3_1_Lexical_Complexity_Binary_Classification_Prediction_Transformers_

April 7, 2025

[1]:
```
#@title Install Packages
```

[2]:
```
!pip install -q transformers
!pip install -q torchinfo
!pip install -q datasets
!pip install -q evaluate
!pip install -q nltk
!pip install -q contractions
!pip install -q hf_xet
!pip install -q sentencepiece
```

[3]:
```
!sudo apt-get update
! sudo apt-get install tree
```

```
Hit:1 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64
InRelease
Hit:2 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease
Get:3 http://security.ubuntu.com/ubuntu jammy-security InRelease [129 kB]
Hit:4 http://archive.ubuntu.com/ubuntu jammy InRelease
Get:5 http://archive.ubuntu.com/ubuntu jammy-updates InRelease [128 kB]
Hit:6 https://r2u.stat.illinois.edu/ubuntu jammy InRelease
Hit:7 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
Hit:8 https://ppa.launchpadcontent.net/graphics-drivers/ppa/ubuntu jammy
InRelease
Get:9 http://security.ubuntu.com/ubuntu jammy-security/main amd64 Packages
[2,788 kB]
Hit:10 https://ppa.launchpadcontent.net/ubuntugis/ppa/ubuntu jammy InRelease
Hit:11 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Get:12 http://security.ubuntu.com/ubuntu jammy-security/universe amd64 Packages
[1,243 kB]
Fetched 4,288 kB in 2s (1,883 kB/s)
Reading package lists… Done
W: Skipping acquire of configured file 'main/source/Sources' as repository
'https://r2u.stat.illinois.edu/ubuntu jammy InRelease' does not seem to provide
it (sources.list entry misspelt?)
Reading package lists… Done
Building dependency tree… Done
```

```
Reading state information… Done
tree is already the newest version (2.0.2-1).
0 upgraded, 0 newly installed, 0 to remove and 37 not upgraded.
```

[4]:

```
#@title Imports
import nltk
from nltk.tokenize import RegexpTokenizer

import contractions

import evaluate
import transformers
import torch

from torchinfo import summary

from datasets import load_dataset, Dataset, DatasetDict

from transformers import AutoTokenizer, AutoModel,␣
  ↪AutoModelForSequenceClassification, TrainingArguments, Trainer, BertConfig,␣
  ↪BertForSequenceClassification

import os
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns

import sklearn

import spacy

from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report,␣
  ↪precision_recall_fscore_support, accuracy_score

import sentencepiece

from datetime import datetime
```

[5]:

```
# @title Mount Google Drive
```

[6]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[7]:

```
dir_root = '/content/drive/MyDrive/266-final/'
# dir_data = '/content/drive/MyDrive/266-final/data/'
# dir_data = '/content/drive/MyDrive/266-final/data/se21-t1-comp-lex-master/'
dir_data = '/content/drive/MyDrive/266-final/data/266-comp-lex-master'
dir_models = '/content/drive/MyDrive/266-final/models/'
dir_results = '/content/drive/MyDrive/266-final/results/'
```

[8]:

```
wandbai_api_key = "5236444b7e96f5cf74038116d8c1efba161a4310"
```

[9]:

```
!tree /content/drive/MyDrive/266-final/data/266-comp-lex-master/
```

```
/content/drive/MyDrive/266-final/data/266-comp-lex-master/
  fe-test-labels
      test_multi_df.csv
      test_single_df.csv
  fe-train
      train_multi_df.csv
      train_single_df.csv
  fe-trial-val
      trial_val_multi_df.csv
      trial_val_single_df.csv
  test-labels
      lcp_multi_test.tsv
      lcp_single_test.tsv
  train
      lcp_multi_train.tsv
      lcp_single_train.tsv
  trial
       lcp_multi_trial.tsv
       lcp_single_trial.tsv

6 directories, 12 files
```

[10]:

```
!ls -R /content/drive/MyDrive/266-final/data/266-comp-lex-master/
```

```
/content/drive/MyDrive/266-final/data/266-comp-lex-master/:
fe-test-labels  fe-train  fe-trial-val  test-labels  train  trial

/content/drive/MyDrive/266-final/data/266-comp-lex-master/fe-test-labels:
test_multi_df.csv  test_single_df.csv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/fe-train:
train_multi_df.csv  train_single_df.csv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/fe-trial-val:
trial_val_multi_df.csv  trial_val_single_df.csv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/test-labels:
```

```
lcp_multi_test.tsv  lcp_single_test.tsv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/train:
lcp_multi_train.tsv  lcp_single_train.tsv

/content/drive/MyDrive/266-final/data/266-comp-lex-master/trial:
lcp_multi_trial.tsv  lcp_single_trial.tsv
```

[11]:

```
!tree /content/drive/MyDrive/266-final/data/266-comp-lex-master/
```

```
/content/drive/MyDrive/266-final/data/266-comp-lex-master/
  fe-test-labels
      test_multi_df.csv
      test_single_df.csv
  fe-train
      train_multi_df.csv
      train_single_df.csv
  fe-trial-val
      trial_val_multi_df.csv
      trial_val_single_df.csv
  test-labels
      lcp_multi_test.tsv
      lcp_single_test.tsv
  train
      lcp_multi_train.tsv
      lcp_single_train.tsv
  trial
       lcp_multi_trial.tsv
       lcp_single_trial.tsv

6 directories, 12 files
```

