```python
import pandas as pd
import numpy as np
import torch
from torch import nn
from transformers import AutoTokenizer, AutoModelForSequenceClassification, TrainingArguments, Trainer,
set_seed
from sklearn.model_selection import StratifiedKFold
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import f1_score
from sklearn.utils.class_weight import compute_class_weight
from scipy.optimize import minimize
from datasets import Dataset
import os
import gc
import matplotlib.pyplot as plt
import random

MODEL_NAME = "xlm-roberta-large"
MAXIMUM_SEQUENCE_LENGTH = 160
BATCH_SIZE = 32
NUMBER_OF_EPOCHS = 5
LEARNING_RATE = 1e-5
GLOBAL_RANDOM_SEED = 42
FOCAL_LOSS_GAMMA = 2.0
MINIMUM_SAMPLES_FOR_OVERSAMPLING = 250
NUMBER_OF_FOLDS = 5

device_computation_target = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Using device: {device_computation_target}")
```

```
Using device: cuda
```

```python
def set_deterministic_seed(seed_value):
    random.seed(seed_value)
    np.random.seed(seed_value)
    torch.manual_seed(seed_value)
    torch.cuda.manual_seed_all(seed_value)
    set_seed(seed_value)
    torch.backends.cudnn.deterministic = True
    torch.backends.cudnn.benchmark = False

class FocalLossTrainer(Trainer):
    def compute_loss(self, model, inputs, return_outputs=False, num_items_in_batch=None):
        labels = inputs.get("labels")
        outputs = model(**inputs)
        logits = outputs.get("logits")

        if hasattr(self, 'class_weights') and self.class_weights is not None:
             alpha_weights = self.class_weights.to(logits.device)
        else:
             alpha_weights = None

        cross_entropy_loss = nn.functional.cross_entropy(
            logits.view(-1, self.model.config.num_labels),
            labels.view(-1),
            reduction='none',
            weight=alpha_weights
        )

        probability_of_true_class = torch.exp(-cross_entropy_loss)
        focal_loss = ((1 - probability_of_true_class) ** FOCAL_LOSS_GAMMA) * cross_entropy_loss
        mean_loss = focal_loss.mean()

        return (mean_loss, outputs) if return_outputs else mean_loss

def compute_evaluation_metrics(prediction_output):
    labels = prediction_output.label_ids
    predictions = prediction_output.predictions.argmax(-1)
    macro_f1_score = f1_score(labels, predictions, average='macro')
    return {'macro_f1': macro_f1_score}

def oversample_minority_classes(dataframe, label_column_name, minimum_samples):
    class_counts = dataframe[label_column_name].value_counts()
    dataframe_chunks = [dataframe]

    for label, count in class_counts.items():
        if count < minimum_samples:
            difference = minimum_samples - count
            samples_of_class = dataframe[dataframe[label_column_name] == label]
            if len(samples_of_class) > 0:
                upsampled_chunk = samples_of_class.sample(
                    n=difference,
```

```
                    replace=True,
                    random_state=GLOBAL_RANDOM_SEED
                )
                dataframe_chunks.append(upsampled_chunk)

    return pd.concat(dataframe_chunks).sample(
        frac=1,
        random_state=GLOBAL_RANDOM_SEED
    ).reset_index(drop=True)

def optimization_objective(weights, probabilities, true_labels):
    weighted_probs = probabilities * weights
    final_predictions = np.argmax(weighted_probs, axis=1)
    score = f1_score(true_labels, final_predictions, average='macro')
    return -score

def find_best_class_weights(oof_probabilities, true_labels, num_classes):
    print("\nStarting Threshold/Weight Optimization...")

    initial_weights = np.ones(num_classes)

    bounds = [(0.01, 10.0)] * num_classes

    result = minimize(
        optimization_objective,
        initial_weights,
        args=(oof_probabilities, true_labels),
        method='Powell',
        bounds=bounds,
        tol=1e-4
    )

    best_weights = result.x
    best_score = -result.fun
    print(f"Optimization Complete. Best Validation F1: {best_score:.4f}")
    print(f"Optimized Weights: {best_weights}")

    return best_weights

def execute_training_and_prediction_pipeline(target_column_name, training_dataframe, testing_dataframe):
    print(f"\n{'='*30}\n Processing Target: {target_column_name}\n{'='*30}")

    label_encoder = LabelEncoder()
    training_dataframe['label'] = label_encoder.fit_transform(training_dataframe[target_column_name])
    number_of_labels = len(label_encoder.classes_)

    tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)

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

