```
import pandas as pd
from sklearn.model_selection import train_test_split
from transformers import BertTokenizer, TFBertForSequenceClassification
import tensorflow as tf
```

```
df = pd.read_csv('/content/Dataset.csv')
df.head()
```

	Unnamed: 0	Subject	Body	Type
0	1.0	Webinar on The Emergence and Future of Generat...	Dear Members,IEEE Industrial Electronics Socie...	yes
1	2.0	Invitation to Attend Zoom Meeting	Dear [Student's Name],\n\nI hope this email fi...	yes
2	3.0	Zoom Meeting Reminder	Hi John, Just a quick reminder about our Zoom ...	yes
3	4.0	Get caught up in May Madness and save up to £5...	Hi ,\nMay is a bit of a mad month.\n\nWe dance...	no
4	5.0	✨NEW✨ Microsoft Professional Certificates!	\nNEW Microsoft Professional Certificates\nThe...	no

```
df['Type'] = df['Type'].replace(['yes', 'no'], [1, 0])
df = df.drop_duplicates(keep='first')
df.head()
```

```
<ipython-input-4-6dd09ced509a>:1: FutureWarning: Downcasting behavior in `replace` is deprecated and will be removed in a future version
  df['Type'] = df['Type'].replace(['yes', 'no'], [1, 0])
```

	Unnamed: 0	Subject	Body	Type
0	1.0	Webinar on The Emergence and Future of Generat...	Dear Members,IEEE Industrial Electronics Socie...	1.0
1	2.0	Invitation to Attend Zoom Meeting	Dear [Student's Name],\n\nI hope this email fi...	1.0
2	3.0	Zoom Meeting Reminder	Hi John, Just a quick reminder about our Zoom ...	1.0
3	4.0	Get caught up in May Madness and save up to £5...	Hi ,\nMay is a bit of a mad month.\n\nWe dance...	0.0
4	5.0	✨NEW✨ Microsoft Professional Certificates!	\nNEW Microsoft Professional Certificates\nThe...	0.0

```
df = df.drop(['Unnamed: 0', 'Subject'], axis=1)
```

```
df.head()
```

	Body	Type
0	Dear Members,IEEE Industrial Electronics Socie...	1.0
1	Dear [Student's Name],\n\nI hope this email fi...	1.0
2	Hi John, Just a quick reminder about our Zoom ...	1.0
3	Hi ,\nMay is a bit of a mad month.\n\nWe dance...	0.0
4	\nNEW Microsoft Professional Certificates\nThe...	0.0

```
df.dropna(inplace=True)
```

```
df.shape
```

```
(496, 2)
```

```
train_df, remaining = train_test_split(df, random_state=42, train_size=0.8, stratify=df.Type.values)
valid_df, test_df = train_test_split(remaining, random_state=42, train_size=0.5, stratify=remaining.Type.values)
```

```
print(train_df.shape)
print(valid_df.shape)
print(test_df.shape)
```

```
(396, 2)
(50, 2)
(50, 2)
```

```
# Tokenize the data
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

```
tokenizer_config.json:  100%                                   48.0/48.0  [00:00<00:00,  823B/s]

vocab.txt:  100%                                            232k/232k  [00:00<00:00,  1.07MB/s]

tokenizer.json:  100%                                         466k/466k  [00:00<00:00,  2.17MB/s]

config.json:  100%                                           570/570  [00:00<00:00,  13.3kB/s]

/usr/local/lib/python3.10/dist-packages/transformers/tokenization_utils_base.py:1601: FutureWarning: `clean_up_tokenization_spaces` was
  warnings.warn(
```

```
def encode_data(data, tokenizer, max_len):
    input_ids = []
    attention_masks = []
    for body in data:
        encoded = tokenizer.encode_plus(
            body,
            add_special_tokens=True,
            max_length=max_len,
            padding='max_length',
            truncation=True,
            return_attention_mask=True
        )
        input_ids.append(encoded['input_ids'])
        attention_masks.append(encoded['attention_mask'])
    return tf.convert_to_tensor(input_ids), tf.convert_to_tensor(attention_masks)
```

```
# Encode the data
max_len = 128
train_input_ids, train_attention_masks = encode_data(train_df.Body.values, tokenizer, max_len)
valid_input_ids, valid_attention_masks = encode_data(valid_df.Body.values, tokenizer, max_len)

train_labels = tf.convert_to_tensor(train_df.Type.values)
valid_labels = tf.convert_to_tensor(valid_df.Type.values)
test_labels = tf.convert_to_tensor(test_df.Type.values)

# Create TensorFlow datasets
train_dataset = tf.data.Dataset.from_tensor_slices(({'input_ids': train_input_ids, 'attention_mask': train_attention_masks}, train_labels))
valid_dataset = tf.data.Dataset.from_tensor_slices(({'input_ids': valid_input_ids, 'attention_mask': valid_attention_masks}, valid_labels))
test_dataset = tf.data.Dataset.from_tensor_slices(({'input_ids': valid_input_ids, 'attention_mask': valid_attention_masks}, test_labels))

batch_size = 8
train_dataset = train_dataset.shuffle(len(train_df)).batch(batch_size)
valid_dataset = valid_dataset.batch(batch_size)
test_dataset = test_dataset.batch(batch_size)
```

```
# Load the model
model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)
```

```
All PyTorch model weights were used when initializing TFBertForSequenceClassification.