[12]:

```
#@title Import Data
```

[13]:

```
df_names = [
    "train_single_df",
    "train_multi_df",
    "trial_val_single_df",
    "trial_val_multi_df",
    "test_single_df",
    "test_multi_df"
]

loaded_dataframes = {}

for df_name in df_names:
    if "train" in df_name:
        subdir = "fe-train"
    elif "trial_val" in df_name:
        subdir = "fe-trial-val"
    elif "test" in df_name:
        subdir = "fe-test-labels"
    else:
        subdir = None

    if subdir:
        read_path = os.path.join(dir_data, subdir, f"{df_name}.csv")
        loaded_df = pd.read_csv(read_path)
        loaded_dataframes[df_name] = loaded_df
        print(f"Loaded {df_name} from {read_path}")

# for df_name, df in loaded_dataframes.items():
#     print(f"\n>>> {df_name} shape: {df.shape}")
#     if 'binary_complexity' in df.columns:
#         print(df['binary_complexity'].value_counts())
#         print(df.info())
#         print(df.head())

for df_name, df in loaded_dataframes.items():
    globals()[df_name] = df
    print(f"{df_name} loaded into global namespace.")
```

```
Loaded train_single_df from /content/drive/MyDrive/266-final/data/266-comp-lex-
master/fe-train/train_single_df.csv
Loaded train_multi_df from /content/drive/MyDrive/266-final/data/266-comp-lex-
master/fe-train/train_multi_df.csv
Loaded trial_val_single_df from /content/drive/MyDrive/266-final/data/266-comp-
lex-master/fe-trial-val/trial_val_single_df.csv
Loaded trial_val_multi_df from /content/drive/MyDrive/266-final/data/266-comp-
lex-master/fe-trial-val/trial_val_multi_df.csv
Loaded test_single_df from /content/drive/MyDrive/266-final/data/266-comp-lex-
master/fe-test-labels/test_single_df.csv
Loaded test_multi_df from /content/drive/MyDrive/266-final/data/266-comp-lex-
master/fe-test-labels/test_multi_df.csv
train_single_df loaded into global namespace.
train_multi_df loaded into global namespace.
trial_val_single_df loaded into global namespace.
trial_val_multi_df loaded into global namespace.
test_single_df loaded into global namespace.
test_multi_df loaded into global namespace.
```

- Functional tests pass, we can proceed with Baseline Modeling

0.1 Experiments with Transformers Models

[14]:

```
# def get_model_and_tokenizer(model_name: str):
#     """
#     Loads the specified pretrained model & tokenizer for classification.
#     """
#     tokenizer = AutoTokenizer.from_pretrained(model_name)
#     model = AutoModelForSequenceClassification.from_pretrained(model_name)
#     return model, tokenizer

# new prod version to support local model checkpoints, to be used after␣
 ↪experiment 1.0
def get_model_and_tokenizer(
    remote_model_name: str = None,
    local_model_path: str = None
):
    """
    Loads the model & tokenizer for classification.
    If 'local_model_path' is specified, load from that path.
    Otherwise, fall back to 'remote_model_name'.
    """
    from transformers import AutoTokenizer, AutoModelForSequenceClassification

    if local_model_path:
        # Local load
        print(f"Loading from local path: {local_model_path}")
        tokenizer = AutoTokenizer.from_pretrained(local_model_path)
        model = AutoModelForSequenceClassification.
 ↪from_pretrained(local_model_path)
    elif remote_model_name:
        # Load from HF Hub
        print(f"Loading from Hugging Face model: {remote_model_name}")
        tokenizer = AutoTokenizer.from_pretrained(remote_model_name)
        model = AutoModelForSequenceClassification.
 ↪from_pretrained(remote_model_name)
    else:
        raise ValueError("You must provide either a remote_model_name or a␣
 ↪local_model_path!")

    return model, tokenizer
```

[15]:

```
def freeze_unfreeze_layers(model, layers_to_unfreeze=None):
    """
    Toggles requires_grad = False for all parameters
    except for those whose names contain any string in layers_to_unfreeze.
    By default, always unfreeze classifier/heads.
    """
    if layers_to_unfreeze is None:
        layers_to_unfreeze = ["classifier.", "pooler."]

    for name, param in model.named_parameters():
        # If any layer substring matches, we unfreeze
        if any(substring in name for substring in layers_to_unfreeze):
            param.requires_grad = True
        else:
            param.requires_grad = False
```

[16]:

```
def encode_examples(examples, tokenizer, text_col, max_length=256):
    """
    Tokenizes a batch of texts from 'examples[text_col]' using the given␣
 ↪tokenizer.
    Returns a dict with 'input_ids', 'attention_mask', etc.
    """
    texts = examples[text_col]
    encoded = tokenizer(
        texts,
        truncation=True,
        padding='max_length',
        max_length=max_length
    )
    return encoded
```

[17]:

```
def prepare_dataset(df, tokenizer, text_col, label_col, max_length=256):
    """
    Converts a Pandas DataFrame to a Hugging Face Dataset,
    then applies 'encode_examples' to tokenize.
    """
    # Convert to HF Dataset
    dataset = Dataset.from_pandas(df)

    # Map the encode function
    dataset = dataset.map(
        lambda batch: encode_examples(batch, tokenizer, text_col, max_length),
        batched=True
    )

    # Rename the label column to 'labels' for HF Trainer
    dataset = dataset.rename_column(label_col, "labels")
    # HF often requires removing any columns that cannot be converted or are␣
 ↪not needed
    dataset.set_format(type='torch',
                       columns=['input_ids', 'attention_mask', 'labels'])
    return dataset
```

[18]:

```
def compute_metrics(eval_pred):
    """
    Computes classification metrics, including accuracy, precision, recall, and␣
 ↪F1.
    """
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=1)

    metric_accuracy  = evaluate.load("accuracy")
    metric_precision = evaluate.load("precision")
    metric_recall    = evaluate.load("recall")
    metric_f1        = evaluate.load("f1")

    accuracy_result  = metric_accuracy.compute(predictions=preds,␣
 ↪references=labels)
    precision_result = metric_precision.compute(predictions=preds,␣
 ↪references=labels, average="binary")
    recall_result    = metric_recall.compute(predictions=preds,␣
 ↪references=labels, average="binary")
    f1_result        = metric_f1.compute(predictions=preds, references=labels,␣
 ↪average="binary")

    return {
        "accuracy"       : accuracy_result["accuracy"],
        "precision": precision_result["precision"],
        "recall"   : recall_result["recall"],
        "f1"       : f1_result["f1"]
    }
```