    testing_dataset_raw = Dataset.from_pandas(testing_dataframe[['text']])
    testing_dataset_encoded = testing_dataset_raw.map(tokenize_function, batched=True)

    stratified_k_fold = StratifiedKFold(
        n_splits=NUMBER_OF_FOLDS,
        shuffle=True,
        random_state=GLOBAL_RANDOM_SEED
    )

    accumulated_test_probabilities = np.zeros((len(testing_dataframe), number_of_labels))
    oof_probabilities = np.zeros((len(training_dataframe), number_of_labels))

    valid_folds_count = 0

    for fold_index, (train_indices, validation_indices) in
enumerate(stratified_k_fold.split(training_dataframe, training_dataframe['label'])):
        print(f"\n--- Fold {fold_index + 1}/{NUMBER_OF_FOLDS} ---")

        training_fold_dataframe = training_dataframe.iloc[train_indices]
        validation_fold_dataframe = training_dataframe.iloc[validation_indices]

        training_fold_dataframe = oversample_minority_classes(
            training_fold_dataframe,
            'label',
            MINIMUM_SAMPLES_FOR_OVERSAMPLING
        )
```

```
        class_weights_array = compute_class_weight(
            class_weight='balanced',
            classes=np.unique(training_fold_dataframe['label']),
            y=training_fold_dataframe['label']
        )
        class_weights_tensor = torch.tensor(class_weights_array, dtype=torch.float)

        training_dataset_raw = Dataset.from_pandas(training_fold_dataframe[['text', 'label']])
        validation_dataset_raw = Dataset.from_pandas(validation_fold_dataframe[['text', 'label']])

        training_dataset_encoded = training_dataset_raw.map(tokenize_function, batched=True)
        validation_dataset_encoded = validation_dataset_raw.map(tokenize_function, batched=True)

        model = AutoModelForSequenceClassification.from_pretrained(
            MODEL_NAME, num_labels=number_of_labels
        ).to(device_computation_target)

        training_arguments = TrainingArguments(
            output_dir=f"./results_{target_column_name}_fold{fold_index}",
            eval_strategy="epoch",
            save_strategy="epoch",
            learning_rate=LEARNING_RATE,
            per_device_train_batch_size=BATCH_SIZE,
            per_device_eval_batch_size=BATCH_SIZE * 2,
            num_train_epochs=NUMBER_OF_EPOCHS,
            weight_decay=0.01,
            load_best_model_at_end=True,
            metric_for_best_model="macro_f1",
            save_total_limit=1,
            fp16=True,
            report_to="none",
            label_smoothing_factor=0.0,
            seed=GLOBAL_RANDOM_SEED
        )

        trainer = FocalLossTrainer(
            model=model,
            args=training_arguments,
            train_dataset=training_dataset_encoded,
            eval_dataset=validation_dataset_encoded,
            compute_metrics=compute_evaluation_metrics,
        )
        trainer.class_weights = class_weights_tensor

        trainer.train()

        eval_metrics = trainer.evaluate()
        fold_score = eval_metrics['eval_macro_f1']

        val_preds_output = trainer.predict(validation_dataset_encoded).predictions
        val_probs = torch.nn.functional.softmax(torch.tensor(val_preds_output), dim=-1).numpy()
        oof_probabilities[validation_indices] = val_probs

        if fold_score > 0.45:
            print(f"Keeping Fold {fold_index + 1} (Score: {fold_score:.4f})")

            test_preds_output = trainer.predict(testing_dataset_encoded).predictions
            test_probs = torch.nn.functional.softmax(torch.tensor(test_preds_output), dim=-1).numpy()
            accumulated_test_probabilities += test_probs
            valid_folds_count += 1
        else:
            print(f"DISCARDING Fold {fold_index + 1} (Score: {fold_score:.4f}) - Model Collapsed")

        del model, trainer
        torch.cuda.empty_cache()
        gc.collect()

    if valid_folds_count > 0:
        average_test_probabilities = accumulated_test_probabilities / valid_folds_count
    else:
        print("CRITICAL WARNING: All folds failed! Returning raw probabilities (likely garbage).")
        average_test_probabilities = accumulated_test_probabilities

    best_weights = find_best_class_weights(
        oof_probabilities,
        training_dataframe['label'].values,
        number_of_labels
    )

    weighted_test_probs = average_test_probabilities * best_weights

    final_prediction_indices = np.argmax(weighted_test_probs, axis=1)
```

```
    final_prediction_labels = label_encoder.inverse_transform(final_prediction_indices)

    return final_prediction_labels

set_deterministic_seed(GLOBAL_RANDOM_SEED)

training_dataframe_main = pd.read_csv('train.csv')
testing_dataframe_main = pd.read_csv('test.csv')

training_dataframe_main['text'] = training_dataframe_main['text'].fillna("")
testing_dataframe_main['text'] = testing_dataframe_main['text'].fillna("")

emotion_predictions = execute_training_and_prediction_pipeline(
    'emotion',
    training_dataframe_main,
    testing_dataframe_main
)
category_predictions = execute_training_and_prediction_pipeline(
    'category',
    training_dataframe_main,
    testing_dataframe_main
)

submission_dataframe = pd.DataFrame({
    'index': testing_dataframe_main.iloc[:, 0],
    'emotion': emotion_predictions,
    'category': category_predictions
})

output_filename = 'submission_optimized_thresholds.csv'
submission_dataframe.to_csv(output_filename, index=False)
print(f"Done! Submission saved to {output_filename}")


==============================
 Processing Target: emotion
==============================

/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":"310343bd33e04bea809743726c064e4d","version_major":2,"version_minor":0}

{"model_id":"1ab6ba266b2a47b48608fc391ac10aa6","version_major":2,"version_minor":0}

{"model_id":"baf8f0c4044843f1a577456acd4d6fcc","version_major":2,"version_minor":0}

{"model_id":"263a952b969843e2b2c865880446f416","version_major":2,"version_minor":0}

{"model_id":"bab78614ee7c4cb88a6d77fed1b8c21c","version_major":2,"version_minor":0}


--- Fold 1/5 ---

{"model_id":"2e9565cf43304fee9bdf4bb9ed0e95cc","version_major":2,"version_minor":0}

{"model_id":"f16aea4a3ed447918b34924a4752975d","version_major":2,"version_minor":0}

{"model_id":"dbab1cd3dc8f438cb30f3fd5b9dff4aa","version_major":2,"version_minor":0}

Some weights of XLMRobertaForSequenceClassification were not initialized from the model checkpoint at
xlm-roberta-large and are newly initialized: ['classifier.dense.bias', 'classifier.dense.weight',
'classifier.out_proj.bias', 'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Keeping Fold 1 (Score: 0.5939)

<IPython.core.display.HTML object>


--- Fold 2/5 ---

{"model_id":"09ffdc1ec22a486680c358725bb1f44b","version_major":2,"version_minor":0}

{"model_id":"caa05c4dbfd143208ee8cbdc05837e11","version_major":2,"version_minor":0}
```

```
Some weights of XLMRobertaForSequenceClassification were not initialized from the model checkpoint at xlm-roberta-large and are newly initialized: ['classifier.dense.bias', 'classifier.dense.weight', 'classifier.out_proj.bias', 'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Keeping Fold 2 (Score: 0.5590)

<IPython.core.display.HTML object>


--- Fold 3/5 ---

{"model_id":"068b698291044386883c197f161aa043","version_major":2,"version_minor":0}

{"model_id":"8f3e353aaac9439c88a7abf5033736ba","version_major":2,"version_minor":0}

Some weights of XLMRobertaForSequenceClassification were not initialized from the model checkpoint at xlm-roberta-large and are newly initialized: ['classifier.dense.bias', 'classifier.dense.weight', 'classifier.out_proj.bias', 'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Keeping Fold 3 (Score: 0.5782)

<IPython.core.display.HTML object>


--- Fold 4/5 ---

{"model_id":"e3441b79ecf0493e979eea1d366d3ddd","version_major":2,"version_minor":0}

{"model_id":"7baa7c90dca24629ad3cc1d1e561128d","version_major":2,"version_minor":0}

Some weights of XLMRobertaForSequenceClassification were not initialized from the model checkpoint at xlm-roberta-large and are newly initialized: ['classifier.dense.bias', 'classifier.dense.weight', 'classifier.out_proj.bias', 'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Keeping Fold 4 (Score: 0.6275)

<IPython.core.display.HTML object>


--- Fold 5/5 ---

{"model_id":"647b527625a24e7fb827e2812910a3ea","version_major":2,"version_minor":0}

{"model_id":"2eb872196e2744c3aff9b38517929866","version_major":2,"version_minor":0}

Some weights of XLMRobertaForSequenceClassification were not initialized from the model checkpoint at xlm-roberta-large and are newly initialized: ['classifier.dense.bias', 'classifier.dense.weight', 'classifier.out_proj.bias', 'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Keeping Fold 5 (Score: 0.5074)

<IPython.core.display.HTML object>


Starting Threshold/Weight Optimization...
Optimization Complete. Best Validation F1: 0.6073
Optimized Weights: [3.26710038 2.34583049 6.04221147 6.40367413 2.02035235 3.49757959
 6.11166831]

==============================
 Processing Target: category
==============================
```

