**오픈 소스 전문 프로젝트**

**Report #06 데이터 분석 결과**

****

|  |  |
| --- | --- |
| **학과** | **컴퓨터공학과** |
| **학번** | **2018037002 손지현**  **2018037004 이진희**  **2018037025 이도희**  **2018037026 윤소영**  **2018037054 신지애** |
| **교수님** | **박수창 교수님** |
| **제출일** | **2020/06/09** |

**목 차**

1. **센서 사용 및 수집 정보**
2. **MLP 사용**

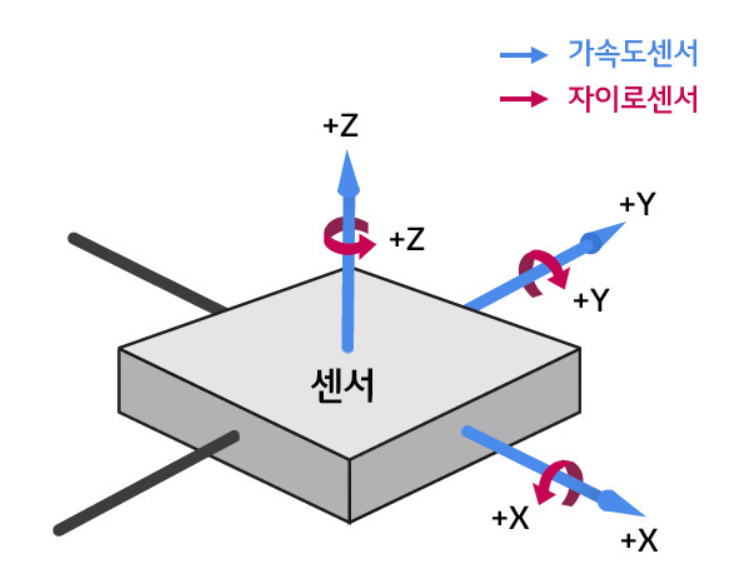
**1. 센서 사용 및 수집 정보**

**STEP 1) 센서 활용**

1. **가속도 센서 정의**

* 물체의 속도의 변화(가속도)나 운동량의 변화(충격량) 등을 측정하는 센서이다. 물체의 운동 상태를 상세하게 감지할 수 있기 때문에 다양한 분야에서 필수적으로 사용되며, 그 활용도 또한 높다.

1. **가속도 센서 이해**



스마트폰의 기울기에 따라 가속도 센서 x/y/z축의 측정값이 변화함.

* 스마트폰이 주머니/가방 속에 있을 경우, 다양한 방향(x/y/z축)으로 가속도 센서 측정값이 나올 수 있기 때문에, 스마트폰 기울기를 다르게 하여 가속도 센서 측정값을 수집함.