0.1.1 Experiment Design

[20]:

```
# Define Experiment Parameters

named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = "" # modern bert

# learning_rate = 1e-3
# learning_rate = 1e-4
# learning_rate = 1e-5
# learning_rate = 5e-6
learning_rate = 5e-7
# learning_rate = 5e-8

# num_epochs = 3
```

```
num_epochs = 5
# num_epochs = 10
# num_epochs = 15
# num_epochs = 20

length_max = 128
# length_max = 256
# length_max = 348
# length_max = 512

# size_batch = 1
# size_batch = 4
size_batch = 8
# size_batch = 16
# size_batch = 24
# size_batch = 32

regularization_weight_decay = 0
# regularization_weight_decay = 0.1
# regularization_weight_decay = 0.5

# dropout???

# layers to freeze and unfreeze?

y_col = "binary_complexity"
# y_col = "complexity"

x_task = "single"
# x_task = "multi"

# x_col = "sentence"
x_col = "sentence_no_contractions"
# x_col = "pos_sequence"
# x_col = "dep_sequence"
# x_col = "morph_sequence"

if x_task == "single":
    df_train = train_single_df
    df_val   = trial_val_single_df
    df_test  = test_single_df
else:
    df_train = train_multi_df
    df_val   = trial_val_multi_df
    df_test  = test_multi_df
```

[21]:

```
def train_transformer_model(
    model,
    tokenizer,
    train_dataset,
    val_dataset,
    output_dir=dir_results,
    num_epochs=num_epochs,
    batch_size=size_batch,
    lr=learning_rate,
    weight_decay=regularization_weight_decay
):
    """
    Sets up a Trainer and trains the model for 'num_epochs' using the given␣
 ↪dataset.
    Returns the trained model and the Trainer object for possible re-use or␣
 ↪analysis.
    """

    training_args = TrainingArguments(
        output_dir=output_dir,
        num_train_epochs=num_epochs,
        per_device_train_batch_size=batch_size,
        per_device_eval_batch_size=batch_size,
        evaluation_strategy="epoch",
        save_strategy="no",
        logging_strategy="epoch",
        learning_rate=lr,
        weight_decay=weight_decay,
        report_to=["none"],  # or "wandb"
    )

    trainer = Trainer(
        model=model,
        args=training_args,
        train_dataset=train_dataset,
        eval_dataset=val_dataset,
        tokenizer=tokenizer,  # optional
        compute_metrics=compute_metrics
    )

    trainer.train()
    return model, trainer
```

0.1.2 Experiment 1: from pretrained bert-base-cased with single task

Model Inspection

[22]:

```python
print("model checkpoints:", dir_models)
!ls /content/drive/MyDrive/266-final/models/
```

```
model checkpoints: /content/drive/MyDrive/266-final/models/
bert-base-cased_20250407_232900  model_20250407_232826  nltk_data
```

[23]:

```python
# Load Model & Tokenizer
model, tokenizer = get_model_and_tokenizer(named_model) # deprecated argument structure
# model, tokenizer = get_model_and_tokenizer("/content/drive/MyDrive/266-final/models/bert-base-cased_20250407_232900") # proposed argument usage for checkpointed models

for name, param in model.named_parameters():
    print(name)

print("=============")
print(named_model, ":")
print("=============")
print(model)
print("=============")
print(model.config)
print("=============")
print("num_parameters:", model.num_parameters())
print("=============")
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
```

```
Loading from Hugging Face model: bert-base-cased
```

```
Some weights of BertForSequenceClassification were not initialized from the
model checkpoint at bert-base-cased and are newly initialized:
['classifier.bias', 'classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

