05_linear_probe_pretrained_encoder

December 14, 2025

1 Linear Probe on a Pretrained Encoder

Goal: Evaluate how well a **frozen pretrained text encoder** separates human-written vs LLM-generated text using a **linear classifier** on top.

- Encoder is frozen (no fine-tuning).
- Only a lightweight classifier is trained.
- Uses the fixed `splits_v1` created in notebook 03.

[2]:
```
import numpy as np
import pandas as pd
from pathlib import Path
from tqdm.auto import tqdm
import json
from datetime import datetime
```

[3]:
```
!pip -q install sentence-transformers scikit-learn pandas numpy tqdm
```

[1]:
```
from google.colab import drive
drive.mount("/content/drive")
```

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

[4]:
```
# === LOAD FIXED SPLITS (exported from baseline notebook) ===

# from google.colab import drive
# drive.mount("/content/drive")

import json
from pathlib import Path
import pandas as pd
import numpy as np

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:
```

```python
    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
```

[5]:
```python
from sentence_transformers import SentenceTransformer
```

```
ENCODER_NAME = "sentence-transformers/all-MiniLM-L6-v2"

encoder = SentenceTransformer(ENCODER_NAME)
encoder.max_seq_length = 256
print("Loaded encoder:", ENCODER_NAME)
```

```
/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(

modules.json:   0%|          | 0.00/349 [00:00<?, ?B/s]

config_sentence_transformers.json:   0%|          | 0.00/116 [00:00<?, ?B/s]

README.md: 0.00B [00:00, ?B/s]

sentence_bert_config.json:   0%|          | 0.00/53.0 [00:00<?, ?B/s]

config.json:   0%|          | 0.00/612 [00:00<?, ?B/s]

model.safetensors:   0%|          | 0.00/90.9M [00:00<?, ?B/s]

tokenizer_config.json:   0%|          | 0.00/350 [00:00<?, ?B/s]

vocab.txt: 0.00B [00:00, ?B/s]

tokenizer.json: 0.00B [00:00, ?B/s]

special_tokens_map.json:   0%|          | 0.00/112 [00:00<?, ?B/s]

config.json:   0%|          | 0.00/190 [00:00<?, ?B/s]

Loaded encoder: sentence-transformers/all-MiniLM-L6-v2
```

[6]:

```
# Cache embeddings so we don't re-encode every time
CACHE_DIR = Path("/content/drive/MyDrive/artifacts/linear_probe/cache")
CACHE_DIR.mkdir(parents=True, exist_ok=True)

def embed_texts(texts, cache_path: Path, batch_size: int = 64):
    if cache_path.exists():
        return np.load(cache_path)
    emb = encoder.encode(
        texts,
        batch_size=batch_size,
        show_progress_bar=True,
        convert_to_numpy=True,
```

```python
        normalize_embeddings=True
    )
    np.save(cache_path, emb)
    return emb

X_train = embed_texts(train_df["text"].tolist(), CACHE_DIR / "X_train.npy")
X_val   = embed_texts(val_df["text"].tolist(),   CACHE_DIR / "X_val.npy")
X_test  = embed_texts(test_df["text"].tolist(),  CACHE_DIR / "X_test.npy")

print(" Embeddings shapes:", X_train.shape, X_val.shape, X_test.shape)
```

```
Batches:   0%|          | 0/510 [00:00<?, ?it/s]

Batches:   0%|          | 0/90 [00:00<?, ?it/s]

Batches:   0%|          | 0/26 [00:00<?, ?it/s]

 Embeddings shapes: (32615, 384) (5756, 384) (1629, 384)
```

[7]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, f1_score, classification_report

clf = LogisticRegression(max_iter=2000, n_jobs=-1)
clf.fit(X_train, y_train)

val_pred = clf.predict(X_val)
val_prob = clf.predict_proba(X_val)[:, 1]

test_pred = clf.predict(X_test)
test_prob = clf.predict_proba(X_test)[:, 1]

val_acc = accuracy_score(y_val, val_pred)
val_f1  = f1_score(y_val, val_pred)

test_acc = accuracy_score(y_test, test_pred)
test_f1  = f1_score(y_test, test_pred)

print("VAL  acc/f1:", val_acc, val_f1)
print("TEST acc/f1:", test_acc, test_f1)

report = classification_report(y_test, test_pred, output_dict=True)
report_df = pd.DataFrame(report).transpose()
report_df.round(4)
```

```
VAL  acc/f1: 0.9279013203613621 0.9256938227394808
TEST acc/f1: 0.7562922038060159 0.8566269411339834
```

[7]:

	precision	recall	f1-score	support
0	0.4220	0.1211	0.1881	380.0000