Some weights or buffers of the TF 2.0 model TFBertForSequenceClassification were not initialized from the PyTorch model and are newly in
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.
```

```
# Compile the model
optimizer = tf.keras.optimizers.Adam(learning_rate=3e-5, epsilon=1e-08)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metric = tf.keras.metrics.SparseCategoricalAccuracy('accuracy')

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

```
# Train the model
epochs = 7
history = model.fit(train_dataset, validation_data=valid_dataset, epochs=epochs)
```

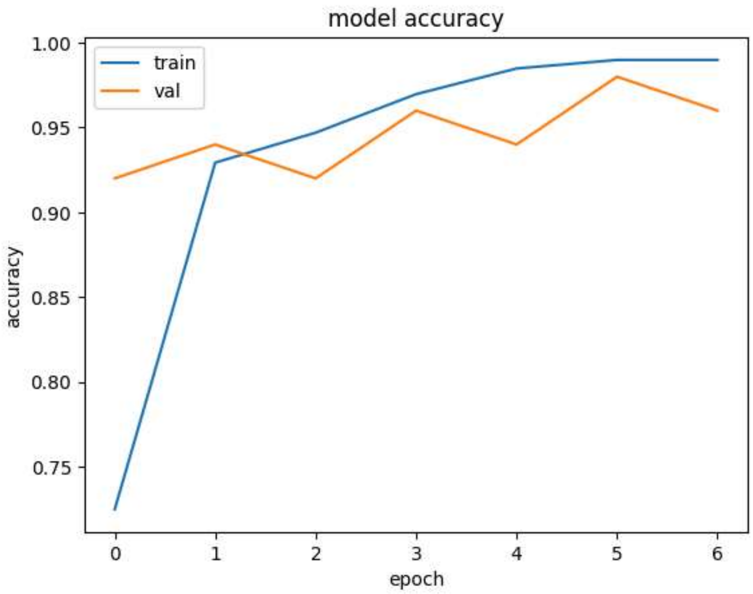
```
Epoch 1/7
50/50 [==============================] - 53s 379ms/step - loss: 0.5463 - accuracy: 0.7247 - val_loss: 0.2678 - val_accuracy: 0.9200
Epoch 2/7
50/50 [==============================] - 12s 246ms/step - loss: 0.2250 - accuracy: 0.9293 - val_loss: 0.2152 - val_accuracy: 0.9400
Epoch 3/7
50/50 [==============================] - 12s 246ms/step - loss: 0.1547 - accuracy: 0.9470 - val_loss: 0.1957 - val_accuracy: 0.9200
Epoch 4/7
```

```
50/50 [==============================] - 12s 247ms/step - loss: 0.0855 - accuracy: 0.9697 - val_loss: 0.1534 - val_accuracy: 0.9600
Epoch 5/7
50/50 [==============================] - 12s 244ms/step - loss: 0.0481 - accuracy: 0.9848 - val_loss: 0.1932 - val_accuracy: 0.9400
Epoch 6/7
50/50 [==============================] - 12s 241ms/step - loss: 0.0323 - accuracy: 0.9899 - val_loss: 0.1128 - val_accuracy: 0.9800
Epoch 7/7
50/50 [==============================] - 12s 241ms/step - loss: 0.0286 - accuracy: 0.9899 - val_loss: 0.2036 - val_accuracy: 0.9600
```

```
# prompt: plotthe accuracy graph

# Import the matplotlib.pyplot module
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()
```



```
# Define the input string
text_mail = """


See the step-by-step video tutorial to train and deploy YOLO11, learn how to build applications with Florence-2, and a guide to YOLO11 segmen


Step-by-step Video Tutorial on YOLO11
Learn how to use YOLO11 for object detection with this complete video tutorial. From finding datasets to labeling images, training the model,

SEE VIDEO

Build Applications with Florence-2
Florence-2, developed by Microsoft and released with an MIT license, is a state-of-the-art multimodal model architecture. With the base Flore

READ MORE

Guide to YOLO11 for Instance Segmentation
YOLO11 is a family of models for object detection, classification, instance segmentation, keypoint detection, and oriented bounding box detec

SEE GUIDE

Community Spotlights

Making Gesture Controlled Robots »

Learn how to build gesture controlled robotics using computer vision [youtube]


Vision-Language Models Explained »
```

```
Learn about the latest vision-language models and how multimodality is the next frontier in AI [youtube]


Roboflow Project of the Week

Build applications for the railway industry with this Rail Anchor API. View Roboflow Universe »


CV & ML You Can Use
Depth Pro: Sharp Monocular Metric Depth in Less Than a Second [github] »

Dolphins: Multimodal Language Model for Driving [github] »



Research of the Week
Depth Pro: Sharp Monocular Metric Depth in Less Than a Second [arxiv] »

Minimalist Vision with Freeform Pixels [columbia.edu] »


Want to join the Roboflow Team?
We're hiring!

VIEW OPEN JOBS

Facebook  Twitter LinkedIn  YouTube
© 2024 Roboflow, Inc. All rights reserved.
500 Locust St, Suite 104, Des Moines, IA 50309

Manage Subscription Preferences | Unsubscribe



"""


# Tokenize the input string
inputs = tokenizer.encode_plus(
    text_mail,
    add_special_tokens=True,
    max_length=128,
    padding='max_length',
    truncation=True,
    return_tensors='tf'
)

# Get the input tensors
input_ids = inputs['input_ids']
attention_mask = inputs['attention_mask']

# Get the model's predictions
outputs = model(input_ids, attention_mask=attention_mask)
logits = outputs.logits

# Convert logits to probabilities
probabilities = tf.nn.softmax(logits, axis=-1)

# Get the predicted class
predicted_class = tf.argmax(probabilities, axis=1).numpy()[0]

# Map the predicted class to the sentiment
sentiment = 'Event' if predicted_class == 1 else 'Non-Event'

print(f"Sentiment: {sentiment}")
```

```
Sentiment: Non-Event
```

```
test_list = list(test_df.Body)
type_list = list(test_df.Type)


pred_test_list =[]


for i in range(len(test_list)):
  inputs = tokenizer.encode_plus(
    test_list[i],
    add_special_tokens=True,
    max_length=128,
    padding='max_length',
    truncation=True,
    return_tensors='tf'
)

  # Get the input tensors
  input_ids = inputs['input_ids']
  attention_mask = inputs['attention_mask']

  # Get the model's predictions
  outputs = model(input_ids, attention_mask=attention_mask)
  logits = outputs.logits

  # Convert logits to probabilities
  probabilities = tf.nn.softmax(logits, axis=-1)

  # Get the predicted class
  predicted_class = tf.argmax(probabilities, axis=1).numpy()[0]

  # Map the predicted class to the sentiment
  sentiment = 'Event' if predicted_class == 1 else 'Non-Event'

  if sentiment == 'Event':
    pred_test_list.append(1)

  else:
    pred_test_list.append(0)
```

```
print(len(pred_test_list), len(type_list))
```

```
50 50
```