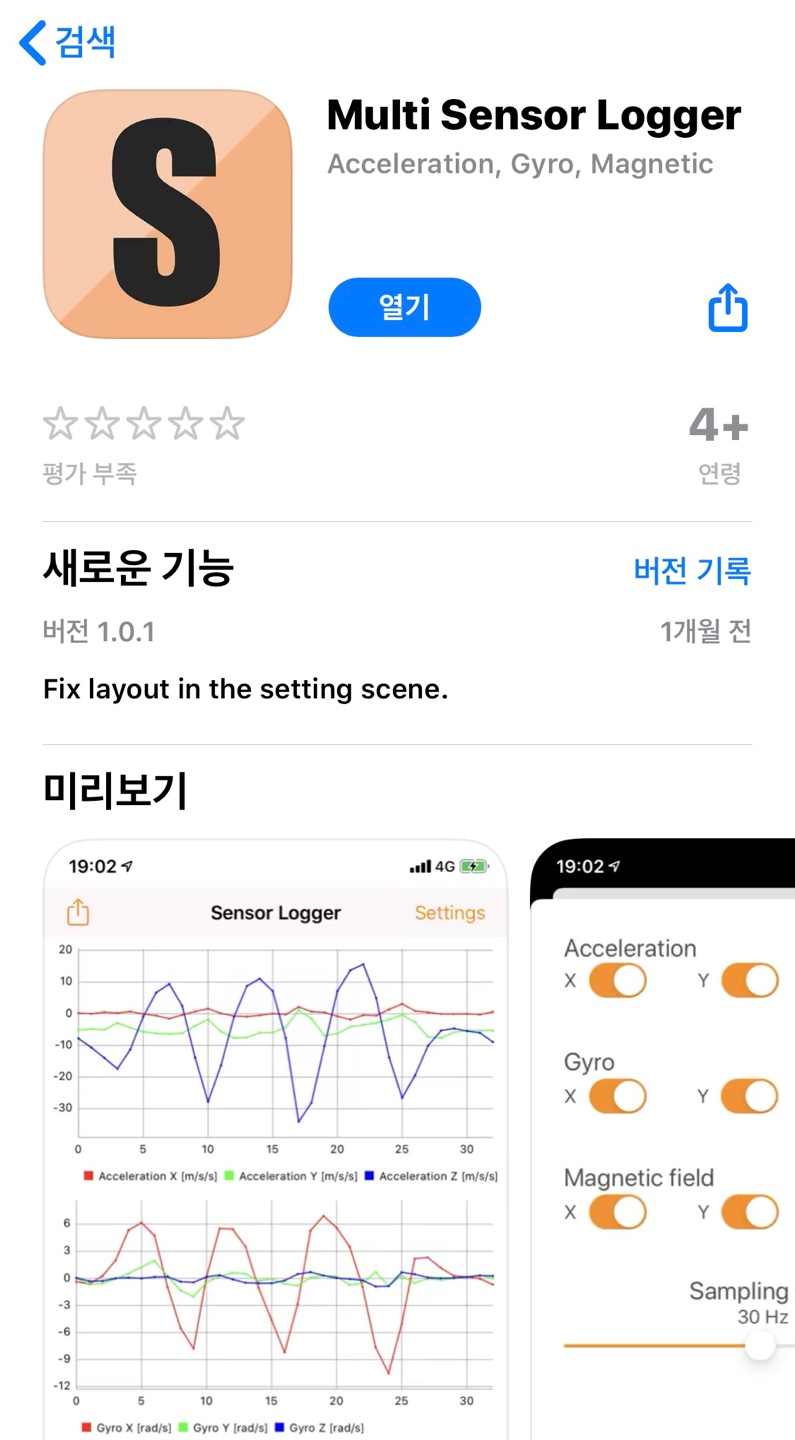
1. **가속도 센서 활용**

* 포켓폴리스는 스마트폰을 주머니/가방 속에 넣고 귀가하는 상황에서 위험 상황으로 감지될 경우, 주저앉음으로써 저장된 위치 정보를 전송 및 신고하는 치안 관련 어플.
* 즉 주저앉았음을 인식하기 위해, 물체의 운동 상태를 상세하게 감지할 수 있는 스마트폰에 내장된 가속도 센서를 활용.