```
bert.embeddings.word_embeddings.weight
bert.embeddings.position_embeddings.weight
bert.embeddings.token_type_embeddings.weight
bert.embeddings.LayerNorm.weight
bert.embeddings.LayerNorm.bias
bert.encoder.layer.0.attention.self.query.weight
bert.encoder.layer.0.attention.self.query.bias
bert.encoder.layer.0.attention.self.key.weight
bert.encoder.layer.0.attention.self.key.bias
bert.encoder.layer.0.attention.self.value.weight
bert.encoder.layer.0.attention.self.value.bias
bert.encoder.layer.0.attention.output.dense.weight
bert.encoder.layer.0.attention.output.dense.bias
bert.encoder.layer.0.attention.output.LayerNorm.weight
bert.encoder.layer.0.attention.output.LayerNorm.bias
bert.encoder.layer.0.intermediate.dense.weight
bert.encoder.layer.0.intermediate.dense.bias
bert.encoder.layer.0.output.dense.weight
bert.encoder.layer.0.output.dense.bias
bert.encoder.layer.0.output.LayerNorm.weight
bert.encoder.layer.0.output.LayerNorm.bias
bert.encoder.layer.1.attention.self.query.weight
bert.encoder.layer.1.attention.self.query.bias
bert.encoder.layer.1.attention.self.key.weight
bert.encoder.layer.1.attention.self.key.bias
bert.encoder.layer.1.attention.self.value.weight
bert.encoder.layer.1.attention.self.value.bias
bert.encoder.layer.1.attention.output.dense.weight
bert.encoder.layer.1.attention.output.dense.bias
bert.encoder.layer.1.attention.output.LayerNorm.weight
bert.encoder.layer.1.attention.output.LayerNorm.bias
bert.encoder.layer.1.intermediate.dense.weight
bert.encoder.layer.1.intermediate.dense.bias
bert.encoder.layer.1.output.dense.weight
bert.encoder.layer.1.output.dense.bias
bert.encoder.layer.1.output.LayerNorm.weight
bert.encoder.layer.1.output.LayerNorm.bias
bert.encoder.layer.2.attention.self.query.weight
bert.encoder.layer.2.attention.self.query.bias
bert.encoder.layer.2.attention.self.key.weight
bert.encoder.layer.2.attention.self.key.bias
bert.encoder.layer.2.attention.self.value.weight
bert.encoder.layer.2.attention.self.value.bias
bert.encoder.layer.2.attention.output.dense.weight
bert.encoder.layer.2.attention.output.dense.bias
bert.encoder.layer.2.attention.output.LayerNorm.weight
bert.encoder.layer.2.attention.output.LayerNorm.bias
bert.encoder.layer.2.intermediate.dense.weight
bert.encoder.layer.2.intermediate.dense.bias
bert.encoder.layer.2.output.dense.weight
bert.encoder.layer.2.output.dense.bias
bert.encoder.layer.2.output.LayerNorm.weight
bert.encoder.layer.2.output.LayerNorm.bias
bert.encoder.layer.3.attention.self.query.weight
bert.encoder.layer.3.attention.self.query.bias
bert.encoder.layer.3.attention.self.key.weight
bert.encoder.layer.3.attention.self.key.bias
bert.encoder.layer.3.attention.self.value.weight
bert.encoder.layer.3.attention.self.value.bias
bert.encoder.layer.3.attention.output.dense.weight
bert.encoder.layer.3.attention.output.dense.bias
bert.encoder.layer.3.attention.output.LayerNorm.weight
bert.encoder.layer.3.attention.output.LayerNorm.bias
bert.encoder.layer.3.intermediate.dense.weight
bert.encoder.layer.3.intermediate.dense.bias
bert.encoder.layer.3.output.dense.weight
bert.encoder.layer.3.output.dense.bias
bert.encoder.layer.3.output.LayerNorm.weight
bert.encoder.layer.3.output.LayerNorm.bias
bert.encoder.layer.4.attention.self.query.weight
bert.encoder.layer.4.attention.self.query.bias
bert.encoder.layer.4.attention.self.key.weight
bert.encoder.layer.4.attention.self.key.bias
bert.encoder.layer.4.attention.self.value.weight
bert.encoder.layer.4.attention.self.value.bias
bert.encoder.layer.4.attention.output.dense.weight
bert.encoder.layer.4.attention.output.dense.bias
bert.encoder.layer.4.attention.output.LayerNorm.weight
bert.encoder.layer.4.attention.output.LayerNorm.bias
bert.encoder.layer.4.intermediate.dense.weight
bert.encoder.layer.4.intermediate.dense.bias
bert.encoder.layer.4.output.dense.weight
bert.encoder.layer.4.output.dense.bias
bert.encoder.layer.4.output.LayerNorm.weight
bert.encoder.layer.4.output.LayerNorm.bias
bert.encoder.layer.5.attention.self.query.weight
bert.encoder.layer.5.attention.self.query.bias
bert.encoder.layer.5.attention.self.key.weight
bert.encoder.layer.5.attention.self.key.bias
bert.encoder.layer.5.attention.self.value.weight
bert.encoder.layer.5.attention.self.value.bias
bert.encoder.layer.5.attention.output.dense.weight
bert.encoder.layer.5.attention.output.dense.bias
bert.encoder.layer.5.attention.output.LayerNorm.weight
bert.encoder.layer.5.attention.output.LayerNorm.bias
bert.encoder.layer.5.intermediate.dense.weight
bert.encoder.layer.5.intermediate.dense.bias
bert.encoder.layer.5.output.dense.weight
bert.encoder.layer.5.output.dense.bias
bert.encoder.layer.5.output.LayerNorm.weight
bert.encoder.layer.5.output.LayerNorm.bias
bert.encoder.layer.6.attention.self.query.weight
bert.encoder.layer.6.attention.self.query.bias
bert.encoder.layer.6.attention.self.key.weight
bert.encoder.layer.6.attention.self.key.bias
bert.encoder.layer.6.attention.self.value.weight
bert.encoder.layer.6.attention.self.value.bias
bert.encoder.layer.6.attention.output.dense.weight
bert.encoder.layer.6.attention.output.dense.bias
bert.encoder.layer.6.attention.output.LayerNorm.weight
bert.encoder.layer.6.attention.output.LayerNorm.bias
bert.encoder.layer.6.intermediate.dense.weight
bert.encoder.layer.6.intermediate.dense.bias
bert.encoder.layer.6.output.dense.weight
bert.encoder.layer.6.output.dense.bias
bert.encoder.layer.6.output.LayerNorm.weight
bert.encoder.layer.6.output.LayerNorm.bias
bert.encoder.layer.7.attention.self.query.weight
bert.encoder.layer.7.attention.self.query.bias
bert.encoder.layer.7.attention.self.key.weight
bert.encoder.layer.7.attention.self.key.bias
bert.encoder.layer.7.attention.self.value.weight
bert.encoder.layer.7.attention.self.value.bias
bert.encoder.layer.7.attention.output.dense.weight
bert.encoder.layer.7.attention.output.dense.bias
bert.encoder.layer.7.attention.output.LayerNorm.weight
bert.encoder.layer.7.attention.output.LayerNorm.bias
bert.encoder.layer.7.intermediate.dense.weight
bert.encoder.layer.7.intermediate.dense.bias
bert.encoder.layer.7.output.dense.weight
bert.encoder.layer.7.output.dense.bias
bert.encoder.layer.7.output.LayerNorm.weight
bert.encoder.layer.7.output.LayerNorm.bias
bert.encoder.layer.8.attention.self.query.weight
bert.encoder.layer.8.attention.self.query.bias
bert.encoder.layer.8.attention.self.key.weight
bert.encoder.layer.8.attention.self.key.bias
bert.encoder.layer.8.attention.self.value.weight
bert.encoder.layer.8.attention.self.value.bias
bert.encoder.layer.8.attention.output.dense.weight
bert.encoder.layer.8.attention.output.dense.bias
bert.encoder.layer.8.attention.output.LayerNorm.weight
bert.encoder.layer.8.attention.output.LayerNorm.bias
bert.encoder.layer.8.intermediate.dense.weight
bert.encoder.layer.8.intermediate.dense.bias
bert.encoder.layer.8.output.dense.weight
bert.encoder.layer.8.output.dense.bias
bert.encoder.layer.8.output.LayerNorm.weight
bert.encoder.layer.8.output.LayerNorm.bias
bert.encoder.layer.9.attention.self.query.weight
bert.encoder.layer.9.attention.self.query.bias
bert.encoder.layer.9.attention.self.key.weight
bert.encoder.layer.9.attention.self.key.bias
bert.encoder.layer.9.attention.self.value.weight
bert.encoder.layer.9.attention.self.value.bias
bert.encoder.layer.9.attention.output.dense.weight
bert.encoder.layer.9.attention.output.dense.bias
bert.encoder.layer.9.attention.output.LayerNorm.weight
bert.encoder.layer.9.attention.output.LayerNorm.bias
bert.encoder.layer.9.intermediate.dense.weight
bert.encoder.layer.9.intermediate.dense.bias
bert.encoder.layer.9.output.dense.weight
bert.encoder.layer.9.output.dense.bias
bert.encoder.layer.9.output.LayerNorm.weight
bert.encoder.layer.9.output.LayerNorm.bias
bert.encoder.layer.10.attention.self.query.weight
bert.encoder.layer.10.attention.self.query.bias
bert.encoder.layer.10.attention.self.key.weight
bert.encoder.layer.10.attention.self.key.bias
bert.encoder.layer.10.attention.self.value.weight
bert.encoder.layer.10.attention.self.value.bias
bert.encoder.layer.10.attention.output.dense.weight
bert.encoder.layer.10.attention.output.dense.bias
bert.encoder.layer.10.attention.output.LayerNorm.weight
bert.encoder.layer.10.attention.output.LayerNorm.bias
bert.encoder.layer.10.intermediate.dense.weight
bert.encoder.layer.10.intermediate.dense.bias
bert.encoder.layer.10.output.dense.weight
bert.encoder.layer.10.output.dense.bias
bert.encoder.layer.10.output.LayerNorm.weight
bert.encoder.layer.10.output.LayerNorm.bias
bert.encoder.layer.11.attention.self.query.weight
bert.encoder.layer.11.attention.self.query.bias
bert.encoder.layer.11.attention.self.key.weight
bert.encoder.layer.11.attention.self.key.bias
bert.encoder.layer.11.attention.self.value.weight
bert.encoder.layer.11.attention.self.value.bias
bert.encoder.layer.11.attention.output.dense.weight
bert.encoder.layer.11.attention.output.dense.bias
bert.encoder.layer.11.attention.output.LayerNorm.weight
bert.encoder.layer.11.attention.output.LayerNorm.bias
bert.encoder.layer.11.intermediate.dense.weight
bert.encoder.layer.11.intermediate.dense.bias
bert.encoder.layer.11.output.dense.weight
bert.encoder.layer.11.output.dense.bias
bert.encoder.layer.11.output.LayerNorm.weight
bert.encoder.layer.11.output.LayerNorm.bias
bert.pooler.dense.weight
bert.pooler.dense.bias
classifier.weight
classifier.bias
=============
bert-base-cased :
=============
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(28996, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSdpaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): Tanh()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
=============
BertConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "torch_dtype": "float32",
  "transformers_version": "4.50.3",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 28996
}

