

3_2 10 e, 16 bs, 256 ml, 0.1 wd, 1e-5 lr

April 13, 2025

0.1 Packages, Library Imports, File Mounts, & Data Imports ** Run All **

```
[1]: !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
```

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

```
Hit:1 https://cloud.r-project.org/bin/linux/ubuntu jammy-cran40/ InRelease
Hit:2 https://developer.download.nvidia.com/compute/cuda/repos/ubuntu2204/x86_64
InRelease
Hit:3 https://r2u.stat.illinois.edu/ubuntu jammy InRelease
Hit:4 http://archive.ubuntu.com/ubuntu jammy InRelease
Hit:5 http://archive.ubuntu.com/ubuntu jammy-updates InRelease
Hit:6 http://security.ubuntu.com/ubuntu jammy-security InRelease
Hit:7 http://archive.ubuntu.com/ubuntu jammy-backports InRelease
Hit:8 https://ppa.launchpadcontent.net/deadsnakes/ppa/ubuntu jammy InRelease
Hit:9 https://ppa.launchpadcontent.net/graphics-drivers/ppa/ubuntu jammy
InRelease
Hit:10 https://ppa.launchpadcontent.net/ubuntugis/ppa/ubuntu jammy InRelease
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 47 not upgraded.
```

```
[3]: #@title Imports
import nltk
```

```

from nltk.tokenize import RegexpTokenizer
import sentencepiece
import contractions
import spacy

import evaluate
from datasets import load_dataset, Dataset, DatasetDict

import torch
import torch.nn as nn
from torchinfo import summary

import transformers
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
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 json
import datetime
import zoneinfo
from datetime import datetime

```

```
[4]: # @title Mount Google Drive
```

```
[5]: 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).

```
[6]: 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/'
log_filename = "experiment_runs.txt"
log_filepath = os.path.join(dir_results, log_filename)

```

```
[7]: wandbai_api_key = ""
```

```
[8]: !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

```
[9]: !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
```

```
[10]: !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

```
[11]: #@title Import Data
```

```
[12]: 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

0.2 Experiments

0.2.1 Helper Functions **** Run ****

```
[19]: MODEL_LINEAGE = {}

def get_model_and_tokenizer(
    remote_model_name: str = None,
    local_model_path: str = None,
    config=None
):
    """
    Loads the model & tokenizer for classification.
    If 'local_model_path' is specified, load from that path.
    Otherwise, fall back to 'remote_model_name'.

    Optional: 'config' can be a custom BertConfig/AutoConfig object
              to override certain configuration parameters.

    Records complete traceable lineage in the global MODEL_LINEAGE.
    """
    global MODEL_LINEAGE

    if local_model_path:
        print(f"Loading from local path: {local_model_path}")
        tokenizer = AutoTokenizer.from_pretrained(local_model_path)

        # If a config object is provided, we pass it to from_pretrained.
        # Otherwise, it just uses the config that is part of local_model_path.
        if config is not None:
            model = AutoModelForSequenceClassification.from_pretrained(
                local_model_path,
                config=config
            )
        else:
            model = AutoModelForSequenceClassification.
↳ from_pretrained(local_model_path)

        MODEL_LINEAGE = {
            "type": "offline_checkpoint",
            "path": local_model_path,
            "timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
        }
    elif remote_model_name:
        print(f"Loading from Hugging Face model: {remote_model_name}")
        tokenizer = AutoTokenizer.from_pretrained(remote_model_name)

        if config is not None:
```

```

        model = AutoModelForSequenceClassification.from_pretrained(
            remote_model_name,
            config=config
        )
    else:
        model = AutoModelForSequenceClassification.
↳from_pretrained(remote_model_name)

    MODEL_LINEAGE = {
        "type": "huggingface_hub",
        "path": remote_model_name,
        "timestamp": datetime.now().strftime("%Y-%m-%d %H:%M:%S")
    }
    else:
        raise ValueError("You must provide either a remote_model_name or a
↳local_model_path!")

    return model, tokenizer

```

```

[20]: 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(substring in name for substring in layers_to_unfreeze):
            param.requires_grad = True
        else:
            param.requires_grad = False

```

```

[21]: 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
```

```
[22]: 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.
    """
    dataset = Dataset.from_pandas(df)

    dataset = dataset.map(
        lambda batch: encode_examples(batch, tokenizer, text_col, max_length),
        batched=True
    )

    dataset = dataset.rename_column(label_col, "labels")
    dataset.set_format(type='torch',
                       columns=['input_ids', 'attention_mask', 'labels'])
    return dataset

[23]: 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"]
    }
```



```
[24]: def gather_config_details(model):
    """
    Enumerates every attribute in model.config
    """
    config_items = {}
    for attr_name, attr_value in vars(model.config).items():
        config_items[attr_name] = attr_value
    return config_items

def gather_model_details(model):
    """
    Extracts total layers, total params, trainable params, and activation_
    ↪function
    from a Transformers model. Adjust logic as needed for different_
    ↪architectures.
    """
    details = {}

    try:
        total_params = model.num_parameters()
        trainable_params = model.num_parameters(only_trainable=True)
    except AttributeError:
        all_params = list(model.parameters())
        total_params = sum(p.numel() for p in all_params)
        trainable_params = sum(p.numel() for p in all_params if p.requires_grad)

    details["model_total_params"] = total_params
    details["model_trainable_params"] = trainable_params

    if hasattr(model, "bert") and hasattr(model.bert, "pooler"):
        act_obj = getattr(model.bert.pooler, "activation", None)
        details["pooler_activation_function"] = act_obj.__class__.__name__ if_
    ↪act_obj else "N/A"
    else:
        details["pooler_activation_function"] = "N/A"

    details["config_attributes"] = gather_config_details(model)
    return details

def gather_all_run_metrics(trainer, train_dataset=None, val_dataset=None,
    ↪test_dataset=None):
    """
    Gathers final training metrics, final validation metrics, final test_
    ↪metrics.
    Instead of only parsing the final train_loss from the log, we also do a full
    trainer.evaluate(train_dataset) to get the same set of metrics that val/
    ↪test have.

```

```

"""
results = {}

if train_dataset is not None:
    train_metrics = trainer.evaluate(train_dataset)
    for k, v in train_metrics.items():
        results[f"train_{k}"] = v
else:
    results["train_metrics"] = "No train dataset provided"

if val_dataset is not None:
    val_metrics = trainer.evaluate(val_dataset)
    for k, v in val_metrics.items():
        results[f"val_{k}"] = v
else:
    results["val_metrics"] = "No val dataset provided"

if test_dataset is not None:
    test_metrics = trainer.evaluate(test_dataset)
    for k, v in test_metrics.items():
        results[f"test_{k}"] = v
else:
    results["test_metrics"] = "No test dataset provided"

return results

# def log_experiment_results_json(experiment_meta, model_details, run_metrics,
# ↪ log_file):
#     """
#     Logs experiment metadata, model details, and metrics to a JSON lines file.
#     Automatically concatenates the 'checkpoint_path' to the 'model_lineage'.
#     """
#     checkpoint_path = model_details.get("checkpoint_path")
#     if checkpoint_path:
#         if "model_lineage" not in model_details:
#             model_details["model_lineage"] = ""
#         if model_details["model_lineage"]:
#             model_details["model_lineage"] += " -> "
#         model_details["model_lineage"] += checkpoint_path

#     record = {
#         "timestamp": str(datetime.datetime.now()),
#         "experiment_meta": experiment_meta,
#         "model_details": model_details,
#         "run_metrics": run_metrics
#     }

```

```

#     with open(log_file, "a", encoding="utf-8") as f:
#         json.dump(record, f)
#         f.write("\n")

def log_experiment_results_json(experiment_meta, model_details, run_metrics,
    ↪log_file):
    """
    Logs experiment metadata, model details, and metrics to a JSON lines file.
    Automatically concatenates the 'checkpoint_path' to the 'model_lineage'
    and uses Pacific time for the timestamp.
    """
    checkpoint_path = model_details.get("checkpoint_path")
    if checkpoint_path:
        if "model_lineage" not in model_details:
            model_details["model_lineage"] = ""
        if model_details["model_lineage"]:
            model_details["model_lineage"] += " -> "
        model_details["model_lineage"] += checkpoint_path

    pacific_time = datetime.now(zoneinfo.ZoneInfo("America/Los_Angeles")) # ↪
    ↪update to support pacific time
    timestamp_str = pacific_time.isoformat()

    record = {
        "timestamp": timestamp_str,
        "experiment_meta": experiment_meta,
        "model_details": model_details,
        "run_metrics": run_metrics
    }

    with open(log_file, "a", encoding="utf-8") as f:
        json.dump(record, f)
        f.write("\n")