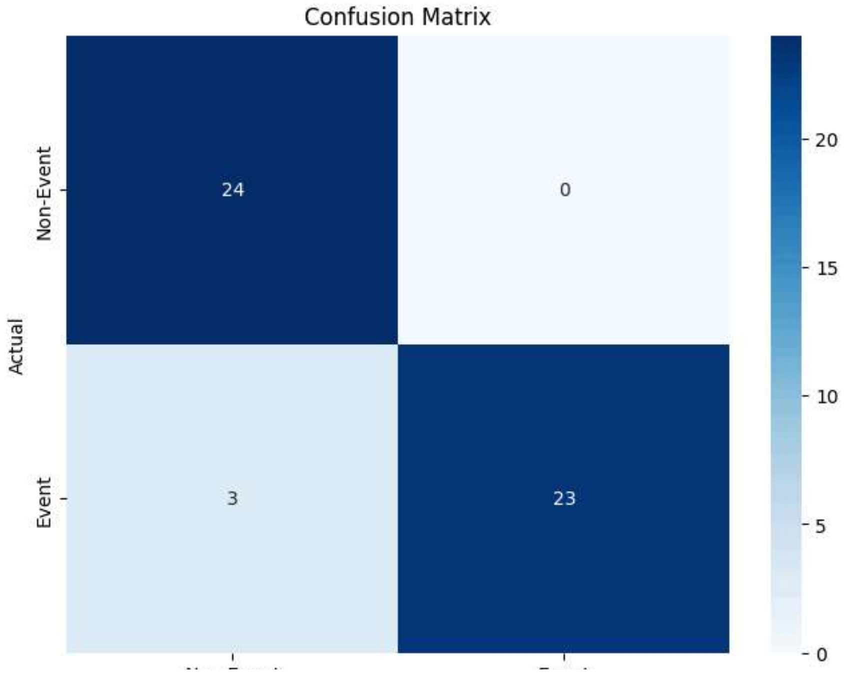
```
pred_train_list_float = [float(x) for x in pred_test_list]
```

```
# prompt: create confusion matix using pred_train_list_float and type list as a heatmap

import seaborn as sns
from sklearn.metrics import confusion_matrix

cm = confusion_matrix(type_list, pred_train_list_float)

plt.figure(figsize=(8, 6))
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues', xticklabels=['Non-Event', 'Event'], yticklabels=['Non-Event', 'Event'])
plt.xlabel('Predicted')
plt.ylabel('Actual')
plt.title('Confusion Matrix')
plt.show()
```


Confusion Matrix
Actual
Non-Event
Event
24
0
3
23
20
15
10
5
0