=============
num_parameters: 108311810
=============
num_trainable_parameters: 108311810
```

Layer Configuration

[24]:

```python
# Freeze/Unfreeze Layers & Additional Configuration Parameters

import torch.nn as nn

layers_to_unfreeze = [
    "bert.encoder.layer.9.",
    "bert.encoder.layer.10.",
    "bert.encoder.layer.11.",
    "pooler.",
    "classifier.",
]

freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)


bert_config = BertConfig(
    # vocab_size=28996,
    hidden_size=768,
    # num_hidden_layers=12,
    # num_attention_heads=12,
    intermediate_size=3072,
    # max_position_embeddings=512,
    type_vocab_size=2,

    hidden_dropout_prob=0.3,
    attention_probs_dropout_prob=0.2,
    # classifier_dropout=None,
    # initializer_range=0.02,
    # layer_norm_eps=1e-12,

    hidden_act="gelu",
    gradient_checkpointing=False,
    position_embedding_type="absolute",
    use_cache=True,
    pad_token_id=0
)

model.bert.pooler.activation = nn.ReLU() # Tanh() replaced as the pooler layer␣
 ↪activation function

for name, param in model.named_parameters():
    print(name, "requires_grad=", param.requires_grad)

print("\nLayers that are 'True' are trainable. 'False' are frozen.")

print("=============")
print(named_model, ":")
print("=============")
print(model)
print("=============")
print(model.config)
print("=============")
print("num_parameters:", model.num_parameters())
print("=============")
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
```

