```
# -*- coding: utf-8 -*-
"""ling441predictanauthor.ipynb

Automatically generated by Colab.

Original file is located at
    https://colab.research.google.com/drive/1IlvS2j-oS2U0sHTXL1vyI500tBH0eBvN
"""

import sys
import re
import random
from collections import defaultdict
import torch
import torch.nn as nn


if len(sys.argv) > 2 or len(sys.argv) == 1:
    num_iter = 30000
else:
    num_iter = int(sys.argv[1])  # MUST be wrapped in int()


TRAIN_FILES = {
    "Coleridge": "Coleridge_train.txt",
    "EBBrowning": "EBBrowning_train.txt",
    "Eliot": "Eliot_train.txt",
    "Emily Dickinson": "Emily_Dickinson_train.txt",
    "Shakespeare": "Shakespeares_sonnets_train.txt",
}

TEST_FILES = {
    "Coleridge": "Coleridge_test.txt",
    "EBBrowning": "EBBrowning_test.txt",
    "Eliot": "Eliot_test.txt",
    "Emily Dickinson": "Emily_Dickinson_test.txt",
    "Shakespeare": "Shakespeares_sonnets_test.txt",
}


EMBED_DIM = 64
LR = 0.003
N = 5
NUM_DATA_POINTS = 50000
PRINT_INTERVAL = 1000
WEIGHT_DECAY = 0.0

def preprocess_line(line):
    line = line.lower()
    line = re.sub(r"[^a-z']", " ", line)
    return line.split()

def build_vocab_and_ngrams(train_files, n):
    vocab = set()
    author_ngrams = defaultdict(list)
    for author, path in train_files.items():
        with open(path, 'r', encoding='utf-8') as f:
            for line in f:
                words = preprocess_line(line)
            if len(words) < n:
                continue
            for w in words:
                vocab.add(w)
            for i in range(len(words) - n + 1):
                ngram = words[i:i+n]
                author_ngrams[author].append(ngram)
    vocab_list = list(vocab)
    vocab_list.append('OOV')
    wd2ix = {w: i for i, w in enumerate(vocab_list)}
    return vocab_list, wd2ix, author_ngrams


class AuthorNet(nn.Module):
    def __init__(self, vocab_size, embed_dim, n, num_authors):
        super().__init__()
        self.embeddings = nn.Embedding(vocab_size, embed_dim)
        self.linear = nn.Linear(embed_dim * n, num_authors)
        self.softmax = nn.Softmax(dim=1)

    def forward(self, idx_tensor):
        embeds = self.embeddings(idx_tensor)
        concat = embeds.view(1, -1)
        out = self.linear(concat)
        return out


print("Building vocabulary and extracting n-grams from training files...")
vocab_list, wd2ix, author_ngrams = build_vocab_and_ngrams(TRAIN_FILES, N)
vocab_size = len(vocab_list)
oov_idx = wd2ix['OOV']
authors = sorted(list(TRAIN_FILES.keys()))
author2ix = {a: i for i, a in enumerate(authors)}
ix2author = {i: a for a, i in author2ix.items()}


device = torch.device('cpu')
model = AuthorNet(vocab_size, EMBED_DIM, N, len(authors)).to(device)
loss_fn = nn.CrossEntropyLoss()
optimizer = torch.optim.Adam(model.parameters(), lr=LR, weight_decay=WEIGHT_DECAY)
random.seed(42)
torch.manual_seed(42)


def ngram_to_idx_tensor(ngram, wd2ix_local, oov_index):
    idxs = [wd2ix_local.get(w, oov_index) for w in ngram]
    return torch.tensor(idxs, dtype=torch.long, device=device)

model.train()
window_correct = 0
window_total = 0
for it in range(1, num_iter + 1):
    author = random.choice(authors)
    ngram = random.choice(author_ngrams[author])
    idx_tensor = ngram_to_idx_tensor(ngram, wd2ix, oov_idx)
    target_idx = torch.tensor([author2ix[author]], dtype=torch.long, device=device)
    logits = model(idx_tensor)
    loss = loss_fn(logits, target_idx)
    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
    with torch.no_grad():
        pred_label = torch.argmax(logits, dim=1).item()
        correct = 1 if pred_label == target_idx.item() else 0
        window_correct += correct
        window_total += 1
    if it % PRINT_INTERVAL == 0:
        print(f"{window_correct} out of {window_total} correct; avg ep loss on last
{window_total} {loss.item():.4f}")
        window_correct = 0
        window_total = 0

print("Training complete.\n")


print("Now testing.")
print("Log prob scores for each test file:")

model.eval()
authors_print_order = ["Coleri", "Eliot_", "Emily_", "Shakes", "EBBrow"]
author_key_map = {
    "Coleridge": "Coleri",
    "Eliot": "Eliot_",
    "Emily Dickinson": "Emily_",
    "Shakespeare": "Shakes",
    "EBBrowning": "EBBrow"
}

for true_author, test_path in TEST_FILES.items():
    agg_log_probs = torch.zeros(1, len(authors), device=device)
    with open(test_path, 'r', encoding='utf-8') as f:
        for line in f:
            words = preprocess_line(line)
            if len(words) < N:
                continue
            for i in range(len(words) - N + 1):
                ngram = words[i:i+N]
                idx_tensor = ngram_to_idx_tensor(ngram, wd2ix, oov_idx)
                with torch.no_grad():
                    logits = model(idx_tensor)
                    probs = model.softmax(logits)
                    log_probs = torch.log(probs + 1e-12)
                    agg_log_probs += log_probs

    scores = agg_log_probs.squeeze().tolist()
    best_idx = scores.index(max(scores))
    best_author = author_key_map[ix2author[best_idx]]

    short_name = author_key_map[true_author]
    print(f"{short_name}", end=" ")
    for s in scores:
        print(f"{s:.4f}", end=" ")
    print(f"Best score index: {best_idx + 1}")
    print(f"which is {best_author}")
```