrative` and `Seriousness` into a list?

✅ Yes

- HuggingFace `Tokenizer` expects a **list of strings**, where each item is one **sentence** (narrative).
- Your data is in a `pandas DataFrame`. Each column is a `Series`, not a list.

🔷2. What is `max_length=256`?

Each sentence is converted into a maximum of 256 **tokens**.

If the sentence is longer → it gets **truncated.**

If it's shorter → it gets **padded** (extra `[PAD]` tokens added to reach 256 length).

Evaluate and Predict

🔷 3. Why we used `SparseCategoricalCrossentropy` as loss?

Because our labels are:

- **Categorical**, not continuous (e.g., "non-serious", "serious", "death")
- But **integer-encoded**, not one-hot encoded

You have 3 classes:

```
Non-serious → 0
D/LT → 1
Serious → 2
```

And you pass them as integers. So you can't use `CategoricalCrossentropy` (which expects one-hot labels like `[1, 0, 0]`).

🔷 4. What is `from_logits=True`?

A **logit** is a raw output from the model **before applying softmax**.

Let's say:

```
model output → [1.5, 0.5, -2.0]  (called logits)
```

To convert it into probabilities:

```
softmax([1.5, 0.5, -2.0]) → [0.73, 0.26, 0.01]
```

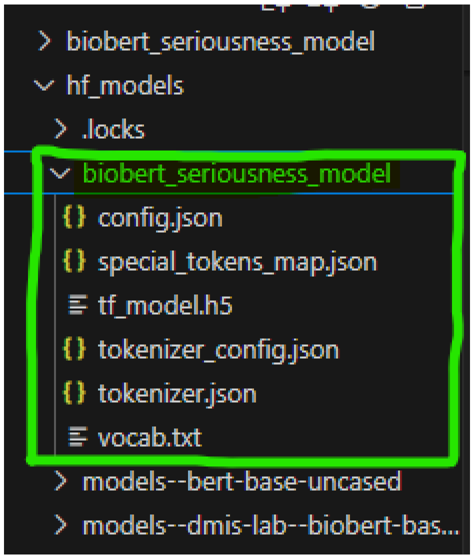
If you're using the model output directly, you should tell the loss function:

> “Hey, I’m giving you logits, please apply softmax inside.”

That’s what `from_logits=True` does. If you set it to `False`, the model output must already be softmaxed.

Save the Model

```
model.save_pretrained("./hf_models/biobert_seriousness_model")
tokenizer.save_pretrained("./hf_models/biobert_seriousness_model")
```

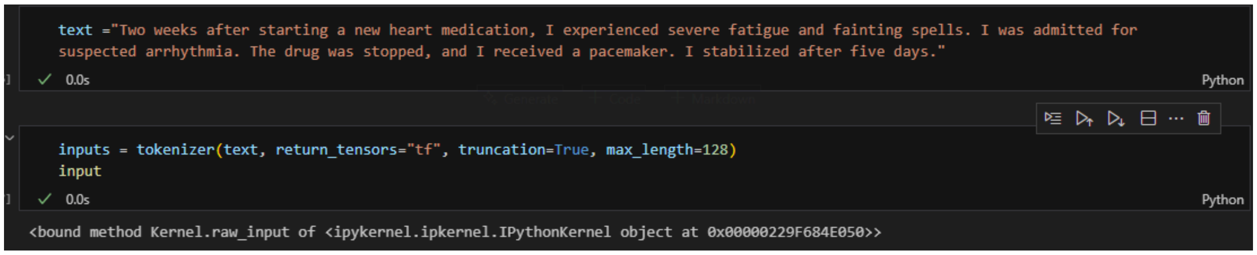


```
biobert_seriousness_model
hf_models
  .locks
  biobert_seriousness_model
    config.json
    special_tokens_map.json
    tf_model.h5
    tokenizer_config.json
    tokenizer.json
    vocab.txt
  models--bert-base-uncased
  models--dmis-lab--biobert-bas...
```

