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

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

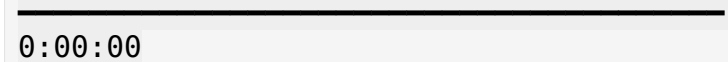
```
!pip install numpy==1.26.4 pandas==2.1.4 --force-reinstall
```

```
Collecting numpy==1.26.4
  Downloading numpy-1.26.4-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (61 kB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0.0/61.0 kB ? eta -:--:--
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 61.0/61.0 kB 2.5 MB/s eta
0:00:00
anylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (18 kB)
Collecting python-dateutil>=2.8.2 (from pandas==2.1.4)
  Downloading python_dateutil-2.9.0.post0-py2.py3-none-
any.whl.metadata (8.4 kB)
Collecting pytz>=2020.1 (from pandas==2.1.4)
  Downloading pytz-2025.2-py2.py3-none-any.whl.metadata (22 kB)
Collecting tzdata>=2022.1 (from pandas==2.1.4)
  Downloading tzdata-2025.2-py2.py3-none-any.whl.metadata (1.4 kB)
Collecting six>=1.5 (from python-dateutil>=2.8.2->pandas==2.1.4)
  Downloading six-1.17.0-py2.py3-none-any.whl.metadata (1.7 kB)
Downloading numpy-1.26.4-cp312-cp312-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl (18.0 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 18.0/18.0 MB 129.0 MB/s eta
0:00:00
anylinux_2_17_x86_64.manylinux2014_x86_64.whl (11.7 MB)
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.7/11.7 MB 140.7 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 229.9/229.9 kB 25.5 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 509.2/509.2 kB 43.4 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 347.8/347.8 kB 36.9 MB/s eta
0:00:00
py, python-dateutil, pandas
  Attempting uninstall: pytz
    Found existing installation: pytz 2025.2
    Uninstalling pytz-2025.2:
      Successfully uninstalled pytz-2025.2
  Attempting uninstall: tzdata
    Found existing installation: tzdata 2025.2
    Uninstalling tzdata-2025.2:
      Successfully uninstalled tzdata-2025.2
  Attempting uninstall: six
    Found existing installation: six 1.17.0
    Uninstalling six-1.17.0:
      Successfully uninstalled six-1.17.0
  Attempting uninstall: numpy
```

```
    Found existing installation: numpy 2.0.2
    Uninstalling numpy-2.0.2:
      Successfully uninstalled numpy-2.0.2
  Attempting uninstall: python-dateutil
    Found existing installation: python-dateutil 2.9.0.post0
    Uninstalling python-dateutil-2.9.0.post0:
      Successfully uninstalled python-dateutil-2.9.0.post0
  Attempting uninstall: pandas
    Found existing installation: pandas 2.2.2
    Uninstalling pandas-2.2.2:
      Successfully uninstalled pandas-2.2.2
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
google-colab 1.0.0 requires pandas==2.2.2, but you have pandas 2.1.4
which is incompatible.
jax 0.7.2 requires numpy>=2.0, but you have numpy 1.26.4 which is
incompatible.
plotnine 0.14.5 requires pandas>=2.2.0, but you have pandas 2.1.4
which is incompatible.
opencv-python-headless 4.12.0.88 requires numpy<2.3.0,>=2;
python_version >= "3.9", but you have numpy 1.26.4 which is
incompatible.
opencv-python 4.12.0.88 requires numpy<2.3.0,>=2; python_version >=
"3.9", but you have numpy 1.26.4 which is incompatible.
shap 0.50.0 requires numpy>=2, but you have numpy 1.26.4 which is
incompatible.
opencv-contrib-python 4.12.0.88 requires numpy<2.3.0,>=2;
python_version >= "3.9", but you have numpy 1.26.4 which is
incompatible.
mizani 0.13.5 requires pandas>=2.2.0, but you have pandas 2.1.4 which
is incompatible.
xarray 2025.11.0 requires pandas>=2.2, but you have pandas 2.1.4 which
is incompatible.
pytensor 2.35.1 requires numpy>=2.0, but you have numpy 1.26.4 which
is incompatible.
jaxlib 0.7.2 requires numpy>=2.0, but you have numpy 1.26.4 which is
incompatible.
Successfully installed numpy-1.26.4 pandas-2.1.4 python-dateutil-
2.9.0.post0 pytz-2025.2 six-1.17.0 tzdata-2025.2
```


```
{"id":"de5ebe00f3b6498c94316f07966af08d","pip_warning":{"packages":
["dateutil","numpy","pandas","pytz","six"]}}
```


```
!pip install --upgrade transformers datasets scikit-learn matplotlib
accelerate -q
!pip install numpy==1.26.4 pandas==2.2.2 scipy==1.11.4 -q
```



```
12.0/12.0 MB 122.5 MB/s eta
0:00:00
```

```
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 511.6/511.6 kB 42.1 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 9.5/9.5 MB 131.4 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 8.7/8.7 MB 152.9 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 47.7/47.7 MB 50.1 MB/s eta
0:00:00
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
plotnine 0.14.5 requires pandas>=2.2.0, but you have pandas 2.1.4
which is incompatible.
shap 0.50.0 requires numpy>=2, but you have numpy 1.26.4 which is
incompatible.
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 60.4/60.4 kB 2.5 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 12.7/12.7 MB 138.6 MB/s eta
0:00:00
━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 35.8/35.8 MB 80.6 MB/s eta
0:00:00
ERROR: pip's dependency resolver does not currently take into account
all the packages that are installed. This behaviour is the source of
the following dependency conflicts.
inequality 1.1.2 requires scipy>=1.12, but you have scipy 1.11.4 which
is incompatible.
jax 0.7.2 requires numpy>=2.0, but you have numpy 1.26.4 which is
incompatible.
jax 0.7.2 requires scipy>=1.13, but you have scipy 1.11.4 which is
incompatible.
tsfresh 0.21.1 requires scipy>=1.14.0; python_version >= "3.10", but
you have scipy 1.11.4 which is incompatible.
mapclassify 2.10.0 requires scipy>=1.12, but you have scipy 1.11.4
which is incompatible.
shap 0.50.0 requires numpy>=2, but you have numpy 1.26.4 which is
incompatible.
spopt 0.7.0 requires scipy>=1.12.0, but you have scipy 1.11.4 which is
incompatible.
pytensor 2.35.1 requires numpy>=2.0, but you have numpy 1.26.4 which
is incompatible.
