山口大学教育学部研究論叢　第 73 巻　91 頁～ 100 頁　2024 年 1 月

様々な機械学習を用いた手の動作識別について

伊藤　正剛*・北本　卓也**

Hand Movement Identification Using Various Machine Learning Methods

ITO Masataka*, KITAMOTO Takuya**

（Received September 29, 2023）

現在、様々な機械学習手法が考案され、比較的簡単に使うことができるようになっている。前回の研究［1］では、手の動作識別やそれを利用したカーソルの動きについて、深層学習を用いて学習回数が少なく高精度に識別する手法を用いたが、本研究ではその他の機械学習手法ではどうなのかを調査する。一つ一つ機械学習の手法を試していくのは、時間がかかるため、かなりの部分を自動化し、複数のモデルを比較できるPyCaretというPythonのライブラリを用いて実験を行う。PyCaretは、環境によってインストールに少し時間がかかることもあるが、それ以上に自動化によって受ける恩恵は大きい。PyCaretを使い、性能が高いモデルを探索し、それを使って 3動作識別を行うことで、遅延が少なく正確なカーソル操作を目指す。本手法では動作識別のたびに再学習が必要となるが、比較的シンプルな構成でコストも低いことから、筋電義手への応用に有用ではないかと考えられる。また、今回も前回の研究[1]を引き継ぎ、Arduino Uno R3 やMyoWare筋電センサといった安価で、手軽に実験できる機器を用いる。

１．はじめに

1.1　現状分析と先行研究

上肢切断者が使用する義手（prosthetic hand）は、歴史が古く紀元前から使われている。古代では、戦いにより手や腕を負傷したときに、外観を補ったり、完全とは行かないまでも、ある程度、物をつかんだりするのに使用されてきた。現代でも事故や病気により義手を用い、手やその機能の代替として用いることがある。また、義手の役割自体は変化してないものの、コンピュータの発展により、義手にコンピュータの役割を持たせることで手の機能を拡張することも可能となっている。

さらに、近年では、義手を正確に動かすため、意図した手の動きを識別する方法が開発されている。本研究では、義手作成には深く立ち入らず、正確な動作識別とそれを利用したカーソルの操作を中心に述べる。手の動作識別で、具体的には、センサを用いることにより、生体信号の一つである筋電位を計測し、機械学習を用いてセンサの値から動作識別を行うことで、義手の動きを制御しようという方法である。ここで、動作識別とは、例えば、手を握る動作を行えば、筋電位から「握った」と識別することである。

筋電位を用いて動作識別を行い操作する義手を筋電義手と言う。

筋電位による動作識別は、日本では1960年代から力を入れて開発が行われたが、広く実用化されるまでには至らなかった。現在、筋電義手は活発な開発が行われているが、これは、機械学習で動作識別するプログラムや手法が作られたことが一因にある。例えば、k-NN法を用いた表面筋電位による指動作識別[2]や筋電位を利用したサポートベクトルマシンによる手のリアルタイム動作識別[3]など、機械学習を用いて手の動作識別を行い、義手を動かそうとする試みは多い。

また、深層学習の登場で、筋電位信号のビッグデータを用いて、モデルを作成する手法もある。このように、手の動作識別について様々なアプローチで研究が行われているが、機械学習の手法では深層学習が使われることが散見される。

その要因は、以前は主にグラフィックス用途だったGPUが、深層学習の計算に応用されるようになったこと、TensorFlowなどの深層学習向けの機械学習ライブ

＊ 放送大学教養学部（自然と環境コース）
＊＊ 山口大学教育学部　〒753-8513　山口市吉田1677－1　kitamoto@yamaguchi-u.ac.jp

ラリが作られ、深層学習の敷居が下がっていることにもある。

結果として、大量の筋電位測定データを用いて、学習を行う手法が登場し完全な義手のコントロールに近い精度で実用化しているものもある。しかし、深層学習以外の方法もあり、そのような手法も有用な可能性もある。

1.2　研究動機

現状分析で、深層学習以外の動作識別方法もあることを紹介した。深層学習を使って義手を制御するのは、有用なことだと思われる。以前の論文でも引用した[4]のような、大量のデータを用いることにより、義手のつけ外しや日を跨ぐことによる、信号特性の変化に強い識別モデルを作り義手として動作させている研究もある。

前回の研究[1]では、動作識別を行い、カーソルの操作を目標に研究を行ったが、その実験は、主に、計測したデータをパソコンに一旦保存し、そのデータに対して教師ラベルをつけた後、深層学習で学習し学習済みモデルを使い予測することであった。リアルタイムでの動作識別処理は、以前の研究論文に明示していないが、その時点で２動作を目標に行っていた。この研究[1]からプログラムを改良し、卒業研究論文[5]で２動作の「握り」と「動作無し」を識別し、カーソルを動かすことができた。ただし、２動作識別では、カーソルの動作方向は１方向に限られている。

深層学習での研究も続けているが、研究が停滞しているのと、深層学習以外の機械学習手法を用いて、動作識別やカーソル動作を確認することも新しい視点や利点となると考え、それを動機として本研究を行った。今回は、深層学習のようなビッグデータの取得というアプローチは行わず、逆に少ないデータで、使用前に短い時間、事前学習を行った後、動作識別を行う。その精度を上げることを目的としている。

動作識別では、３動作識別（「握り」「反り」「動作無し」）を行い、MyoWare筋電センサを付けて、リアルタイムに筋電位データを取得し、手を握ったら「握った」、手を反ったら「反った」、動作無しのときは「動作無し」と識別することである。また、その動作識別の値から、カーソルを操作することも目標である。