✅ Load the Saved Model

```
from transformers import TFBertForSequenceClassification, BertTokenizer

# Load from the directory you saved to
model = TFBertForSequenceClassification.from_pretrained("./hf_models/biob
ert_seriousness_model")
tokenizer = BertTokenizer.from_pretrained("./hf_models/biobert_seriousness_
model")
```

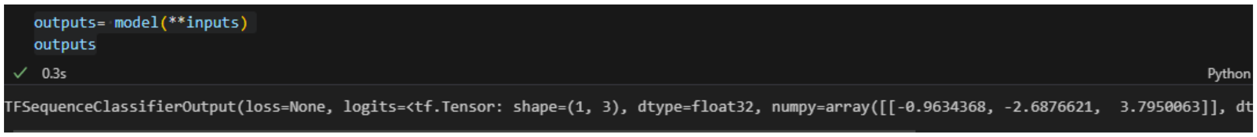
```
import tensorflow as tf

def predict_seriousness(text):
    # Tokenize text
    inputs = tokenizer(text, return_tensors="tf", truncation=True, max_length=1
28)

    # Get model predictions
    outputs = model(**inputs)

    # Convert logits to probabilities (for all 3 classes)
    proba = tf.nn.softmax(outputs.logits).numpy()[0]

    # Get predicted class (0, 1, or 2)
    predicted_class = tf.argmax(proba).numpy()

    # Map to label names
    label_names = ["DLT", "Non-serious", "Serious"]  # Must match your trainin
g labels
    return label_names[predicted_class], "Confidence:", proba[predicted_class]
```



```
text ="Two weeks after starting a new heart medication, I experienced severe fatigue and fainting spells. I was admitted for
suspected arrhythmia. The drug was stopped, and I received a pacemaker. I stabilized after five days."
```

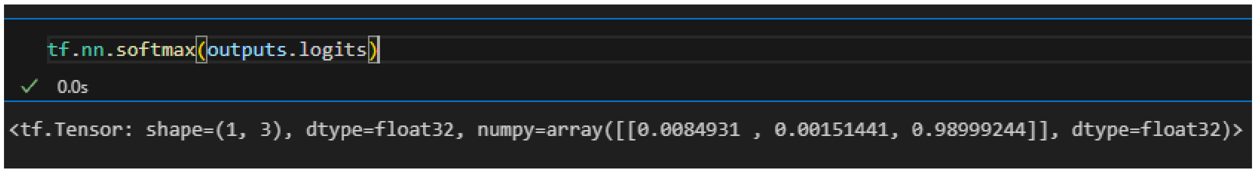
```
inputs = tokenizer(text, return_tensors="tf", truncation=True, max_length=128)
input
```

```
<bound method Kernel.raw_input of <ipykernel.ipkernel.IPythonKernel object at 0x00000229F684E050>>
```

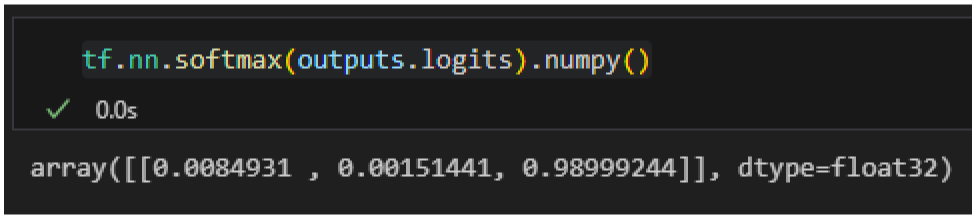


```
outputs= model(**inputs)
outputs
```