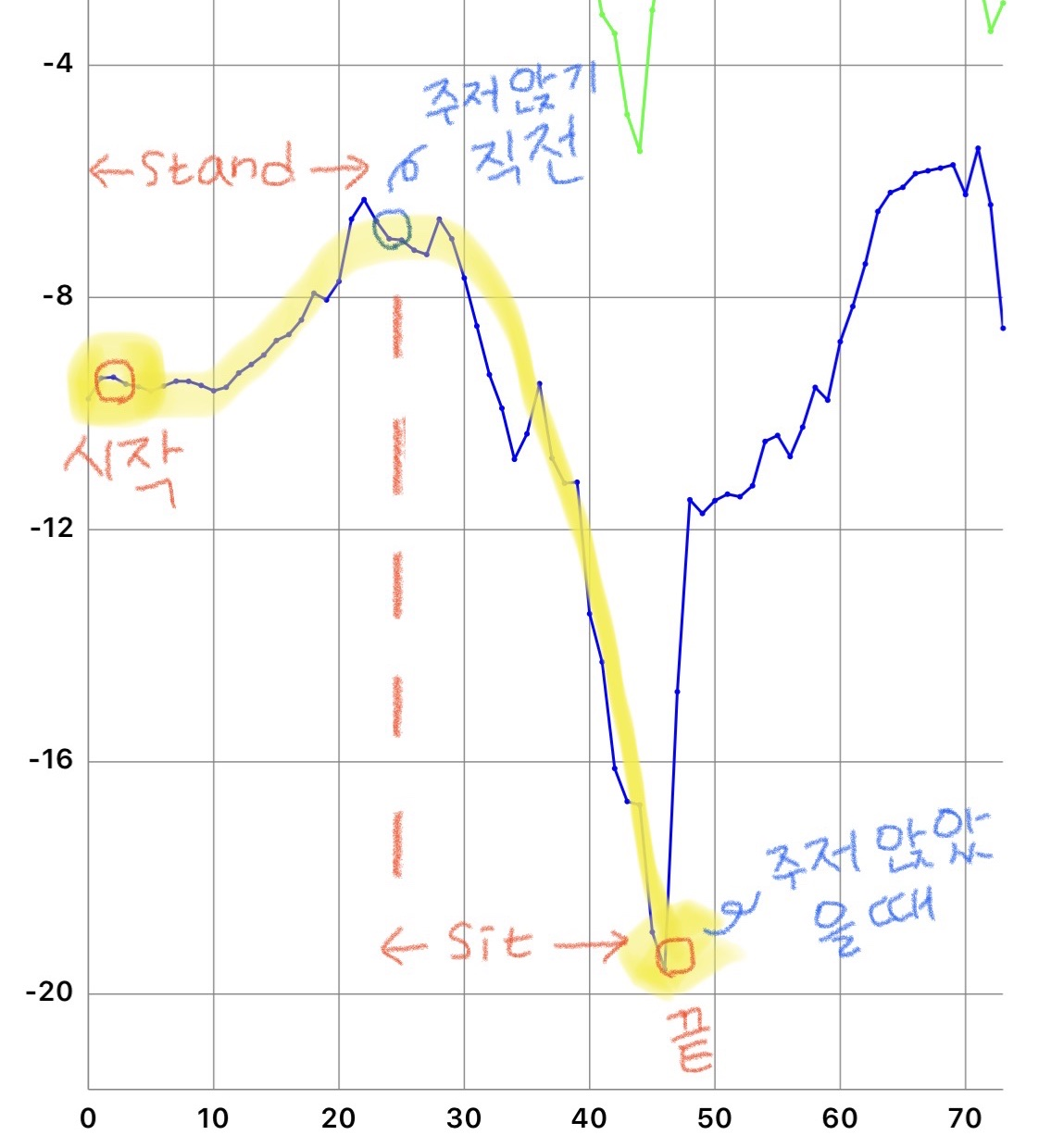
**STEP 2) 데이터 수집 정보**

1. **데이터 측정**

* 스마트폰 앱 Multi Sensor Logger을 통해 데이터를 측정.
* **Multi Sensor Logger**는 가속도, 자이로 스코프 및 자기장을 동시에 측정할 수 있는 로거 응용 프로그램. 이 앱은 측정 값을 CSV 파일로 출력 가능함.

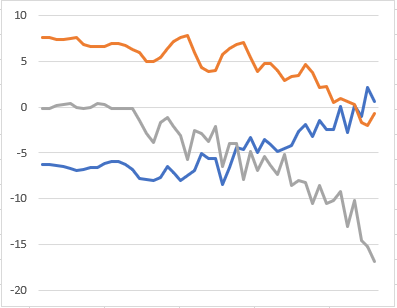


1. **데이터 수집**



* 가속도 그래프를 보았을 때, **서 있을 때의 값부터 주저 앉았을 때까지의 값**을 도출하여 CSV 파일로 데이터 수집
* 그 구간에서 시간에 따른 변화량 데이터 분석 및 머신 러닝 기법을 이용하여 사용자가 주저앉았을 때, 위험 상황으로 감지하여 서비스를 실행할 수 있도록 활용.