詳しくは後述するが、本研究はPyCaretというライブラリを用いて、筋電位信号という時系列のデータに良く当てはまり、予測してくれるモデルを探索している。

２．計測機器と表面筋電位について

計測機器として、前回の研究[1]や卒業研究論文[5]のときとマイコンやセンサは変えていない。

Arduinoは、機械としての「Arduinoボード」と「Arduino IDE」から構成されている。図１にArduino Uno R3の写真を掲載する。

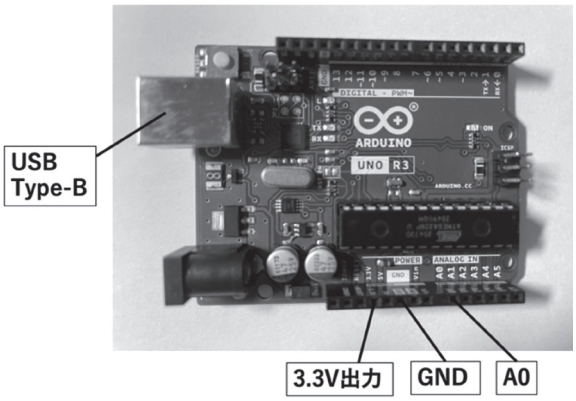

図１　Arduino Uno R3 ボード

Arduino Uno R3と、後述するMyoWare筋電センサとの接続部分、コンピュータに接続するUSB Type-Bの部分が分かるように記載した。

「Arduino IDE」は、Arduinoのプログラムを作成する統合開発環境で、Arduino言語が使われる。Arduino言語は、C言語やC++言語を元に設計されている。C言語やC++言語に馴染みのある場合、比較的理解しやすいと思われる。「Arduino IDE」には、Arduinoのスケッチと呼ばれるプログラムの例が多数あり、自分が作成しようとしているプログラムに近いスケッチがあれば、多少手直しすることによって目的のものを作成することができる。本研究では、「Arduino IDE」に付属するAnalogReadSerialと言うスケッチ例を、少し改変して使っている。

価格だが、Arduino Uno R3は、価格の変動で一つには決められないが、2023年８月時点では、4,000円程度と比較的安価である。

Arduinoは、MyoWare筋電センサからの表面筋電位の値を受信すると同時に、センサ値をコンピュータに送信する。（表面筋電位は、本論文では単に筋電位とも書いている。）

このセンサ値は「Arduino IDE」の中にあるシリアルモニタやシリアルプロッタを使い可視化できる。シリアルモニタではシリアル通信でのデータの表示あるいは送信を行うことができ、シリアルプロッタでは、時間経過に沿ってグラフにプロットし、表示することができる。

次に、筋電位を取得する機械であるMyoWare筋電センサについて説明する。MyoWare筋電センサは、Advancer Technologies社の低価格な筋電位センサで、図２に写真を載せる。

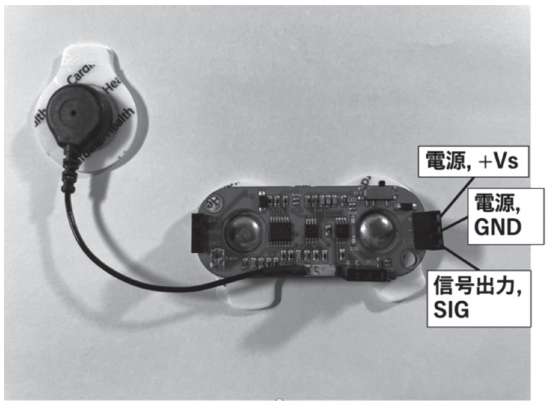

図２　MyoWare筋電センサ

MyoWare筋電センサは、センサパッドを使うEMGセンサとなっている。機器同士の接続の概略図を図３に掲載する。

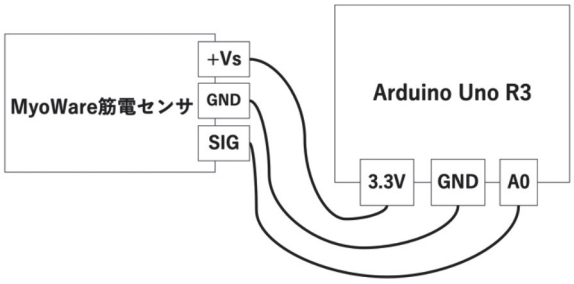

図３　Arduinoとセンサの接続概略図

また、MyoWare筋電センサは、本研究で用いるIEMG（積分筋電位）を出力することができる。

次に、この製品の価格情報を紹介する。現在、新しいモデルMyoWare 2.0筋電センサが登場したため、この研究で用いたモデルは、ほとんど販売されておらず、2023年８月では正確な価格はわからない。2022年８月の価格だと、4,000円前後であった。

MyoWare筋電センサは、Arduinoとジャンパワイヤを用いて、簡単に接続できるため使いやすく、また、提示した価格のように、筋電センサの中では低価格帯に位置し、低コストで導入できる。

上でも紹介したように、MyoWare筋電センサは、主に表在筋の筋線維全体から発生する活動電位である、表面筋電位を記録することがきる。ただし、MyoWare筋電センサのような表面筋電位を測定するセンサは、深層筋の活動は計測することは難しい[6]。