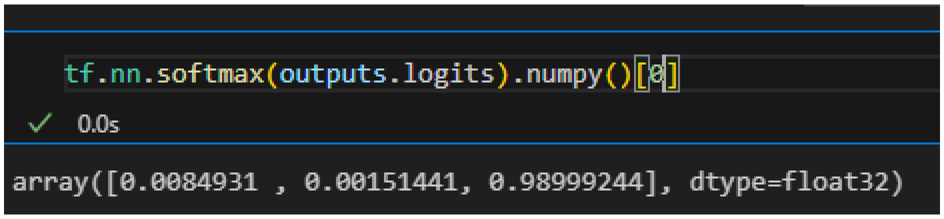
```
TFSequenceClassifierOutput(loss=None, logits=<tf.Tensor: shape=(1, 3), dtype=float32, numpy=array([[-0.9634368, -2.6876621,  3.7950063]], dt
```



```
tf.nn.softmax(outputs.logits)
✓ 0.0s
<tf.Tensor: shape=(1, 3), dtype=float32, numpy=array([[0.0084931 , 0.00151441, 0.98999244]], dtype=float32)>
```

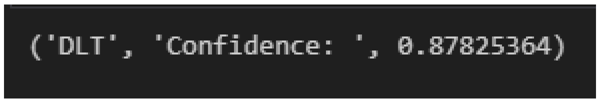


```
tf.nn.softmax(outputs.logits).numpy()
✓ 0.0s
array([[0.0084931 , 0.00151441, 0.98999244]], dtype=float32)
```



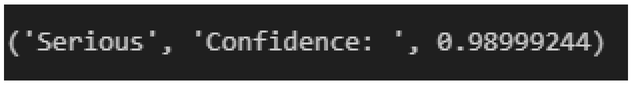
```
tf.nn.softmax(outputs.logits).numpy()[0]
✓ 0.0s
array([0.0084931 , 0.00151441, 0.98999244], dtype=float32)
```

```
predict_seriousness("patient died")
```



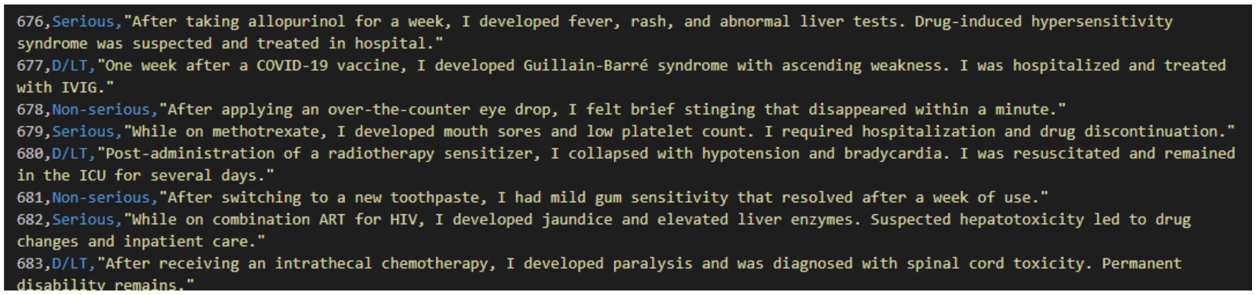
```
('DLT', 'Confidence: ', 0.87825364)
```

```
predict_seriousness("Two weeks after starting a new heart medication, I expe
rienced severe fatigue and fainting spells. I was admitted for suspected arrhyt
hmia. The drug was stopped, and I received a pacemaker. I stabilized after fiv
e days.")
```



