Strategy Pivot Grouping

```
import os
import numpy as np
import pandas as pd
import torch
import torch.nn as nn
import torch.optim as optim
import librosa
from tqdm import tqdm
from transformers import Wav2Vec2Model, Wav2Vec2Processor
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score, f1_score, classification_report, confusion_matrix
import seaborn as sns
import matplotlib.pyplot as plt

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print("Device:", device)
```

```
Device: cuda
```

```
def merge_age(age):
    if age in ["teens", "twenties"]:
        return "young"
    elif age in ["thirties", "fourties", "fifties"]:
        return "middle"
    else:
        return "senior"

df = df[df["age"].notna()].copy()
df["age_group"] = df["age"].apply(merge_age)
```

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

num_classes = len(le.classes_)
print("Classes:", le.classes_)
```

```
Classes: ['middle' 'senior' 'young']
```

```
speakers = df["client_id"].unique()

train_spk, val_spk = train_test_split(
    speakers,
    test_size=0.2,
    random_state=42
)

train_df = df[df["client_id"].isin(train_spk)].copy()
val_df = df[df["client_id"].isin(val_spk)].copy()

print("Train:", len(train_df), "Val:", len(val_df))
```

```
Train: 44896 Val: 7988
```

```
processor = Wav2Vec2Processor.from_pretrained("facebook/wav2vec2-base")
wav2vec = Wav2Vec2Model.from_pretrained("facebook/wav2vec2-base")
wav2vec = wav2vec.to(device)
wav2vec.eval()
```

```
Loading weights: 100%                                                    211/211 [00:00<00:00, 595.80it/s, Materializing param=masked_spec_embed]
Wav2Vec2Model LOAD REPORT from: facebook/wav2vec2-base
Key                           | Status     |  |
------------------------------+------------+--+-
quantizer.weight_proj.weight | UNEXPECTED |  |
project_hid.weight           | UNEXPECTED |  |
project_q.weight             | UNEXPECTED |  |
project_q.bias               | UNEXPECTED |  |
project_hid.bias             | UNEXPECTED |  |
quantizer.weight_proj.bias   | UNEXPECTED |  |
quantizer.codevectors        | UNEXPECTED |  |

Notes:
- UNEXPECTED    :can be ignored when loading from different task/architecture; not ok if you expect identical arch.
Wav2Vec2Model(
  (feature_extractor): Wav2Vec2FeatureEncoder(
    (conv_layers): ModuleList(
      (0): Wav2Vec2GroupNormConvLayer(
        (conv): Conv1d(1, 512, kernel_size=(10,), stride=(5,), bias=False)
        (activation): GELUActivation()
        (layer_norm): GroupNorm(512, 512, eps=1e-05, affine=True)
      )
      (1-4): 4 x Wav2Vec2NoLayerNormConvLayer(
        (conv): Conv1d(512, 512, kernel_size=(3,), stride=(2,), bias=False)
        (activation): GELUActivation()
      )
      (5-6): 2 x Wav2Vec2NoLayerNormConvLayer(
        (conv): Conv1d(512, 512, kernel_size=(2,), stride=(2,), bias=False)
        (activation): GELUActivation()
      )
    )
  )
  (feature_projection): Wav2Vec2FeatureProjection(
    (layer_norm): LayerNorm((512,), eps=1e-05, elementwise_affine=True)
    (projection): Linear(in_features=512, out_features=768, bias=True)
    (dropout): Dropout(p=0.1, inplace=False)
  )
  (encoder): Wav2Vec2Encoder(
    (pos_conv_embed): Wav2Vec2PositionalConvEmbedding(
      (conv): ParametrizedConv1d(
        768, 768, kernel_size=(128,), stride=(1,), padding=(64,), groups=16
        (parametrizations): ModuleDict(
          (weight): ParametrizationList(
            (0): _WeightNorm()
          )
        )
      )
      (padding): Wav2Vec2SamePadLayer()
      (activation): GELUActivation()
    )
    (layer_norm): LayerNorm((768,), eps=1e-05, elementwise_affine=True)
    (dropout): Dropout(p=0.1, inplace=False)
    (layers): ModuleList(
      (0-11): 12 x Wav2Vec2EncoderLayer(
        (attention): Wav2Vec2Attention(
          (k_proj): Linear(in_features=768, out_features=768, bias=True)
          (v_proj): Linear(in_features=768, out_features=768, bias=True)
```

```
def extract_w2v_batched(dataframe, batch_size=8):
    X = []
    y = []

    paths = dataframe["full_path"].values
    labels = dataframe["label"].values

    for i in tqdm(range(0, len(paths), batch_size)):
        batch_paths = paths[i:i+batch_size]
        batch_labels = labels[i:i+batch_size]

        signals = []
        for p in batch_paths:
            signal, sr = librosa.load(p, sr=16000)
            signals.append(signal)

        inputs = processor(
            signals,
            sampling_rate=16000,
            return_tensors="pt",
            padding=True
        )
```

```
        with torch.no_grad():
            outputs = wav2vec(inputs.input_values.to(device))
            hidden = outputs.last_hidden_state

        pooled = hidden.mean(dim=1)