計測では、センサパッドを皮膚に貼り付けるだけなので、例えば電極のついた針を刺入する方法などと比べて侵襲性が低く、短時間で測定する際にそれほど不快な感じはない。ただ、MyoWare筋電センサをセンサパッドを使い長時間皮膚につける時は、汗による不快感や、配線により手や腕を動かせる範囲が狭い場合は、手や腕が疲労してしまうということがある。その場合、配線の長さを変えることや、無線通信を行うことにより解決できる場合もある。[7]では、体の動かせる範囲を拡大するとともに、計測装置とパソコンを電気的に絶縁状態にすることで安全性確保も考慮した計測装置を提案している。

３．プログラミング言語PythonとPyCaretについて

3.1　Pythonとその実行環境

Pythonは、プログラミングの初心者でも学習しやすく、ライブラリと呼ばれる、よく使われるプログラムの集まりが多数あり、場合によってはC言語などと比べて、少ない行数でプログラムを作ることができる。ライブラリには、例えば、科学技術計算でよく使われるNumPyやグラフの描画などに用いられるMatplotlib、機械学習分野ではscikit-learnなどがある。また、Pythonでは、インデント（日本語で字下げ）を行い、プログラムのブロックを区別するため、可読性が高い。

Pythonを導入するには様々な方法があるが、簡単な方法として前回の研究[1]で用いたGoogle Colaboratoryがある。詳しくは、[1]を参照されたい。

他にPythonの実行環境を作る方法として、Anacondaがある。本研究では、AnacondaでPythonの環境を構築し、Spyderと呼ばれる統合開発環境を用いて実験を行っている。これは、Spyderが、筋電位をリアルタイムで取得・解析し、その動作を識別するといった一連の流れを行うのに適しているからである。また、カーソルの動き具合を見るのに、エディタペインにあるソースコードなどのテキストを利用すればすぐに確認できる。

Anacondaでは、仮想環境を作成し、そこにSpyderをcondaコマンドでインストールした。pipを用いて仮想環境にSpyderをインストールすると、その仮想環境では起動しないことがある。

3.2　PyCaretについて

本研究では、より少ないコードの量で、複数の機械学習のライブラリの読み込みや前処理、モデルの学習と比較、チューニング、予測まで行ってくれるライブラリを用いることにした。

現在、多数の機械学習の手法があり、機械学習手法１つにつき１つコードを書いていくと、行数が多くなりかなりの労力が必要になる。PyCaretは複数のモデルを、様々な指標を基準に自動で比較してくれ、最も当てはま

りの良いモデルをハイパーパラメータとともに探索してくれる。これらは、少ない行数で行うことが可能である。また、PyCaretでは、ハイパーパラメータの値を明示的に記述することなしに、１行のコードで探索してくれる。

図４は、PyCaretの実行の流れを表している。

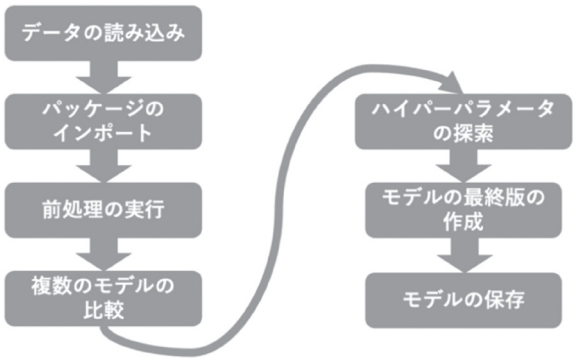

図４　PyCaret実行の流れ

図のそれぞれの段階が、１行から２行程度で記述できるので、モデルの保存まで行っても、10行程度で最適なモデルを探し出すことができる。PyCaretでは、モデルを網羅的に探索するために、計算に少し時間がかかる。

今回、実験で用いたコンピュータのスペックを表１に掲載する。

表１　実験コンピュータスペック

OS	Windows 10 Pro 64bit
CPU	Intel Core i7-8565U (1.80GHz-4.60GHz 4 コア/8 スレッド)
RAM	16GB LPDDR3 2133MHz

モデルの学習・探索の計算時間は、コンピュータによって異なる。ここではデータの詳細は解説しないが、例えば、(30, 20) のデータ（30行20列のデータ、以下この書き方でデータの形状を表す。）を表１のコンピュータでPyCaretを用いモデルの探索を行うと１分以内に終わった。この程度のデータであれば、それほど時間はかからない。

PyCaretを扱う上で、つまずきがちなのは、最初のインストールである。インストールするとき、Pythonのバージョンは、2023年８月14日では、Python3.7、Python3.8、Python3.9、Python3.10でなければならなかった。また、他のライブラリとの依存関係もあるので、もしインストール中に問題が出たなら、その部分も解決する必要がある。本研究でのPyCaretのバージョン、Pythonのバージョン、Spyderのバージョンを表２に載せる。

表２　ライブラリバージョン

Python	3.9.17
PyCaret	3.0.4
Spyder	5.4.3

４．実験方法

リアルタイム動作識別とカーソルの操作を行うまでの実験方法を説明する。筋電位データは全て筆者の腕から計測し取得したデータである。

１）識別の目標とする動作は、「握り」「反り」「動作無し」の３つである。

「握り」は、腕を机の上に置き、じゃんけんで言うグーの形に素早く握り開くこと、「反り」は、腕を机の上に置き、手を握らない状態で、指同士が開かない状態で後ろに反ること、「動作無し」は、手を握らない状態で、指同士が開かない状態で机の上に置き、力を抜いて動かさないということである。詳しい手の形は、前回の論文[1]の図を修正した図５を参照してほしい。コンピュータに学習する際、対応する教師データを用意する必要があるため、「握り」は０、「反り」は１、「動作無し」は２としている。