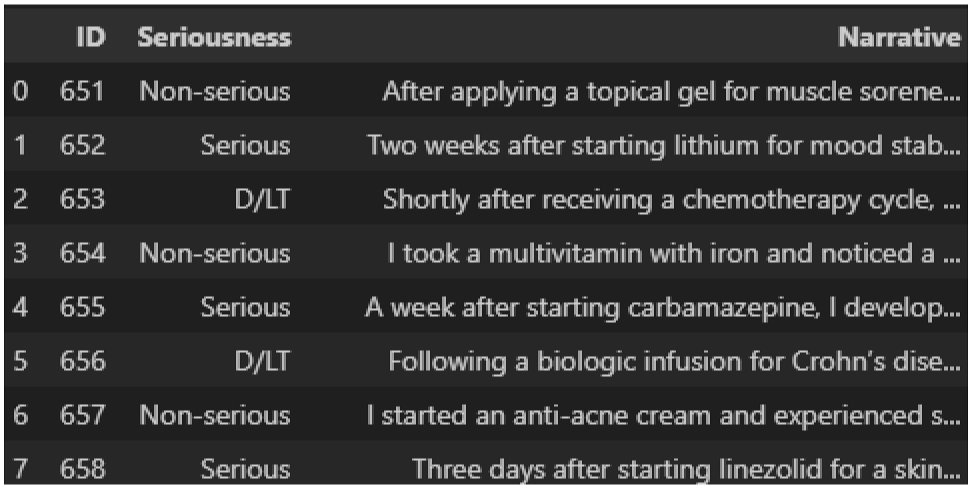
```
('Serious', 'Confidence: ', 0.98999244)
```

Test on unseen data



```
676,Serious,"After taking allopurinol for a week, I developed fever, rash, and abnormal liver tests. Drug-induced hypersensitivity
syndrome was suspected and treated in hospital."
677,D/LT,"One week after a COVID-19 vaccine, I developed Guillain-Barré syndrome with ascending weakness. I was hospitalized and treated
with IVIG."
678,Non-serious,"After applying an over-the-counter eye drop, I felt brief stinging that disappeared within a minute."
679,Serious,"While on methotrexate, I developed mouth sores and low platelet count. I required hospitalization and drug discontinuation."
680,D/LT,"Post-administration of a radiotherapy sensitizer, I collapsed with hypotension and bradycardia. I was resuscitated and remained
in the ICU for several days."
681,Non-serious,"After switching to a new toothpaste, I had mild gum sensitivity that resolved after a week of use."
682,Serious,"While on combination ART for HIV, I developed jaundice and elevated liver enzymes. Suspected hepatotoxicity led to drug
changes and inpatient care."
683,D/LT,"After receiving an intrathecal chemotherapy, I developed paralysis and was diagnosed with spinal cord toxicity. Permanent
disability remains."
```

```
import pandas as pd

# Load your test CSV (same format as training data)
test_df = pd.read_csv("test.csv")  # Columns: ID, Seriousness, Narrative

test_df
```



	ID	Seriousness	Narrative
0	651	Non-serious	After applying a topical gel for muscle sorene...
1	652	Serious	Two weeks after starting lithium for mood stab...
2	653	D/LT	Shortly after receiving a chemotherapy cycle, ...
3	654	Non-serious	I took a multivitamin with iron and noticed a ...
4	655	Serious	A week after starting carbamazepine, I develop...
5	656	D/LT	Following a biologic infusion for Crohn's dise...
6	657	Non-serious	I started an anti-acne cream and experienced s...
7	658	Serious	Three days after starting linezolid for a skin...

🔷Load Model

```
import tensorflow as tf
from transformers import TFBertForSequenceClassification, BertTokenizerFas
t



model = TFBertForSequenceClassification.from_pretrained("./hf_models/biob
ert_seriousness_model")
```

```
tokenizer = BertTokenizer.from_pretrained("./hf_models/biobert_seriousness_
model")
encodings = tokenizer(texts, truncation=True, padding=True, max_length=25
6, return_tensors="tf")
```

🔷 3.
Load and prepare test data

```
texts = test_df["Narrative"].tolist()
labels = test_df["Seriousness"].tolist()
```