```
bert.embeddings.word_embeddings.weight requires_grad= False
bert.embeddings.position_embeddings.weight requires_grad= False
bert.embeddings.token_type_embeddings.weight requires_grad= False
bert.embeddings.LayerNorm.weight requires_grad= False
bert.embeddings.LayerNorm.bias requires_grad= False
bert.encoder.layer.0.attention.self.query.weight requires_grad= False
bert.encoder.layer.0.attention.self.query.bias requires_grad= False
bert.encoder.layer.0.attention.self.key.weight requires_grad= False
bert.encoder.layer.0.attention.self.key.bias requires_grad= False
bert.encoder.layer.0.attention.self.value.weight requires_grad= False
bert.encoder.layer.0.attention.self.value.bias requires_grad= False
bert.encoder.layer.0.attention.output.dense.weight requires_grad= False
bert.encoder.layer.0.attention.output.dense.bias requires_grad= False
bert.encoder.layer.0.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.0.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.0.intermediate.dense.weight requires_grad= False
bert.encoder.layer.0.intermediate.dense.bias requires_grad= False
bert.encoder.layer.0.output.dense.weight requires_grad= False
bert.encoder.layer.0.output.dense.bias requires_grad= False
bert.encoder.layer.0.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.0.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.1.attention.self.query.weight requires_grad= False
bert.encoder.layer.1.attention.self.query.bias requires_grad= False
bert.encoder.layer.1.attention.self.key.weight requires_grad= False
bert.encoder.layer.1.attention.self.key.bias requires_grad= False
bert.encoder.layer.1.attention.self.value.weight requires_grad= False
bert.encoder.layer.1.attention.self.value.bias requires_grad= False
bert.encoder.layer.1.attention.output.dense.weight requires_grad= False
bert.encoder.layer.1.attention.output.dense.bias requires_grad= False
bert.encoder.layer.1.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.1.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.1.intermediate.dense.weight requires_grad= False
bert.encoder.layer.1.intermediate.dense.bias requires_grad= False
bert.encoder.layer.1.output.dense.weight requires_grad= False
bert.encoder.layer.1.output.dense.bias requires_grad= False
bert.encoder.layer.1.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.1.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.2.attention.self.query.weight requires_grad= False
bert.encoder.layer.2.attention.self.query.bias requires_grad= False
bert.encoder.layer.2.attention.self.key.weight requires_grad= False
bert.encoder.layer.2.attention.self.key.bias requires_grad= False
bert.encoder.layer.2.attention.self.value.weight requires_grad= False
bert.encoder.layer.2.attention.self.value.bias requires_grad= False
bert.encoder.layer.2.attention.output.dense.weight requires_grad= False
bert.encoder.layer.2.attention.output.dense.bias requires_grad= False
bert.encoder.layer.2.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.2.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.2.intermediate.dense.weight requires_grad= False
bert.encoder.layer.2.intermediate.dense.bias requires_grad= False
bert.encoder.layer.2.output.dense.weight requires_grad= False
bert.encoder.layer.2.output.dense.bias requires_grad= False
bert.encoder.layer.2.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.2.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.3.attention.self.query.weight requires_grad= False
bert.encoder.layer.3.attention.self.query.bias requires_grad= False
bert.encoder.layer.3.attention.self.key.weight requires_grad= False
bert.encoder.layer.3.attention.self.key.bias requires_grad= False
bert.encoder.layer.3.attention.self.value.weight requires_grad= False
bert.encoder.layer.3.attention.self.value.bias requires_grad= False
bert.encoder.layer.3.attention.output.dense.weight requires_grad= False
bert.encoder.layer.3.attention.output.dense.bias requires_grad= False
bert.encoder.layer.3.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.3.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.3.intermediate.dense.weight requires_grad= False
bert.encoder.layer.3.intermediate.dense.bias requires_grad= False
bert.encoder.layer.3.output.dense.weight requires_grad= False
bert.encoder.layer.3.output.dense.bias requires_grad= False
bert.encoder.layer.3.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.3.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.4.attention.self.query.weight requires_grad= False
bert.encoder.layer.4.attention.self.query.bias requires_grad= False
bert.encoder.layer.4.attention.self.key.weight requires_grad= False
bert.encoder.layer.4.attention.self.key.bias requires_grad= False
bert.encoder.layer.4.attention.self.value.weight requires_grad= False
bert.encoder.layer.4.attention.self.value.bias requires_grad= False
bert.encoder.layer.4.attention.output.dense.weight requires_grad= False
bert.encoder.layer.4.attention.output.dense.bias requires_grad= False
bert.encoder.layer.4.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.4.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.4.intermediate.dense.weight requires_grad= False
bert.encoder.layer.4.intermediate.dense.bias requires_grad= False
bert.encoder.layer.4.output.dense.weight requires_grad= False
bert.encoder.layer.4.output.dense.bias requires_grad= False
bert.encoder.layer.4.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.4.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.5.attention.self.query.weight requires_grad= False
bert.encoder.layer.5.attention.self.query.bias requires_grad= False
bert.encoder.layer.5.attention.self.key.weight requires_grad= False
bert.encoder.layer.5.attention.self.key.bias requires_grad= False
bert.encoder.layer.5.attention.self.value.weight requires_grad= False
bert.encoder.layer.5.attention.self.value.bias requires_grad= False
bert.encoder.layer.5.attention.output.dense.weight requires_grad= False
bert.encoder.layer.5.attention.output.dense.bias requires_grad= False
bert.encoder.layer.5.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.5.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.5.intermediate.dense.weight requires_grad= False
bert.encoder.layer.5.intermediate.dense.bias requires_grad= False
bert.encoder.layer.5.output.dense.weight requires_grad= False
bert.encoder.layer.5.output.dense.bias requires_grad= False
bert.encoder.layer.5.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.5.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.6.attention.self.query.weight requires_grad= False
bert.encoder.layer.6.attention.self.query.bias requires_grad= False
bert.encoder.layer.6.attention.self.key.weight requires_grad= False
bert.encoder.layer.6.attention.self.key.bias requires_grad= False
bert.encoder.layer.6.attention.self.value.weight requires_grad= False
bert.encoder.layer.6.attention.self.value.bias requires_grad= False
bert.encoder.layer.6.attention.output.dense.weight requires_grad= False
bert.encoder.layer.6.attention.output.dense.bias requires_grad= False
bert.encoder.layer.6.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.6.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.6.intermediate.dense.weight requires_grad= False
bert.encoder.layer.6.intermediate.dense.bias requires_grad= False
bert.encoder.layer.6.output.dense.weight requires_grad= False
bert.encoder.layer.6.output.dense.bias requires_grad= False
bert.encoder.layer.6.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.6.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.7.attention.self.query.weight requires_grad= False
bert.encoder.layer.7.attention.self.query.bias requires_grad= False
bert.encoder.layer.7.attention.self.key.weight requires_grad= False
bert.encoder.layer.7.attention.self.key.bias requires_grad= False
bert.encoder.layer.7.attention.self.value.weight requires_grad= False
bert.encoder.layer.7.attention.self.value.bias requires_grad= False
bert.encoder.layer.7.attention.output.dense.weight requires_grad= False
bert.encoder.layer.7.attention.output.dense.bias requires_grad= False
bert.encoder.layer.7.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.7.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.7.intermediate.dense.weight requires_grad= False
bert.encoder.layer.7.intermediate.dense.bias requires_grad= False
bert.encoder.layer.7.output.dense.weight requires_grad= False
bert.encoder.layer.7.output.dense.bias requires_grad= False
bert.encoder.layer.7.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.7.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.8.attention.self.query.weight requires_grad= False
bert.encoder.layer.8.attention.self.query.bias requires_grad= False
bert.encoder.layer.8.attention.self.key.weight requires_grad= False
bert.encoder.layer.8.attention.self.key.bias requires_grad= False
bert.encoder.layer.8.attention.self.value.weight requires_grad= False
bert.encoder.layer.8.attention.self.value.bias requires_grad= False
bert.encoder.layer.8.attention.output.dense.weight requires_grad= False
bert.encoder.layer.8.attention.output.dense.bias requires_grad= False
bert.encoder.layer.8.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.8.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.8.intermediate.dense.weight requires_grad= False
bert.encoder.layer.8.intermediate.dense.bias requires_grad= False
bert.encoder.layer.8.output.dense.weight requires_grad= False
bert.encoder.layer.8.output.dense.bias requires_grad= False
bert.encoder.layer.8.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.8.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.9.attention.self.query.weight requires_grad= True
bert.encoder.layer.9.attention.self.query.bias requires_grad= True
bert.encoder.layer.9.attention.self.key.weight requires_grad= True
bert.encoder.layer.9.attention.self.key.bias requires_grad= True
bert.encoder.layer.9.attention.self.value.weight requires_grad= True
bert.encoder.layer.9.attention.self.value.bias requires_grad= True
bert.encoder.layer.9.attention.output.dense.weight requires_grad= True
bert.encoder.layer.9.attention.output.dense.bias requires_grad= True
bert.encoder.layer.9.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.9.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.9.intermediate.dense.weight requires_grad= True
bert.encoder.layer.9.intermediate.dense.bias requires_grad= True
bert.encoder.layer.9.output.dense.weight requires_grad= True
bert.encoder.layer.9.output.dense.bias requires_grad= True
bert.encoder.layer.9.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.9.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.10.attention.self.query.weight requires_grad= True
bert.encoder.layer.10.attention.self.query.bias requires_grad= True
bert.encoder.layer.10.attention.self.key.weight requires_grad= True
bert.encoder.layer.10.attention.self.key.bias requires_grad= True
bert.encoder.layer.10.attention.self.value.weight requires_grad= True
bert.encoder.layer.10.attention.self.value.bias requires_grad= True
bert.encoder.layer.10.attention.output.dense.weight requires_grad= True
bert.encoder.layer.10.attention.output.dense.bias requires_grad= True
bert.encoder.layer.10.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.10.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.10.intermediate.dense.weight requires_grad= True
bert.encoder.layer.10.intermediate.dense.bias requires_grad= True
bert.encoder.layer.10.output.dense.weight requires_grad= True
bert.encoder.layer.10.output.dense.bias requires_grad= True
bert.encoder.layer.10.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.10.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.11.attention.self.query.weight requires_grad= True
bert.encoder.layer.11.attention.self.query.bias requires_grad= True
bert.encoder.layer.11.attention.self.key.weight requires_grad= True
bert.encoder.layer.11.attention.self.key.bias requires_grad= True
bert.encoder.layer.11.attention.self.value.weight requires_grad= True
bert.encoder.layer.11.attention.self.value.bias requires_grad= True
bert.encoder.layer.11.attention.output.dense.weight requires_grad= True
bert.encoder.layer.11.attention.output.dense.bias requires_grad= True
bert.encoder.layer.11.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.11.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.11.intermediate.dense.weight requires_grad= True
bert.encoder.layer.11.intermediate.dense.bias requires_grad= True
bert.encoder.layer.11.output.dense.weight requires_grad= True
bert.encoder.layer.11.output.dense.bias requires_grad= True
bert.encoder.layer.11.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.11.output.LayerNorm.bias requires_grad= True
bert.pooler.dense.weight requires_grad= True
bert.pooler.dense.bias requires_grad= True
classifier.weight requires_grad= True
classifier.bias requires_grad= True
```