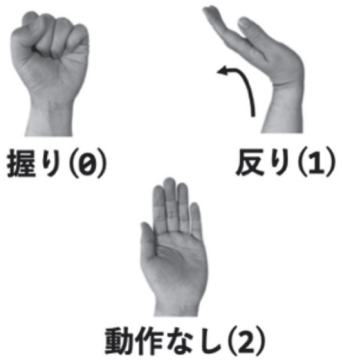

図５　動作区分

２）次に、３つの動作を識別できそうな筋肉を決定する。過去の実験から、経験的に橈側手根屈筋が３つの動作をうまく識別できそうなことが分かっているので、この筋肉を使う。

３）決定した筋肉に、MyoWare筋電センサを貼付し、Arduino Uno R3と接続し、さらにArduino Uno R3をコンピュータと接続する。また、センサデータをシリアル通信を通して、コンピュータに送信するスケッチをArduino Uno R3に書き込む。

なお、MyoWare筋電センサを用い、10秒間でどれくらいデータを取得できるか計算したところ、１秒間

あたり、150個程度であった。コード１に、使用したArduinoのスケッチ（AnalogReadSerial）を掲載する。

```
void setup() {
  //9600bit/sec でシリアル通信を初期化
  Serial.begin(9600);
}

void loop() {
  //analog ピン 0 (A0) の入力を読み取る
  int sensorValue = analog(A0);
  //読み取った値を出力する
  Serial.println(sensorValue);
  delay(1);
}
```

コード１　AnalogReadSerial スケッチ

４）作成したデータ取得プログラムを使って、学習のためのデータを取得し、学習する。「握り」「反り」を、それぞれ12回行い（例えば「握り」「反り」を動作間の時間を少しおいて12回繰り返す）、IEMGのデータを取得する。握る時間間隔を４秒にして12回握ると、「握り」のデータのプロットは図６のように、「反り」は、図７のようなプロットになる。

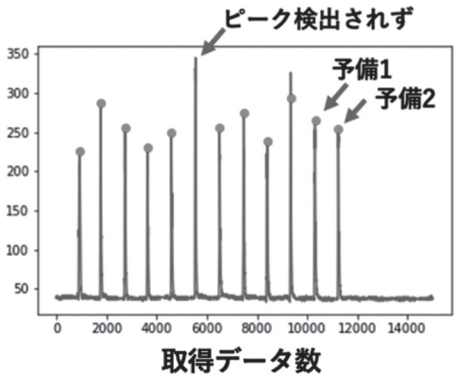

図６　握りデータプロット

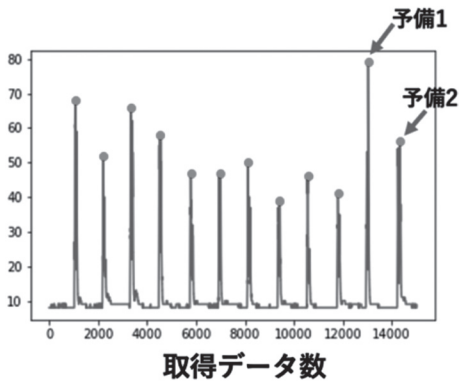

図７　反りデータプロット

全てのデータを学習に使うのではなく、黒丸が付いたピークのときのインデックスにプラスマイナス10した値を元に、リストのスライスを行い、合計20個のデータを得ている。形状は（10, 20）となる。データ取得については、図８、図９に具体的に示した。

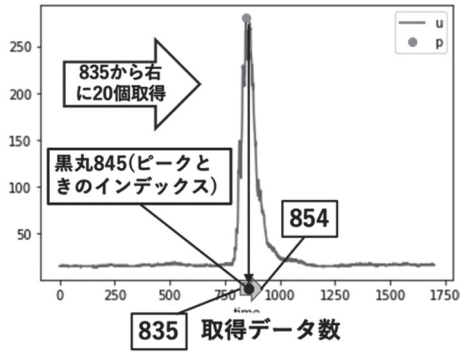

図８　握りデータプロット（１波形拡大）

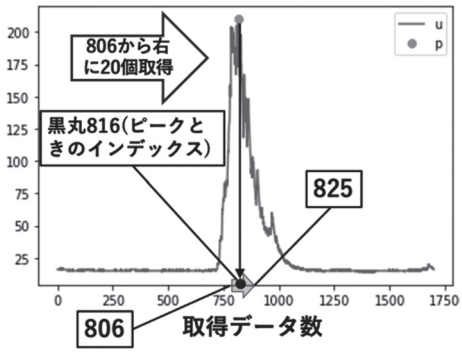

図９　反りデータプロット（１波形拡大）

また、10回ほど、「握り」や「反り」の動作を行っても、データの特性から、ピークを10回中必ず10回得ることが難しいため、10回の動作に予備として２回の動作を加えて12回にしている。実際にデータを取得しているのは、先頭から数えて10個のピーク両側データである。例えば、図６の握りデータプロットで言うと、６番目のデータのピークが検出されていないため、予備１を使用している。

IEMGデータは、予備も考慮すると12回「握り」を行ったときのデータの形状は、先ほど述べたように、(10, 20）となる。その他の「反り」と「動作無し」もデータの形状は（10, 20）となる。

具体的には、「握り」「反り」は、ピーク10個分、ピーク両側データ20個分で（10, 20）である。「動作無し」も（10, 20）のデータである。３つのデータを縦方向に結合すると、（30, 20）の学習データとなる。「動作無し」については、データ取得方法についてふれていないが、その部分は、実験を行うにつれて分かってきたことがあるので、実験の結果で説明する。