1. **데이터 수집 정보**



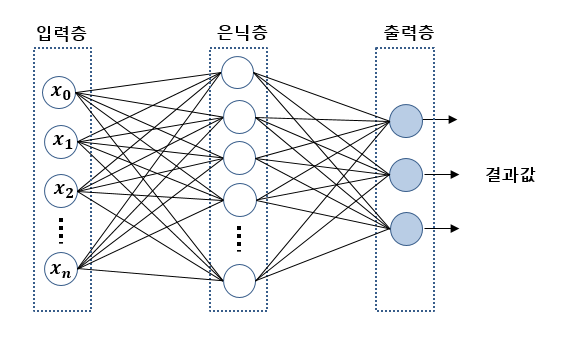
**<한번 앉았을 때의 그래프>**

* 주저앉기 시작할 때부터 주저앉았을 때까지의 데이터를 1000개 수집함.

**2. MLP 사용**

**MLP (다중 퍼셉트론, Multilayer Perceptron)**

: 단층 퍼셉트론의 비선형적으로 분리되는 데이터에 대해서는 제대로 된 학습이 불가능하다는 한계를 극복하기 위해 고안되었다. 다층 퍼셉트론은 **입력층과 출력층 사이에 하나 이상의 중간층을 두어 비선형적으로 분리되는 데이터에 대해서도 학습이 가능**하도록 한다.

단층 퍼셉트론의 동작 원리는 활성 함수가 내놓는 결과값이 실제값과 오류가 최소가 되도록 입력층에서 전달되는 가중치의 값을 결정하는 것인데, 다층 퍼셉트론의 동작원리 또한 이와 크게 다를 바 없지만 다층 퍼셉트론은 은닉층과 출력층에 존재하는 활성함수가 여러 개고, 이에 다른 가중치도 여러 개다.

**다층 퍼셉트론의 동작 순서**

1. 각 층에서 가중치를 임의의 값(일반적으로 0에 가까운 값)으로 설정한다. 각 층에서 바이어스 값은 1로 설정한다.

2. 하나의 트레이닝 데이터에 대해서 각 층에서의 순입력 함수값을 계산하고 최종적으로 활성 함수에 의한 출력값을 계산한다.

3. 출력층의 활성 함수에 의한 결과값과 실제값이 허용 오차 이내가 되도록 각층에서의 가중치를 업데이터 한다.

4. 모든 트레이닝 데이터에 대해서 출력층의 활성 함수에 의한 결과값과 실제값이 허용 오차 이내가 되면 학습을 종료한다.