```

0.2.2 Experiment Cohort Design

```

[25]: # 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 = 1
# 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
# size_batch = 64
# size_batch = 128

# regularization_weight_decay = 0
regularization_weight_decay = 0.1
# regularization_weight_decay = 0.5

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 = train_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

custom_config = BertConfig.from_pretrained("bert-base-cased")

custom_config.hidden_dropout_prob = 0.1
# custom_config.intermediate_size = 3072
# custom_config.intermediate_size = 6144
# custom_config.num_attention_heads = 12
# custom_config.num_hidden_layers = 12
custom_config.gradient_checkpointing = False
custom_config.attention_probs_dropout_prob = 0.1
# custom_config.max_position_embeddings = 512
# custom_config.type_vocab_size = 2
custom_config.hidden_act = "gelu" # alts: "relu" "silu"
# custom_config.vocab_size = 28996 # must match

# model.bert.pooler.activation = nn.ReLU() # Tanh() replaced as the pooler
↳ layer activation function in side-by-side with 1.1

```

```

[26]: 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"
        warmup_steps=100
    )

    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

```

Model Inspection ** Run **

```

[27]: print("model checkpoints:", dir_models)
      !ls /content/drive/MyDrive/266-final/models/

```

```

model checkpoints: /content/drive/MyDrive/266-final/models/
multi_bert-base-cased_binary_complexity_20250408_143322
multi_bert-base-cased_binary_complexity_20250409_175804
multi_bert-base-cased_binary_complexity_20250409_175954
multi_bert-base-cased_binary_complexity_20250409_180139
multi_bert-base-cased_binary_complexity_20250409_185057
multi_bert-base-cased_binary_complexity_20250409_185213
multi_bert-base-cased_binary_complexity_20250409_185333
multi_bert-base-cased_binary_complexity_20250409_234934
single_bert-base-cased_binary_complexity_20250408_043117
single_bert-base-cased_binary_complexity_20250408_043334
single_bert-base-cased_binary_complexity_20250408_043750
single_bert-base-cased_binary_complexity_20250409_175702
single_bert-base-cased_binary_complexity_20250409_175900
single_bert-base-cased_binary_complexity_20250409_180045
single_bert-base-cased_binary_complexity_20250409_185027
single_bert-base-cased_binary_complexity_20250409_185141
single_bert-base-cased_binary_complexity_20250409_185303
single_bert-base-cased_binary_complexity_20250409_234236

```

```

[28]: # 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/....") # proposed argument usage for checkpointed models

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

model, tokenizer = get_model_and_tokenizer(
    remote_model_name="bert-base-cased",
    local_model_path=None,
    config=custom_config
)

# model, tokenizer = get_model_and_tokenizer(
#     local_model_path="my_local_bert_path",
#     config=custom_config
# )

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-base-cased :

=====

=====

```

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 ** Run **

[29]: *# Freeze/Unfreeze Layers & Additional Activation Function Configuration*

```

layers_to_unfreeze = [
    # "bert.embeddings.",
    "bert.encoder.layer.0.",
    # "bert.encoder.layer.1.",
    "bert.encoder.layer.8.",
    "bert.encoder.layer.9.",
    "bert.encoder.layer.10.",
    "bert.encoder.layer.11.",
    "bert.pooler.",
    "classifier.",
]

freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)

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= True
bert.encoder.layer.0.attention.self.query.bias requires_grad= True
bert.encoder.layer.0.attention.self.key.weight requires_grad= True
bert.encoder.layer.0.attention.self.key.bias requires_grad= True
bert.encoder.layer.0.attention.self.value.weight requires_grad= True
bert.encoder.layer.0.attention.self.value.bias requires_grad= True
bert.encoder.layer.0.attention.output.dense.weight requires_grad= True
bert.encoder.layer.0.attention.output.dense.bias requires_grad= True
bert.encoder.layer.0.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.0.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.0.intermediate.dense.weight requires_grad= True
bert.encoder.layer.0.intermediate.dense.bias requires_grad= True
bert.encoder.layer.0.output.dense.weight requires_grad= True
bert.encoder.layer.0.output.dense.bias requires_grad= True
bert.encoder.layer.0.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.0.output.LayerNorm.bias requires_grad= True
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

```


[illegible]

bert.encoder.layer.8.attention.self.key.weight requires_grad= True
bert.encoder.layer.8.attention.self.key.bias requires_grad= True
bert.encoder.layer.8.attention.self.value.weight requires_grad= True
bert.encoder.layer.8.attention.self.value.bias requires_grad= True
bert.encoder.layer.8.attention.output.dense.weight requires_grad= True
bert.encoder.layer.8.attention.output.dense.bias requires_grad= True
bert.encoder.layer.8.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.8.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.8.intermediate.dense.weight requires_grad= True
bert.encoder.layer.8.intermediate.dense.bias requires_grad= True
bert.encoder.layer.8.output.dense.weight requires_grad= True
bert.encoder.layer.8.output.dense.bias requires_grad= True
bert.encoder.layer.8.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.8.output.LayerNorm.bias requires_grad= True
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 :

```

```

=====

```

```

=====

```

```

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: 36031490

```

Dataset Preparation ** Run **

```

[30]: # 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,
```



```

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: 1e-05
number of epochs: 10
maximum sequence length: 256
batch size used: 16
regularization value: 0.1
outcome variable: binary_complexity
task: single
input column: sentence_no_contractions

```

[32]: *# 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-26-c2ee9f934517>:31: 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': 0.8265407085418701, 'eval_accuracy': 0.6080760095011877, 'eval_precision': 0.5754189944134078, 'eval_recall':


```
0.5364583333333334, 'eval_f1': 0.555256064690027, 'eval_runtime': 2.4756,
'eval_samples_per_second': 170.057, 'eval_steps_per_second': 10.906, 'epoch':
10.0}
Test metrics: {'eval_loss': 0.8846817016601562, 'eval_accuracy':
0.5768811341330425, 'eval_precision': 0.5667506297229219, 'eval_recall':
0.5102040816326531, 'eval_f1': 0.5369928400954654, 'eval_runtime': 4.4947,
'eval_samples_per_second': 204.017, 'eval_steps_per_second': 12.904, 'epoch':
10.0}
```

[33]: *# save model checkpoint*

```
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
model_save_path = os.path.join(dir_models,
    ↪f"{x_task}_{named_model}_{y_col}_{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/single_bert-base-cased_binary_complexity_20250410_000508

[34]:

```
experiment_info = {
    "model_name": named_model,
    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,
    "x_col": x_col,
    "y_col": y_col,
    "layers_to_unfreeze": layers_to_unfreeze
}

model_info = gather_model_details(trained_model)

all_run_metrics = gather_all_run_metrics(
    trainer=trainer_obj,
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    test_dataset=test_data_hf
)

log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath
)
```

```
print(f"EXPERIMENT LOGGED TO: {log_filepath}")
```

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:

/content/drive/MyDrive/266-final/results/experiment_runs.txt

0.2.4 3.1.1.1 from checkpoint 3.1.1 Y: single task 1 & X: sentence_no_contractions — X ** skip **

```
[ ]: # Load Model & Tokenizer

# model, tokenizer = get_model_and_tokenizer(
#     remote_model_name="bert-base-cased",
#     config=custom_config
# )

model, tokenizer = get_model_and_tokenizer(
    remote_model_name=None,
    local_model_path="/content/drive/MyDrive/266-final/models/
↪single_bert-base-cased_binary_complexity_20250408_043117",
    config=custom_config
)

print("=====")
print(named_model, ":")
print("=====")
# print(model)
print("=====")
print(model.config)
# print("=====")
```