５）得られたデータをもとにPyCaretを用いて学習し、最も性能が高いモデルを探索する。その流れを説明する。また、実験に使用しているPyCaret部分を抜き出したPythonのプログラムをコード２に載せる。

```
from pycaret.classification import *
from pycaret.classification import \
ClassificationExperiment

setup(data=x_train,target=t_train)
model = compare_models(fold=6, n_select=1,\
, include = [‘lr’, ‘dummy]’)
df_result = pull()
save_model(model, model_name = ‘best_model’)
final_model = finalize_model(model)
```

コード２　Pythonを用いたPyCaretコード

最初に、PyCaretや機械学習のライブラリをimportする必要がある。本研究の場合、「握り」「反り」「動作無し」を動作識別、つまり分類するので、pycaret.classificationを用いている。setupで学習データと教師データの読み込みを行うと同時に前処理も行う。ヒントとしてsetupのところで、PyCaretのチュートリアルには、引数が 'data' だけ（例えば、'x_train' だけ）で読み込みができるように見えたが、NumPy配列を読み込むには、'data=x_train' とする必要があった。

次に、複数のモデルの学習と比較だが、compare_models()の一行で行う。しかし、この部分も、引数の 'fold=6' を指定しなければうまくいかなかった。利用可能なモデルを表３に載せる。

表３　利用可能なモデル

'lr' - Logistic Regression	'qda' - Quadratic Discriminant Analysis
'knn' - K Neighbors Classifier	'ada' - Ada Boost Classifier
'nb' - Naive Bayes	'gbc' - Gradient Boosting Classifier
'dt' - Decision Tree Classifier	'lda' - Linear Discriminant Analysis
'svm' - SVM - Linear Kernel	'et' - Extra Trees Classifier
'rbfsvm' - SVM - Radial Kernel	'xgboost' - Extreme Gradient Boosting
'gpc' - Gaussian Process Classifier	'lightgbm' - Light Gradient Boosting Machine
'mlp' - MLP Classifier	'catboost' - CatBoost Classifier
'ridge' - Ridge Classifier	
'rf' - Random Forest Classifier	

compare_models()は、この上に挙げたモデル全て（例外あり）を比較することができるが、モデル数を指定して、比較することができる。そのときに、'lr' や 'knn' のように、表に載せた略記を引数に入力する必要がある。

ただし、実験したところ、PyCaretのバージョンのせいか、Extreme Gradient BoostingとCatBoost Classifierは使用することができなかった。

６）compare_models()の比較が終了すると、一番性能が高いモデルを得られる。また、finalize_model()でモデルが完成すると、final_modelオブジェクトのインスタンス変数の_final_estimatorに性能の一番高いモデルが入っているので、今回は、それを使用する。

７）動作識別を行い、カーソルを動かす関数をコード３に載せる。この関数を使いカーソルを右や左に動かす。ただし、カーソルを動かす場所は、簡単のためと場所を統一するためSpyderのエディタペインの中にした。この関数は流れを把握するために一例として掲載した。実際は学習プログラム等が無いと動作しない。後述するが、「動作無し」の識別については、モデルで識別するものを載せた。

５．実験結果

5.1　深層学習の実験結果（PyCaretの実験を行う以前）

5.1.1　筋電位データの時変性と信号出現パターンなど

PyCaretでの実験を行う前に、以前[1][5]と引き続いてセンサ数１つで深層学習を使った動作識別の実験を行っていた。本研究は、PyCaretの研究とも関連があるので、説明することにする。

MyoWare筋電センサのセンサ値は、起床時に、目視の判定だが、きれいな波形が得られ、起床から時間が立つと、波形が乱れた。自律神経系が関連しているのかと考えたが、解決策は得られなかった。

起床後に得られた目視での「握り」のきれいな波形を図10に示す。

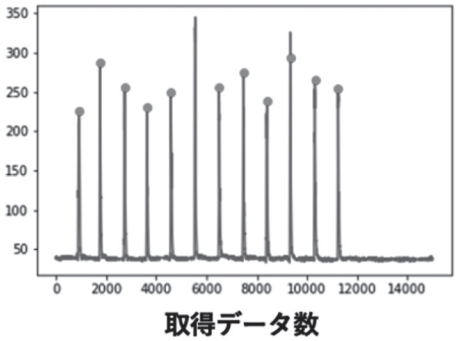

図10　握り起床後

起床から時間が経って計測した波形を図11に示す。

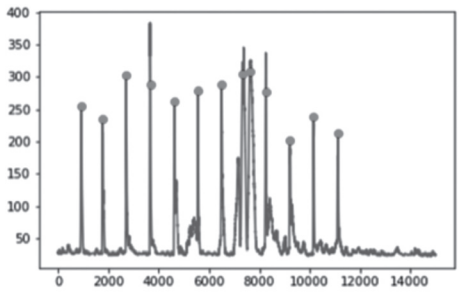

図11　握り起床後時間経過

また、実験結果であるが、放送大学卒業研究論文[5]の時点で、得られた各値を試しに２乗すると、「握り」と「動作無し」の２動作識別がうまく識別できた。しかし、なぜ２乗したらうまくいくか分からなかった。

さらに、深層学習を用いて３動作識別の実験を行った。そして、2022年12月の時点で、３動作識別を行うことができた。また、３動作識別のモデルを用いて、遅延がかなりあったが、カーソルを握ったら右や反ったら左に動かすことができた。しかし、動作識別の精度や付随するカーソルの動きの良さは、得られるデータにかなり依存した。また、データを２動作識別同様、各データを２乗や３乗し試してみたが、３動作識別の精度の向上には寄与しないことが分かった。