출처: <https://m.blog.naver.com/samsjang/221030487369>

1. **목적:**

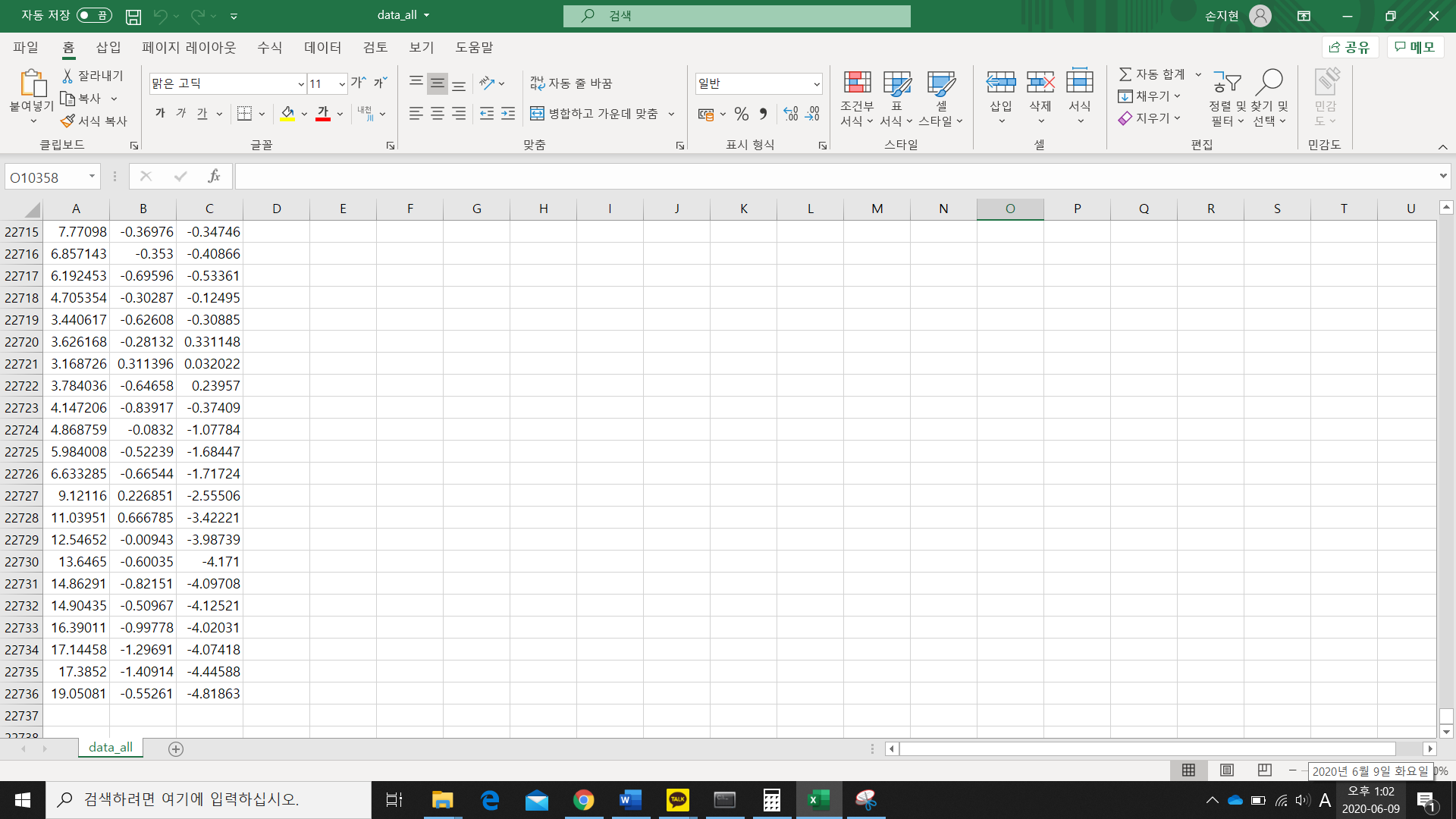
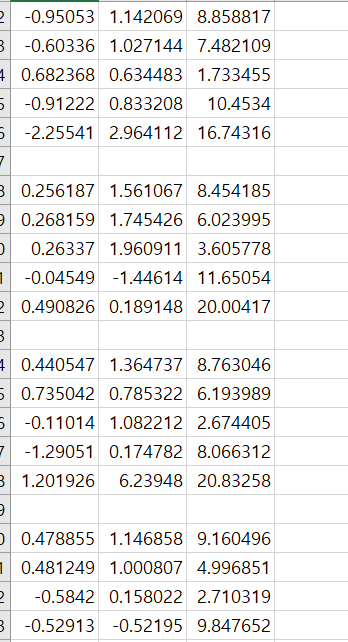
가속센서를 통해 얻은 데이터 값을 입력으로 하고 이진분류모델을 사용하여 **사용자가 주저 앉았는가를 예측**

1. **데이터 및 전처리과정:**

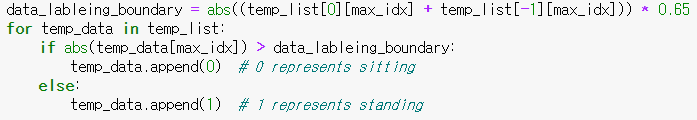
* **데이터셋 구성 필요조건**

MLP는 지도학습에 기반한 모델로서 학습을 위해서는 입력데이터와 출력층의 정답여부를 확인할 수 있는 라벨이 필요하다.

데이터는 **입력 데이터 + 라벨 데이터**로 구성된다.

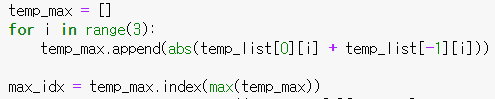
* **데이터셋 수집 및 라벨링**
  + 1. 가속 센서의 연속적인 측정값을 저장한 데이터가 있다.

**한번 앉을 때를 기준으로 나눈다**

* + 1. 연속된 데이터에서 앉음과 서있음을 구별하기위해 측정값의 시작과 끝 사이의 특정 지점을 기준으로 라벨링을 진행하도록 전처리를 함.

최저점과 최고점의 65%지점을 섬과 앉음의 구분점으로 사용

* + 1. 측정값의 양상이 상황에 따라 음수일때도 있고 양수일때도 있다. 따라서 구분점과 축의 데이터를 절대값 처리 후 비교
    2. 주저앉았을 때 가속 센서의 측정값이 크게 변화하는 양상을 보인다. 즉 축별로 측정된 값의 절대값이 커진다는 점을 참고하여 라벨링을 위한 특정지점을 정하는 데 활용하였다.
    3. 각 축의 변화량(기울기)을 체크하여 가장 큰 값이 앉고 서있음을 구별하도록 하는 구분자이므로 이를 특이점을 결정하는데 사용

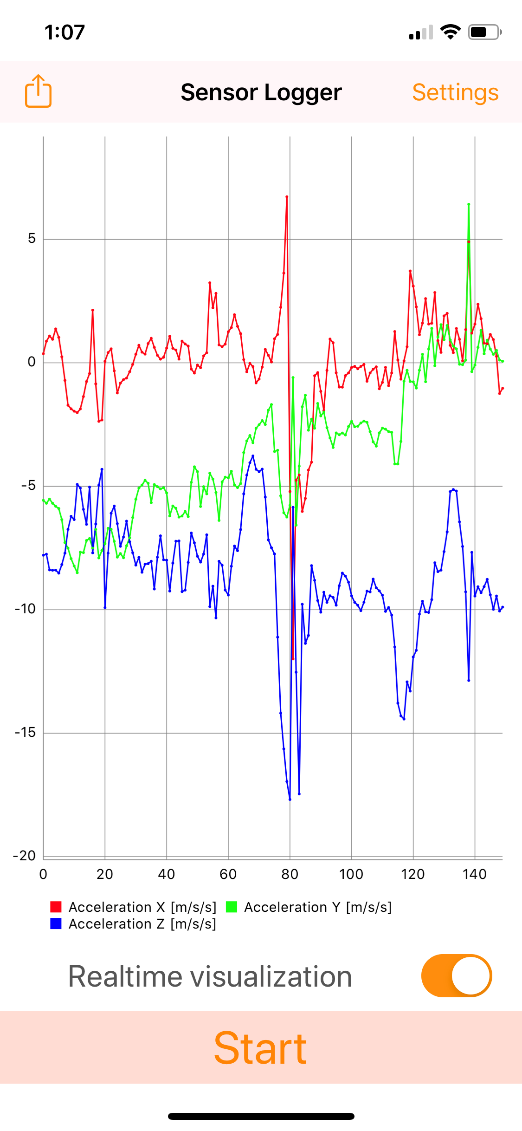


* **데이터 활용**

: 학습데이터셋과 테스트데이터셋은 70:30의 비율로 나누어 사용하였다.

* **데이터 결과**

: Standing status: 1/ Sitting status: 0



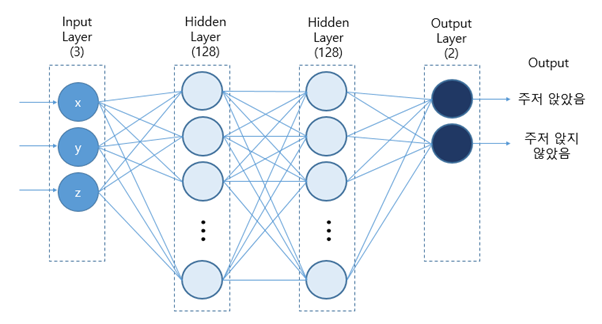
Input

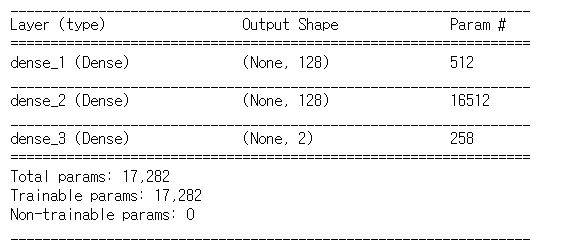
측정된 가속도 센서의 x, y, z 값

Output

사용자가 주저 앉았음 여부

1. **MLP 구성**

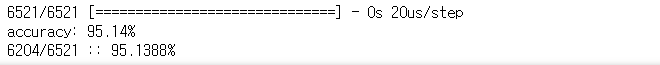




**▶ 파이썬을 통해 구현한 MLP모델 Summury**

1. **MLP 학습 결과**

* 분류 모델을 통한 예측 성능 -> **정확도: 95.14%**

(학습 환경: epoch 150, batch size: 100)

