```
"""
正常ベースライン学習プログラム（CSV→JSON変換対応）
CSV形式の訓練データを各行ごとにJSON形式に変換し、正常パターンを学習

使い方:
    python train_baseline.py --input data/train_data_network_100_5.csv
    python train_baseline.py --input data/train_data_log_150_10.csv --model llama3.2:1b
    python train_baseline.py --input data/train_data_sensor_200_3.csv --text-columns timestamp sensor_id value

必要なライブラリ:
    pip install numpy requests pandas

ディレクトリ構造:
    data/           - 入力CSVファイル
    baseline/       - 出力ベースラインファイル（.npz + _metadata.json）
"""

import json
import numpy as np
import requests
import argparse
from datetime import datetime
from typing import List
import sys
import time
import pandas as pd
import os

# 出力ディレクトリの定義
OUTPUT_DIR = "baseline"

def ensure_directories():
    """必要なディレクトリを作成"""
    os.makedirs(OUTPUT_DIR, exist_ok=True)
    print(f"✓ 出力ディレクトリ: {OUTPUT_DIR}/")

class BaselineTrainer:
    """正常ベースライン学習クラス（JSON形式対応）"""

    def __init__(self, model_name: str = "llama3.2:3b", base_url: str = "http://localhost:11434"):
        self.model_name = model_name
        self.base_url = base_url
        self.embedding_endpoint = f"{base_url}/api/embeddings"
        self.check_connection()

    def check_connection(self):
        """Ollamaサーバーへの接続確認"""
        try:
            response = requests.get(f"{self.base_url}/api/tags", timeout=5)
            response.raise_for_status()
            print(f"✓ Ollamaサーバーに接続しました: {self.base_url}")
        except requests.exceptions.ConnectionError:
            print(f"✗ エラー: Ollamaサーバーに接続できません: {self.base_url}")
            print(f"  Ollamaが起動しているか確認してください: ollama serve")
            sys.exit(1)
        except Exception as e:
            print(f"⚠ 警告: サーバー確認中にエラーが発生しました: {str(e)}")

    def load_csv(self, filename: str, text_columns: List[str] = None) -> List[str]:
        """
        CSVファイルを読み込み、各行をJSON形式のテキストデータに変換

Args:
    filename: CSVファイル名
    text_columns: JSON化する列名のリスト（Noneの場合は全列）

Returns:
    JSON形式のテキストに変換されたサンプルのリスト
"""
try:
    df = pd.read_csv(filename)
except FileNotFoundError:
    print(f"✗ エラー: ファイルが見つかりません: {filename}")
    sys.exit(1)
except Exception as e:
    print(f"✗ エラー: CSVファイルの読み込みに失敗しました: {str(e)}")
    sys.exit(1)

# labelが'normal'のデータのみ抽出
if 'label' in df.columns:
    df = df[df['label'] == 'normal']

print(f"  総行数: {len(df)}件")
print(f"  列: {', '.join(df.columns.tolist())}")

#JSON化する列を決定
if text_columns is None:
    # label列を除外
```

```
        text_columns = [col for col in df.columns if col != 'label']

    print(f"  JSON化する列: {', '.join(text_columns)}")

    # 各行をJSON形式に変換
    samples = []
    for _, row in df.iterrows():
        # 各列の値を辞書形式で格納
        data_dict = {}
        for col in text_columns:
            if col in df.columns:
                # NaN値をNoneに変換
                value = row[col]
                if pd.isna(value):
                    data_dict[col] = None
                else:
                    data_dict[col] = value

        # JSON文字列に変換 (ensure_ascii=Falseで日本語も対応)
        json_text = json.dumps(data_dict, ensure_ascii=False)
        samples.append(json_text)

    return samples

def get_embedding(self, text: str, retry_count: int = 3) -> np.ndarray:
    """
    テキストをEmbeddingベクトルに変換

    Args:
        text: 変換するテキスト（JSON形式）
        retry_count: リトライ回数

    Returns:
        Embeddingベクトル（失敗時はNone）
    """
    for attempt in range(retry_count):
        try:
            payload = {
                "model": self.model_name,
                "prompt": text
            }

            response = requests.post(
                self.embedding_endpoint,
                json=payload,
                timeout=30
            )
            response.raise_for_status()

            embedding = response.json()["embedding"]
            return np.array(embedding)

        except requests.exceptions.Timeout:
            if attempt < retry_count - 1:
                print(f"  ⚠ タイムアウト。リトライ中... ({attempt + 1}/{retry_count})")
                time.sleep(1)
            else:
                print(f"  ✗ タイムアウト: Embedding取得失敗")
                return None

        except requests.exceptions.HTTPError as e:
            print(f"  ✗ HTTPエラー: {e}")
            return None
        except Exception as e:
            print(f"  ✗ Embedding取得エラー: {str(e)}")
            return None

    return None

def train(self, train_samples: List[str], verbose: bool = True) -> dict:
    """
    訓練データからベースラインを学習

    Args:
        train_samples: 訓練用の正常データリスト（JSON形式）
        verbose: 詳細な進捗表示

    Returns:
        ベースライン情報を含む辞書
    """
    if verbose:
        print("¥n" + "=" * 80)
        print("正常ベースライン学習開始 (JSON形式)")
        print("=" * 80)
        print(f"  モデル: {self.model_name}")
        print(f"  訓練サンプル数: {len(train_samples)}")
        print()
```

```
embeddings = []
failed_indices = []
start_time = time.time()

for i, sample in enumerate(train_samples):
    if verbose:
        # 進捗表示
        progress = (i + 1) / len(train_samples) * 100
        elapsed = time.time() - start_time
        eta = (elapsed / (i + 1)) * (len(train_samples) - i - 1) if i > 0 else 0

        print(f"  進行状況: [{i+1}/{len(train_samples)}] "
              f"({progress:.1f}%) - "
              f"経過: {elapsed:.1f}秒, 残り: {eta:.1f}秒",
              end='¥r')

    embedding = self.get_embedding(sample)

    if embedding is not None:
        embeddings.append(embedding)
    else:
        failed_indices.append(i)

if verbose:
    print()  # 改行
    print()

if len(embeddings) == 0:
    raise ValueError("すべてのEmbedding取得に失敗しました")

embeddings = np.array(embeddings)

# 統計情報を計算
mean_embedding = np.mean(embeddings, axis=0)
std_embedding = np.std(embeddings, axis=0)
median_embedding = np.median(embeddings, axis=0)

# 成功率を計算
success_rate = len(embeddings) / len(train_samples) * 100

result = {
    "model_name": self.model_name,
    "train_count": len(train_samples),
    "success_count": len(embeddings),
    "failed_count": len(failed_indices),
    "success_rate": success_rate,
    "embedding_dim": embeddings.shape[1],
    "embeddings": embeddings,
    "mean_embedding": mean_embedding,
    "std_embedding": std_embedding,
    "median_embedding": median_embedding,
    "train_samples": train_samples,
    "failed_indices": failed_indices,
    "trained_at": datetime.now().isoformat(),
    "training_time": time.time() - start_time,
    "data_format": "json"  # データ形式を記録
}

if verbose:
    print("=" * 80)
    print("学習完了")
    print("=" * 80)
    print(f"  成功: {len(embeddings)}件 ({success_rate:.1f}%)")
    print(f"  失敗: {len(failed_indices)}件")
    print(f"  Embedding次元数: {embeddings.shape[1]}")
    print(f"  学習時間: {result['training_time']:.1f}秒")

    if len(failed_indices) > 0:
        print(f"¥n  ⚠ 失敗したサンプルのインデックス: {failed_indices[:10]}")
        if len(failed_indices) > 10:
            print(f"     ... 他 {len(failed_indices) - 10}件")

return result

def save_baseline(self, baseline: dict, filename: str, source_csv: str, verbose: bool = True):
    """
    ベースラインをファイルに保存

    Args:
        baseline: 保存するベースライン情報
        filename: 出力ファイル名 (.npz)
        source_csv: 元のCSVファイル名
        verbose: 詳細表示

    Returns:
        保存したファイルのフルパス
    """
```

```
if verbose:
    print("¥n" + "=" * 80)
    print("ベースライン保存中")
    print("=" * 80)

# ファイル名にディレクトリを追加
output_path = os.path.join(OUTPUT_DIR, filename)

# NumPy配列はnpzで保存（圧縮）
np.savez_compressed(
    output_path,
    embeddings=baseline["embeddings"],
    mean_embedding=baseline["mean_embedding"],
    std_embedding=baseline["std_embedding"],
    median_embedding=baseline["median_embedding"]
)

# メタデータはJSONで保存
metadata_file = output_path.replace('.npz', '_metadata.json')
metadata = {
    "model_name": baseline["model_name"],
    "train_count": baseline["train_count"],
    "success_count": baseline["success_count"],
    "failed_count": baseline["failed_count"],
    "success_rate": baseline["success_rate"],
    "embedding_dim": baseline["embedding_dim"],
    "trained_at": baseline["trained_at"],
    "training_time": baseline["training_time"],
    "source_csv": source_csv,
    "failed_indices": baseline["failed_indices"],
    "data_format": baseline.get("data_format", "json")
}

with open(metadata_file, 'w', encoding='utf-8') as f:
    json.dump(metadata, f, ensure_ascii=False, indent=2)

if verbose:
    print(f"  ✓ Embeddingデータ: {output_path}")
    print(f"    ファイルサイズ: {self._get_file_size(output_path)}")
    print(f"  ✓ メタデータ: {metadata_file}")
    print(f"    ファイルサイズ: {self._get_file_size(metadata_file)}")

return output_path

def _get_file_size(self, filename: str) -> str:
    """ファイルサイズを人間が読みやすい形式で取得"""
    try:
        size = os.path.getsize(filename)
        for unit in ['B', 'KB', 'MB', 'GB']:
            if size < 1024.0:
                return f"{size:.2f} {unit}"
            size /= 1024.0
        return f"{size:.2f} TB"
    except:
        return "不明"


def main():
    parser = argparse.ArgumentParser(
        description='正常ベースラインを学習（CSV→JSON変換対応）',
        formatter_class=argparse.RawDescriptionHelpFormatter,
        epilog="""
使用例:
  # 基本的な使用
  python train_baseline.py --input data/train_data_network_100_5.csv

  # 出力ファイル名を指定
  python train_baseline.py --input data/train_data_log_150_10.csv --output my_baseline.npz

  # 軽量モデルを使用
  python train_baseline.py --input data/train_data_sensor_200_3.csv --model llama3.2:1b

  # JSON化する列を指定
  python train_baseline.py --input data/train_data_log_150_10.csv --text-columns timestamp level event detail

データ形式:
  各CSV行を以下のようなJSON形式に変換して学習します:
  {"timestamp": "2024-01-01 10:00:00", "level": "INFO", "event": "login", "detail": "user123"}

ディレクトリ構造:
  data/           - 入力CSVファイルの場所
  baseline/       - 出力先（自動作成）
    """
    )

    parser.add_argument('--input', type=str, required=True,
                        help='入力CSVファイル (data/train_data_*.csv)')
    parser.add_argument('--output', type=str, default=None,
```

```
                    help='出力ベースラインファイル名 (指定しない場合は自動生成)')
parser.add_argument('--model', type=str, default='llama3.2:3b',
                    help='使用するOllamaモデル (デフォルト: llama3.2:3b)')
parser.add_argument('--url', type=str, default='http://localhost:11434',
                    help='Ollamaサーバー URL (デフォルト: http://localhost:11434)')
parser.add_argument('--text-columns', nargs='*', default=None,
                    help='JSON化する列名 (指定しない場合はlabel以外の全列) ')
parser.add_argument('--quiet', action='store_true',
                    help='詳細な進捗表示を抑制')

args = parser.parse_args()
verbose = not args.quiet

# ヘッダー表示
if verbose:
    print("=" * 80)
    print("正常ベースライン学習プログラム (CSV→JSON変換対応)")
    print("=" * 80)

print(args.text_columns)
# JSON化する列を決定
text_columns=args.text_columns
if text_columns is None:
    print(text_columns)
else:
    text_columns = args.text_columns[0]
    # カンマ区切りの文字列をリストに変換
    text_columns = [col.strip() for col in text_columns.split(',')]

# ディレクトリ作成
ensure_directories()

# CSVデータ読み込み
if verbose:
    print(f"CSVデータ読み込み中: {args.input}")

trainer = BaselineTrainer(model_name=args.model, base_url=args.url)
train_samples = trainer.load_csv(args.input, text_columns)

if len(train_samples) == 0:
    print("¥n✗ エラー: 訓練データが見つかりません")
    sys.exit(1)

if verbose:
    print(f"¥n訓練データ: {len(train_samples)}件 (正常データのみ)")

    # サンプルデータを表示 (JSON形式)
    print(f"¥nサンプルデータ (JSON形式、最初の3件):")
    for i, sample in enumerate(train_samples[:3], 1):
        # JSONを整形して表示
        try:
            json_obj = json.loads(sample)
            formatted = json.dumps(json_obj, ensure_ascii=False, indent=2)
            print(f"  {i}. {formatted}")
        except:
            preview = sample[:150] + "..." if len(sample) > 150 else sample
            print(f"  {i}. {preview}")

# 学習実行
try:
    baseline = trainer.train(train_samples, verbose=verbose)
except KeyboardInterrupt:
    print("¥n¥n✗ 学習を中断しました")
    sys.exit(1)
except Exception as e:
    print(f"¥n✗ エラー: 学習中にエラーが発生しました: {str(e)}")
    if verbose:
        import traceback
        traceback.print_exc()
    sys.exit(1)

# 出力ファイル名を決定
if args.output is None:
    # 入力ファイル名からベースライン名を生成
    base_name = os.path.basename(args.input)
    base_name = base_name.replace('train_data_', 'baseline_').replace('.csv', '.npz')
    output_file = base_name
else:
    output_file = args.output
    if not output_file.endswith('.npz'):
        output_file += '.npz'

# 保存
output_path = trainer.save_baseline(baseline, output_file, args.input, verbose=verbose)

# 完了メッセージ
if verbose:
    print("¥n" + "=" * 80)
```

```
        print("学習完了!")
        print("=" * 80)

        # テストファイル名を推測
        test_file = args.input.replace('train_data_', 'test_data_')

        print(f"¥n次のステップ:")
        print(f"  異常検知を実行:")
        print(f"    python detect_anomaly_unified.py --baseline {output_path} --input {test_file}")
        print()


if __name__ == "__main__":
    try:
        main()
    except KeyboardInterrupt:
        print("¥n¥n処理を中断しました")
        sys.exit(1)
    except Exception as e:
        print(f"¥n¥n✗ 予期しないエラーが発生しました: {str(e)}")
        import traceback
        traceback.print_exc()
        sys.exit(1)
```