Loading from local path: /content/drive/MyDrive/266-final/models/single_bert-base-cased_binary_complexity_20250408_043117

=====

bert-base-cased :

=====

=====

```
BertConfig {
  "_attn_implementation_autoset": true,
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0,
  "classifier_dropout": null,
  "gradient_checkpointing": false,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0,
```

```

"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
}

```

```

[ ]: # Define Experiment Parameters

num_epochs = 10

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

```

```

custom_config = BertConfig.from_pretrained("bert-base-cased")

custom_config.hidden_dropout_prob = 0.1
# custom_config.intermediate_size = 3072
# custom_config.intermediate_size = 6144
# custom_config.num_attention_heads = 12
# custom_config.num_hidden_layers = 12
custom_config.gradient_checkpointing = False
custom_config.attention_probs_dropout_prob = 0.1
# custom_config.max_position_embeddings = 512
# custom_config.type_vocab_size = 2
custom_config.hidden_act = "gelu" # alts: "relu" "silu"
# custom_config.vocab_size = 28996 # must match

# model.bert.pooler.activation = nn.ReLU() # Tanh() replaced as the pooler_
↳ layer activation function in side-by-side with 1.1

```

```

[ ]: # 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 = 1
# num_epochs = 3
# num_epochs = 15
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
# size_batch = 64
# size_batch = 128

# regularization_weight_decay = 0
regularization_weight_decay = 0.1
# regularization_weight_decay = 0.5

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

custom_config = BertConfig.from_pretrained("bert-base-cased")

custom_config.hidden_dropout_prob = 0.1
# custom_config.intermediate_size = 3072
# custom_config.intermediate_size = 6144
# custom_config.num_attention_heads = 12
# custom_config.num_hidden_layers = 12
custom_config.gradient_checkpointing = False
custom_config.attention_probs_dropout_prob = 0.1
# custom_config.max_position_embeddings = 512
# custom_config.type_vocab_size = 2
custom_config.hidden_act = "gelu" # alts: "relu" "silu"
# custom_config.vocab_size = 28996 # must match

# model.bert.pooler.activation = nn.ReLU() # Tanh() replaced as the pooler_
↳ layer activation function in side-by-side with 1.1

```

```

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= True
bert.encoder.layer.0.attention.self.query.bias requires_grad= True
bert.encoder.layer.0.attention.self.key.weight requires_grad= True
bert.encoder.layer.0.attention.self.key.bias requires_grad= True
bert.encoder.layer.0.attention.self.value.weight requires_grad= True
bert.encoder.layer.0.attention.self.value.bias requires_grad= True
bert.encoder.layer.0.attention.output.dense.weight requires_grad= True
bert.encoder.layer.0.attention.output.dense.bias requires_grad= True
bert.encoder.layer.0.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.0.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.0.intermediate.dense.weight requires_grad= True
bert.encoder.layer.0.intermediate.dense.bias requires_grad= True
bert.encoder.layer.0.output.dense.weight requires_grad= True
bert.encoder.layer.0.output.dense.bias requires_grad= True
bert.encoder.layer.0.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.0.output.LayerNorm.bias requires_grad= True
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.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= False
bert.encoder.layer.9.attention.self.query.bias requires_grad= False
bert.encoder.layer.9.attention.self.key.weight requires_grad= False
bert.encoder.layer.9.attention.self.key.bias requires_grad= False
bert.encoder.layer.9.attention.self.value.weight requires_grad= False
bert.encoder.layer.9.attention.self.value.bias requires_grad= False
bert.encoder.layer.9.attention.output.dense.weight requires_grad= False
bert.encoder.layer.9.attention.output.dense.bias requires_grad= False
bert.encoder.layer.9.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.9.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.9.intermediate.dense.weight requires_grad= False
bert.encoder.layer.9.intermediate.dense.bias requires_grad= False
bert.encoder.layer.9.output.dense.weight requires_grad= False
bert.encoder.layer.9.output.dense.bias requires_grad= False
bert.encoder.layer.9.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.9.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.10.attention.self.query.weight requires_grad= False
bert.encoder.layer.10.attention.self.query.bias requires_grad= False
bert.encoder.layer.10.attention.self.key.weight requires_grad= False
bert.encoder.layer.10.attention.self.key.bias requires_grad= False
bert.encoder.layer.10.attention.self.value.weight requires_grad= False
bert.encoder.layer.10.attention.self.value.bias requires_grad= False
bert.encoder.layer.10.attention.output.dense.weight requires_grad= False
bert.encoder.layer.10.attention.output.dense.bias requires_grad= False
bert.encoder.layer.10.attention.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.10.attention.output.LayerNorm.bias requires_grad= False
bert.encoder.layer.10.intermediate.dense.weight requires_grad= False
bert.encoder.layer.10.intermediate.dense.bias requires_grad= False
bert.encoder.layer.10.output.dense.weight requires_grad= False
bert.encoder.layer.10.output.dense.bias requires_grad= False
bert.encoder.layer.10.output.LayerNorm.weight requires_grad= False
bert.encoder.layer.10.output.LayerNorm.bias requires_grad= False
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 :

```

```

=====

```

```

=====

```

```

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: 14767874

```

```
[ ]: 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-06
number of epochs: 1
maximum sequence length: 128
batch size used: 128
regularization value: 0.5
outcome variable: binary_complexity
task: multi
input column: sentence_no_contractions
```

```
[ ]: # 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-22-295bdbf803a2>: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': 0.681236982345581, 'eval_accuracy':  
0.5653206650831354, 'eval_precision': 0.5217391304347826, 'eval_recall': 0.5625,  
'eval_f1': 0.5413533834586466, 'eval_runtime': 5.4089,  
'eval_samples_per_second': 77.835, 'eval_steps_per_second': 0.74, 'epoch': 3.0}  
Test metrics: {'eval_loss': 0.6863542199134827, 'eval_accuracy':  
0.5627044711014176, 'eval_precision': 0.5540540540540541, 'eval_recall':  
0.46485260770975056, 'eval_f1': 0.5055487053020962, 'eval_runtime': 6.2945,  
'eval_samples_per_second': 145.682, 'eval_steps_per_second': 1.271, 'epoch':  
3.0}
```

```
[ ]: # save model checkpoint
```

```
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")  
model_save_path = os.path.join(dir_models,   
    ↪f"{x_task}_{named_model}_{y_col}_{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/single_bert-base-cased_binary_complexity_20250408_043750

```
[ ]: experiment_info = {  
    "model_name": named_model,  
    "learning_rate": learning_rate,  
    "epochs": num_epochs,  
    "batch_size": size_batch,  
    "weight_decay": regularization_weight_decay,  
    "x_task": x_task,  
    "x_col": x_col,  
    "y_col": y_col,  
    "layers_to_unfreeze": layers_to_unfreeze  
}  
  
model_info = gather_model_details(trained_model)  
  
all_run_metrics = gather_all_run_metrics(  
    trainer=trainer_obj,  
    train_dataset=train_data_hf,  
    val_dataset=val_data_hf,  
    test_dataset=test_data_hf  
)
```

```

log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath
)

print(f"EXPERIMENT LOGGED TO: {log_filepath}")

```

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:
/content/drive/MyDrive/266-final/results/experiment_runs.txt

0.2.5 3.1.2: from pretrained bert-base-cased Y: multi task 2 & X: sentence_no_contractions — Y

```

[35]: # 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 = 1
# 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
# size_batch = 64
# size_batch = 128

# regularization_weight_decay = 0
regularization_weight_decay = 0.1
# regularization_weight_decay = 0.5

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

custom_config = BertConfig.from_pretrained("bert-base-cased")

custom_config.hidden_dropout_prob = 0.1
# custom_config.intermediate_size = 3072
# custom_config.intermediate_size = 6144
# custom_config.num_attention_heads = 12
# custom_config.num_hidden_layers = 12
custom_config.gradient_checkpointing = False
custom_config.attention_probs_dropout_prob = 0.1
# custom_config.max_position_embeddings = 512
# custom_config.type_vocab_size = 2
custom_config.hidden_act = "gelu" # alts: "relu" "silu"
# custom_config.vocab_size = 28996 # must match