次に、重要なこととして、動作識別を行うための信号の出現パターンを説明する。出現パターンは、他の機械学習手法でも同じである。動作識別の信号には特性があり、３動作識別では、「握り」は「握り」と「反り」が混在した信号、「反り」は「反り」のみの信号としてあらわれた。

動作開始から終了までの判定は、「動作無し」から別の信号に切り替わったときを動作の始まりとし、その信号がまた「動作無し」に戻ったときに設定している。

具体的に説明すると、動作識別が以下のように左から右のようになっているとすると（動作識別結果は実際では動作区分にある数値）、

動作無し 反り 反り 握り 握り 反り 動作無し…

（センサを付けている間、識別は続く）

のように信号が連なる場合、「握り」と「反り」が混在している点から、これは「握った」と判定し、

動作無し 反り 反り 反り 反り 反り 動作無し…

（センサを付けている間、識別は続く）

```
import serial
form collections import deque
import keyboard
from pycaret.classification import *
deta_num = 10

def serial_communication_predict(final_model):
  q = deque()
  r = deque(maxlen=2)
  s = deque()
  nigiri_flag = 0
  sori_flag = 0
  c = 0
  flag = 0
  clf = final_model._final_estimator
  ser = serial.Serial('COM3',9600)
  while True:
    val_arduino = ser.readline()
    try:
      val_int = int(val_arduino)
    except ValueError:
      val_int = 0
    q.append(val_int)
    print(val_int)
    if c >= 2*data_num:
      q.popleft()
      if c % 5 == 0:
        y = clf.predict([q])
        y = y.tolist()
        r.append(y[0])

        if len(r) > 2:
          if r[0] == 2 and r[1] == 0 or r[1] ¥
== 1:
            flag = 1
            s.append(y[0])

          if flag:
            s.append(y[0])
            print("判別した値->"my[0])

          if r[1] == 2:
            flag = 0
            nigiri_flag = 1
            sori_flag = 1

            a = set(s)
            print("set->",a)
            if nigiri_flag and (0 in a) == True¥
and len(a) == 3:
              print("握っています")
              keyboard.press_and_release¥
("right")
              nigiri_flag = 0
            if sori_flag and (1 in a) and ¥
len(a) == 2:
              print("反っています")
              keyboard.press_and_release("left")
              sori_flag = 0
            s.clear()
c += 1
```

コード３　動作識別とカーソルを動かすプログラム

のような信号の連なりなら、素直に「反り」と判定している。なぜそう動作を決めるかは、「握り」や「反り」のとき、そういう信号の連なりになるからである。

5.1.2　生波形によるデータ取得実験

深層学習での研究は、IEMGの代わりに生波形を使った実験も考え、生波形のデータ取得を行ってみた。そのデータ取得基準は、[6]にあるように、「筋活動の開始や終了を決定する方法には、安静時の振幅から求められる平均した筋活動±標準偏差（SD）の２倍もしくは（もしくは３倍）を超えた時点とすることがある。」とあり、筋活動の開始・終了をこの方法で判定した。

コード４のような、標準偏差２倍を超えたら、動作開始・終了を判定するプログラムを作った。

ただし、以下のプログラムは関連部分だけ抜き出しているので、省略している。if文の条件部分を注目してみてほしい。

```
if next_i > mean_rest + 2*stddev_rest\
or next_i < mean_rest - 2*stddev_rest\
 and r[0] == 1:
  r_data_active_tmp.append(next_i)
  r.append(1)
elif next_i < mean_rest + 2*stddev_rest \
or next_i > mean_rest - s*stddev_rest: \
  r.append(0)
```

コード４　標準偏差による動作開始終了判別

計算方法は省いてあるが、next_iが筋電位の値で、mean_restが安静時振幅の平均、stddev_restが標準偏差である。

このプログラムを実行してみたが、筋電位波形の振幅の大きさの問題で、動作の始まりや終了をうまく判定できなかった。例えば、５回手を握る動作をして、データが５個取れるはずが、何十個ものデータが取れる不具合で実用には至らなかった。

以上が、深層学習を用いた動作識別の研究の一部であるが、その問題点を述べる。深層学習では重みの数が増すにつれて、動作識別に時間がかかり、カーソルの移動に関して、遅延がおこることである。また、重みの数が増えれば、学習に時間がかかってしまう。深層学習は、性能が高く大量のデータを用いれば、再学習の回数が減り精度も高い良い手法だが、現在の実験環境では、学習に時間がかかり、カーソルの移動の遅延が起こる。それらを解消するために、新たに別の機械学習手法に目を向け、網羅的に探索してくれるPyCaretを用いて実験を行った。

5.2　様々な機械学習を用いた手の動作識別の実験結果

5.2.1　３動作識別実験とハイブリッドの手法

PyCaretのインストールでは、実験方法にもある通り、Pythonのバージョンが指定されていること、ライブラリの依存関係の解消が必要で、インストールし、使える状態にするのに１週間かかった。

最初に、得られるデータ数をピーク両側600個にして、３動作識別の実験を行った。2023年３月26日～27日にかけて、PyCaretを用い、「握り」「反り」「動作無し」を学習・識別するプログラムを作成した。「握り」を（10, 20）のデータ、「反り」を（10, 20）のデータ、「動作無し」を（10, 20）のデータを学習させ、50回ほど、カーソル動作を確認したが、正確に動いたのは２回程度であった。実験後に分かったことだが、うまくいかなかった理由は「動作無し」を学習させるかさせないかにあった。その部分について以下に詳しく述べる。