{"model_id":"aed0105486c3481ab6619dab3588a8e7","version_major":2,"version_minor":0}

```
--- Fold 1/5 ---
```

{"model_id":"634cc31ae5774febbbe584b11d46f7fb","version_major":2,"version_minor":0}

{"model_id":"ea7d15cf234e4e5792e13bde50b04ce5","version_major":2,"version_minor":0}

```
Some weights of XLMRobertaForSequenceClassification were not initialized from the model checkpoint at xlm-roberta-large and are newly initialized: ['classifier.dense.bias', 'classifier.dense.weight', 'classifier.out_proj.bias', 'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Keeping Fold 1 (Score: 0.8373)

<IPython.core.display.HTML object>


--- Fold 2/5 ---
```

{"model_id":"f11acd07077b4b4286050e02cf0e8178","version_major":2,"version_minor":0}

{"model_id":"950b77c0b9a04b37899ec346aa8adc16","version_major":2,"version_minor":0}

```
Some weights of XLMRobertaForSequenceClassification were not initialized from the model checkpoint at xlm-roberta-large and are newly initialized: ['classifier.dense.bias', 'classifier.dense.weight', 'classifier.out_proj.bias', 'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Keeping Fold 2 (Score: 0.8472)

<IPython.core.display.HTML object>


--- Fold 3/5 ---
```

{"model_id":"c6d58a38b58b4389a014759b84c13d04","version_major":2,"version_minor":0}

{"model_id":"868ba5daf0134fa580adcb14e8ffe2ae","version_major":2,"version_minor":0}

```
Some weights of XLMRobertaForSequenceClassification were not initialized from the model checkpoint at xlm-roberta-large and are newly initialized: ['classifier.dense.bias', 'classifier.dense.weight', 'classifier.out_proj.bias', 'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Keeping Fold 3 (Score: 0.8226)

<IPython.core.display.HTML object>


--- Fold 4/5 ---
```

{"model_id":"1490b99299074647830a4ce22616613e","version_major":2,"version_minor":0}

{"model_id":"b27054c836a2471fa4e8198cc253c090","version_major":2,"version_minor":0}

```
Some weights of XLMRobertaForSequenceClassification were not initialized from the model checkpoint at xlm-roberta-large and are newly initialized: ['classifier.dense.bias', 'classifier.dense.weight', 'classifier.out_proj.bias', 'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and inference.

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Keeping Fold 4 (Score: 0.8361)

<IPython.core.display.HTML object>


--- Fold 5/5 ---
```

{"model_id":"3140685fced749bc8f53fcd1c79b4db3","version_major":2,"version_minor":0}

{"model_id":"8fae584e5ce74a098eef148d9069613b","version_major":2,"version_minor":0}

```
Some weights of XLMRobertaForSequenceClassification were not initialized from the model checkpoint at
xlm-roberta-large and are newly initialized: ['classifier.dense.bias', 'classifier.dense.weight',
'classifier.out_proj.bias', 'classifier.out_proj.weight']
You should probably TRAIN this model on a down-stream task to be able to use it for predictions and
inference.

<IPython.core.display.HTML object>

<IPython.core.display.HTML object>

Keeping Fold 5 (Score: 0.8376)

<IPython.core.display.HTML object>


Starting Threshold/Weight Optimization...
Optimization Complete. Best Validation F1: 0.8558
Optimized Weights: [5.56787027 8.54260477 2.5496499  2.74324404 1.01610507]
Done! Submission saved to submission_optimized_thresholds.csv
```