esda 2.8.0 requires scipy>=1.12, but you have scipy 1.11.4 which is
incompatible.
jaxlib 0.7.2 requires numpy>=2.0, but you have numpy 1.26.4 which is
incompatible.
jaxlib 0.7.2 requires scipy>=1.13, but you have scipy 1.11.4 which is
incompatible.
```

```python
import os

WORKDIR = "/content/drive/MyDrive/nlp_proj/emotion_classifier"
```

```
os.makedirs(WORKDIR, exist_ok=True)
WORKDIR
```

```
{"type":"string"}
```

```
%cd /content
!git clone https://github.com/google-research/google-research.git
```

```
/content
Cloning into 'google-research'...
remote: Enumerating objects: 107349, done.ote: Counting objects: 100%
(127/127), done.ote: Compressing objects: 100% (121/121), done.ote:
Total 107349 (delta 37), reused 7 (delta 6), pack-reused 107222 (from
7)
```

```
DATA_DIR = "/content/google-research/goemotions/data"
os.listdir(DATA_DIR)
```

```
['emotions.txt',
 'test.tsv',
 'sentiment_dict.json',
 'train.tsv',
 'dev.tsv',
 'full_dataset',
 'ekman_mapping.json',
 'sentiment_mapping.json']
```

```
import pandas as pd

train_df = pd.read_csv(
    f"{DATA_DIR}/train.tsv",
    sep="\t",
    header=None,
    names=["text", "labels", "id"]
)

dev_df = pd.read_csv(
    f"{DATA_DIR}/dev.tsv",
    sep="\t",
    header=None,
    names=["text", "labels", "id"]
)

test_df = pd.read_csv(
    f"{DATA_DIR}/test.tsv",
    sep="\t",
    header=None,
    names=["text", "labels", "id"]
)

print(train_df.head())
```

```
                                                text labels       id
0  My favourite food is anything I didn't have to...     27  eebbqej
1  Now if he does off himself, everyone will thin...     27  ed00q6i
2                     WHY THE FUCK IS BAYLESS ISOING      2  eezlygj
3                        To make her feel threatened     14  ed7ypvh
4                             Dirty Southern Wankers      3  ed0bdzj
```

```
with open(f"{DATA_DIR}/emotions.txt") as f:
    EMOTIONS = [e.strip() for e in f.readlines()]

print(len(EMOTIONS))
print(EMOTIONS)
```

```
28
['admiration', 'amusement', 'anger', 'annoyance', 'approval',
'caring', 'confusion', 'curiosity', 'desire', 'disappointment',
'disapproval', 'disgust', 'embarrassment', 'excitement', 'fear',
'gratitude', 'grief', 'joy', 'love', 'nervousness', 'optimism',
'pride', 'realization', 'relief', 'remorse', 'sadness', 'surprise',
'neutral']
```

```
import numpy as np
import pandas as pd

def decode_label_string(label_str):
    vec = np.zeros(len(EMOTIONS), dtype=int)
    if pd.isna(label_str) or label_str == "":
        return vec

    for idx_str in str(label_str).split(","):
        idx_str = idx_str.strip()
        if idx_str == "":
            continue
        try:
            idx = int(idx_str)
        except ValueError:
            continue
        if 0 <= idx < len(EMOTIONS):
            vec[idx] = 1
    return vec

import numpy as np
import torch
import torch.nn.functional as F

EMOTION_GROUPS = {
    "joy": [
        "admiration", "amusement", "approval", "excitement",
"gratitude",
        "joy", "love", "optimism", "pride", "relief"
```

```
    ],
    "anger": [
        "anger", "annoyance", "disapproval", "disgust"
    ],
    "sadness": [
        "disappointment", "grief", "sadness", "remorse"
    ],
    "fear": [
        "fear", "nervousness"
    ],
    "surprise": [
        "surprise", "realization"
    ],
    "neutral": [
        "neutral"
    ],
}
group_names = list(EMOTION_GROUPS.keys())
NUM_GROUPED_LABELS = len(group_names)
emo_to_idx = {emo: i for i, emo in enumerate(EMOTIONS)}

def get_grouped_label_vec(fine_grained_vec, groups=EMOTION_GROUPS,
group_names=group_names, emo_to_idx=emo_to_idx):
    """
    Converts a 28-element fine-grained label vector to a 6-element
grouped label vector
    using MAX pooling for multi-label classification.
    """
    grouped_vec = np.zeros(len(group_names), dtype=int)
    for g_idx, group_name in enumerate(group_names):
        fine_grained_emos = groups[group_name]
        fg_idxs = [emo_to_idx[e] for e in fine_grained_emos if e in
emo_to_idx]

        if any(fine_grained_vec[i] == 1 for i in fg_idxs):
            grouped_vec[g_idx] = 1
    return grouped_vec


train_df["fine_grained_vec"] =
train_df["labels"].apply(decode_label_string)
dev_df["fine_grained_vec"] =
dev_df["labels"].apply(decode_label_string)
test_df["fine_grained_vec"] =
test_df["labels"].apply(decode_label_string)

train_df["label_vec"] =
train_df["fine_grained_vec"].apply(get_grouped_label_vec)
dev_df["label_vec"] =
dev_df["fine_grained_vec"].apply(get_grouped_label_vec)
```

```
test_df["label_vec"] =
test_df["fine_grained_vec"].apply(get_grouped_label_vec)

print(f"Number of new grouped labels: {NUM_GROUPED_LABELS}")
print(train_df[["text", "labels", "label_vec"]].head())
```

```
Number of new grouped labels: 6
                                                text labels  \
0  My favourite food is anything I didn't have to...     27
1  Now if he does off himself, everyone will thin...     27
2                    WHY THE FUCK IS BAYLESS ISOING      2
3                       To make her feel threatened     14
4                            Dirty Southern Wankers      3

            label_vec
0  [0, 0, 0, 0, 0, 1]
1  [0, 0, 0, 0, 0, 1]
2  [0, 1, 0, 0, 0, 0]
3  [0, 0, 0, 1, 0, 0]
4  [0, 1, 0, 0, 0, 0]
```

```
import numpy as np
np.__version__
```

```
{"type":"string"}
```

```
from transformers import BertTokenizerFast

MODEL_NAME = "bert-base-uncased"
tokenizer = BertTokenizerFast.from_pretrained(MODEL_NAME)
print("Tokenizer loaded.")
```

```
/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(
```


```
{"model_id":"ed699152d0374c41bb6c6a0a05d8c5b8","version_major":2,"vers
ion_minor":0}

{"model_id":"2316df03efd94f86ad0b896951294f56","version_major":2,"vers
ion_minor":0}

{"model_id":"cbd01f961b2e477d9e92af55d1181421","version_major":2,"vers
ion_minor":0}
```