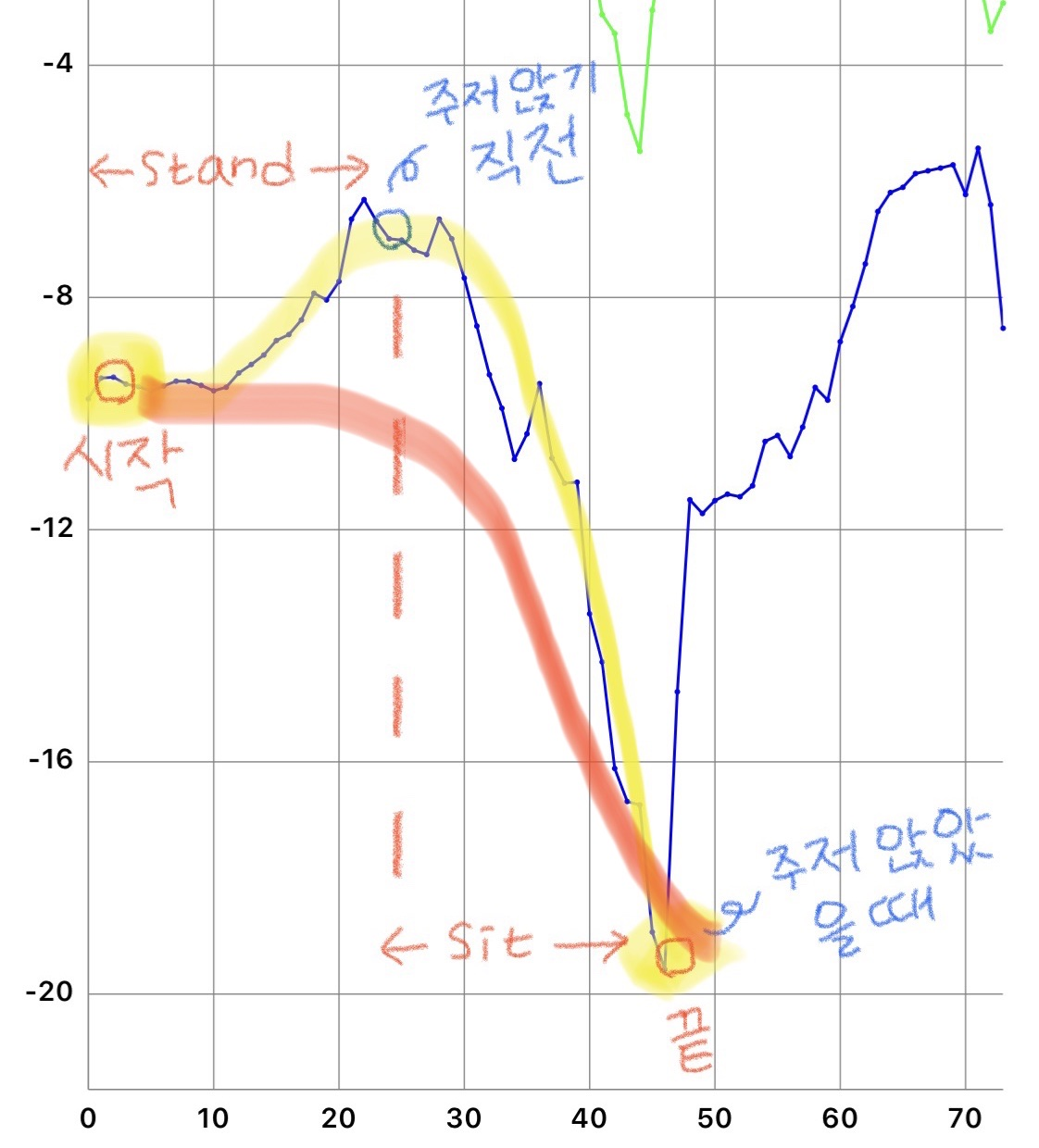
▶ 6521개의 학습 데이터 중 6204개

1. **고찰**

데이터셋 전처리과정에서 라벨링을 진행할 때 수집된 데이터가 연속적 값을 가져왔기 때문에 앉은상태, 선상태의 두개의 값으로 특정하기 어려웠다.

