프로젝트 최종 보고

Weltried

2022년 1학기
산학협력프로젝트(3211)
이희연 교수님

짱잘하조킹받조어쩔티비조
201811182 박원준
201811165 김영길
202011274 김주혜
201911186 신지수
201912550 윤진성
201711430 차준혁

목차

0 프로젝트 요약

1 필요성 및 현황

2 목표

3 방법

4 효과

5 결과물

6 참여인원

7 평가 방법

1) 자체 평가

2) 사용성 평가

0 프로젝트 요약

프로젝트 Weltried는 자세 교정을 위한 어플리케이션 개발 프로젝트입니다. 클라이언트, 서버, 인공지능 기술이 복합적으로 작용합니다.

1 필요성 및 현황

평균 연령이 증가하는 시대적 배경에 따라 하나 뿐인 신체를 올바르게 오래 사용하자는 움직임이 증가하고 있습니다. 이에 힘 입어 자세 교정 장치의 특허 수는 해마다 증가하고 있습니다. 실제로 의자, 방석 등 직접적으로 자세를 교정해주는 하드웨어 제품들은 시중에서 많이 볼 수 있습니다.

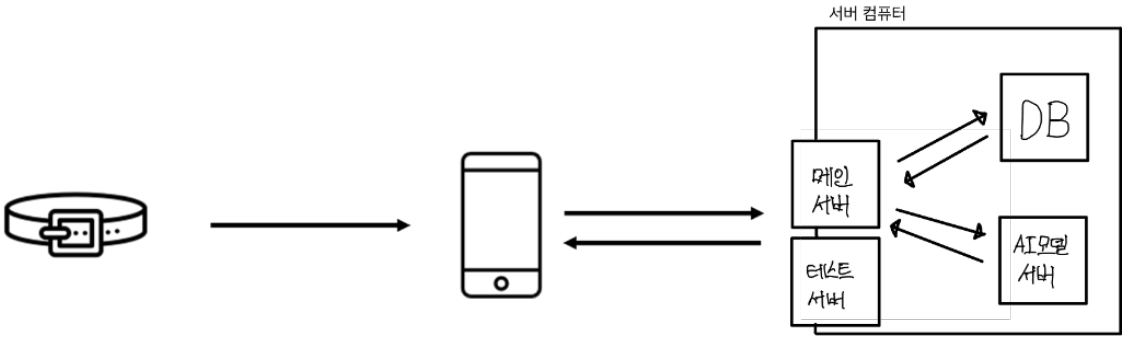
하지만 실생활에서 사용자의 자세를 측정하고 피드백을 줄 수 있는 제품, 다시 말해 소프트웨어적인 해결책을 제시하는 서비스인 디지털 헬스, 디지털 치료제 등의 시장은 앞으로 개척해나가야 합니다.

2 목표

Weltried는 사용자의 리얼 월드 데이터를 수집해, 사용자와 담당 의사가 자신의 자세를 모니터링하고 의식적으로 자세 교정에 힘쓰도록 하는 등의 목표를 가지고 있습니다.

3 방법

Weltried는 다음과 같은 시스템 구조를 갖습니다.

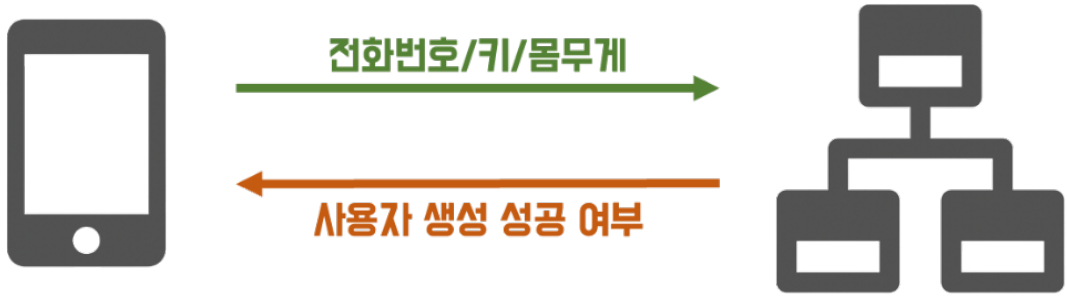


벨트는 웰트(주)의 웨어러블 제품, 스마트 벨트를 이용합니다. 벨트에는 여러 센서가 내장되어 있는데, Weltried는 그 중 가속도 센서를 이용합니다. 사용자가 벨트를 착용하고 앉아 있으면, 벨트는 Weltried 어플리케이션으로 x,y,z 가속도 센서 값을 전송합니다. 벨트와 스마트폰의 통신은 블루투스 통신을 이용합니다. 가속도 센서 값은 1Hz 주기로 받아옵니다. 스마트폰은 Weltried의 서버와 통신하며 사용자의 개인정보나 가속도 센서 값, 인공지능 모델의 결과값 등을 주고 받습니다. 스마트폰과 서버와의 통신은 http 프로토콜을 이용합니다.