```
Layers that are 'True' are trainable. 'False' are frozen.
=============
bert-base-cased :
=============
BertForSequenceClassification(
  (bert): BertModel(
    (embeddings): BertEmbeddings(
      (word_embeddings): Embedding(28996, 768, padding_idx=0)
      (position_embeddings): Embedding(512, 768)
      (token_type_embeddings): Embedding(2, 768)
      (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
      (dropout): Dropout(p=0.1, inplace=False)
    )
    (encoder): BertEncoder(
      (layer): ModuleList(
        (0-11): 12 x BertLayer(
          (attention): BertAttention(
            (self): BertSdpaSelfAttention(
              (query): Linear(in_features=768, out_features=768, bias=True)
              (key): Linear(in_features=768, out_features=768, bias=True)
              (value): Linear(in_features=768, out_features=768, bias=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
            (output): BertSelfOutput(
              (dense): Linear(in_features=768, out_features=768, bias=True)
              (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
              (dropout): Dropout(p=0.1, inplace=False)
            )
          )
          (intermediate): BertIntermediate(
            (dense): Linear(in_features=768, out_features=3072, bias=True)
            (intermediate_act_fn): GELUActivation()
          )
          (output): BertOutput(
            (dense): Linear(in_features=3072, out_features=768, bias=True)
            (LayerNorm): LayerNorm((768,), eps=1e-12, elementwise_affine=True)
            (dropout): Dropout(p=0.1, inplace=False)
          )
        )
      )
    )
    (pooler): BertPooler(
      (dense): Linear(in_features=768, out_features=768, bias=True)
      (activation): ReLU()
    )
  )
  (dropout): Dropout(p=0.1, inplace=False)
  (classifier): Linear(in_features=768, out_features=2, bias=True)
)
=============
BertConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "position_embedding_type": "absolute",
  "torch_dtype": "float32",
  "transformers_version": "4.50.3",
  "type_vocab_size": 2,
  "use_cache": true,
  "vocab_size": 28996
}

=============
num_parameters: 108311810
=============
num_trainable_parameters: 21855746
```

Dataset Preparation

[25]:

```python
# Tokenize & Prepare Datasets

train_data_hf = prepare_dataset(
    df_train,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max
)

val_data_hf = prepare_dataset(
    df_val,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max
)

test_data_hf = prepare_dataset(
    df_test,
    tokenizer,
    text_col=x_col,
    label_col=y_col,
    max_length=length_max