学習させる方法として、コード５を使用して「動作無し」を学習データに組み込んだ。「握り」「反り」「動作無し」の３動作識別を行うといった流れである。

```
executor = ThreadPoolExecutor(2)
executor.submit(dousanashi_timer)
future_dousanashi \
= executor.submit(dousanashi_get_data)
r4 = future_dousanashi.result()
r4 = np.array(r4)
r4 = Hampel(r4,10,thr=2)
r4 = get_dousanashi_data(r4[0])
```

コード５　「動作無し」データ取得流れ

また、「動作無し」のデータを人工的に作り出し、学習データに用いることも試みた。

次は、全く学習させない方法である。「動作無し」は、ノイズはあるものの、ほぼ直線的でフラットなデータなので、「動作無し」は、学習に組み込まず、「握り」と「反り」の２動作識別として、「動作無し」の判定は、識別時に工夫しようと言う方法である。

結論を言うと、「動作無し」を学習させる方法とさせない方法で、PyCaretを用いて、動作識別がうまくいくのは後者の「動作無し」を学習させない方法である。

具体的には、「動作無し」は学習せず、動作識別の段階で、閾値を手動で設定し、その閾値以下を「動作無し」とすることである。機械学習と人の手を使うハイブリット方法である。

なぜ、これがうまくいくのかを考えたが、「動作無し」は信号がフラットであること、さらに、「握り」と「反り」から得られる信号は、「動作無し」の閾値より高いことにある。また、センサの具合で、「動作無し」の値の大きさが微妙に変化してしまうため、学習に組み込む方法ではこの変化を捉えきれなかったのではないだろう

か。

以上から、動作識別がうまくいく、機械学習で識別する動作としない動作を決める手法、つまり、ハイブリッドの手法を採用した。

なお、「動作無し」の閾値をどうやって決めるかは、現段階では、「動作無し」のときに出力される値を目測で、観察し、その値を閾値としている。

その部分を改良するためには、センサの値から、例えば何個か値を取り出し平均をとって、それを閾値に決めるといった方法も考えられる。

例をあげると、「動作無し」の値が以下であるとすると、

38 38 38 39 40 39 38 38 39・・・

（センサーを付けている間値は続く）

簡単のため少ないデータ数だが、例えばセンサ値が十分安定したとして、３番目から７番目の平均を取ると、(38＋39＋40＋39＋38)/5＝38.8となるので、閾値を例えば、38.8とする手法が考えられる。

5.2.2　最適な機械学習モデルの調査と補足実験

最適な機械学習モデルの調査について説明する。説明の流れとして、600個データを得た時の最適なモデルの探索、「差分」と「差分の差分」を用いた実験、20個データを使ったときのモデルの比較となっている。ハイブリッドの手法の開発以前からデータ数を変化させて性能の高いモデルを探索する実験は行っていたので、600個のデータでの探索と「差分」と「差分の差分」の実験は、ハイブリッドの手法以前のモデルでの比較結果、20個のデータでの実験はカーソルが正確に動くハイブリッド後のものを提示した。

まず、ピーク両側600個のデータ得ていたときの各モデルの順位を図12に掲載する。紙面の都合上、省略して載せている。見えにくいが、左からAccuracy、AUC、Recallで、実際には８つの指標からなる。（見えていない）残りの指標は左から順に、Precision、F1、Kappa、MCC、TT（Sec）である。

インデックス	Model	Accuracy	AUC	Recall
lr	Logistic Regression	1	1	1
knn	K Neighbors Classifier	1	1	1
dt	Decision Tree Classifier	1	1	1
rf	Random Forest Classifier	1	1	1
ada	Ada Boost Classifier	1	1	1
gbc	Gradient Boosting Classifier	1	1	1
et	Extra Trees Classifier	1	1	1
lda	Linear Discriminant Analysis	0.9444	1	0.9444
svm	SVM - Linear Kernel	0.8611	0	0.8611

図12　600データでの解析結果

この表から、ベストなモデルは、Logistic Regressionとなっており、TT（Sec）以外、すべての指標で1（100%）の値である。

ただし、このときは、「動作無し」を学習させていたので、指標で1（100%）が出たとしても、3動作識別を行い、カーソルの操作はうまくいかなかった。しかし、過学習している可能性があるが、指標の値は高く、期待を持てる結果であった。

次に、3動作識別を正確に行い、カーソルの動きを改良するため、意味のあるデータ量を増やす方法を考えた。増減の情報と変曲点の情報である。それは、「差分」と「差分の差分」である。これらは、`np.diff()`を使えば簡単に実現できる。「差分」と「差分の差分」の情報を、ピーク両側600個のデータから作り出し、3動作全部で（30, 1797）のデータを使いPyCaretで解析してみた。図13が結果である。

インデックス	Model	Accuracy	AUC	Recall
dt	Decision Tree Classifier	0.8472	0.8889	0.8472
gbc	Gradient Boosting Classifier	0.6528	0.8125	0.6528
et	Extra Trees Classifier	0.6389	0.8958	0.6389
rf	Random Forest Classifier	0.5417	0.8681	0.5417
svm	SVM - Linear Kernel	0.5139	0	0.5139
lr	Logistic Regression	0.4722	0.7014	0.4722
ridge	Ridge Classifier	0.4306	0	0.4306
qda	Quadratic Discriminant Analysis	0.4306	0.5694	0.4306
ada	Ada Boost Classifier	0.4028	0.7708	0.4028