# model.bert.pooler.activation = nn.ReLU() # Tanh() replaced as the pooler_
↳ layer activation function in side-by-side with 1.1

```

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

```
model checkpoints: /content/drive/MyDrive/266-final/models/
multi_bert-base-cased_binary_complexity_20250408_143322
multi_bert-base-cased_binary_complexity_20250409_175804
multi_bert-base-cased_binary_complexity_20250409_175954
multi_bert-base-cased_binary_complexity_20250409_180139
multi_bert-base-cased_binary_complexity_20250409_185057
multi_bert-base-cased_binary_complexity_20250409_185213
multi_bert-base-cased_binary_complexity_20250409_185333
multi_bert-base-cased_binary_complexity_20250409_234934
single_bert-base-cased_binary_complexity_20250408_043117
single_bert-base-cased_binary_complexity_20250408_043334
single_bert-base-cased_binary_complexity_20250408_043750
single_bert-base-cased_binary_complexity_20250409_175702
single_bert-base-cased_binary_complexity_20250409_175900
single_bert-base-cased_binary_complexity_20250409_180045
single_bert-base-cased_binary_complexity_20250409_185027
single_bert-base-cased_binary_complexity_20250409_185141
single_bert-base-cased_binary_complexity_20250409_185303
single_bert-base-cased_binary_complexity_20250409_234236
single_bert-base-cased_binary_complexity_20250410_000508
```

```
[37]: # 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/...") # proposed argument usage for checkpointed models

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

model, tokenizer = get_model_and_tokenizer(
    remote_model_name="bert-base-cased",
    local_model_path=None,
    config=custom_config
)

# model, tokenizer = get_model_and_tokenizer(
#     local_model_path="my_local_bert_path",
#     config=custom_config
# )

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-base-cased :

=====

=====

```
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

```
[38]: # Freeze/Unfreeze Layers & Additional Activation Function Configuration
layers_to_unfreeze = [
    # "bert.embeddings.",
    "bert.encoder.layer.0.",
    # "bert.encoder.layer.1.",
    "bert.encoder.layer.8.",
    "bert.encoder.layer.9.",
    "bert.encoder.layer.10.",
    "bert.encoder.layer.11.",
    "bert.pooler.",
    "classifier.",
]

freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)

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= True
bert.encoder.layer.0.attention.self.query.bias requires_grad= True
bert.encoder.layer.0.attention.self.key.weight requires_grad= True
bert.encoder.layer.0.attention.self.key.bias requires_grad= True
bert.encoder.layer.0.attention.self.value.weight requires_grad= True
bert.encoder.layer.0.attention.self.value.bias requires_grad= True
bert.encoder.layer.0.attention.output.dense.weight requires_grad= True
bert.encoder.layer.0.attention.output.dense.bias requires_grad= True
bert.encoder.layer.0.attention.output.LayerNorm.weight requires_grad= True
```

[illegible]

[illegible]

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= True
bert.encoder.layer.8.attention.self.query.bias requires_grad= True
bert.encoder.layer.8.attention.self.key.weight requires_grad= True
bert.encoder.layer.8.attention.self.key.bias requires_grad= True
bert.encoder.layer.8.attention.self.value.weight requires_grad= True
bert.encoder.layer.8.attention.self.value.bias requires_grad= True
bert.encoder.layer.8.attention.output.dense.weight requires_grad= True
bert.encoder.layer.8.attention.output.dense.bias requires_grad= True
bert.encoder.layer.8.attention.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.8.attention.output.LayerNorm.bias requires_grad= True
bert.encoder.layer.8.intermediate.dense.weight requires_grad= True
bert.encoder.layer.8.intermediate.dense.bias requires_grad= True
bert.encoder.layer.8.output.dense.weight requires_grad= True
bert.encoder.layer.8.output.dense.bias requires_grad= True
bert.encoder.layer.8.output.LayerNorm.weight requires_grad= True
bert.encoder.layer.8.output.LayerNorm.bias requires_grad= True
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 :

=====

```

=====
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: 36031490

```

```

[39]: 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: 1e-05
number of epochs: 10

```

```

maximum sequence length: 256
batch size used: 16
regularization value: 0.1
outcome variable: binary_complexity
task: multi
input column: sentence_no_contractions

```

```

[40]: def validate_dataframe(df, df_name):
    """
    Performs basic functional tests on a pandas DataFrame
    to ensure it matches expected structure and content.
    """
    print(f"\n[VALIDATION] Checking {df_name}...")

    # 1) Check shape
    print(f" - Shape: {df.shape}")

    # 2) Check columns
    print(f" - Columns: {list(df.columns)}")

    # 3) Check label distribution (assuming 'binary_complexity' is the label)
    if "binary_complexity" in df.columns:
        label_counts = df["binary_complexity"].value_counts(dropna=False)
        print(f" - Label distribution:\n{label_counts}")
    else:
        print(" - WARNING: 'binary_complexity' column not found!")

    # 4) Peek at top few rows
    print(" - Sample rows:\n", df.head(3))

    validate_dataframe(train_multi_df, "train_multi_df")
    validate_dataframe(trial_val_multi_df, "trial_val_multi_df")
    validate_dataframe(test_multi_df, "test_multi_df")

```

```

[VALIDATION] Checking train_multi_df...
 - Shape: (1517, 12)
 - Columns: ['id', 'corpus', 'sentence', 'token', 'complexity',
'sentence_no_contractions', 'contraction_expanded', 'pos_sequence',
'dep_sequence', 'morph_sequence', 'morph_complexity', 'binary_complexity']
 - Label distribution:
binary_complexity
0      759
1      758
Name: count, dtype: int64
 - Sample rows:
               id corpus \
0  3S37Y8CWI80N8KVM53U4E6JKCDC4WE  bible

```

```

1 3WGCNLZJKF877FYC1Q6COKNWDWD11 bible
2 3UOMW19E6D6WQ5TH2HDD74IVKTP5CB bible

```

```

                                sentence          token \
0 but the seventh day is a Sabbath to Yahweh you... seventh day
1 But let each man test his own work, and then h... own work
2 To him who by understanding made the heavens; ... loving kindness

```

```

complexity          sentence_no_contractions \
0 0.027778 but the seventh day is a Sabbath to Yahweh you...
1 0.050000 But let each man test his own work, and then h...
2 0.050000 To him who by understanding made the heavens; ...

```

```

contraction_expanded          pos_sequence \
0 False ['CCONJ', 'DET', 'ADJ', 'NOUN', 'AUX', 'DET', ...
1 False ['CCONJ', 'VERB', 'DET', 'NOUN', 'VERB', 'PRON...
2 False ['ADP', 'PRON', 'PRON', 'ADP', 'VERB', 'VERB',...

```

```

                                dep_sequence \
0 ['cc', 'det', 'amod', 'nsubj', 'ccomp', 'det',...
1 ['cc', 'ROOT', 'det', 'nsubj', 'ccomp', 'poss'...
2 ['prep', 'pobj', 'nsubj', 'prep', 'pcomp', 'ad...

```

```

                                morph_sequence morph_complexity \
0 [ConjType=Cmp, Definite=Def|PronType=Art, Degr... 1.341772
1 [ConjType=Cmp, VerbForm=Inf, , Number=Sing, Ve... 1.608696
2 [, Case=Acc|Gender=Masc|Number=Sing|Person=3|P... 1.562500

```

```

binary_complexity
0 0
1 0
2 0

```

```

[VALIDATION] Checking trial_val_multi_df...
- Shape: (99, 12)
- Columns: ['id', 'corpus', 'sentence', 'token', 'complexity',
'sentence_no_contractions', 'contraction_expanded', 'pos_sequence',
'dep_sequence', 'morph_sequence', 'morph_complexity', 'binary_complexity']
- Label distribution:

```

```
binary_complexity
```

```

1 51
0 48

```

```
Name: count, dtype: int64
```

```
- Sample rows:
```

```

                                id corpus \
0 31HLTCK4BLVQ5B01AUR91TX9V9IVGH bible
1 389A2A3040IXVY7G5B71Q9M43LEOCL bible
2 31N9JPQXIPIRX2A3S9NOCCFX06TNHR bible

```



```

                                sentence      token \
0 The name of one son was Gershom, for Moses sai... foreign land
1 unleavened bread, unleavened cakes mixed with ... wheat flour
2 However the high places were not taken away; t... burnt incense

complexity                      sentence_no_contractions \
0 0.000000 The name of one son was Gershom, for Moses sai...
1 0.157895 unleavened bread, unleavened cakes mixed with ...
2 0.200000 However the high places were not taken away; t...

contraction_expanded              pos_sequence \
0 False ['DET', 'NOUN', 'ADP', 'NUM', 'NOUN', 'AUX', '...'
1 False ['ADJ', 'NOUN', 'PUNCT', 'ADJ', 'NOUN', 'VERB'...'
2 False ['ADV', 'DET', 'ADJ', 'NOUN', 'AUX', 'PART', '...'

                                dep_sequence \
0 ['det', 'nsubj', 'prep', 'nummod', 'pobj', 'RO...'
1 ['amod', 'dep', 'punct', 'amod', 'appos', 'acl...'
2 ['advmod', 'det', 'amod', 'nsubjpass', 'auxpas...'

                                morph_sequence morph_complexity \
0 [Definite=Def|PronType=Art, Number=Sing, , Num... 1.520000
1 [Degree=Pos, Number=Sing, PunctType=Comm, Degr... 1.200000
2 [, Definite=Def|PronType=Art, Degree=Pos, Numb... 1.190476

binary_complexity
0 0
1 0
2 0

[VALIDATION] Checking test_multi_df...
- Shape: (184, 12)
- Columns: ['id', 'corpus', 'sentence', 'token', 'complexity',
'sentence_no_contractions', 'contraction_expanded', 'pos_sequence',
'dep_sequence', 'morph_sequence', 'morph_complexity', 'binary_complexity']
- Label distribution:
binary_complexity
1 99
0 85
Name: count, dtype: int64
- Sample rows:
                                id corpus \
0 3UXQ63NLAAMRIP4WG4XPD98A0YOBLX bible
1 3FJ2RVH25Z62TA3R8E1077EBUYU92W bible
2 3Y04AH2FPDK1PZHAT8WAEBL70EQOF bible

```

```

                                sentence      token \

```

```

0 for he had an only daughter, about twelve year... only daughter
1 All these were cities fortified with high wall... high walls
2 In the morning, 'It will be foul weather today... weather today

complexity sentence_no_contractions \
0 0.025 for he had an only daughter, about twelve year...
1 0.100 All these were cities fortified with high wall...
2 0.125 In the morning, 'It will be foul weather today...

contraction_expanded pos_sequence \
0 False ['SCONJ', 'PRON', 'VERB', 'DET', 'ADJ', 'NOUN'...
1 False ['DET', 'PRON', 'AUX', 'NOUN', 'VERB', 'ADP', ...
2 False ['ADP', 'DET', 'NOUN', 'PUNCT', 'PUNCT', 'PRON...

dep_sequence \
0 ['mark', 'nsubj', 'ROOT', 'det', 'amod', 'dobj...
1 ['predet', 'nsubj', 'ROOT', 'attr', 'acl', 'pr...
2 ['prep', 'det', 'pobj', 'punct', 'punct', 'nsu...

morph_sequence morph_complexity \
0 [, Case=Nom|Gender=Masc|Number=Sing|Person=3|P... 1.722222
1 [, Number=Plur|PronType=Dem, Mood=Ind|Tense=Pa... 1.136364
2 [, Definite=Def|PronType=Art, Number=Sing, Pun... 1.476190

binary_complexity
0 0
1 0
2 0

```

```
[41]: # 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-26-c2ee9f934517>:31: 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': 0.8370015621185303, 'eval_accuracy':
0.6104513064133017, 'eval_precision': 0.57, 'eval_recall': 0.59375, 'eval_f1':
0.5816326530612245, 'eval_runtime': 4.2073, 'eval_samples_per_second': 100.064,
'eval_steps_per_second': 6.417, 'epoch': 10.0}
Test metrics: {'eval_loss': 0.9402342438697815, 'eval_accuracy':
0.5670665212649946, 'eval_precision': 0.5491071428571429, 'eval_recall':
0.5578231292517006, 'eval_f1': 0.5534308211473565, 'eval_runtime': 4.1098,
'eval_samples_per_second': 223.123, 'eval_steps_per_second': 14.112, 'epoch':
10.0}
```

```
[42]: # save model checkpoint
```

```
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
model_save_path = os.path.join(dir_models,
    f"{x_task}_{named_model}_{y_col}_{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/multi_bert-base-cased_binary_complexity_20250410_001637

```
[43]: experiment_info = {
    "model_name": named_model,
    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,
    "x_col": x_col,
    "y_col": y_col,
    "layers_to_unfreeze": layers_to_unfreeze
}
```

```

model_info = gather_model_details(trained_model)

all_run_metrics = gather_all_run_metrics(
    trainer=trainer_obj,
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    test_dataset=test_data_hf
)

log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath
)

print(f"EXPERIMENT LOGGED TO: {log_filepath}")

```

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:
/content/drive/MyDrive/266-final/results/experiment_runs.txt

0.2.6 3.1.3 from pretrained bert-base-cased Y: single task 1 & X: pos_sequence —

```

[44]: # Define Experiment Parameters
named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = "" # modern bert
#####
regularization_weight_decay = 0.1
learning_rate = 1e-5
size_batch = 16
length_max = 256
num_epochs = 10
#####
# x_col = "sentence"
# x_col = "sentence_no_contractions"
x_col = "pos_sequence"
# x_col = "dep_sequence"
# x_col = "morph_sequence"
#####
y_col = "binary_complexity"
# y_col = "complexity"
#####
x_task = "single"

```

```

# x_task = "multi"
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
#####
# 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])
#####
custom_config = BertConfig.from_pretrained("bert-base-cased")
custom_config.hidden_act = "gelu" # alts: "relu" "silu"
custom_config.attention_probs_dropout_prob = 0.1
custom_config.hidden_dropout_prob = 0.1
custom_config.gradient_checkpointing = False
#####
model, tokenizer = get_model_and_tokenizer(
    remote_model_name="bert-base-cased",
    local_model_path=None,
    config=custom_config)
#####
# model, tokenizer = get_model_and_tokenizer(
#     remote_model_name=None
#     local_model_path="...CONFIGURE_PATH...",

```

```

#     config=custom_config)
print("=====")
print(named_model, ":")
print("=====")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters at load:", model.
    ↪ num_parameters(only_trainable=True))
print("=====")
print("model lineage:", MODEL_LINEAGE)
print("=====")
#####
layers_to_unfreeze = [
    # "bert.embeddings.",
    "bert.encoder.layer.0.",
    # "bert.encoder.layer.1.",
    "bert.encoder.layer.8.",
    "bert.encoder.layer.9.",
    "bert.encoder.layer.10.",
    "bert.encoder.layer.11.",
    "bert.pooler.",
    "classifier.",
]
freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)
print(model.config)
print("=====")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
print("=====")
#####
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)

```

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, 164, 112, 21362, 11414,
4538, 112, 117, 112, 5844,
```



```

=====
model lineage: {'type': 'huggingface_hub', 'path': 'bert-base-cased',
'timestamp': '2025-04-10 00:17:14'}
=====

```

```

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: 36031490
=====

```

```

Experiment configuration used with this experiment:
model used: bert-base-cased
learning rate used: 1e-05
number of epochs: 10
maximum sequence length: 256
batch size used: 16
regularization value: 0.1
outcome variable: binary_complexity
task: single
input column: pos_sequence

```



```
[45]: # 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-26-c2ee9f934517>:31: 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>
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565:
UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 due to no
predicted samples. Use `zero_division` parameter to control this behavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
<IPython.core.display.HTML object>
Validation metrics: {'eval_loss': 0.6857385635375977, 'eval_accuracy':
0.5486935866983373, 'eval_precision': 0.5043859649122807, 'eval_recall':
0.5989583333333334, 'eval_f1': 0.5476190476190477, 'eval_runtime': 2.6282,
'eval_samples_per_second': 160.183, 'eval_steps_per_second': 10.273, 'epoch':
10.0}
Test metrics: {'eval_loss': 0.6894793510437012, 'eval_accuracy':
0.5539803707742639, 'eval_precision': 0.5333333333333333, 'eval_recall':
0.5804988662131519, 'eval_f1': 0.5559174809989142, 'eval_runtime': 5.3822,
'eval_samples_per_second': 170.376, 'eval_steps_per_second': 10.776, 'epoch':
10.0}
```

```
[46]: # save model checkpoint
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
model_save_path = os.path.join(dir_models,
    f"{x_task}_{named_model}_{y_col}_{timestamp}")
```

```

trainer_obj.save_model(model_save_path)
print(f"Model checkpoint saved to: {model_save_path}")
# log experiment results
experiment_info = {
    "model_name": named_model,
    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,
    "x_col": x_col,
    "y_col": y_col,
    "layers_to_unfreeze": layers_to_unfreeze}
model_info = gather_model_details(trained_model)
all_run_metrics = gather_all_run_metrics(
    trainer=trainer_obj,
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    test_dataset=test_data_hf)
log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath)
print(f"EXPERIMENT LOGGED TO: {log_filepath}")

```

Model checkpoint saved to: /content/drive/MyDrive/266-final/models/single_bert-base-cased_binary_complexity_20250410_002813

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:

/content/drive/MyDrive/266-final/results/experiment_runs.txt

0.2.7 3.1.4 from pretrained bert-base-cased Y: multi task 2 & X: pos_sequence —

```

[47]: # Define Experiment Parameters
named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = "" # modern bert
#####
regularization_weight_decay = 0.1
learning_rate = 1e-5
size_batch = 16
length_max = 256
num_epochs = 10

```

```
#####
# x_col = "sentence"
# x_col = "sentence_no_contractions"
x_col = "pos_sequence"
# x_col = "dep_sequence"
# x_col = "morph_sequence"
#####
y_col = "binary_complexity"
# y_col = "complexity"
#####
# x_task = "single"
x_task = "multi"
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
#####
# 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])
#####
custom_config = BertConfig.from_pretrained("bert-base-cased")
custom_config.hidden_act = "gelu" # alts: "relu" "silu"
custom_config.attention_probs_dropout_prob = 0.1
```

```

custom_config.hidden_dropout_prob = 0.1
custom_config.gradient_checkpointing = False
#####
model, tokenizer = get_model_and_tokenizer(
    remote_model_name="bert-base-cased",
    local_model_path=None,
    config=custom_config)
#####
# model, tokenizer = get_model_and_tokenizer(
#     remote_model_name=None
#     local_model_path="...CONFIGURE_PATH...",
#     config=custom_config)
print("=====")
print(named_model, ":")
print("=====")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters at load:", model.
    ↪ num_parameters(only_trainable=True))
print("=====")
print("model lineage:", MODEL_LINEAGE)
print("=====")
#####
layers_to_unfreeze = [
    # "bert.embeddings.",
    "bert.encoder.layer.0.",
    # "bert.encoder.layer.1.",
    "bert.encoder.layer.8.",
    "bert.encoder.layer.9.",
    "bert.encoder.layer.10.",
    "bert.encoder.layer.11.",
    "bert.pooler.",
    "classifier.",
]
freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)
print(model.config)
print("=====")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
print("=====")
#####
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)

```


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-base-cased :

=====

num_parameters: 108311810

num_trainable_parameters at load: 108311810

=====

model lineage: {'type': 'huggingface_hub', 'path': 'bert-base-cased',
'timestamp': '2025-04-10 00:28:50'}

=====

```
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: 36031490

=====

Experiment configuration used with this experiment:

model used: bert-base-cased
learning rate used: 1e-05
number of epochs: 10
maximum sequence length: 256
batch size used: 16
regularization value: 0.1
outcome variable: binary_complexity
task: multi
input column: pos_sequence

```
[48]: # #QA

# def validate_dataframe(df, df_name):
#     """
#     Performs basic functional tests on a pandas DataFrame
#     to ensure it matches expected structure and content.
#     """
#     print(f"\n[VALIDATION] Checking {df_name}...")

#     # 1) Check shape
#     print(f" - Shape: {df.shape}")

#     # 2) Check columns
#     print(f" - Columns: {list(df.columns)}")

#     # 3) Check label distribution (assuming 'binary_complexity' is the label)
#     if "binary_complexity" in df.columns:
#         label_counts = df["binary_complexity"].value_counts(dropna=False)
#         print(f" - Label distribution:\n{label_counts}")
#     else:
#         print(" - WARNING: 'binary_complexity' column not found!")

#     # 4) Peek at top few rows
#     print(" - Sample rows:\n", df.head(3))

# # Example usage for multi data:
# validate_dataframe(train_multi_df, "train_multi_df")
# validate_dataframe(trial_val_multi_df, "trial_val_multi_df")
# validate_dataframe(test_multi_df, "test_multi_df")
```

```
[49]: def check_dataframe_invariants(df, df_name, expected_shape, expected_columns):
#     """
#     Ensures that df has the exact shape and columns expected.
#     Raises AssertionError if not.
#     """
#     print(f"\n[CHECK] {df_name}")
```

```

actual_shape = df.shape
actual_columns = set(df.columns)

# 1) Check shape
assert actual_shape == expected_shape, (
    f"[ERROR] {df_name} shape mismatch. "
    f"Expected {expected_shape}, got {actual_shape}."
)

# 2) Check columns
assert actual_columns == set(expected_columns), (
    f"[ERROR] {df_name} columns mismatch. "
    f"Expected {set(expected_columns)}, got {actual_columns}."
)

print(" - PASS: shape and columns match expectations")

# Suppose the actual columns are exactly:
my_expected_cols = [
    "id", "sentence", "sentence_no_contractions", "token",
    "contraction_expanded", "pos_sequence", "morph_sequence",
    "dep_sequence", "morph_complexity", "complexity",
    "binary_complexity", "corpus"
]

check_dataframe_invariants(
    train_multi_df,
    "train_multi_df",
    expected_shape=(1517, 12), # example only
    expected_columns=my_expected_cols
)

```

```

[CHECK] train_multi_df
- PASS: shape and columns match expectations

```

```

[50]: # 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-26-c2ee9f934517>:31: 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': 0.720668375492096, 'eval_accuracy':
0.47474747474747475, 'eval_precision': 0.49122807017543857, 'eval_recall':
0.5490196078431373, 'eval_f1': 0.5185185185185185, 'eval_runtime': 1.3625,
'eval_samples_per_second': 72.659, 'eval_steps_per_second': 5.138, 'epoch':
10.0}
Test metrics: {'eval_loss': 0.72453373670578, 'eval_accuracy':
0.5108695652173914, 'eval_precision': 0.5473684210526316, 'eval_recall':
0.5252525252525253, 'eval_f1': 0.5360824742268041, 'eval_runtime': 1.6266,
'eval_samples_per_second': 113.12, 'eval_steps_per_second': 7.377, 'epoch':
10.0}

```

```

[51]: # save model checkpoint
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
model_save_path = os.path.join(dir_models,
    f"{x_task}_{named_model}_{y_col}_{timestamp}")
trainer_obj.save_model(model_save_path)
print(f"Model checkpoint saved to: {model_save_path}")
# log experiment results
experiment_info = {
    "model_name": named_model,
    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,
    "x_col": x_col,
    "y_col": y_col,
    "layers_to_unfreeze": layers_to_unfreeze}
model_info = gather_model_details(trained_model)
all_run_metrics = gather_all_run_metrics(

```

```

        trainer=trainer_obj,
        train_dataset=train_data_hf,
        val_dataset=val_data_hf,
        test_dataset=test_data_hf)
log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath)
print(f"EXPERIMENT LOGGED TO: {log_filepath}")

```

Model checkpoint saved to: /content/drive/MyDrive/266-final/models/multi_bert-base-cased_binary_complexity_20250410_003117

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:

/content/drive/MyDrive/266-final/results/experiment_runs.txt

0.2.8 3.1.5 from pretrained bert-base-cased Y: single task 1 & X: morph_sequence

```

[52]: # Define Experiment Parameters
named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = "" # modern bert
#####
regularization_weight_decay = 0.1
learning_rate = 1e-5
size_batch = 16
length_max = 256
num_epochs = 10
#####
# x_col = "sentence"
# x_col = "sentence_no_contractions"
# x_col = "pos_sequence"
# x_col = "dep_sequence"
x_col = "morph_sequence"
#####
y_col = "binary_complexity"
# y_col = "complexity"
#####
x_task = "single"
# x_task = "multi"
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
#####
# 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])
#####
custom_config = BertConfig.from_pretrained("bert-base-cased")
custom_config.hidden_act = "gelu" # alts: "relu" "silu"
custom_config.attention_probs_dropout_prob = 0.1
custom_config.hidden_dropout_prob = 0.1
custom_config.gradient_checkpointing = False
#####
model, tokenizer = get_model_and_tokenizer(
    remote_model_name="bert-base-cased",
    local_model_path=None,
    config=custom_config)
#####
# model, tokenizer = get_model_and_tokenizer(
#     remote_model_name=None
#     local_model_path="...CONFIGURE_PATH...",
#     config=custom_config)
print("=====")
print(named_model, ":")
print("=====")

```

```

print("num_parameters:", model.num_parameters())
print("num_trainable_parameters at load:", model.
    ↪ num_parameters(only_trainable=True))
print("=====")
print("model lineage:", MODEL_LINEAGE)
print("=====")
#####
layers_to_unfreeze = [
    # "bert.embeddings.",
    "bert.encoder.layer.0.",
    # "bert.encoder.layer.1.",
    "bert.encoder.layer.8.",
    "bert.encoder.layer.9.",
    "bert.encoder.layer.10.",
    "bert.encoder.layer.11.",
    "bert.pooler.",
    "classifier.",
]
freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)
print(model.config)
print("=====")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
print("=====")
#####
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)

```

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,  164, 16752,  3361,  1942,
16726,  134,  140,  8223,  117,
        117,  3177, 16598,  3150,  134,  3177,  2087,  197,  5096,  1179,
1942, 16726,  134,  2051,  117,  7421,  134,  153,  7535,  1197,
        117,  117,  7421,  134, 13315,  117,  9060,  134,  1302,  1306,
        197, 21108,  134,  7085,  1116,  1665,  197,  7421,  134, 13315,

```



```

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: 36031490

```

=====

Experiment configuration used with this experiment:

```

model used: bert-base-cased
learning rate used: 1e-05
number of epochs: 10
maximum sequence length: 256
batch size used: 16
regularization value: 0.1
outcome variable: binary_complexity
task: single
input column: morph_sequence

```

```

[53]: # 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-26-c2ee9f934517>:31: 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': 0.6723506450653076, 'eval_accuracy':
0.5748218527315915, 'eval_precision': 0.5320197044334976, 'eval_recall': 0.5625,
'eval_f1': 0.5468354430379747, 'eval_runtime': 3.5306,
'eval_samples_per_second': 119.243, 'eval_steps_per_second': 7.647, 'epoch':
10.0}
Test metrics: {'eval_loss': 0.6912660598754883, 'eval_accuracy':
0.5528898582333697, 'eval_precision': 0.5354691075514875, 'eval_recall':
0.5306122448979592, 'eval_f1': 0.5330296127562643, 'eval_runtime': 4.5394,
'eval_samples_per_second': 202.009, 'eval_steps_per_second': 12.777, 'epoch':
10.0}

```

```

[54]: # save model checkpoint
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
model_save_path = os.path.join(dir_models,
    ↪f"{x_task}_{named_model}_{y_col}_{timestamp}")
trainer_obj.save_model(model_save_path)
print(f"Model checkpoint saved to: {model_save_path}")
# log experiment results
experiment_info = {
    "model_name": named_model,
    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,

```

```

        "x_col": x_col,
        "y_col": y_col,
        "layers_to_unfreeze": layers_to_unfreeze}
model_info = gather_model_details(trained_model)
all_run_metrics = gather_all_run_metrics(
    trainer=trainer_obj,
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    test_dataset=test_data_hf)
log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath)
print(f"EXPERIMENT LOGGED TO: {log_filepath}")

```

Model checkpoint saved to: /content/drive/MyDrive/266-final/models/single_bert-base-cased_binary_complexity_20250410_004230

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:

/content/drive/MyDrive/266-final/results/experiment_runs.txt

0.2.9 3.1.6 from pretrained bert-base-cased Y: multi task 2 & X: morph_sequence

```

[55]: # Define Experiment Parameters
named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = "" # modern bert
#####
regularization_weight_decay = 0.1
learning_rate = 1e-5
size_batch = 16
length_max = 256
num_epochs = 10
#####
# x_col = "sentence"
# x_col = "sentence_no_contractions"
# x_col = "pos_sequence"
# x_col = "dep_sequence"
x_col = "morph_sequence"
#####
y_col = "binary_complexity"
# y_col = "complexity"
#####

```



```

# x_task = "single"
x_task = "multi"
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
#####
# 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])
#####
custom_config = BertConfig.from_pretrained("bert-base-cased")
custom_config.hidden_act = "gelu" # alts: "relu" "silu"
custom_config.attention_probs_dropout_prob = 0.1
custom_config.hidden_dropout_prob = 0.1
custom_config.gradient_checkpointing = False
#####
model, tokenizer = get_model_and_tokenizer(
    remote_model_name="bert-base-cased",
    local_model_path=None,
    config=custom_config)
#####
# model, tokenizer = get_model_and_tokenizer(
#     remote_model_name=None

```

```

#     local_model_path="...CONFIGURE_PATH...",
#     config=custom_config)
print("=====")
print(named_model, ":")
print("=====")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters at load:", model.
    ↪num_parameters(only_trainable=True))
print("=====")
print("model lineage:", MODEL_LINEAGE)
print("=====")
#####
layers_to_unfreeze = [
    # "bert.embeddings.",
    "bert.encoder.layer.0.",
    # "bert.encoder.layer.1.",
    "bert.encoder.layer.8.",
    "bert.encoder.layer.9.",
    "bert.encoder.layer.10.",
    "bert.encoder.layer.11.",
    "bert.pooler.",
    "classifier.",
]
freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)
print(model.config)
print("=====")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
print("=====")
#####
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)

```

Map: 0%| | 0/1517 [00:00<?, ? examples/s]

Map: 0%| | 0/99 [00:00<?, ? examples/s]

Map: 0%| | 0/184 [00:00<?, ? examples/s]

Datasets prepared. Sample from train_data_hf:

```
{'labels': tensor(0), 'input_ids': tensor([ 101, 164, 117, 117, 3177,
```



```

num_trainable_parameters at load: 108311810
=====
model lineage: {'type': 'huggingface_hub', 'path': 'bert-base-cased',
'timestamp': '2025-04-10 00:43:05'}
=====
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: 36031490
=====
Experiment configuration used with this experiment:
model used: bert-base-cased
learning rate used: 1e-05
number of epochs: 10
maximum sequence length: 256
batch size used: 16
regularization value: 0.1
outcome variable: binary_complexity
task: multi
input column: morph_sequence

```

```
[56]: # 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-26-c2ee9f934517>:31: 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': 0.6641461253166199, 'eval_accuracy':
0.6363636363636364, 'eval_precision': 0.6744186046511628, 'eval_recall':
0.5686274509803921, 'eval_f1': 0.6170212765957447, 'eval_runtime': 1.4877,
'eval_samples_per_second': 66.544, 'eval_steps_per_second': 4.705, 'epoch':
10.0}
Test metrics: {'eval_loss': 0.726580798625946, 'eval_accuracy':
0.5217391304347826, 'eval_precision': 0.582089552238806, 'eval_recall':
0.3939393939393939, 'eval_f1': 0.46987951807228917, 'eval_runtime': 1.8381,
'eval_samples_per_second': 100.105, 'eval_steps_per_second': 6.529, 'epoch':
10.0}
```

```
[57]: # save model checkpoint
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
model_save_path = os.path.join(dir_models,
    f"{x_task}_{named_model}_{y_col}_{timestamp}")
trainer_obj.save_model(model_save_path)
print(f"Model checkpoint saved to: {model_save_path}")
# log experiment results
experiment_info = {
    "model_name": named_model,
```

```

    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,
    "x_col": x_col,
    "y_col": y_col,
    "layers_to_unfreeze": layers_to_unfreeze}
model_info = gather_model_details(trained_model)
all_run_metrics = gather_all_run_metrics(
    trainer=trainer_obj,
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    test_dataset=test_data_hf)
log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath)
print(f"EXPERIMENT LOGGED TO: {log_filepath}")

```

Model checkpoint saved to: /content/drive/MyDrive/266-final/models/multi_bert-base-cased_binary_complexity_20250410_004527

<IPython.core.display.HTML object>

EXPERIMENT LOGGED TO:

/content/drive/MyDrive/266-final/results/experiment_runs.txt

0.2.10 3.1.0.1 from pretrained bert-base-cased Y: single task 1 & X: sentence —

```

[58]: # Define Experiment Parameters
named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = "" # modern bert
#####
regularization_weight_decay = 0.1
learning_rate = 1e-5
size_batch = 16
length_max = 256
num_epochs = 10
#####
x_col = "sentence"
# x_col = "sentence_no_contractions"
# x_col = "pos_sequence"
# x_col = "dep_sequence"

```

```

# x_col = "morph_sequence"
#####
y_col = "binary_complexity"
# y_col = "complexity"
#####
x_task = "single"
# x_task = "multi"
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
#####
# 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])
#####
custom_config = BertConfig.from_pretrained("bert-base-cased")
custom_config.hidden_act = "gelu" # alts: "relu" "silu"
custom_config.attention_probs_dropout_prob = 0.1
custom_config.hidden_dropout_prob = 0.1
custom_config.gradient_checkpointing = False
#####
model, tokenizer = get_model_and_tokenizer(
    remote_model_name="bert-base-cased",

```

```

    local_model_path=None,
    config=custom_config)
#####
# model, tokenizer = get_model_and_tokenizer(
#     remote_model_name=None
#     local_model_path="...CONFIGURE_PATH...",
#     config=custom_config)
print("=====")
print(named_model, ":")
print("=====")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters at load:", model.
    ↪ num_parameters(only_trainable=True))
print("=====")
print("model lineage:", MODEL_LINEAGE)
print("=====")
#####
layers_to_unfreeze = [
    # "bert.embeddings.",
    "bert.encoder.layer.0.",
    # "bert.encoder.layer.1.",
    "bert.encoder.layer.8.",
    "bert.encoder.layer.9.",
    "bert.encoder.layer.10.",
    "bert.encoder.layer.11.",
    "bert.pooler.",
    "classifier.",
]
freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)
print(model.config)
print("=====")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
print("=====")
#####
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)

```

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:

[illegible]

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-base-cased :
=====
num_parameters: 108311810
num_trainable_parameters at load: 108311810
=====
model lineage: {'type': 'huggingface_hub', 'path': 'bert-base-cased',
'timestamp': '2025-04-10 00:45:40'}
=====
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: 36031490
=====
Experiment configuration used with this experiment:
model used: bert-base-cased
learning rate used: 1e-05
number of epochs: 10
maximum sequence length: 256
batch size used: 16

```

```
regularization value: 0.1
outcome variable: binary_complexity
task: single
input column: sentence
```

```
[ ]: # 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-26-c2ee9f934517>:31: 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>
```

```
[ ]: # save model checkpoint
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
model_save_path = os.path.join(dir_models,
    f"{x_task}_{named_model}_{y_col}_{timestamp}")
trainer_obj.save_model(model_save_path)
print(f"Model checkpoint saved to: {model_save_path}")
# log experiment results
experiment_info = {
    "model_name": named_model,
    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,
    "x_col": x_col,
    "y_col": y_col,
```

```

        "layers_to_unfreeze": layers_to_unfreeze}
model_info = gather_model_details(trained_model)
all_run_metrics = gather_all_run_metrics(
    trainer=trainer_obj,
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    test_dataset=test_data_hf)
log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath)
print(f"EXPERIMENT LOGGED TO: {log_filepath}")

```

0.2.11 3.1.0.2 from pretrained bert-base-cased Y: multi task 2 & X: sentence —

```

[ ]: # Define Experiment Parameters
named_model = "bert-base-cased"
# named_model = "roberta-base"
# named_model = "bert-large"
# named_model = "roberta-large"
# named_model = "" # modern bert
#####
regularization_weight_decay = 0.1
learning_rate = 1e-5
size_batch = 16
length_max = 256
num_epochs = 10
#####
x_col = "sentence"
# x_col = "sentence_no_contractions"
# x_col = "pos_sequence"
# x_col = "dep_sequence"
# x_col = "morph_sequence"
#####
y_col = "binary_complexity"
# y_col = "complexity"
#####
# x_task = "single"
x_task = "multi"
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
#####
# 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])
#####
custom_config = BertConfig.from_pretrained("bert-base-cased")
custom_config.hidden_act = "gelu" # alts: "relu" "silu"
custom_config.attention_probs_dropout_prob = 0.1
custom_config.hidden_dropout_prob = 0.1
custom_config.gradient_checkpointing = False
#####
model, tokenizer = get_model_and_tokenizer(
    remote_model_name="bert-base-cased",
    local_model_path=None,
    config=custom_config)
#####
# model, tokenizer = get_model_and_tokenizer(
#     remote_model_name=None
#     local_model_path="...CONFIGURE_PATH...",
#     config=custom_config)
print("=====")
print(named_model, ":")
print("=====")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters at load:", model.
    ↪ num_parameters(only_trainable=True))
print("=====")

```

```

print("model lineage:", MODEL_LINEAGE)
print("=====")
#####
layers_to_unfreeze = [
    # "bert.embeddings.",
    "bert.encoder.layer.0.",
    # "bert.encoder.layer.1.",
    "bert.encoder.layer.8.",
    "bert.encoder.layer.9.",
    "bert.encoder.layer.10.",
    "bert.encoder.layer.11.",
    "bert.pooler.",
    "classifier.",
]
freeze_unfreeze_layers(model, layers_to_unfreeze=layers_to_unfreeze)
print(model.config)
print("=====")
print("num_parameters:", model.num_parameters())
print("num_trainable_parameters:", model.num_parameters(only_trainable=True))
print("=====")
#####
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)

```

```

[ ]: # 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)

```

```
[ ]: # save model checkpoint
timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
model_save_path = os.path.join(dir_models,
    ↪f"{x_task}_{named_model}_{y_col}_{timestamp}")
trainer_obj.save_model(model_save_path)
print(f"Model checkpoint saved to: {model_save_path}")
# log experiment results
experiment_info = {
    "model_name": named_model,
    "learning_rate": learning_rate,
    "epochs": num_epochs,
    "batch_size": size_batch,
    "weight_decay": regularization_weight_decay,
    "x_task": x_task,
    "x_col": x_col,
    "y_col": y_col,
    "layers_to_unfreeze": layers_to_unfreeze}
model_info = gather_model_details(trained_model)
all_run_metrics = gather_all_run_metrics(
    trainer=trainer_obj,
    train_dataset=train_data_hf,
    val_dataset=val_data_hf,
    test_dataset=test_data_hf)
log_experiment_results_json(
    experiment_meta=experiment_info,
    model_details=model_info,
    run_metrics=all_run_metrics,
    log_file=log_filepath)
print(f"EXPERIMENT LOGGED TO: {log_filepath}")
```

0.2.12 3.1.7 from pretrained roberta-base Y: single task 1 & X: sentence —

[]:

[]:

[]:

0.2.13 3.1.8 from pretrained roberta-base Y: multi task 2 & X: sentence —

[]:

[]:

[]:

0.2.14 3.1.9 from pretrained roberta-base Y: single task 1 & X: sentence_no_contractions —

[]:

[]:

[]:

0.2.15 3.1.10 from pretrained roberta-base Y: multi task 2 & X: sentence_no_contractions —

[]:

[]:

[]: