```
!pip install torch torchvision torchaudio accelerate datasets transformers scikit-learn matplotlib panda
```

```
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-packages (from torch) (2025.3.
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-pac
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packa
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-package
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib/python3.12/dist-packages (fr
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (f
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packag
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12/dist-packages (from torch) (
Requirement already satisfied: pillow!=8.3.*,>=5.3.0 in /usr/local/lib/python3.12/dist-packages (from
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.12/dist-packages (from accele
Requirement already satisfied: psutil in /usr/local/lib/python3.12/dist-packages (from accelerate) (5.
Requirement already satisfied: pyyaml in /usr/local/lib/python3.12/dist-packages (from accelerate) (6.
Requirement already satisfied: huggingface_hub>=0.21.0 in /usr/local/lib/python3.12/dist-packages (frc
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/dist-packages (from acc
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.12/dist-packages (from datase
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.12/dist-packages (from dat
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.12/dist-packages (from datas
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.12/dist-packages (from datasets)
Requirement already satisfied: xxhash in /usr/local/lib/python3.12/dist-packages (from datasets) (3.6.
Requirement already satisfied: multiprocess<0.70.17 in /usr/local/lib/python3.12/dist-packages (from d
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-packages (from tran
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/python3.12/dist-packages
Requirement already satisfied: scipy>=1.6.0 in /usr/local/lib/python3.12/dist-packages (from scikit-le
Requirement already satisfied: joblib>=1.2.0 in /usr/local/lib/python3.12/dist-packages (from scikit-l
Requirement already satisfied: threadpoolctl>=3.1.0 in /usr/local/lib/python3.12/dist-packages (from s
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.12/dist-packages (from matpl
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.12/dist-packages (from matplotli
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.12/dist-packages (from matp
Requirement already satisfied: kiwisolver>=1.3.1 in /usr/local/lib/python3.12/dist-packages (from matp
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.12/dist-packages (from matpl
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.12/dist-packages (from m
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas) (
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas)
Requirement already satisfied: aiohttp!=4.0.0a0,!=4.0.0a1 in /usr/local/lib/python3.12/dist-packages (
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from h
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateut
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (fr
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.12/dist-packages (from requests>
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from req
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.12/dist-packages (from req
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sym
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.12/dist-packages (from jinja2
Requirement already satisfied: aiohappyeyeballs>=2.5.0 in /usr/local/lib/python3.12/dist-packages (frc
Requirement already satisfied: aiosignal>=1.4.0 in /usr/local/lib/python3.12/dist-packages (from aioht
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.12/dist-packages (from aiohttp!
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.12/dist-packages (from aioh
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.12/dist-packages (from ai
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.12/dist-packages (from aioht
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.12/dist-packages (from aioh
```

```
import numpy as np
import matplotlib.pyplot as plt
import torch
#import torch.nn as nn
#import accelerate
```

```
from datasets import load_dataset

from sklearn.metrics import accuracy_score, f1_score
from collections import Counter

from transformers import BertTokenizer, BertForSequenceClassification
from transformers import Trainer, TrainingArguments, pipeline

from peft import LoraConfig, get_peft_model, TaskType

import os
os.environ["WANDB_DISABLED"] = "true"
```

```
# Check if GPU is available, becouse training on CPU is very slow!!!
print(torch.cuda.is_available())
print(torch.cuda.get_device_name(0) if torch.cuda.is_available() else "No GPU")
```

```
True
Tesla T4
```

```
dataset = load_dataset("go_emotions")

emotions = dataset["train"].features["labels"].feature.names
num_labels = len(emotions)
print(len(dataset["train"]))
print(num_labels)
print(emotions)
```

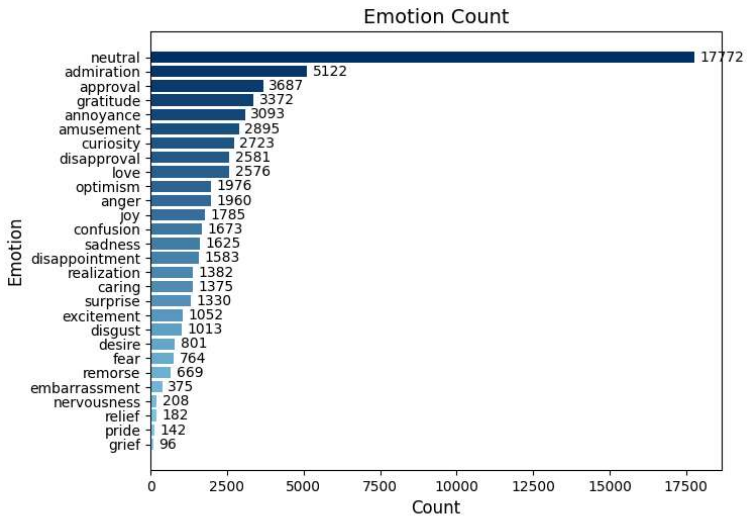
```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
README.md:       9.40k/? [00:00<00:00, 218kB/s]
simplified/train-00000-of-00001.parquet: 100%  2.77M/2.77M [00:02<00:00, 1.31MB/s]
simplified/validation-00000-of-00001.par(…): 100%  350k/350k [00:00<00:00, 576kB/s]
simplified/test-00000-of-00001.parquet: 100%  347k/347k [00:00<00:00, 632kB/s]
Generating train split: 100%  43410/43410 [00:00<00:00, 326540.50 examples/s]
Generating validation split: 100%  5426/5426 [00:00<00:00, 101983.33 examples/s]
Generating test split: 100%  5427/5427 [00:00<00:00, 117730.51 examples/s]
43410
28
['admiration', 'amusement', 'anger', 'annoyance', 'approval', 'caring', 'confusion', 'curiosity', 'desi
```

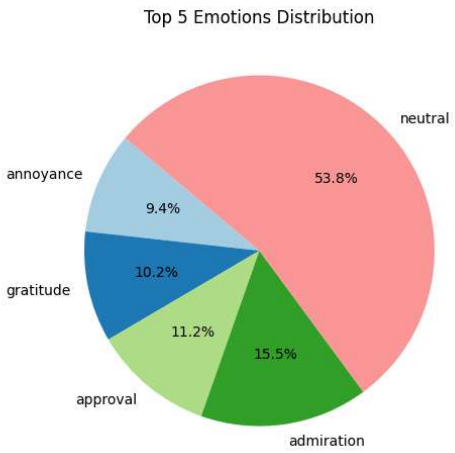
```
# Get the distribution of emotions in the dataset
emotion_counter = Counter()

for split, split_data in dataset.items():
    for sample in split_data:
        emotion_counter.update(sample["labels"])
```

```
emotions_count = {emotion: emotion_counter.get(i, 0) for i, emotion in enumerate(emotions)}

print(emotions_count)
```

```
{'admiration': 5122, 'amusement': 2895, 'anger': 1960, 'annoyance': 3093, 'approval': 3687, 'caring': 1
```

```
# --- Sort by ascending count ---
sorted_items = sorted(emotions_count.items(), key=lambda x: x[1])
emotions_sorted, counts_sorted = zip(*sorted_items)

fig, axes = plt.subplots(1, 2, figsize=(14, 6))

# --- Horizontal bar plot (left) ---
import matplotlib.colors as mcolors

# Create a colormap from darkblue to lightblue
cmap = mcolors.LinearSegmentedColormap.from_list("blue_gradient", ["#87CEEB", "#003366"])
# Generate colors from the colormap for each bar
colors = [cmap(i / (len(emotions_sorted) - 1)) for i in range(len(emotions_sorted))]
bars = axes[0].barh(emotions_sorted, counts_sorted, color=colors)
for bar, count in zip(bars, counts_sorted):
    axes[0].text(
        bar.get_width() + max(counts_sorted) * 0.01,
        bar.get_y() + bar.get_height() / 2,
        str(count),
        va='center',
        ha='left',
        fontsize=10,
        fontweight='normal'
    )
axes[0].set_xlabel("Count", fontsize=12)
axes[0].set_ylabel("Emotion", fontsize=12)
axes[0].set_title("Emotion Count", fontsize=14)

# Show total train data at the bottom right
axes[0].text(
    1.0, -0.15, f"Total train samples: {len(dataset['train'])}",
    transform=axes[0].transAxes,
    fontsize=12,
    color="navy",
    ha="right",
    va="top",
    fontweight="bold"
)

# --- Pie chart of top 5 emotions (right) ---
top5_emotions = emotions_sorted[-5:]
top5_counts = counts_sorted[-5:]
axes[1].pie(top5_counts, labels=top5_emotions, autopct='%1.1f%%', startangle=140, colors=plt.cm.Paired
axes[1].set_title("Top 5 Emotions Distribution")

plt.tight_layout(pad=2)
plt.show()
```





Total train samples: 43410

```
def multi_hot_encode(samples):
    multi_hot = np.zeros(len(emotions))

    for label in samples["labels"]:
        multi_hot[label] = 1

    return {"labels": multi_hot.tolist()}

dataset = dataset.map(multi_hot_encode)
```

```
Map: 100% 43410/43410 [00:08<00:00, 9729.81 examples/s]
Map: 100% 5426/5426 [00:00<00:00, 9315.14 examples/s]
Map: 100% 5427/5427 [00:00<00:00, 9027.01 examples/s]
```

```
model_name = 'bert-base-uncased'

tokenizer = BertTokenizer.from_pretrained(model_name)

def tokenize_function(examples):
    return tokenizer(examples["text"], padding="max_length", truncation=True, max_length=128)

tokenized_dataset = dataset.map(tokenize_function, batched=True)

# Set the format of the dataset to PyTorch tensors
tokenized_dataset.set_format("torch")
```

```
tokenizer_config.json: 100% 48.0/48.0 [00:00<00:00, 2.57kB/s]
vocab.txt: 100% 232k/232k [00:00<00:00, 4.46MB/s]
tokenizer.json: 100% 466k/466k [00:00<00:00, 2.08MB/s]
config.json: 100% 570/570 [00:00<00:00, 15.7kB/s]
Map: 100% 43410/43410 [00:35<00:00, 2818.75 examples/s]
Map: 100% 5426/5426 [00:01<00:00, 2942.17 examples/s]
Map: 100% 5427/5427 [00:02<00:00, 1890.03 examples/s]
```

```
convert_to_float = lambda sample: {"float_labels": sample["labels"].to(torch.float)}

tokenized_dataset = (tokenized_dataset
                     .map(convert_to_float, remove_columns=["labels"])
                     .rename_column("float_labels", "labels"))
```

```
Map: 100%                                                    43410/43410 [00:24<00:00, 981.13 examples/s]
Map: 100%                                                    5426/5426 [00:02<00:00, 1853.17 examples/s]
Map: 100%                                                    5427/5427 [00:02<00:00, 1947.40 examples/s]
```

```
model = BertForSequenceClassification.from_pretrained(
    model_name,
    num_labels=len(emotions),
    problem_type="multi_label_classification"
)
```

```
model.safetensors: 100%                                      440M/440M [00:08<00:00, 72.9MB/s]
Some weights of BertForSequenceClassification were not initialized from the model checkpoint at bert-ba
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inf
```

```
device = "cuda:0" if torch.cuda.is_available() else "cpu"
print(device)
model = model.to(device)
```

```
cuda:0
```

```
# Define parameters
LEARNING_RATE = 5e-5
BATCH_SIZE = 16
EPOCHS = 10
WEIGHT_DECAY = 1e-2
```

```
training_args = TrainingArguments(
    output_dir="./results",
    eval_strategy="epoch",
    learning_rate=LEARNING_RATE,
    per_device_train_batch_size=BATCH_SIZE,
    per_device_eval_batch_size=BATCH_SIZE,
    num_train_epochs=EPOCHS,
    weight_decay=WEIGHT_DECAY,
    fp16=True
)
```

```
Using the `WANDB_DISABLED` environment variable is deprecated and will be removed in v5. Use the --repo
```

```
def compute_metrics(pred):
    logits, labels = pred
    preds = (logits > 0.5).astype(int) # Convert probabilities to binary predictions

    f1 = f1_score(labels, preds, average='micro')
    acc = accuracy_score(labels, preds)

    return {
        'f1': f1,
```

```
        'accuracy': acc
    }
```

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset["train"],
    eval_dataset=tokenized_dataset["validation"],
    compute_metrics=compute_metrics
)
```

```
trainer.train()
```

[27140/27140 1:11:12, Epoch 10/10]

Epoch	Training Loss	Validation Loss	F1	Accuracy
1	0.092400	0.087820	0.483314	0.337818
2	0.077500	0.083093	0.536127	0.401032
3	0.061100	0.086912	0.542366	0.417803
4	0.048200	0.097814	0.553207	0.444895
5	0.033800	0.110940	0.551506	0.449871
6	0.023700	0.119850	0.554570	0.450240
7	0.017200	0.127860	0.566341	0.454847
8	0.012100	0.137151	0.568068	0.449502
9	0.009000	0.144258	0.572369	0.450240
10	0.006500	0.147231	0.573902	0.451345

```
TrainOutput(global_step=27140, training_loss=0.04027653253508914, metrics={'train_runtime': 4274.0708,
'train_samples_per_second': 101.566, 'train_steps_per_second': 6.35, 'total_flos':
2.85607930586112e+16, 'train_loss': 0.04027653253508914, 'epoch': 10.0})
```

```
multilabel_pipeline = pipeline(
    "text-classification",
    model=model,
    tokenizer=tokenizer,
    top_k=None,  # return all pedictions first
)
```

```
Device set to use cuda:0
```

```
#result = multilabel_pipeline("I simply can't stand when my computer freezes")
#result = multilabel_pipeline("I hate you, but i also like you")
result = multilabel_pipeline("I’m Not Mad")
result
```

```
[[{'label': 'LABEL_27', 'score': 0.9061560034751892},
  {'label': 'LABEL_10', 'score': 0.8906751871109009},
  {'label': 'LABEL_3', 'score': 0.0427221953868866},
  {'label': 'LABEL_2', 'score': 0.005179712548851967},
  {'label': 'LABEL_4', 'score': 0.0006361912237480283},
  {'label': 'LABEL_5', 'score': 0.0005172813544049859},
  {'label': 'LABEL_11', 'score': 0.0005112578510306776},
  {'label': 'LABEL_15', 'score': 0.0003997710591647774},
  {'label': 'LABEL_23', 'score': 0.000347345310728997},
  {'label': 'LABEL_0', 'score': 0.00029365462251007557},
```

```
  {'label': 'LABEL_22', 'score': 0.00026947917649522424},
  {'label': 'LABEL_6', 'score': 0.000263238325715065},
  {'label': 'LABEL_1', 'score': 0.00026119034737348557},
  {'label': 'LABEL_25', 'score': 0.00017536993254907 43},
  {'label': 'LABEL_26', 'score': 0.00017536993254907 43},
  {'label': 'LABEL_9', 'score': 0.00016219283861573786},
  {'label': 'LABEL_17', 'score': 0.0001511819864390418},
  {'label': 'LABEL_7', 'score': 0.00011866795102832839},
  {'label': 'LABEL_16', 'score': 0.00011412239109631628},
  {'label': 'LABEL_18', 'score': 0.00010804956400534138},
  {'label': 'LABEL_21', 'score': 0.00010637458035489544},
  {'label': 'LABEL_20', 'score': 0.00010071399447042495},
  {'label': 'LABEL_13', 'score': 7.086326513672248e-05},
  {'label': 'LABEL_8', 'score': 6.205049430718645e-05},
  {'label': 'LABEL_24', 'score': 6.061318345018 6625e-05},
  {'label': 'LABEL_14', 'score': 5.649793456541374e-05},
  {'label': 'LABEL_19', 'score': 3.70529705833178e-05},
  {'label': 'LABEL_12', 'score': 1.8631746570463292e-05}]]
```

```
def get_emotion(pred):
    """
    Gets the int number after the _ in "LABEL_X"
    e.g.: LABEL_18 yields just the integer 18
    We use this integer as index to get the original emotion name
    """
    pred_emotion_idx = int(pred["label"].split('_')[1])
    return emotions[pred_emotion_idx]

predicted_emotions = [get_emotion(pred) for pred in result[0] if pred["score"] > 0.1]
predicted_emotions
```

```
['neutral', 'disapproval']
```

```
# === Per-label threshold search (validation) and test evaluation ===
import numpy as np
import pandas as pd
from sklearn.metrics import f1_score, precision_recall_fscore_support, accuracy_score

def _sigmoid(x): return 1 / (1 + np.exp(-x))

def _predict_split(trainer, split_ds):
    out = trainer.predict(split_ds)
    y_true = out.label_ids
    y_score = _sigmoid(out.predictions)
    if y_score.ndim == 1: y_score = y_score.reshape(-1, 1)
    return y_true, y_score

# 1) Get validation scores
y_true_val, y_score_val = _predict_split(trainer, tokenized_dataset["validation"])
L = y_true_val.shape[1]
if "label_names" not in globals():
    label_names = [f"L{i}" for i in range(L)]

# 2) Grid-search a separate threshold for each label
grid = np.round(np.arange(0.05, 0.95 + 1e-9, 0.01), 2)
best_thr_per_label = np.zeros(L)
best_f1_per_label  = np.zeros(L)

for j in range(L):
    yj_true  = y_true_val[:, j]
    yj_score = y_score_val[:, j]
    best_f1, best_t = -1, 0.5
```

```
        for t in grid:
            yj_pred = (yj_score >= t).astype(int)
            f1 = f1_score(yj_true, yj_pred, zero_division=0)
            if f1 > best_f1:
                best_f1, best_t = f1, t
        best_thr_per_label[j] = best_t
        best_f1_per_label[j]  = best_f1

    thr_table = pd.DataFrame({
        "label": label_names,
        "best_thr": best_thr_per_label,
        "val_f1_at_best_thr": best_f1_per_label
    }).sort_values("val_f1_at_best_thr", ascending=False).reset_index(drop=True)
    display(thr_table.head(15))

    # 3) Evaluate on TEST using per-label thresholds
    y_true_test, y_score_test = _predict_split(trainer, tokenized_dataset["test"])
    y_pred_test = (y_score_test >= best_thr_per_label[None, :]).astype(int)

    # Summary metrics
    subset_acc = accuracy_score(y_true_test, y_pred_test)
    micro_f1   = f1_score(y_true_test, y_pred_test, average="micro", zero_division=0)
    macro_f1   = f1_score(y_true_test, y_pred_test, average="macro", zero_division=0)
    samples_f1 = f1_score(y_true_test, y_pred_test, average="samples", zero_division=0)

    print("=== TEST (per-label thresholds) ===")
    print(f"subset_accuracy: {subset_acc:.4f}")
    print(f"micro_f1:       {micro_f1:.4f}")
    print(f"macro_f1:       {macro_f1:.4f}")
    print(f"samples_f1:     {samples_f1:.4f}")

    # Per-label breakdown
    p, r, f, s = precision_recall_fscore_support(y_true_test, y_pred_test, average=None, zero_division=0)
    per_label = pd.DataFrame({
        "label": label_names, "precision": p, "recall": r, "f1": f, "support": s, "thr_used": best_thr_per
    }).sort_values("support", ascending=False).reset_index(drop=True)
    display(per_label.head(20))
```

	label	best_thr	val_f1_at_best_thr
0	L15	0.89	0.909884

```
# === Surface interesting errors on TEST ===
import pandas as pd
import numpy as np

# Ensure we have y_true_test/y_score_test/y_pred_test from A1 cell; otherwise recompute
if 'y_true_test' not in globals() or 'y_pred_test' not in globals():
    y_true_test, y_score_test = _predict_split(trainer, tokenized_dataset["test"])
    y_pred_test = (y_score_test >= 0.5).astype(int)

texts_test = tokenized_dataset["test"]["text"]

def labels_to_str(row):
    return ", ".join(np.array(label_names)[np.where(row==1)[0]])

df_err = pd.DataFrame({
    "text": texts_test,
    "true": [labels_to_str(y_true_test[i]) for i in range(len(texts_test))],
    "pred": [labels_to_str(y_pred_test[i]) for i in range(len(texts_test))]
})
df_err = df_err[df_err["true"] != df_err["pred"]]

print("Total misclassified samples:", len(df_err))
display(df_err.sample(8, random_state=42))
```

```
micro_f1:       0.5714
Total misclassified samples: 3282
macro_f1:       0.5029
samples_f1:     0.5880
```

	text	true	pred
2773	Bad sarcasm in response to a bad joke seems ab...	L4	L3, L10
2192	It's refreshing to not see comments with anti-...	L27	L17
3980	Honestly, fuck [NAME] after the bounty gate shit	L2, L3	L2
2764	Good luck and Godspeed.	L0	L0, L20
4202	DID YOU KNOW THAT CASHEWS COME FROM A FRUIT!?!?	L7	L7, L22
4796	A fork and spoon could be a fapoon?	L7	L27
4385	By his comment I would say that he is in fact,...	L3	L22, L27
399	Yes. Ignore and move on	L4, L5	L3

```
# === Best & worst labels by F1 (TEST) ===
from sklearn.metrics import precision_recall_fscore_support

p, r, f, s = precision_recall_fscore_support(y_true_test, y_pred_test, average=None, zero_division=0)
table = pd.DataFrame({"label": label_names, "precision": p, "recall": r, "f1": f, "support": s})
best5  = table.sort_values("f1", ascending=False).head(5)
worst5 = table.sort_values("f1", ascending=True).head(5)
print("Top-5 labels by F1:")
display(best5)
print("Bottom-5 labels by F1:")
display(worst5)
```

16	L26	0.577236	0.503546	0.537879	141	0.53
17	L5	0.415385	0.400000	0.407547	135	0.42

```
Top-5 labels by F1:
```

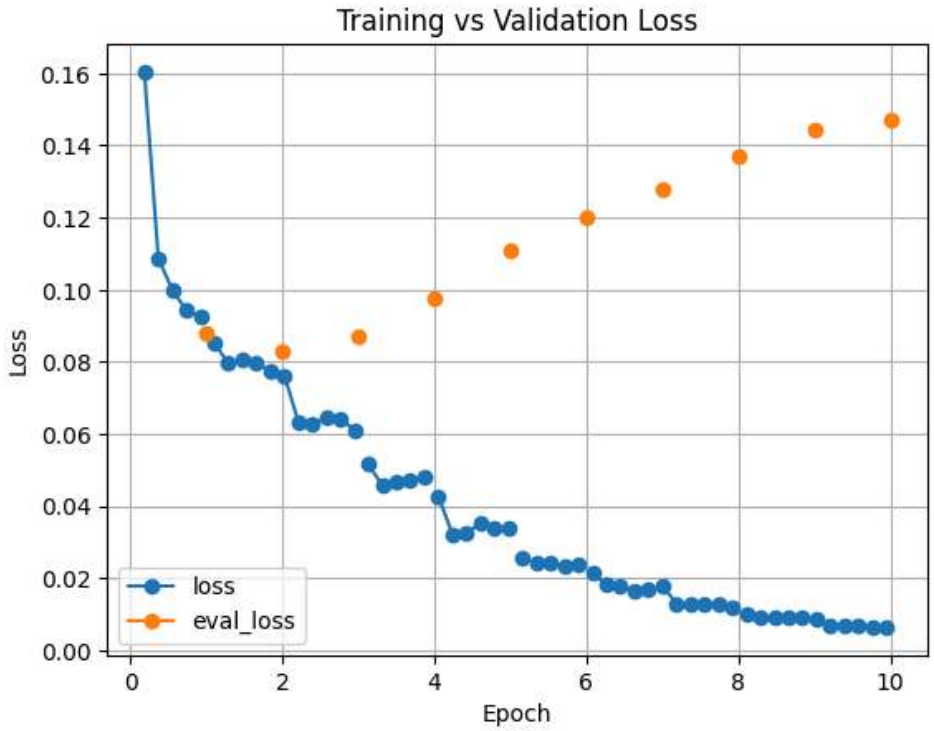
	label	precision	recall	f1	support
18	L18	0.531915	0.406504	0.460829	123
19	L13	0.560000	0.407767	0.471910	103
15	L15	0.944785	0.875000	0.908555	352
1	L1	0.777003	0.844697	0.809437	264
18	L18	0.709790	0.852941	0.774809	238
0	L0	0.656604	0.690476	0.673114	504
27	L27	0.591837	0.762731	0.666504	1787

