Inference: 2-stage-training

The core idea of two-stage training is to train on the entire dataset, then filter the data based on the kl scatter of the first training result, and train again in the second stage on the filtered (here when the threshold is 9 we can get the best result) data, because we have sampled the dataset, and we'd like to have more useful data being trained.

- First stage: training with the full data
- Second stage: training with samples (KL Loss < 9).

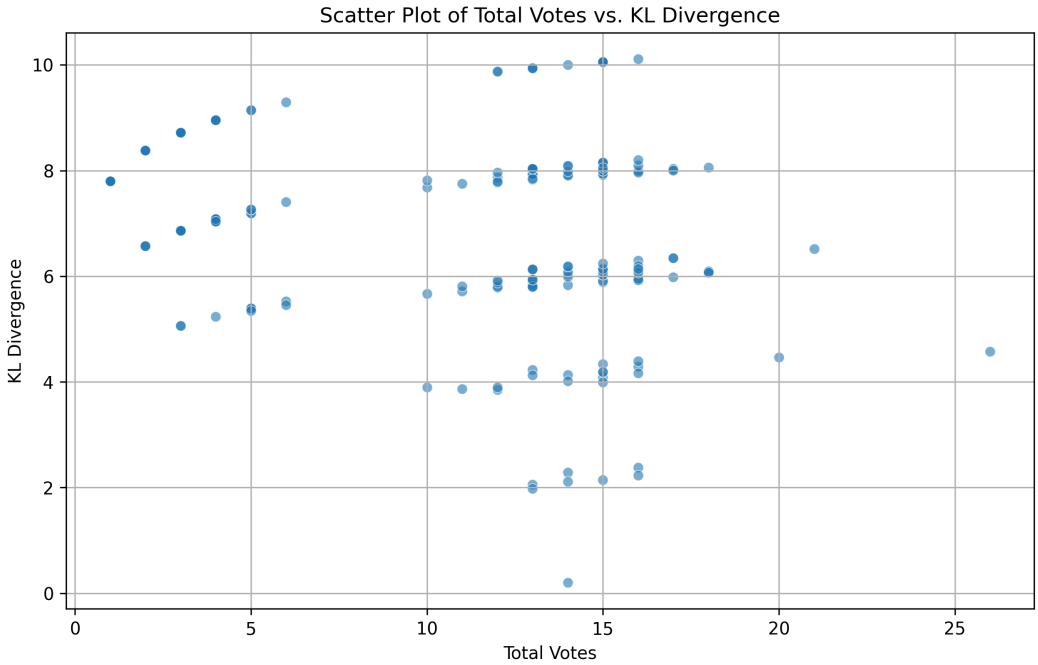
Core Code:

```
# ----------based on kl loss--------------------
if stage == 1:
    # all data
    print("Training Stage 1: Using all data")
elif stage == 2:
    # KL Loss < 9 data
    print("Training Stage 2: Filtering data based on KL Loss
< 9")
    train_folds = train_folds[train_folds['kl_loss'] < 9]

train_dataset = CustomDataset(train_folds, config, mode="trai
n", augment=True, specs=all_spectrograms, eeg_specs=all_eegs,
wavelets_spectrograms = all_wavelet_spectrograms )

valid_dataset = CustomDataset(valid_folds, config, mode="trai
n", augment=False, specs=all_spectrograms, eeg_specs=all_eeg
s,wavelets_spectrograms = all_wavelet_spectrograms)

...
...
```

```
# the second stage uses the best model obtained from the firs
t stage
if stage == 2:
    model_path = paths.OUTPUT_DIR / f"{config.MODEL.replace
('/', '_')}_fold_{fold}_stage_1_best.pth"
    model.load_state_dict(torch.load(model_path)["model"])
    model.to(device)


# I added one more column of kl_loss into the data
def compute_kl_divergence(data, label_cols):
    labels = data[label_cols].values + 1e-5
    labels /= labels.sum(axis=1, keepdims=True)
    kl_div = torch.nn.functional.kl_div(
        torch.log(torch.tensor(labels, dtype=torch.float)),
        torch.tensor([[1/6] * 6], dtype=torch.float),
        reduction='none'
    ).sum(dim=1).numpy()
    return kl_div
```

Why use two-stage training?

MOTIVATION: Based on the observation of the results of training on the entire dataset, samples with peak distributions cause the model to make predictions with exaggerated confidence, which leads to performance degradation.



Methodology: two-stage training method (first stage using full data and then fine-tuning the model using samples with kl loss<9.

Kullback-Leibler Divergence：

KL Scatter calculates the difference between two probability distributions. If you have two probability distributions P and Q, where P usually represents the true distribution and Q represents the distribution predicted or assumed by the model, then the KL scatter tells you how much information is lost when you approximate P with Q.

$$D_{KL}(P \parallel Q) = \sum_i P(i) \log \left(\frac{P(i)}{Q(i)} \right)$$

P(x) is the probability of observing event x in distribution P, while $Q(x)$ is the probability of observing event x in distribution Q. This formula calculates how many extra bits (or units of information) you would produce, on average, if you used the distribution Q to simulate the real distribution P.

Results

OVERALL KL LOSS CV: 1.2792(base) - 1.2527 (Inference)

Fold	Stage 1 KL LOSS	Stage 2 KL LOSS
0	1.2593	1.2543 ↑
1	1.2588	1.2518 ↑
2	1.1740	1.1641 ↑
3	1.3382	1.3233 ↑

We can observe that the results for each fold of training, after a second stage of more refined training, have improved

Parameter setting

```
AMP = True
BATCH_SIZE_TRAIN = 8
BATCH_SIZE_VALID = 8
EPOCHS = 16
FOLDS = 4
FREEZE = False
GRADIENT_ACCUMULATION_STEPS = 1
MAX_GRAD_NORM = 1e7
MODEL = "tf_efficientnet_b0"
NUM_FROZEN_LAYERS = 39
NUM_WORKERS = 0 # multiprocessing.cpu_count()
PRINT_FREQ = 20
SEED = 20
TRAIN_FULL_DATA = False
VISUALIZE = False
WEIGHT_DECAY = 0.01
```

Reference

HMS - Harmful Brain Activity Classification | Kaggle