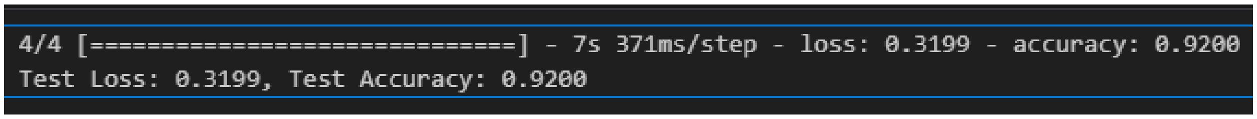
🔷 4.
Preprocess test data

```
label_encoder = LabelEncoder()
labels_encoded = label_encoder.fit_transform(labels)
```

```
test_dataset = tf.data.Dataset.from_tensor_slices((dict(encodings), labels_enc
oded)).batch(16)
```

🔷 5.
Evaluate using model.evaluate()

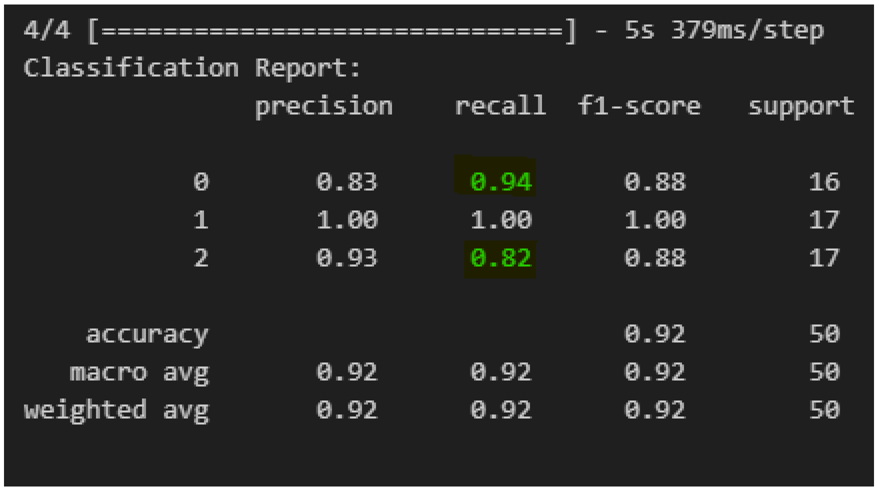
```
# Compile the model for evaluation
from tensorflow.keras.losses import SparseCategoricalCrossentropy
from tensorflow.keras.optimizers import Adam

model.compile(
    optimizer=Adam(learning_rate=5e-5),
    loss=SparseCategoricalCrossentropy(from_logits=True),
    metrics=["accuracy"]
)

# Evaluate
loss, accuracy = model.evaluate(test_dataset)
print(f"Test Loss: {loss:.4f}, Test Accuracy: {accuracy:.4f}")
```



```
4/4 [==============================] - 7s 371ms/step - loss: 0.3199 - accuracy: 0.9200
Test Loss: 0.3199, Test Accuracy: 0.9200
```

confusion_matrix

```
from sklearn.metrics import classification_report, confusion_matrix

# Get predictions
y_pred_logits = model.predict(test_dataset).logits
y_pred = tf.argmax(y_pred_logits, axis=1).numpy()

# Decode labels (optional)
y_true = labels_encoded

print("Classification Report:\n", classification_report(y_true, y_pred))
```



```
4/4 [==============================] - 5s 379ms/step
Classification Report:
               precision    recall  f1-score   support

           0       0.83      0.94      0.88        16
           1       1.00      1.00      1.00        17
           2       0.93      0.82      0.88        17

    accuracy                           0.92        50
   macro avg       0.92      0.92      0.92        50
weighted avg       0.92      0.92      0.92        50
```

✅per-class recall:

```
Class 0 ("DLT")      → Recall = 0.94
Class 1 ("Non-serious") → Recall = 1.00
Class 2 ("Serious") → Recall = 0.82
```

his means:

- You are **very good at catching DLTs** ✅
- You're **missing 18% of serious cases** ❌ ← needs attention