24/11/05

Motion Decoding Using Biosignals スケートボードトリック分類チャレンジ
ソースコード説明書

1.はじめに

1.1 概要

本システムは、脳波データから行動の予測を行うモデルである。センサーによる時系列データを、以下の 3 つの主要部分で処理する。

- ✓ 前処理部分:データの読み込みと整形
- ✓ 学習部分:遺伝的アルゴリズムによるセンサー選択と ROCKET モデルの学習
- ✓ 推論部分:学習済みモデルによる予測とアンサンブル

1.2 構成

本システムでは、脳波データに対してマスクを適用し、使用するセンサーデータを制御している。前処理で整形した時系列データを、遺伝的アルゴリズムにより最適化したマスクを用いて Rocket モデルで学習し、予測を行う。最終的な予測は、上位 3 つのモデルのスコアを基にアンサンブル処理を行い、推論結果ファイル(submit.csv)を生成する。

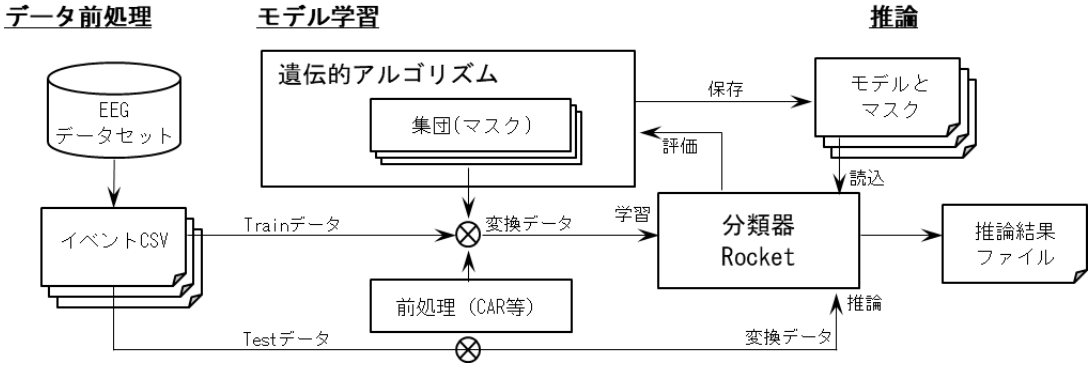

図 1 モデルの構成

2.操作方法

システムは、前処理「01.Preprocessing.ipynb」、学習「02.Learning.ipynb」、推論「03.Predicting.ipynb」の 3 つのプログラムで構成される。以下に、それぞれの操作方法を示す

2.1 前処理

(1)Google Drive にて作業用フォルダを作成し、「01.データセット作成プログラム.ipynb」をアップロードする。以下、当フォルダのパスを{BASE_DIR}と記す

(2)同フォルダ配下に data1,data2 フォルダを作成する

(3)データ①②の学習用データ・評価用データを解凍し、下記のフォルダ構成で保存する

```
{BASE_DIR}/
 └── data1/
        ├── train/   ※データ①の train.zip の内容
        └── test/   ※データ①の test.zip の内容
 └── data2/
        ├── train/   ※データ②の train1.zip,train2.zip の内容
        └── test/   ※データ②の test.zip の内容
```

(4)Google Coloab 上で「01.データセット作成プログラム.ipynb」を開く

(5)「0.初期設定」のプログラムの定数 BASE_DIR を上記のフォルダに書き換える

なお Google Drive の Google Cloab へのマウント時のルートパスは「/content/drive/MyDrive/」である。

(6)「0.初期設定」の 2 つのセルを、順次実行する

(7)学習用データ生成のため、セル「1.学習用データ作成」を実行する

```
make_train_data(TRAIN_MAT_DIR1, TRAIN_SAVE_DIR1)
make_train_data(TRAIN_MAT_DIR2, TRAIN_SAVE_DIR2)
```

(8)下記構成でデータが格納できたことを確認する

```
{BASE_DIR}/
 └── data{*}/  ※data1,2
        └── modeling_data/  生成されたデータフォルダ
              └── subject{*]}/  ※被験者 0～4
                    └── train{*}.mat/  ※mat0～2
                          └── {トリック種別}/  ※frontside_kickturn,backside_kickturn,pumping
                                └── [連番].csv  ※000 からの連番
```

(9)推論用データ作成のため、セル「2.推論用データ作成」を実行する

```
make_val_data(TEST_MAT_DIR1, TEST_SAVE_DIR1)
make_val_data(TEST_MAT_DIR2, TEST_SAVE_DIR2)
```

(10)下記構成でデータが格納できたことを確認する

```
{BASE_DIR}/
 └── data{*}/  ※data1,2
        └── modeling_data/  生成されたデータフォルダ
              └── subject{*]}/  ※被験者 0～4
                    └── subject{*}.mat/  ※mat0～2
                          └── unknown/
                                └── [連番].csv  ※000 からの連番
```

2.2 学習

(1)2.1 で作成した、Google Drive 上の作業用フォルダに、「02.Learning.ipynb」をアップロードしオープンする

(2)「0.初期設定」の「今回の実行の設定」の定数を、今回のプロジェクトに合わせて設定する。

BASE_DIR = '(パス)'　　※2.1 で設定した BASE_DIR と同じパスを指定
MODEL_NAME = '(名称) '　※今回の実行を識別する任意の名称。「trial01」など
ENV_MODE =1　　※動作モード:データ①、2:データ②、3:データ①コンペ時設定
NUM_POPULATIONS = 30　※遺伝的アルゴリズムでの集団数
NUM_GENERATIONS = 50　※遺伝的アルゴリズムでの実行世代数
RANDOM_SEED = 42　　※乱数シード

(3)「0.初期設定」の 2 つのセルを、順次実行する

(4)「1.データローダ定義」のセルを実行する

(5)「2.学習実行」のセルを実行する

これにより、学習が実行される。なお Google Colab Pro+環境のハイメモリ CPU にて、1 世代あたり約 24 分を要する。途中でセッションが終了した場合には、再実行することにより、前回から継続いて学習が進められる。

(6)実行結果を確認する

以下の通り、モデル名を付与したフォルダが生成され、成績上位 3 つのモデル・フォルダおよび配下にそれぞれれのモデルファイル群が保存される。

```
{BASE_DIR}/
└── models_{MODEL_NAME}/
        └── mask_models/
                └── score{score}_gen{generation}/  ※例「score0.8787_gen24」
                └── score{score}_gen{generation}/
                └── score{score}_gen{generation}/
```

2.3 推論

(1)2.1 で作成した、Google Drive 上の作業用フォルダに、「03.Predicting.ipynb」をアップロードしオープンする

(2)「0.初期設定」の「今回の実行の設定」の定数を、今回のプロジェクトに合わせて設定する。

BASE_DIR = '(パス)'　　※2.1 で設定した BASE_DIR と同じパスを指定
MODEL_NAME = '(名称) '　※今回の実行を識別する任意の名称。「trial01」など
ENV_MODE =1　　※動作モード:データ①、2:データ②、3:データ①コンペ時設定
RANDOM_SEED = 42　　※乱数シード

(3)「0.初期設定」の 2 つのセルを、順次実行する

(4)「1.データローダ定義」のセルを実行する

(5)「2.予測実行(submit 生成)」のセルを実行する

これにより推論が mask_models フォルダ配下の学習結果ファイルに基づいて推論が実行され、結果が csv ファイルで保存される。

(6)実行結果を確認する

以下の通り、mask_models_predictions フォルダが生成され、配下に成績上位 3 つのモデルによる予測、およびアンサンブルでの予測結果が CSV 形式で保存される。

```
{BASE_DIR}/
└── models_{MODEL_NAME}/
        └── mask_models_predictions/
                ├── submit_score{score}_gen{generation}.csv  ※例「submit_score0.8787_gen24.csv」
                ├── submit_score{score}_gen{generation}.csv
                ├── submit_score{score}_gen{generation}.csv
                └── submit.csv     ※アンサンブルによる予測結果(投稿用)
```

3.アルゴリズムについて

3.1 データ前処理

生データからモデルの学習と評価に適した形式にデータを変換する。

- Googleドライブから生データ(.mat ファイル)を読み込む。
- イベント情報に基づいてデータをセグメント化し、各クラス(動作)ごとの CSV ファイルとして保存する。
- 評価用データも同様に処理し、未知のラベル(unknown)として保存する。

3.2 モデルの学習

最適なセンサーマスクを探索し、そのマスクを適用したデータでモデルを学習する。

- 遺伝的アルゴリズムを使用してセンサーマスクを最適化する。
  - ✓ 初期集団の生成、適応度評価、選択、交叉、突然変異の各ステップを実施。
  - ✓ 各世代で上位のマスクを保持し、次世代に反映。
- 最適化されたマスクを用いてデータをマスク処理し、モデルを学習。
  - ✓ ROCKET(Random Convolutional Kernel Transform)を用いて特徴量を抽出。
  - ✓ RidgeClassifierCV を用いて分類モデルを構築。

3.3 推論

学習済みモデルを用いてデータの予測を行い、アンサンブルで最終的な予測結果を生成する。

- 上位 3 つの学習済みモデルをロードする。
- 評価用データに対して各モデルで予測を行う。
- 各サンプルについて、モデルの予測結果を多数決で集約し、最終的なクラスを決定する。
- 予測結果を提出用の CSV ファイルとして保存する。

3.4 補助機能

- キャッシング機能:データ読み込みの高速化を実現する
- 途中経過の保存:学習の中断・再開を可能とする

4.ソースコード解説（主要関数の定義）

4.1 データ前処理

(1) make_train_data(mat_dir, save_dir)

機能	学習用データの作成
引数	mat_dir: MAT ファイルの格納ディレクトリ save_dir: 生成データの保存ディレクトリ
返値	なし ※ファイルを生成

(2) make_val_data(mat_dir, save_dir)

機能	推論用データの作成
引数	mat_dir: MAT ファイルの格納ディレクトリ save_dir: 生成データの保存ディレクトリ
返値	なし ※ファイルを生成

4.2 モデルの学習

(1)【クラス】SeqDataset(Dataset)

機能	マスクに対応するモデルファイル群からアンサンブルで推論実行
関数	_init_: データセットフォルダやマスクの設定 _get_item_: レコードの取得 _len_: レコード数取得 blend_data: マスクに基づいてデータセット加工 preprocess: データロード時の前処理 save_data_to_cache,: 読込んだデータセットをキャッシュとして保存(高速化用) load_data_from_cache: キャッシュファイルからデータセット復元

(2) initialize_population(population_size, mask_size)

機能	遺伝的アルゴリズムでの初期集団の生成
引数	population_size: 集団サイズ mask_size: マスクの次元数
返値	population: 初期マスク群

(3) crossover(parent1, parent2)

機能	マスクの交叉処理
引数	parent1, parent2: 親マスク
返値	child1, child2: 子マスク

(4) mutate(mask, mutation_rate)

機能	突然変異処理
引数	mask: 対象マスク mutation_rate: 突然変異率
返値	mutated_mask: 変異後のマスク

(5) genetic_algorithm(model_name, population_size, generations, mutation_rate)

機能	遺伝的アルゴリズムによるマスク最適化
引数	model_name: モデル識別名 population_size: 集団サイズ generations: 世代数 mutation_rate: 突然変異率
返値	best_mask: 最優秀のマスク ※コンペティションでは未使用

(6) apply_car_and_normalize(data)

機能	CAR フィルタと正規化の適用
引数	data: 入力データ
返値	processed_data: 処理済みデータ

(7) train_rocket_model(X_train, y_train)

機能	ROCKET モデルの学習
引数	X_train: 学習データ y_train: 教師ラベル
返値	trained_model: 学習済みモデル

4.3 推論

(1)**【クラス】**initialize_population(population_size, mask_size)

機能	マスクに対応するモデルファイル群からアンサンブルで推論実行
関数	_init_: モデル群の読込み _load_models: 保存モデルの復元 predict: アンサンブル予測の実行

(2) predict()

機能	推論の実行
引数	なし
返値	なし　※推論結果を csv ファイルに出力