)

print("Datasets prepared. Sample from train_data_hf:\n", train_data_hf[10])
print("Datasets prepared. Sample from train_data_hf:\n", val_data_hf[10])
print("Datasets prepared. Sample from train_data_hf:\n", test_data_hf[10])
```

```
Map:   0%|          | 0/7662 [00:00<?, ? examples/s]

Map:   0%|          | 0/421 [00:00<?, ? examples/s]

Map:   0%|          | 0/917 [00:00<?, ? examples/s]

Datasets prepared. Sample from train_data_hf:
 {'labels': tensor(0), 'input_ids': tensor([  101,  1252,  1106,  1103,  3824,
1104, 19892, 11220,  1324,  1119,
         1522,  3839,   117,  1272,  1103,  1555,  1104,  1103, 11563,  5609,
         1106,  1172,   132,  1152,  2446,  1122,  1113,  1147,  3221,   119,
          102,     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,     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,     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': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 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, 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, 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])}
Datasets prepared. Sample from train_data_hf:
 {'labels': tensor(0), 'input_ids': tensor([ 101, 6589, 1103, 2226, 1108, 1304,
4259,  117, 1105, 1117, 4470, 4562,
        1107, 1140,  119,  102,    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,    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,    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,    0,    0,    0,    0]), 'attention_mask':
tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 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, 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, 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, 0, 0, 0, 0])}
Datasets prepared. Sample from train_data_hf:
 {'labels': tensor(1), 'input_ids': tensor([  101,  1220,  1508,  1117,  8526,
1107,  1103,  1402,  1104,  1147,
         6807,   117,  1105, 27052,  1117,  1246,  1107,  1103,  1402,  1104,
        10136,  7528,   119,   102,     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,
            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,     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': tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
1, 1, 1, 1, 1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 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, 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, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0])}
```

[26]:

```
# Train & Evaluate

trained_model, trainer_obj = train_transformer_model(
    model=model,
    tokenizer=tokenizer,
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    output_dir=dir_results,
    num_epochs=num_epochs,
    batch_size=size_batch,
    lr=learning_rate,
    weight_decay=regularization_weight_decay
)

metrics = trainer_obj.evaluate()
print("Validation metrics:", metrics)

test_metrics = trainer_obj.evaluate(test_data_hf) if test_data_hf else None
print("Test metrics:", test_metrics)
```

```
/usr/local/lib/python3.11/dist-packages/transformers/training_args.py:1611:
FutureWarning: `evaluation_strategy` is deprecated and will be removed in
version 4.46 of  Transformers. Use `eval_strategy` instead
  warnings.warn(
<ipython-input-21-295bdbf803a2>:30: FutureWarning: `tokenizer` is deprecated and
will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class`
instead.
  trainer = Trainer(
```

```
<IPython.core.display.HTML object>
```

```
<IPython.core.display.HTML object>
```

```
Validation metrics: {'eval_loss': 0.6899044513702393, 'eval_accuracy':
0.5534441805225653, 'eval_precision': 0.5106382978723404, 'eval_recall': 0.5,
'eval_f1': 0.5052631578947369, 'eval_runtime': 5.6434,
'eval_samples_per_second': 74.601, 'eval_steps_per_second': 9.392, 'epoch': 5.0}
Test metrics: {'eval_loss': 0.6933835744857788, 'eval_accuracy':
0.5310796074154853, 'eval_precision': 0.5136476426799007, 'eval_recall':
0.46938775510204084, 'eval_f1': 0.490521327014218, 'eval_runtime': 7.662,
'eval_samples_per_second': 119.681, 'eval_steps_per_second': 15.009, 'epoch':
5.0}
```

[27]:
```
print("Experiment configuration used with this experiment:")
print("model used:", named_model)
print("learning rate used:", learning_rate)
print("number of epochs:", num_epochs)
print("maximum sequence length:", length_max)
print("batch size used:", size_batch)
print("regularization value:", regularization_weight_decay)
print("outcome variable:", y_col)
print("task:", x_task)
print("input column:", x_col)
```

```
Experiment configuration used with this experiment:
model used: bert-base-cased
learning rate used: 5e-07
```

```
number of epochs: 5
maximum sequence length: 128
batch size used: 8
regularization value: 0
outcome variable: binary_complexity
task: single
input column: sentence_no_contractions
```

[28]:

```
# save model checkpoint

timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
model_save_path = os.path.join(dir_models, f"{named_model}_{timestamp}")

trainer_obj.save_model(model_save_path)
print(f"Model checkpoint saved to: {model_save_path}")
```

```
Model checkpoint saved to: /content/drive/MyDrive/266-final/models/bert-base-
cased_20250407_235945
```

0.1.3 Experiment 1.1: bert-base-cased single with additional epochs

[ ]:

```
# Define Experiment Parameters

named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = "" # modern bert

# learning_rate = 1e-3
# learning_rate = 1e-4
# learning_rate = 1e-5
# learning_rate = 5e-6
learning_rate = 5e-7
# learning_rate = 5e-8

# num_epochs = 3
num_epochs = 5
# num_epochs = 10
# num_epochs = 15
# num_epochs = 20

length_max = 128
# length_max = 256
# length_max = 348
# length_max = 512

# size_batch = 1
# size_batch = 4
size_batch = 8
# size_batch = 16
# size_batch = 24
# size_batch = 32

regularization_weight_decay = 0
# regularization_weight_decay = 0.1
# regularization_weight_decay = 0.5

# dropout???

# layers to freeze and unfreeze?

y_col = "binary_complexity"
# y_col = "complexity"

x_task = "single"
# x_task = "multi"

# x_col = "sentence"
x_col = "sentence_no_contractions"
# x_col = "pos_sequence"
# x_col = "dep_sequence"
# x_col = "morph_sequence"

if x_task == "single":
    df_train = train_single_df
    df_val   = trial_val_single_df
    df_test  = test_single_df
else:
    df_train = train_multi_df
    df_val   = trial_val_multi_df
    df_test  = test_multi_df
```

[ ]:

[ ]:

[ ]:

[ ]: