06_transformer_finetune

December 15, 2025

1 Transformer Fine-tuning (Human vs AI)

Goal: Fine-tune a pretrained transformer for binary classification: **human-written vs LLM-generated text**.

- Uses the fixed `train/val/test` splits created earlier.
- Trains an end-to-end transformer classifier (not frozen).
- Reports validation and test metrics.

[1]:
```
!pip -q install transformers datasets evaluate accelerate scikit-learn pandas␣
↪numpy tqdm
```

```
                                        0.0/84.1 kB
? eta -:--:--
                                      84.1/84.1 kB 7.3
MB/s eta 0:00:00
```

[2]:
```
import os
import numpy as np
import pandas as pd
from pathlib import Path
import json
from dataclasses import dataclass
from typing import Dict, List

from sklearn.metrics import accuracy_score, f1_score, classification_report
```

[4]:
```
from google.colab import drive
drive.mount("/content/drive")
```

```
Mounted at /content/drive
```

[5]:
```
# === LOAD FIXED SPLITS (exported from baseline notebook) ===


ART_DIR = Path("/content/drive/MyDrive/artifacts/data_splits_v1")  # same␣
↪folder used in baseline
```

```
# --- load metadata ---
with open(ART_DIR / "meta.json") as f:
    meta = json.load(f)

fmt = meta["format"]
style_cols = meta["style_cols"]

# --- load datasets ---
if fmt == "parquet":
    train_df = pd.read_parquet(ART_DIR / "train_all.parquet")
    val_df   = pd.read_parquet(ART_DIR / "val_all.parquet")
    test_df  = pd.read_parquet(ART_DIR / "test_all.parquet")
else:
    train_df = pd.read_csv(ART_DIR / "train_all.csv")
    val_df   = pd.read_csv(ART_DIR / "val_all.csv")
    test_df  = pd.read_csv(ART_DIR / "test_all.csv")

# --- sanity checks (text + label + style columns) ---
required_cols = ["text", "label", "source"] + style_cols

for name, df in [("train", train_df), ("val", val_df), ("test", test_df)]:
    missing = [c for c in required_cols if c not in df.columns]
    if missing:
        raise ValueError(f"{name} split missing columns: {missing[:15]}{' ...' ␣
 ↪if len(missing) > 15 else ''}")

# --- labels as numpy arrays ---
y_train = train_df["label"].astype(int).values
y_val   = val_df["label"].astype(int).values
y_test  = test_df["label"].astype(int).values

print("Loaded splits from:", ART_DIR)
print("Format:", fmt)
print("Sizes:", len(train_df), len(val_df), len(test_df))
print("Label dist train:", np.bincount(y_train))
print("Label dist val:  ", np.bincount(y_val))
print("Label dist test: ", np.bincount(y_test))
print("Num stylometry features:", len(style_cols))
```

```
Loaded splits from: /content/drive/MyDrive/artifacts/data_splits_v1
Format: parquet
Sizes: 32615 5756 1629
Label dist train: [16677 15938]
Label dist val:   [2943 2813]
Label dist test:  [ 380 1249]
Num stylometry features: 33
```

[6]:
```
from datasets import Dataset

train_ds = Dataset.from_pandas(train_df[["text", "label"]])
val_ds   = Dataset.from_pandas(val_df[["text", "label"]])
test_ds  = Dataset.from_pandas(test_df[["text", "label"]])

print(train_ds)
```

```
Dataset({
    features: ['text', 'label'],
    num_rows: 32615
})
```

[7]:
```
from transformers import AutoTokenizer

MODEL_NAME = "distilbert-base-uncased"
MAX_LEN = 256

tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME, use_fast=True)

def tokenize_batch(batch):
    return tokenizer(
        batch["text"],
        truncation=True,
        max_length=MAX_LEN,
        padding=False,   # we'll pad dynamically in the collator
    )

train_tok = train_ds.map(tokenize_batch, batched=True, remove_columns=["text"])
val_tok   = val_ds.map(tokenize_batch, batched=True, remove_columns=["text"])
test_tok  = test_ds.map(tokenize_batch, batched=True, remove_columns=["text"])

print(" Tokenized.")
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
  warnings.warn(
```

```
tokenizer_config.json:   0%|          | 0.00/48.0 [00:00<?, ?B/s]
```

```
config.json:   0%|          | 0.00/483 [00:00<?, ?B/s]
```

```
vocab.txt:   0%|          | 0.00/232k [00:00<?, ?B/s]
tokenizer.json:   0%|          | 0.00/466k [00:00<?, ?B/s]
Map:   0%|          | 0/32615 [00:00<?, ? examples/s]
Map:   0%|          | 0/5756 [00:00<?, ? examples/s]
Map:   0%|          | 0/1629 [00:00<?, ? examples/s]
 Tokenized.
```

[8]:
```python
from transformers import DataCollatorWithPadding

data_collator = DataCollatorWithPadding(tokenizer=tokenizer)
```

[9]:
```python
from transformers import AutoModelForSequenceClassification

num_labels = 2
model = AutoModelForSequenceClassification.from_pretrained(MODEL_NAME,␣
 ↪num_labels=num_labels)
```

```
model.safetensors:   0%|          | 0.00/268M [00:00<?, ?B/s]
Some weights of DistilBertForSequenceClassification were not initialized from
the model checkpoint at distilbert-base-uncased and are newly initialized:
['classifier.bias', 'classifier.weight', 'pre_classifier.bias',
'pre_classifier.weight']
You should probably TRAIN this model on a down-stream task to be able to use it
for predictions and inference.
```

[10]:
```python
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    preds = np.argmax(logits, axis=-1)

    acc = accuracy_score(labels, preds)
    f1  = f1_score(labels, preds)
    return {"accuracy": acc, "f1": f1}
```

[11]:
```python
## Training setup

from transformers import TrainingArguments, Trainer

OUTPUT_DIR = "./artifacts/transformer_finetune/distilbert_run_v1"

training_args = TrainingArguments(
    output_dir=OUTPUT_DIR,
    # evaluation_strategy="epoch",
    # save_strategy="epoch",
    save_total_limit=2,
```

```
    # load_best_model_at_end=True,
    metric_for_best_model="f1",
    greater_is_better=True,

    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=32,
    num_train_epochs=3,
    weight_decay=0.01,

    fp16=True,  # works on most Colab GPUs; if error, set fp16=False
    logging_steps=50,
    report_to="none",
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_tok,
    eval_dataset=val_tok,
    tokenizer=tokenizer,
    data_collator=data_collator,
    compute_metrics=compute_metrics,
)
```

```
/tmp/ipython-input-724705245.py:27: FutureWarning: `tokenizer` is deprecated and
will be removed in version 5.0.0 for `Trainer.__init__`. Use `processing_class`
instead.
  trainer = Trainer(
```

[12]:
```
trainer.train()
```

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

[12]:
```
TrainOutput(global_step=6117, training_loss=0.03350536731702548,
metrics={'train_runtime': 788.4376, 'train_samples_per_second': 124.1,
'train_steps_per_second': 7.758, 'total_flos': 6480636310748160.0, 'train_loss':
0.03350536731702548, 'epoch': 3.0})
```

[15]:
```
val_metrics = trainer.evaluate(val_tok)
print("Val metrics:", val_metrics)

test_metrics = trainer.evaluate(test_tok)
print("Test metrics:", test_metrics)
```

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

```
Val metrics: {'eval_loss': 0.03343895822763443, 'eval_accuracy':
```

```
0.9939193884642112, 'eval_f1': 0.9938107869142352, 'eval_runtime': 19.5138,
'eval_samples_per_second': 294.97, 'eval_steps_per_second': 9.224, 'epoch': 3.0}
Test metrics: {'eval_loss': 2.240030288696289, 'eval_accuracy':
0.7458563535911602, 'eval_f1': 0.8503253796095445, 'eval_runtime': 3.0192,
'eval_samples_per_second': 539.544, 'eval_steps_per_second': 16.892, 'epoch':
3.0}
```

[19]:
```
print("Validation Results")
print(f"  Accuracy : {val_metrics['eval_accuracy']*100:.2f}%")
print(f"  F1-score : {val_metrics['eval_f1']*100:.2f}%")
print(f"  Loss     : {val_metrics['eval_loss']*100:.2f}%")
print()

print("Test Results")
print(f"  Accuracy : {test_metrics['eval_accuracy']*100:.2f}%")
print(f"  F1-score : {test_metrics['eval_f1']*100:.2f}%")
print(f"  Loss     : {test_metrics['eval_loss']*100:.2f}%")
```

```
Validation Results
  Accuracy : 99.39%
  F1-score : 99.38%
  Loss     : 3.34%

Test Results
  Accuracy : 74.59%
  F1-score : 85.03%
  Loss     : 224.00%
```

Interpretation:

The fine-tuned transformer achieves near-perfect performance on the validation set, indicating strong capacity to fit the training distribution. However, test accuracy drops substantially, while F1 remains relatively high. This suggests that the model generalizes well for detecting AI-generated text but struggles with human-written examples, consistent with observations from the linear probe and LSTM models.

[20]:
```
from sklearn.metrics import confusion_matrix

preds = trainer.predict(test_tok)
test_logits = preds.predictions
test_labels = preds.label_ids
test_preds = np.argmax(test_logits, axis=1)

cm = confusion_matrix(test_labels, test_preds)

cm_df = pd.DataFrame(
    cm,
    index=["Human (0)", "AI (1)"],
    columns=["Pred Human", "Pred AI"]
```

```
)

cm_df
```

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

[20]:

	Pred Human	Pred AI
Human (0)	39	341
AI (1)	73	1176

The confusion matrix shows that most errors come from human-written text being misclassified as AI, while AI-generated text is detected reliably. This explains why F1 remains high despite lower accuracy. The model strongly favors the AI class.

[21]:
```
import torch
from scipy.special import softmax

probs = softmax(test_logits, axis=1)
test_df_analysis = test_df.copy()

test_df_analysis["pred_label"] = test_preds
test_df_analysis["prob_ai"] = probs[:, 1]
test_df_analysis["correct"] = test_df_analysis["label"] ==␣
 ↪test_df_analysis["pred_label"]
```

[23]:
```
test_df_analysis[
    (test_df_analysis["label"] == 1) &
    (test_df_analysis["correct"]) &
    (test_df_analysis["prob_ai"] > 0.9)
][["text", "prob_ai"]].sample(3)
```

[23]:

	text	prob_ai
1133	Here are a few *recent/significant* examples o…	0.999955
1290	public.sp_coupling_results (id bigserial prim…	0.999644
578	A `sync def` is short for **`async def`**, whi…	0.999923

Confident AI detections: Above texts often exhibit fluent structure, neutral tone, and consistent sentence patterns, which the transformer captures effectively after fine-tuning.

[25]:
```
test_df_analysis[
    (test_df_analysis["label"] == 0) &
    (~test_df_analysis["correct"]) &
    (test_df_analysis["prob_ai"] > 0.9)
][["text", "prob_ai"]].head(3)
```

[25]:

	text	prob_ai
1	in Seoul If you want to test yourself, here's …	0.999980
2	Admissibility of solution estimators for stoch…	0.999982

```
9  as well as putting them at risk of becoming ta…  0.999987
```

Confident misclassifications (Human → AI): Above human-written examples are often formal, well-structured, or informational in tone, making them stylistically similar to LLM-generated text. This suggests the model relies heavily on surface fluency cues rather than deeper semantic intent.

[13]:

[ ]:

[ ]:

[14]:
```
# from https://gist.github.com/jonathanagustin/b67b97ef12c53a8dec27b343dca4abba
# install can take a minute

import os
# @title Convert Notebook to PDF. Save Notebook to given directory
NOTEBOOKS_DIR = "/content/drive/MyDrive/" # @param {type:"string"}
NOTEBOOK_NAME = "06_transformer_finetune.ipynb" # @param {type:"string"}
#-------------------------------------------------------------------------------#
from google.colab import drive
drive.mount("/content/drive/", force_remount=True)
NOTEBOOK_PATH = f"{NOTEBOOKS_DIR}/{NOTEBOOK_NAME}"
assert os.path.exists(NOTEBOOK_PATH), f"NOTEBOOK NOT FOUND: {NOTEBOOK_PATH}"
!apt install -y texlive-xetex texlive-fonts-recommended texlive-plain-generic >␣
 ↪/dev/null 2>&1
!apt install pandoc > /dev/null 2>&1
!jupyter nbconvert "$NOTEBOOK_PATH" --to pdf > /dev/null 2>&1
NOTEBOOK_PDF = NOTEBOOK_PATH.rsplit('.', 1)[0] + '.pdf'
assert os.path.exists(NOTEBOOK_PDF), f"ERROR MAKING PDF: {NOTEBOOK_PDF}"
print(f"PDF CREATED: {NOTEBOOK_PDF}")
```

```
Mounted at /content/drive/
```

```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call last)
/tmp/ipython-input-2546151715.py in <cell line: 0>()
     10 drive.mount("/content/drive/", force_remount=True)
     11 NOTEBOOK_PATH = f"{NOTEBOOKS_DIR}/{NOTEBOOK_NAME}"
---> 12 assert os.path.exists(NOTEBOOK_PATH), f"NOTEBOOK NOT FOUND:␣
 ↪{NOTEBOOK_PATH}"
     13 get_ipython().system('apt install -y texlive-xetex␣
 ↪texlive-fonts-recommended texlive-plain-generic > /dev/null 2>&1')
     14 get_ipython().system('apt install pandoc > /dev/null 2>&1')
```

```
AssertionError: NOTEBOOK NOT FOUND: /content/drive/MyDrive//
  ↪06_transformer_finetune.ipynb
```