図13 「差分」や「差分の差分」情報追加解析結果

「差分」と「差分の差分」の情報を入れると、逆にそれぞれの指標の値が下がってしまい、動作識別を行うと、様々な動作識別結果が混ざったような信号となった。

最後に、カーソルの動きがうまくいく、ハイブリッドの手法で、データ数20個のときの表を示したいが、記録が無く、取り直そうとしたが、現在、MyoWare筋電センサの調子が悪いため、載せることができなかった。ただ、傾向として、Logistic Regressionがベストのモデルとなり、その次がK Neighbors Classifier、Decision Tree Classifierとなることが多い。K Neighbors Classifier以外の２つのモデルは動作が非常に軽く、筋電義手やカーソルを動かすデバイスを作る際に使いやすいのではないかと考えられる。筆者の腕から測定したデータではあるが、実験結果として、３動作識別において最適なモデルは、Logistic Regressionである。

一方、このモデルの中で、リアルタイムで機械学習の予測を行うのに適していないモデルがあり、カーソルの動きに遅延がありすぎて、使用に難があるモデルも存在する。そのモデルを挙げると、先ほどのK Neighbors

Classifierに加えて、Random Forest Classifier、Extra Trees Classifier、Ada Boost Classifierであ る。K Neighbors Classifierは、20個のデータでも、精度などの指標の値は高いが、かなり重たい処理となる。

機械学習手法について調べたものを表４に載せる。記録がある部分だけ載せる。ない部分は斜線にした。

表４　機械学習モデル間比較

	遅延	識別力	安定性
Logistic Regression	少ない	高い	高い
K Neighbors Classifier	大きい	高い	高い
Naive Bayes	少ない	低い	
Decision Tree Classifier	少ない	高い	高い
SVM-Linear Kernel	少ない	低い	低い
SVM-Radial Kernel	少ない		低い
Gaussian Process Classifier	少ない		
MLP Classifier	少ない		
Ridge Classifier	少ない	普通	低い
Random Forest Classifier	大きい	高い	高い
Quadratic Discriminant Analysis	少ない	低い	低い
Ada Boost Classifier	大きい	低い	
Gradient Boosting Classifier	少ない	普通	
Linear Discriminant Analysis	少ない	低い	低い
Extra Trees Classifier	大きい	低い	
Light Gradient Boosting Machine	測定不能	低い	

この研究結果において、識別力は筋電位のデータに強く依存している。遅延に関しては、計算量のような機械学習手法の特性だと考えられるので、実験環境によらず、このような結果になると考えられる。

６．まとめ

今回、PyCaretを使用することにより、筋電位という時系列データに関して、様々な機械学習手法を網羅的に探索し、3動作識別において、筆者の筋電位データだが、最も性能が良いモデルを見つけ出すことができた。結果として、Logistic Regressionが、動作識別率が高く、短時間（2-3分程度）のカーソル動作実験ではあるが、カーソルの動作を遅延が少なく、「握り」「反り」「動作無し」をほぼ正確に動かせることが分かった。

さらに、前回の研究と引き続いて、センサ１つでも様々な機能を持たせたカーソルを動かすデバイスを作成できることを示した。センサの数を制限することで低コスト化が見込まれる。

現在、MyoWare筋電センサの調子が悪く、スポーツセンシング社の湿式タイプの筋電センサを導入して、ARVを使い、リアルタイム動作識別の実験を行っている。

現在では、このセンサを使いMyoWare筋電センサ同様、３動作識別が正確に行えることが分かっている。ただし、このセンサでは「動作無し」はハイブリッドの方法を使うより学習させた方がうまくいく。

また、スポーツセンシング社のセンサは、ワイヤレスでコンピュータに筋電位情報を送信できるということや、実験を繰り返したところ値の信頼性が高いということから、これから、この利点を活かした研究を行いたいと考えている。

最後に、カーソルを動かす先の目標として、多機能の筋電義手の作成がある。筋電義手は、装飾用の義手と違って、自分の動かしたい意図に沿って動かすことが可能なので、これからますます、需要が高まると考えられる。筋電義手を必要とされている方のためにも、カーソルの動作も含めて、手の動作識別のさらなる改良を進め、その応用としての筋電義手の開発の助けとなりたい。

参考文献

［1］伊藤正剛・北本卓也（2023）「筋電位を利用した深層学習による手の動作識別について」　山口大学教育学部研究論叢　第72巻 PP.227-235

［2］中村智史・奥野大・淡野公一・田村宏樹・外山貴子「k-NN法を用いた表面筋電位による指動作識別」宮崎大学工学部紀要 第36号 P73-P80

［3］吉川雅博，三河正彦，田中和世「筋電位を利用したサポートベクターマシンによる手のリアルタイム動作識別」電子情報通信学会論文誌D, vol.J92-D, no.1, pp.93-103, 2009.

［4］山野井祐介（2018 年度）『筋電義手制御のための信号特徴の時変性を考慮したビッグデータを用いた手指動作推定法』横浜国立大学大学院工学府システム統合工学専攻機械システム工学コース博士論文

［5］伊藤正剛（2022年度）『筋電位を利用した深層学習における手の動作識別について』放送大学 卒業研究論文

［6］編集 加藤浩・山本澄子 執筆 勝平純司・田中惣治・井川達也・中谷知生・加藤浩『臨床にいかす表面筋電図』医学書院

［7］清水健吾，大村基将，島袋舞子，兼宗進「Arduinoを利用した筋電測定システムの提案」情報処理学会研究報告 Vol.2016-CE-136 No.7 P1-P6