```
1                0.7803  0.9496    0.8566  1249.0000
accuracy         0.7563  0.7563    0.7563     0.7563
macro avg        0.6011  0.5353    0.5224  1629.0000
weighted avg     0.6967  0.7563    0.7007  1629.0000
```

[9]:
```python
def summarize_results(val_acc, val_f1, test_acc, test_f1):
    df = pd.DataFrame({
        "Split": ["Validation", "Test"],
        "Accuracy": [val_acc * 100, test_acc *100],
        "F1": [val_f1*100, test_f1*100],
    })
    return df.round(2)

summarize_results(val_acc, val_f1, test_acc, test_f1)
```

[9]:

	Split	Accuracy	F1
0	Validation	92.79	92.57
1	Test	75.63	85.66

The below confusion matrix shows that the linear probe correctly identifies most AI-generated texts, but frequently misclassifies human-written text as AI. This explains the high F1 for class 1 and low recall for class 0.

[10]:
```python
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, test_pred)
cm_df = pd.DataFrame(
    cm,
    index=["Human (0)", "AI (1)"],
    columns=["Pred Human", "Pred AI"]
)

cm_df
```

[10]:

	Pred Human	Pred AI
Human (0)	46	334
AI (1)	63	1186

##Qualitative Analysis

[11]:
```python
test_results = test_df.copy()
test_results["pred"] = test_pred
test_results["prob_ai"] = test_prob

false_positives = test_results[
    (test_results["label"] == 0) & (test_results["pred"] == 1)
]

false_negatives = test_results[
```

```
    (test_results["label"] == 1) & (test_results["pred"] == 0)
]
```

[12]:
```
false_positives[["text", "prob_ai"]].head(3)
```

[12]:
```
                                                text   prob_ai
1  in Seoul If you want to test yourself, here's …  0.990763
2  Admissibility of solution estimators for stoch…  0.924783
9  as well as putting them at risk of becoming ta…  0.999874
```

[13]:
```
false_negatives[["text", "prob_ai"]].head(3)
```

[13]:
```
                                                 text   prob_ai
8   PIL_AVAILABLE = False # MolScribe for image→SM…  0.496338
21  "We model salary as ( mathcal{N}(mu_k + beta c…  0.249398
24  of all trades the model predicted as violation…  0.291505
```

Observation: Many false positives (human text predicted as AI) are highly structured, neutral in tone, and lack personal context. These stylistic traits resemble LLM-generated text, causing the encoder to overgeneralize.

False negatives (AI predicted as human) often contain informal phrasing or personal language, which reduces stereotypical AI patterns.

[14]:
```
false_positives["prob_ai"].describe()
```

[14]:
```
count    334.000000
mean       0.911384
std        0.114466
min        0.502408
25%        0.878895
50%        0.963300
75%        0.989214
max        0.999973
Name: prob_ai, dtype: float64
```

[15]:
```
false_negatives["prob_ai"].describe()
```

[15]:
```
count    63.000000
mean      0.340110
std       0.129131
min       0.027534
25%       0.257143
50%       0.374614
75%       0.440502
max       0.499187
Name: prob_ai, dtype: float64
```

[ ]:

[ ]:

[ ]:

[ ]:

[16]:

```
# 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 = "05_linear_probe_pretrained_encoder.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}")
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/tmp/ipython-input-3056652466.py in <cell line: 0>()
      8␣
↪#------------------------------------------------------------------------------#
      9 from google.colab import drive
---> 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}"

/usr/local/lib/python3.12/dist-packages/google/colab/drive.py in␣
↪mount(mountpoint, force_remount, timeout_ms, readonly)
     95 def mount(mountpoint, force_remount=False, timeout_ms=120000,␣
↪readonly=False):
     96   """Mount your Google Drive at the specified mountpoint path."""
```

```
---> 97   return _mount(

     98       mountpoint,
     99       force_remount=force_remount,

/usr/local/lib/python3.12/dist-packages/google/colab/drive.py in␣
↪_mount(mountpoint, force_remount, timeout_ms, ephemeral, readonly)
    250
    251   while True:
--> 252     case = d.expect([

    253         success,
    254         prompt,

/usr/local/lib/python3.12/dist-packages/pexpect/spawnbase.py in expect(self,␣
↪pattern, timeout, searchwindowsize, async_, **kw)
    352
    353         compiled_pattern_list = self.compile_pattern_list(pattern)
--> 354         return self.expect_list(compiled_pattern_list,

    355                 timeout, searchwindowsize, async_)
    356

/usr/local/lib/python3.12/dist-packages/pexpect/spawnbase.py in␣
↪expect_list(self, pattern_list, timeout, searchwindowsize, async_, **kw)
    381             return expect_async(exp, timeout)
    382         else:
--> 383             return exp.expect_loop(timeout)
    384
    385     def expect_exact(self, pattern_list, timeout=-1, searchwindowsize=- ,

/usr/local/lib/python3.12/dist-packages/pexpect/expect.py in expect_loop(self,␣
↪timeout)
    169                 incoming = spawn.read_nonblocking(spawn.maxread, timeou )
    170                 if self.spawn.delayafterread is not None:
--> 171                     time.sleep(self.spawn.delayafterread)
    172                 idx = self.new_data(incoming)
    173                 # Keep reading until exception or return.

KeyboardInterrupt:
```