```
!pip install torchmetrics
!pip install portalocker
!pip install torcheval
```

```
Requirement already satisfied: torchmetrics in /usr/local/lib/python3.10/dist-packages (1.2.1)
Requirement already satisfied: numpy>1.20.0 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (1.23.5)
Requirement already satisfied: packaging>17.1 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (23.2)
Requirement already satisfied: torch>=1.8.1 in /usr/local/lib/python3.10/dist-packages (from torchmetrics) (2.1.0+cu118
Requirement already satisfied: lightning-utilities>=0.8.0 in /usr/local/lib/python3.10/dist-packages (from torchmetrics
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-packages (from lightning-utilities>=0.8.0->
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from lightning-utilities>=
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torchmetrics) (3
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torchmetrics) (1.12
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torchmetrics) (3
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torchmetrics) (3.1
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torchmetrics) (202
Requirement already satisfied: triton==2.1.0 in /usr/local/lib/python3.10/dist-packages (from torch>=1.8.1->torchmetric
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/dist-packages (from jinja2->torch>=1.8.1->t
Requirement already satisfied: mpmath>=0.19 in /usr/local/lib/python3.10/dist-packages (from sympy->torch>=1.8.1->torch
Requirement already satisfied: portalocker in /usr/local/lib/python3.10/dist-packages (2.8.2)
Requirement already satisfied: torcheval in /usr/local/lib/python3.10/dist-packages (0.0.7)
Requirement already satisfied: typing-extensions in /usr/local/lib/python3.10/dist-packages (from torcheval) (4.5.0)
```

```
from torchtext.models import T5_BASE_GENERATION
from torchtext.prototype.generate import GenerationUtils
from torchtext.datasets import IMDB
from torchvision.transforms import ToTensor
from functools import partial
from torch.utils.data import DataLoader
import torch
from torch import tensor
from torchmetrics.classification import BinaryCalibrationError
from torchmetrics.classification import MulticlassCalibrationError
from sklearn.isotonic import IsotonicRegression
from torcheval.metrics.functional import multiclass_f1_score
```

```
def batch_prefix(task, x):
    return {
        "article": [f'{task}: ' + y for y in x["article"]],
        "abstract": x["abstract"]
}

def apply_prefix(task, x):
    return f"{task}: " + x[0], x[1]


def process_labels(labels, x):
      return x[1], labels[str(x[0])]


imdb_batch_size = 64
imdb_datapipe = IMDB(split="test")
task = "sst2 sentence"
labels = {"1": "negative", "2": "positive"}


# imdb_datapipe = imdb_datapipe.map(partial(process_labels, labels))
# imdb_datapipe = imdb_datapipe.map(partial(apply_prefix, task))
# imdb_datapipe = imdb_datapipe.batch(imdb_batch_size)
# imdb_datapipe = imdb_datapipe.rows2columnar(["text", "label"])
# imdb_dataloader = DataLoader(imdb_datapipe, batch_size=None, shuffle=True)

test_data = IMDB(
    split="test"
)
imdb_dataloader = DataLoader(test_data, batch_size=imdb_batch_size, shuffle=True)
```

```
t5_base = T5_BASE_GENERATION
transform = t5_base.transform()
model = t5_base.get_model(freeze_model=True)
model.eval()

T5_POSITIVE_LOGITS = 1465
T5_NEGATIVE_LOGITS = 2841

# sequence_generator = GenerationUtils(model)

padding_idx = 0
eos_idx = 1
max_seq_len = 512
```

```
    targets = None
    logits = None

    data_count = 0

    for batch in iter(imdb_dataloader):
        data_count += 1
        print(imdb_batch_size * data_count)

        # Datapipe Implementation:
        # input_text = batch["text"]
        # target = batch["label"]


        # Direct DataLoader Implementation:
        target_tensor = torch.zeros(len(batch[0]), 2)
        for i, data in enumerate(batch[0]):
            if(data == 1):
                target_tensor[i, :] = torch.tensor([1.0, 0.0])
            else:
                target_tensor[i, :] = torch.tensor([0.0, 1.0])

        if targets == None:
            targets = target_tensor
        else:
            targets = torch.cat((targets, target_tensor), dim = 0)

        input_text = list(batch[1])

        model_input = transform(input_text)
        temp = model(model_input)

        pos_logit = temp["decoder_output"][:,:,T5_POSITIVE_LOGITS]
        neg_logit = temp["decoder_output"][:,:,T5_NEGATIVE_LOGITS]

        z = torch.zeros(neg_logit.shape[0], 2)
        # Mention adaptation in report
        z[:, 0] += neg_logit[:, 0] / (neg_logit[:, 0] + pos_logit[:, 0])
        z[:, 1] += pos_logit[:, 0] / (neg_logit[:, 0] + pos_logit[:, 0])
```

```
        if logits == None:
            logits = z
        else:
            logits = torch.cat((logits, z), dim = 0)

        # if logits == None:
        #   logits = logits
        # else:
        #   targets = torch.cat((targets, target), dim = 0)

        # beam_size = 1
        # model_output = sequence_generator.generate(model_input, eos_idx=eos_idx, num_beams=beam_size)
        # output_text = transform.decode(model_output.tolist())

        # if(data_count * imdb_batch_size >= 88):
        #   break

        # print(logits.shape)
        # print(logits)
```

```
12480
```

```
metric = BinaryCalibrationError()
preds = torch.argmax(targets, dim=1)
calibration_error = metric(logits, targets)
f1_score = multiclass_f1_score(logits, preds, num_classes = 2)

print("Histogram Binning ECE:", str(round(100 * calibration_error.item(), 1)) + "%")
print("Histogram Binning F1 Score:", str(round(f1_score.item(), 4)))

# print(preds.shape)
targets_1d = torch.argmax(targets, dim=1)
# print(targets_1d.shape)
print(logits.shape)
train_size = 8 * targets_1d.shape[0] // 10
iso_logits = torch.zeros((logits.shape[0]))
arg_logits = torch.argmax(logits, dim = 1)
for i in range(logits.shape[0]):
    if(arg_logits[i] == 0):
        iso_logits[i] += logits[i, 0]
    else:
        iso_logits[i] -= logits[i, 1]
iso_model = IsotonicRegression().fit(iso_logits[0:train_size], targets_1d[0:train_size])
calibrated_preds = torch.Tensor(iso_model.predict(iso_logits[train_size:]))
calibration_error = metric(calibrated_preds, targets_1d[train_size:])
f1_score =  multiclass_f1_score(calibrated_preds, targets_1d[train_size:], num_classes = 2)

print("Isotonic Regression ECE:", str(100 * calibration_error.item()) + "%")
print("Isotonic Regression F1 Score:", str(round(f1_score.item(), 4)))

# Result of temperature optimization on local device
T = 0.8576
temperature_logits = logits / T

calibration_error = metric(temperature_logits, targets)
f1_score = multiclass_f1_score(temperature_logits, preds, num_classes = 2)

print("Temperature Scaling ECE:", str(round(100 * calibration_error.item(), 1)) + "%")
print("Temperature Scaling F1 Score:", str(round(f1_score.item(), 4)))
```