지도학습의 특성상 라벨 값이 있어야 학습 모델을 구성할 수 있기 때문에 이를 최댓값과 최솟값사이의 특정 값으로 사용하였는데, 이는 실제 앉은 상태와 선 상태를 정확히 입력데이터와 비교하여 라벨링한 것이 아니기 때문에 이를 보완할 방법이 필요하다. 예를 들어

1. 데이터를 수집할 때 앉은 상태인지 선 상태인지를 기록하는 방법
2. 가속도계 외에 고도계를 사용하여 앉은 상태의 고도를 측정하여 라벨링 시 참고하는 방법



1. **Source Code**

import csv  
import random

import numpy  
from keras import Sequential  
from keras.layers import Dense  
from keras.utils import np\_utils  
  
  
csv\_file = './data\_all.csv'  
# This should be consisted  
  
epoch = 150  
batch\_size = 100  
  
# Load data  
total\_dataset = []  
file = open(csv\_file, 'r', encoding='utf-8-sig')  
csv\_reader = csv.reader(file)  
temp\_list = []  
for line in csv\_reader:  
 if line[0] == '':  
   
 temp\_max = []  
 for i in range(3):  
 temp\_max.append(abs(temp\_list[0][i] + temp\_list[-1][i]))  
  
 max\_idx = temp\_max.index(max(temp\_max))  
 data\_lableing\_boundary = abs((temp\_list[0][max\_idx] + temp\_list[-1][max\_idx])) \* 0.65 # last float type value is boundary params  
 for temp\_data in temp\_list:  
 if abs (temp\_data[max\_idx]) < data\_lableing\_boundary:  
 temp\_data.append(0) # 0 represents sitting  
 else:  
 temp\_data.append(1) # 1 represents standing  
 total\_dataset.append(temp\_data)  
 temp\_list = []  
 else:  
 line = [float(item) for item in line]  
 temp\_list.append(line)  
  
random.shuffle(total\_dataset)  
total\_dataset = numpy.array(total\_dataset)  
  
# It represent training and testing dataset percentage n : (100-n)  
dataset\_division\_percentage = 70  
# Dataset Count  
total\_dataset\_count = len(total\_dataset)  
training\_dataset\_count = int(total\_dataset\_count \* dataset\_division\_percentage \* 0.01)  
test\_dataset\_count = total\_dataset\_count - training\_dataset\_count  
  
train\_x = total\_dataset[0:training\_dataset\_count, 0:3]  
train\_y = total\_dataset[0:training\_dataset\_count, 3]  
train\_y = np\_utils.to\_categorical(train\_y)  
  
test\_x = total\_dataset[training\_dataset\_count:-1, 0:3]  
test\_y = total\_dataset[training\_dataset\_count:-1, 3]  
test\_y = np\_utils.to\_categorical(test\_y)  
  
model = Sequential()  
model.add(Dense(128, input\_dim=3, activation='sigmoid'))  
model.add(Dense(128, activation='sigmoid'))  
model.add(Dense(2, activation='softmax'))  
model.compile(loss='categorical\_crossentropy', optimizer='rmsprop', metrics=['accuracy'])  
  
print(model.summary())  
# Train  
  
model.fit(train\_x, train\_y, epochs=epoch, batch\_size=batch\_size, verbose=2, shuffle=False) # 50 is X.shape[0]  
# model.reset\_states()  
  
# Validation  
scores = model.evaluate(test\_x, test\_y, batch\_size=batch\_size)  
print("%s: %.2f%%" % (model.metrics\_names[1], scores[1] \* 100))  
  
# Test  
i = 0  
cnt = 0  
for output in model.predict(test\_x, batch\_size):  
 if numpy.argmax(test\_y[i]) == numpy.argmax(output):  
 # print(i)  
 cnt += 1  
 i += 1  
  
# Results  
print(str(cnt) + '/' + str(i) + ' :: ' + str(round(cnt / i, 6) \* 100) + "%")