```
Bottom-5 labels by F1:
```

	label	precision	recall	f1	support
22	L22	0.177570	0.262069	0.211699	145
9	L9	0.214047	0.423841	0.284444	151
21	L21	0.294118	0.312500	0.303030	16
12	L12	0.727273	0.216216	0.333333	37
19	L19	0.321429	0.391304	0.352941	23

```
# === Plot learning curves from Trainer logs ===
import pandas as pd
import matplotlib.pyplot as plt

logs = pd.DataFrame(trainer.state.log_history)
# Keep useful keys only
keys = ["epoch", "loss", "eval_loss", "eval_f1", "eval_accuracy"]
curves = logs[[k for k in keys if k in logs.columns]].dropna(how="all")

# Loss curve
plt.figure()
curves.plot(x="epoch", y=[c for c in ["loss","eval_loss"] if c in curves.columns], marker="o")
plt.title("Training vs Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.grid(True)
plt.show()
```

```
<Figure size 640x480 with 0 Axes>
```




```
# === Efficiency snapshot ===
import torch, numpy as np, pandas as pd

def count_params(m):
    total = sum(p.numel() for p in m.parameters())
    trainable = sum(p.numel() for p in m.parameters() if p.requires_grad)
    return total, trainable

total_p, trainable_p = count_params(model)
runtime_sec = float(getattr(trainer.state, "train_runtime", np.nan))
steps_per_sec = float(getattr(trainer.state, "train_steps_per_second", np.nan))
samples_per_sec = float(getattr(trainer.state, "train_samples_per_second", np.nan))

try:
    vram_mb = torch.cuda.max_memory_allocated() / (1024**2)
except:
    vram_mb = np.nan

eff = pd.DataFrame([{
    "strategy": "current_model",
    "total_params": total_p,
    "trainable_params": trainable_p,
    "train_runtime_s": runtime_sec,
    "steps_per_s": steps_per_sec,
    "samples_per_s": samples_per_sec,
    "approx_max_VRAM_MB": vram_mb
}])
display(eff)
```

	strategy	total_params	trainable_params	train_runtime_s	steps_per_s	samples_per_s	approx_max
0	current_model	109503772	109503772	NaN	NaN	NaN	22

```
# === Single-threshold sensitivity (VAL) ===
import numpy as np, matplotlib.pyplot as plt
from sklearn.metrics import f1_score, accuracy_score

y_true_val, y_score_val = _predict_split(trainer, tokenized_dataset["validation"])
ths = np.linspace(0.05, 0.95, 19)
micro_f1s, macros, accs = [], [], []

for t in ths:
    y_pred = (y_score_val >= t).astype(int)
    micro_f1s.append(f1_score(y_true_val, y_pred, average="micro", zero_division=0))
    macros.append(f1_score(y_true_val, y_pred, average="macro", zero_division=0))
    accs.append(accuracy_score(y_true_val, y_pred))

plt.figure()
plt.plot(ths, micro_f1s, marker="o", label="micro F1")
plt.plot(ths, macros, marker="o", label="macro F1")
plt.plot(ths, accs, marker="o", label="subset accuracy")
plt.xlabel("Threshold")
```