        X.append(pooled.cpu().numpy())
        y.extend(batch_labels)

    return np.vstack(X), np.array(y)
```

```
print("Extracting train embeddings...")
X_train_w2v, y_train_w2v = extract_w2v_batched(train_df, batch_size=8)

print("Extracting val embeddings...")
X_val_w2v, y_val_w2v = extract_w2v_batched(val_df, batch_size=8)
```

```
Extracting train embeddings...
100%|██████████| 5612/5612 [31:25<00:00,  2.98it/s]
Extracting val embeddings...
100%|██████████| 999/999 [05:30<00:00,  3.02it/s]
```

```
from torch.utils.data import TensorDataset, DataLoader

train_dataset = TensorDataset(
    torch.tensor(X_train_w2v, dtype=torch.float32),
    torch.tensor(y_train_w2v)
)

val_dataset = TensorDataset(
    torch.tensor(X_val_w2v, dtype=torch.float32),
    torch.tensor(y_val_w2v)
)

train_loader = DataLoader(train_dataset, batch_size=64, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=64)
```

```
class MLP(nn.Module):
    def __init__(self, input_dim, num_classes):
        super().__init__()

        self.net = nn.Sequential(
            nn.Linear(input_dim, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(),
            nn.Dropout(0.4),

            nn.Linear(512, 256),
            nn.BatchNorm1d(256),
            nn.ReLU(),
            nn.Dropout(0.3),

            nn.Linear(256, 128),
            nn.ReLU(),
            nn.Dropout(0.2),

            nn.Linear(128, num_classes)
        )

    def forward(self, x):
        return self.net(x)

model = MLP(input_dim=768, num_classes=num_classes).to(device)
```

```
class_counts = np.bincount(y_train_w2v)
class_weights = 1.0 / (class_counts + 1e-6)
class_weights = class_weights / class_weights.sum()

weights = torch.tensor(class_weights, dtype=torch.float32).to(device)

criterion = nn.CrossEntropyLoss(weight=weights)

optimizer = optim.AdamW(
    model.parameters(),
    lr=5e-4,
```

```
    weight_decay=1e-4
)
```

```
EPOCHS = 20

for epoch in range(EPOCHS):

    model.train()
    train_preds, train_labels = [], []

    for x, y in train_loader:
        x, y = x.to(device), y.to(device)

        optimizer.zero_grad()
        outputs = model(x)
        loss = criterion(outputs, y)

        loss.backward()
        optimizer.step()

        train_preds.extend(outputs.argmax(1).cpu().numpy())
        train_labels.extend(y.cpu().numpy())

    train_acc = accuracy_score(train_labels, train_preds)

    model.eval()
    val_preds, val_labels = [], []

    with torch.no_grad():
        for x, y in val_loader:
            x, y = x.to(device), y.to(device)
            outputs = model(x)

            val_preds.extend(outputs.argmax(1).cpu().numpy())
            val_labels.extend(y.cpu().numpy())

    val_acc = accuracy_score(val_labels, val_preds)
    val_f1 = f1_score(val_labels, val_preds, average="macro")

    print(f"Epoch {epoch+1}: "
          f"Train Acc={train_acc:.4f}, "
          f"Val Acc={val_acc:.4f}, "
          f"Val F1={val_f1:.4f}")
```

```
Epoch 1: Train Acc=0.7654, Val Acc=0.5556, Val F1=0.4937
Epoch 2: Train Acc=0.8308, Val Acc=0.6181, Val F1=0.5197
Epoch 3: Train Acc=0.8518, Val Acc=0.6023, Val F1=0.5167
Epoch 4: Train Acc=0.8677, Val Acc=0.6269, Val F1=0.5340
Epoch 5: Train Acc=0.8790, Val Acc=0.6373, Val F1=0.5589
Epoch 6: Train Acc=0.8856, Val Acc=0.6450, Val F1=0.5416
Epoch 7: Train Acc=0.8916, Val Acc=0.6333, Val F1=0.5561
Epoch 8: Train Acc=0.8982, Val Acc=0.6256, Val F1=0.5460
Epoch 9: Train Acc=0.9048, Val Acc=0.6708, Val F1=0.5711
Epoch 10: Train Acc=0.9074, Val Acc=0.6405, Val F1=0.5501
Epoch 11: Train Acc=0.9109, Val Acc=0.6177, Val F1=0.5580
Epoch 12: Train Acc=0.9143, Val Acc=0.6472, Val F1=0.5780
Epoch 13: Train Acc=0.9168, Val Acc=0.6403, Val F1=0.5699
Epoch 14: Train Acc=0.9211, Val Acc=0.6203, Val F1=0.5352
Epoch 15: Train Acc=0.9217, Val Acc=0.6482, Val F1=0.5723
Epoch 16: Train Acc=0.9248, Val Acc=0.6617, Val F1=0.5603
Epoch 17: Train Acc=0.9278, Val Acc=0.6645, Val F1=0.5721
Epoch 18: Train Acc=0.9295, Val Acc=0.5996, Val F1=0.5400
Epoch 19: Train Acc=0.9292, Val Acc=0.6501, Val F1=0.5714
Epoch 20: Train Acc=0.9329, Val Acc=0.6715, Val F1=0.5649
```