{"model_id":"79792d9313bd431ab51cefd6fd32ea5b","version_major":2,"version_minor":0}

```
Tokenizer loaded.
```

```
MAX_LEN = 128

def tokenize_function(text_list):
    return tokenizer(
        text_list.to_list(),
        padding="max_length",
        truncation=True,
        max_length=MAX_LEN
    )

train_encodings = tokenize_function(train_df["text"])
dev_encodings   = tokenize_function(dev_df["text"])
test_encodings  = tokenize_function(test_df["text"])

print("Tokenization complete.")
```

```
Tokenization complete.
```

```
import torch
from torch.utils.data import Dataset

class GoEmotionsDataset(Dataset):
    def __init__(self, encodings, labels):
        self.encodings = encodings
        self.labels = labels

    def __getitem__(self, idx):
        item = {
            "input_ids": torch.tensor(self.encodings["input_ids"]
[idx]),
            "attention_mask":
torch.tensor(self.encodings["attention_mask"][idx]),
            "labels": torch.tensor(self.labels[idx]).float()
        }
        return item

    def __len__(self):
        return len(self.labels)

train_dataset = GoEmotionsDataset(train_encodings,
list(train_df["label_vec"]))
dev_dataset   = GoEmotionsDataset(dev_encodings,
list(dev_df["label_vec"]))
test_dataset  = GoEmotionsDataset(test_encodings,
list(test_df["label_vec"]))
```

```
print("Datasets created:")
print("Train:", len(train_dataset))
print("Dev:", len(dev_dataset))
print("Test:", len(test_dataset))
```

```
Datasets created:
Train: 43410
Dev: 5426
Test: 5427
```

```
from transformers import BertForSequenceClassification

model = BertForSequenceClassification.from_pretrained(
 MODEL_NAME,
 num_labels=NUM_GROUPED_LABELS,
 problem_type="multi_label_classification"
)

model.cuda() # Move to GPU
print("Model loaded on GPU.")
```


```
{"model_id":"360d788b6b814e1db9f0de38ac26d766","version_major":2,"vers
ion_minor":0}
```


```
Some weights of BertForSequenceClassification were not initialized
from the model checkpoint at bert-base-uncased 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.
```

```
Model loaded on GPU.
```

```
from transformers import TrainingArguments

training_args = TrainingArguments(
    output_dir="/content/drive/MyDrive/nlp_proj/goemotions_model",
    eval_strategy="epoch",
    save_strategy="epoch",
    logging_steps=100,
    num_train_epochs=5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    learning_rate=2e-5,
    load_best_model_at_end=True,
    metric_for_best_model="macro_f1",
    greater_is_better=True,
    report_to=[]
)

from sklearn.metrics import f1_score
import numpy as np
```

```
THRESHOLD = 0.3  # you can later tune this

def compute_metrics(eval_pred):
    logits, labels = eval_pred
    probs = 1 / (1 + np.exp(-logits))
    preds = (probs >= THRESHOLD).astype(int)
    macro_f1 = f1_score(labels, preds, average="macro",
zero_division=0)
    return {"macro_f1": macro_f1}

from transformers import Trainer

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=dev_dataset,
    tokenizer=tokenizer,
    compute_metrics=compute_metrics
)
```

```
/tmp/ipython-input-1087349691.py:3: FutureWarning: `tokenizer` is
deprecated and will be removed in version 5.0.0 for
`Trainer.__init__`. Use `processing_class` instead.
  trainer = Trainer(
```

```
trainer.train()
```

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

```
TrainOutput(global_step=13570, training_loss=0.16110830862204117,
metrics={'train_runtime': 1387.0037, 'train_samples_per_second':
156.488, 'train_steps_per_second': 9.784, 'total_flos':
1.42775763934464e+16, 'train_loss': 0.16110830862204117, 'epoch':
5.0})
```

```
model.save_pretrained("/content/drive/MyDrive/nlp_proj/
goemotions_model/best")
tokenizer.save_pretrained("/content/drive/MyDrive/nlp_proj/goemotions_
model/best")

print("Model saved.")
```

```
Model saved.
```

```
from transformers import BertForSequenceClassification,
BertTokenizerFast
import torch

model_path = "/content/drive/MyDrive/nlp_proj/goemotions_model/best"
```

```
model = BertForSequenceClassification.from_pretrained(
    model_path,
    problem_type="multi_label_classification"
)
tokenizer = BertTokenizerFast.from_pretrained(model_path)

model.eval()

def predict(text):
    encoding = tokenizer(
        text,
        return_tensors="pt",
        padding=True,
        truncation=True,
        max_length=128
    )

    with torch.no_grad():
        logits = model(**encoding).logits
        probs = torch.sigmoid(logits)

    return probs.squeeze().tolist()



def decode_predictions(prob_vector, threshold=0.3):
    return [EMOTIONS[i] for i, p in enumerate(prob_vector) if p >=
threshold]


dev_results = trainer.evaluate(eval_dataset=dev_dataset)
print("Dev results:", dev_results)

test_results = trainer.evaluate(eval_dataset=test_dataset)
print("Test results:", test_results)
```

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

```
Dev results: {'eval_loss': 0.2248423546552658, 'eval_macro_f1':
0.6453992183241022, 'eval_runtime': 10.0193,
'eval_samples_per_second': 541.555, 'eval_steps_per_second': 33.935,
'epoch': 5.0}
Test results: {'eval_loss': 0.22151629626750946, 'eval_macro_f1':
0.6387543747639285, 'eval_runtime': 10.2353,
'eval_samples_per_second': 530.226, 'eval_steps_per_second': 33.219,
'epoch': 5.0}
```

```
import numpy as np
from sklearn.metrics import f1_score, precision_score, recall_score
```

```python
dev_results = trainer.evaluate(eval_dataset=dev_dataset)
print("Dev results:")
for k, v in dev_results.items():
    print(f"  {k}: {v}")


test_results = trainer.evaluate(eval_dataset=test_dataset)
print("\nTest results:")
for k, v in test_results.items():
    print(f"  {k}: {v}")
```

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

Dev results:
  eval_loss: 0.2248423546552658
  eval_macro_f1: 0.6453992183241022
  eval_runtime: 10.3133
  eval_samples_per_second: 526.118
  eval_steps_per_second: 32.967
  epoch: 5.0

Test results:
  eval_loss: 0.22151629626750946
  eval_macro_f1: 0.6387543747639285
  eval_runtime: 10.3552
  eval_samples_per_second: 524.087
  eval_steps_per_second: 32.834
  epoch: 5.0
```

```python
test_output = trainer.predict(test_dataset)

logits = test_output.predictions
labels = test_output.label_ids

probs = 1 / (1 + np.exp(-logits))
```

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

```python
threshold = 0.3

# Binary predictions
preds = (probs >= threshold).astype(int)

# Global metrics
macro_f1 = f1_score(labels, preds, average="macro", zero_division=0)
micro_f1 = f1_score(labels, preds, average="micro", zero_division=0)

macro_precision = precision_score(labels, preds, average="macro",
zero_division=0)
```

```python
micro_precision = precision_score(labels, preds, average="micro",
zero_division=0)

macro_recall = recall_score(labels, preds, average="macro",
zero_division=0)
micro_recall = recall_score(labels, preds, average="micro",
zero_division=0)

print(f"Macro F1 : {macro_f1:.4f}")
print(f"Micro F1 : {micro_f1:.4f}")
print(f"Macro P  : {macro_precision:.4f}")
print(f"Micro P  : {micro_precision:.4f}")
print(f"Macro R  : {macro_recall:.4f}")
print(f"Micro R  : {micro_recall:.4f}")
```

```
Macro F1 : 0.6388
Micro F1 : 0.7076
Macro P  : 0.6099
Micro P  : 0.6528
Macro R  : 0.6820
Micro R  : 0.7725
```

```python
print("\nPer-emotion F1, Precision, Recall:")

for i, emo in enumerate(group_names):
    y_true = labels[:, i]
    y_pred = preds[:, i]

    f1  = f1_score(y_true, y_pred, zero_division=0)
    p   = precision_score(y_true, y_pred, zero_division=0)
    r   = recall_score(y_true, y_pred, zero_division=0)

    print(f"{i:2d} - {emo:15s} | F1: {f1:.3f}  P: {p:.3f}  R:
{r:.3f}")
```

```
Per-emotion F1, Precision, Recall:
 0 - joy             | F1: 0.826  P: 0.782  R: 0.876
 1 - anger           | F1: 0.608  P: 0.501  R: 0.774
 2 - sadness         | F1: 0.601  P: 0.604  R: 0.597
 3 - fear            | F1: 0.657  P: 0.632  R: 0.684
 4 - surprise        | F1: 0.450  P: 0.504  R: 0.406
 5 - neutral         | F1: 0.691  P: 0.636  R: 0.755
```

```python
def decode_group_predictions(prob_vec, threshold=0.3,
group_names=group_names):
    labels_pred = [
        group_names[i]
        for i, prob in enumerate(prob_vec)
        if prob >= threshold
```

```
    ]
    return labels_pred

def predict_emotions_grouped(text, threshold=0.3):
    """
    text: str
    returns: (predicted_group_emotions_list,
probability_vector_of_length_6)
    """
    # Tokenize and move to device
    inputs = tokenizer(text, return_tensors="pt", truncation=True,
padding=True).to(model.device)

    # Model forward pass
    with torch.no_grad():
        outputs = model(**inputs)
        logits = outputs.logits
    probs = torch.sigmoid(logits).cpu().numpy()[0]

    labels_pred = decode_group_predictions(probs, threshold=threshold)
    return labels_pred, probs.tolist()
examples = [
 "I am so happy and excited to see you!",
 "This is unacceptable and you should be ashamed of yourself.",
 "I don't feel anything in particular, it's just a regular day.",
]

for text in examples:
    labels_pred, prob_vec = predict_emotions_grouped(text,
threshold=0.3)
    print(f"Text: '{text}'")
    print("Grouped labels:", labels_pred)
    # Print the probability of the 6 groups
    prob_output = ", ".join([f"{name}: {p:.3f}" for name, p in
zip(group_names, prob_vec)])
    print("Grouped probs:", prob_output)
    print("---")
```

```
Text: 'I am so happy and excited to see you!'
Grouped labels: ['joy']
Grouped probs: joy: 0.989, anger: 0.003, sadness: 0.004, fear: 0.003,
surprise: 0.010, neutral: 0.007
---
Text: 'This is unacceptable and you should be ashamed of yourself.'
Grouped labels: ['anger']
Grouped probs: joy: 0.013, anger: 0.881, sadness: 0.023, fear: 0.007,
surprise: 0.010, neutral: 0.044
---
Text: 'I don't feel anything in particular, it's just a regular day.'
Grouped labels: ['neutral']
```

```
Grouped probs: joy: 0.127, anger: 0.163, sadness: 0.011, fear: 0.001,
surprise: 0.035, neutral: 0.722
---
```

Singlish Fine Tuning

```
import os

model_path = "/content/drive/MyDrive/nlp_proj/goemotions_model/best"
print(os.listdir(model_path))
```

```
['model.safetensors', 'tokenizer_config.json', 'config.json',
'special_tokens_map.json', 'vocab.txt', 'tokenizer.json']
```

```
from transformers import BertForSequenceClassification,
BertTokenizerFast

model_path = "/content/drive/MyDrive/nlp_proj/goemotions_model/best"

tokenizer = BertTokenizerFast.from_pretrained(model_path)
model = BertForSequenceClassification.from_pretrained(model_path)

print("Loaded base model + tokenizer.")
```

```
Loaded base model + tokenizer.
```

```
import pandas as pd
import csv

df = pd.read_csv(
    "/content/singlish_dataset.csv",
    sep=',',
    lineterminator='\n',
    skipinitialspace=True,
    quoting=csv.QUOTE_MINIMAL,
    on_bad_lines='warn'
)

print(f"DataFrame loaded successfully with {len(df)} rows.")
df.head()
```

```
DataFrame loaded successfully with 298 rows.
```


```
{"summary":"{\n  \"name\": \"df\",\n  \"rows\": 298,\n  \"fields\": [\
n    {\n      \"column\": \"row_id\",\n      \"properties\": {\n
\"dtype\": \"number\",\n        \"std\": 86,\n        \"min\": 0,\n
\"max\": 297,\n        \"num_unique_values\": 298,\n
\"samples\": [\n          159,\n          264,\n          254\n
],\n        \"semantic_type\": \"\",\n        \"description\": \"\"\n
}\n    },\n    {\n      \"column\": \"singlish_sentence\",\n
\"properties\": {\n        \"dtype\": \"string\",\n
```

\"num_unique_values\": 298,\n   \"samples\": [\n   \"Huh? You buy the new gadget already meh? Didn\\u2019t expect so fast lor.\",\n   \"Yi-lo, this deal too good to be true \\u2014 must double check lah.\",\n   \"Got lobang for free workshop tomorrow, must attend lah.\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"english_translation\",\n   \"properties\": {\n   \"dtype\": \"string\",\n   \"num_unique_values\": 298,\n   \"samples\": [\n   \"Huh? You bought the new gadget already? Didn\\u2019t expect it this fast.\",\n   \"Wow, this deal is too good to be true \\u2014 must double check.\",\n   \"Got an opportunity for a free workshop tomorrow; must attend.\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   },\n   {\n   \"column\": \"sentiment\\r\",\n   \"properties\": {\n   \"dtype\": \"category\",\n   \"num_unique_values\": 7,\n   \"samples\": [\n   \"joy \\r\",\n   \"sadness \\r\",\n   \"neutral \\r\"\n   ],\n   \"semantic_type\": \"\",\n   \"description\": \"\"\n   }\n   }\n]\n}","type":"dataframe","variable_name":"df"}

```
from sklearn.preprocessing import LabelEncoder

df.columns = df.columns.str.strip()

df["sentiment"] = df["sentiment"].astype(str).str.strip().str.lower()

valid_labels = ["joy", "anger", "sadness", "fear", "surprise",
"neutral"]

before = len(df)
df = df[df["sentiment"].isin(valid_labels)].copy()
after = len(df)
print(f"Kept {after}/{before} rows after filtering to 6 labels.")

print("Unique sentiments after cleaning:",
sorted(df["sentiment"].unique()))

le = LabelEncoder()
df["label"] = le.fit_transform(df["sentiment"])

num_labels = df["label"].nunique()
emotion_labels = list(le.classes_)

print("Num labels:", num_labels)
print("Singlish label classes:", emotion_labels)
```

```
Kept 298/298 rows after filtering to 6 labels.
Unique sentiments after cleaning: ['anger', 'fear', 'joy', 'neutral',
'sadness', 'surprise']
Num labels: 6
```

```
Singlish label classes: ['anger', 'fear', 'joy', 'neutral', 'sadness',
'surprise']

from sklearn.preprocessing import LabelEncoder

le = LabelEncoder()
df.columns = df.columns.str.strip()
df["label"] = le.fit_transform(df["sentiment"])

num_labels = df["label"].nunique()
num_labels

6

from transformers import BertConfig

model.config.num_labels = num_labels
model.classifier = torch.nn.Linear(model.config.hidden_size,
num_labels)

from sklearn.model_selection import train_test_split

train_df, val_df = train_test_split(df, test_size=0.1,
random_state=42)

from datasets import Dataset

train_dataset = Dataset.from_pandas(train_df)
val_dataset = Dataset.from_pandas(val_df)

def tokenize(batch):
    return tokenizer(
        batch["singlish_sentence"],
        truncation=True,
        padding="max_length",
        max_length=64
    )

train_dataset = train_dataset.map(tokenize, batched=True)
val_dataset   = val_dataset.map(tokenize, batched=True)

train_dataset = train_dataset.rename_column("label", "labels")
val_dataset   = val_dataset.rename_column("label", "labels")

train_dataset.set_format("torch", columns=["input_ids",
"attention_mask", "labels"])
val_dataset.set_format("torch", columns=["input_ids",
"attention_mask", "labels"])

{"model_id":"791971b502f3409db7bbb78d392f31b9","version_major":2,"vers
ion_minor":0}
```

{"model_id":"a4b4e5b1c81041219f02a699b2d1bf3d","version_major":2,"version_minor":0}

```
from transformers import TrainingArguments, Trainer

training_args = TrainingArguments(

output_dir="/content/drive/MyDrive/nlp_proj/singlish_emotion_model",
    eval_strategy="epoch",
    save_strategy="epoch",
    learning_rate=2e-5,
    per_device_train_batch_size=16,
    per_device_eval_batch_size=16,
    num_train_epochs=10,
    weight_decay=0.01,
    load_best_model_at_end=True,
    report_to="none"
)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset
)

from transformers import BertForSequenceClassification, BertConfig, DataCollatorWithPadding

config = BertConfig.from_pretrained(model_path)
config.num_labels = num_labels
config.problem_type = "single_label_classification"

model = BertForSequenceClassification.from_pretrained(model_path, config=config, ignore_mismatched_sizes=True)

# Move model to GPU
model.to("cuda")


data_collator = DataCollatorWithPadding(tokenizer=tokenizer)

trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=train_dataset,
    eval_dataset=val_dataset,
    tokenizer=tokenizer,
    data_collator=data_collator
)
```

```
trainer.train()
```

```
/tmp/ipython-input-3477996389.py:22: 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>
```

```
TrainOutput(global_step=170, training_loss=0.7486313763786765,
metrics={'train_runtime': 475.5914, 'train_samples_per_second': 5.635,
'train_steps_per_second': 0.357, 'total_flos': 881453691187200.0,
'train_loss': 0.7486313763786765, 'epoch': 10.0})
```

```
trainer.save_model("/content/drive/MyDrive/nlp_proj/
singlish_emotion_classifier")
tokenizer.save_pretrained("/content/drive/MyDrive/nlp_proj/singlish_em
otion_classifier")

print("Saved Singlish emotion classifier!")
```

```
Saved Singlish emotion classifier!
```

```
import torch
from transformers import BertTokenizerFast,
BertForSequenceClassification
import torch.nn.functional as F


model_path =
"/content/drive/MyDrive/nlp_proj/singlish_emotion_classifier"

tokenizer = BertTokenizerFast.from_pretrained(model_path)
model = BertForSequenceClassification.from_pretrained(model_path)
model.eval()

emotion_labels = list(le.classes_)


def predict_singlish_emotion(sentence):
    # tokenize
    inputs = tokenizer(sentence, return_tensors="pt", truncation=True,
padding=True)

    # model forward pass
    with torch.no_grad():
        outputs = model(**inputs)
        logits = outputs.logits

    # softmax → probs
    print(outputs)
```

```
    probs = F.softmax(logits, dim=-1)
    print(probs)
    pred_idx = probs.argmax().item()
    pred_label = emotion_labels[pred_idx]
    pred_conf = probs[0][pred_idx].item()

    return pred_label, pred_conf



sentence = "Alamak, I spill kopi on my shirt"
pred, conf = predict_singlish_emotion(sentence)

print("\nInput:", sentence)
print("Predicted Emotion:", pred)
print("Confidence:", round(conf, 4))
```

```
SequenceClassifierOutput(loss=None, logits=tensor([[-2.4741, -1.8668,
-2.8247,  0.7312, -3.1923, -1.7954]]), hidden_states=None,
attentions=None)
tensor([[0.0326, 0.0599, 0.0230, 0.8044, 0.0159, 0.0643]])

Input: Alamak, I spill kopi on my shirt
Predicted Emotion: neutral
Confidence: 0.8044
```

```
sentence = "Wah shiok lah! Today bonus come already!"
pred, conf = predict_singlish_emotion(sentence)

print("\nInput:", sentence)
print("Predicted Emotion:", pred)
print("Confidence:", round(conf, 4))
```

```
SequenceClassifierOutput(loss=None, logits=tensor([[-0.9075, -5.2540,
2.4849, -2.5152, -2.1465, -2.3609]]), hidden_states=None,
attentions=None)
tensor([[3.1771e-02, 4.1151e-04, 9.4482e-01, 6.3655e-03, 9.2037e-03,
7.4276e-03]])

Input: Wah shiok lah! Today bonus come already!
Predicted Emotion: joy
Confidence: 0.9448
```

```
sentence = "Walao eh, bus come so late again!"
pred, conf = predict_singlish_emotion(sentence)

print("\nInput:", sentence)
print("Predicted Emotion:", pred)
print("Confidence:", round(conf, 4))
```

```
SequenceClassifierOutput(loss=None, logits=tensor([[-3.3165, -5.0116,
-0.8693, -3.3005, -0.7704, -2.2292]]), hidden_states=None,
attentions=None)
tensor([[0.0339, 0.0062, 0.3920, 0.0345, 0.4328, 0.1006]])

Input: Walao eh, bus come so late again!
Predicted Emotion: sadness
Confidence: 0.4328

sentence = "Eh why you like that one, always cut queue!"
pred, conf = predict_singlish_emotion(sentence)

print("\nInput:", sentence)
print("Predicted Emotion:", pred)
print("Confidence:", round(conf, 4))

SequenceClassifierOutput(loss=None, logits=tensor([[-3.4642, -3.2679,
-3.2347, -4.9298, -3.5026, -0.6153]]), hidden_states=None,
attentions=None)
tensor([[0.0456, 0.0555, 0.0573, 0.0105, 0.0439, 0.7872]])

Input: Eh why you like that one, always cut queue!
Predicted Emotion: surprise
Confidence: 0.7872

sentence = "Steady bom pi pi, everything go smoothly today!"
pred, conf = predict_singlish_emotion(sentence)

print("\nInput:", sentence)
print("Predicted Emotion:", pred)
print("Confidence:", round(conf, 4))

SequenceClassifierOutput(loss=None, logits=tensor([[-0.1651, -4.8138,
-0.0610, -1.9200, -3.6440, -1.9054]]), hidden_states=None,
attentions=None)
tensor([[0.4002, 0.0038, 0.4442, 0.0692, 0.0123, 0.0702]])

Input: Steady bom pi pi, everything go smoothly today!
Predicted Emotion: joy
Confidence: 0.4442

sentence = "Aiyo, my phone drop and screen crack again lor."
pred, conf = predict_singlish_emotion(sentence)

print("\nInput:", sentence)
print("Predicted Emotion:", pred)
print("Confidence:", round(conf, 4))

SequenceClassifierOutput(loss=None, logits=tensor([[-1.0160, -3.8047,
-4.4495, -3.9787, -0.4188, -3.2654]]), hidden_states=None,
attentions=None)
```

```
tensor([[0.3259, 0.0200, 0.0105, 0.0168, 0.5923, 0.0344]])

Input: Aiyo, my phone drop and screen crack again lor.
Predicted Emotion: sadness
Confidence: 0.5923
```

```
sentence = "Sian half… whole day rain spoil my plan."
pred, conf = predict_singlish_emotion(sentence)

print("\nInput:", sentence)
print("Predicted Emotion:", pred)
print("Confidence:", round(conf, 4))
```

```
SequenceClassifierOutput(loss=None, logits=tensor([[-0.3184, -4.2479,
-1.4019, -2.0002,  0.0433, -3.1612]]), hidden_states=None,
attentions=None)
tensor([[0.3291, 0.0065, 0.1114, 0.0612, 0.4726, 0.0192]])

Input: Sian half… whole day rain spoil my plan.
Predicted Emotion: sadness
Confidence: 0.4726
```

```
sentence = "Har? You mean the price so cheap meh?"
pred, conf = predict_singlish_emotion(sentence)

print("\nInput:", sentence)
print("Predicted Emotion:", pred)
print("Confidence:", round(conf, 4))
```

```
SequenceClassifierOutput(loss=None, logits=tensor([[-4.0932, -3.8151,
-4.6973, -6.1229, -4.2676, -0.3270]]), hidden_states=None,
attentions=None)
tensor([[0.0213, 0.0281, 0.0116, 0.0028, 0.0179, 0.9184]])

Input: Har? You mean the price so cheap meh?
Predicted Emotion: surprise
Confidence: 0.9184
```

```
sentence = "Wah piang! Didn’t expect him to win sia!"
pred, conf = predict_singlish_emotion(sentence)

print("\nInput:", sentence)
print("Predicted Emotion:", pred)
print("Confidence:", round(conf, 4))
```

```
SequenceClassifierOutput(loss=None, logits=tensor([[-1.7249, -5.3864,
-1.5917, -5.7303, -0.1880, -0.6825]]), hidden_states=None,
attentions=None)
tensor([[0.1034, 0.0027, 0.1181, 0.0019, 0.4808, 0.2932]])
```

```
Input: Wah piang! Didn’t expect him to win sia!
Predicted Emotion: sadness
Confidence: 0.4808

sentence = "Aiyo, later boss ask me why late again then die liao."
pred, conf = predict_singlish_emotion(sentence)

print("\nInput:", sentence)
print("Predicted Emotion:", pred)
print("Confidence:", round(conf, 4))

SequenceClassifierOutput(loss=None, logits=tensor([[-0.4903, -5.6405,
-1.0083, -3.7491, -0.6896, -2.9919]]), hidden_states=None,
attentions=None)
tensor([[0.3935, 0.0023, 0.2344, 0.0151, 0.3224, 0.0323]])

Input: Aiyo, later boss ask me why late again then die liao.
Predicted Emotion: anger
Confidence: 0.3935

sentence = "Siao liao, exam tomorrow and I haven’t study."
pred, conf = predict_singlish_emotion(sentence)

print("\nInput:", sentence)
print("Predicted Emotion:", pred)
print("Confidence:", round(conf, 4))

SequenceClassifierOutput(loss=None, logits=tensor([[-1.0560, -4.9674,
-0.6477, -4.0756, -1.3820, -2.5922]]), hidden_states=None,
attentions=None)
tensor([[0.2849, 0.0057, 0.4285, 0.0139, 0.2056, 0.0613]])

Input: Siao liao, exam tomorrow and I haven’t study.
Predicted Emotion: joy
Confidence: 0.4285
```