Weltried의 현재 스토리는 3가지입니다.

1) 사용자 식별 정보 및 개인 정보 입력

사용자는 어플 설치 시, 전화번호와 키, 몸무게 등을 입력합니다. 해당 정보는 서버의 데이터베이스에 저장됩니다.



2) 사용자의 현 자세 측정 및 출력

1. 측정 시작 버튼을 누르면, 사용자가 착용한 스마트벨트와 블루투스 통신을 시작합니다.
2. 벨트는 가속도 값을 어플리케이션으로 전달하며, 어플리케이션은 해당 값을 받아 서버로 전달합니다.
3. 서버는 가속도 값을 인공지능 모델이 탑재된 내부 네트워크의 다른 서버에 보냅니다. 인공지능 모델은 사용자의 자세를 9가지 방향으로 분류하며, 서버는 분류 결과를 다시 어플리케이션으로 전달합니다.
4. 어플리케이션은 결과값을 사용자에게 시각화 해 보여줍니다. 결과값은 누적되며, 측정 종료 시 해당 측정에서 사용자가 어느 방향으로 얼마나 기울어 있었는지를 알 수 있습니다.

3) 날짜별 사용자 자세 출력

사용자는 과거 측정한 정보를 날짜별로 선택해 볼 수 있습니다. 사용자가 특정 날짜를 클릭하면, 서버로부터 해당 날짜에 측정한 정보를 받습니다. 어플리케이션에서는 해당 날짜의 최대 빈도 자세를 보여줍니다.

Weltried의 자세 분류 인공지능 모델은 다음 과정을 거쳐 만들어졌습니다.

1) 실험 설계 및 수행

실험 설계

WELT사의 웨어러블 벨트는 자체적으로 0.2초마다 3축 가속도 데이터를 전송할 수 있습니다. 따라서 기존 기능을 이용하여 자세를 예측하기 위해 이용할데이터는 3축 가속도 데이터로 정하였습니다. 3축 가속도 센서는 x, y, z 3방향에 가해지는 가속도 값을 계산합니다. 따라서 허리를 특정 방향으로 기울일 경우벨트에 있는 가속도 센서의 위치가 변하게 되고 특정3축 가속도 값을 띄게 됩니다. 이를 이용하여 자세를 예측할 수 있습니다.

각도	조인 정도	1세트
5~15도	꽉 조임(-1칸)	150(30초)
	평범	150
	여유 있게(1칸)	150
15~25도	꽉 조임	150
	평범	150
	여유 있게	150

실험 진행 방식. 해당표의 내용을 9방향으로 모두 실행하여 총 8100개의 데이터 수집

분류할 자세는 총 9가지로 전방, 후방, 좌측, 우측, 대각선 4방향, 중앙으로 총9가지

방향으로 정하였습니다. 실험은6명의 모든 팀원이 참여하였습니다. 실험자 한 명은 총 9가지 방향, 허리를 기울인 정도(많이, 적게), 벨트의 조인 정도(꽉 조임, 평범, 여유있게)를 각각 30초씩 진행하였습니다. 따라서 한 명당 총 9 * 2 * 3 * 150 = 8100 개의 데이터를 생성하였으며 6명이므로 6 * 8100 = 48600개의 데이터를 생성하였습니다.

데이터 생성시 유의한 점

데이터 생성시의 문제점이라고 생각할 수 있는 부분은, 체형, 벨트를 조이는 정도 등이 사람마다 다를 수 있고 이로 인해 실험 데이터와 실제 데이터의 괴리가 생길 수 있다는점입니다.이를 해결하기위한 방법으로 실험자 개개인이 9방위만 돌아가며 수집하는 것이 아니라, 벨트를 꽉 조이기도 하고, 느슨하게 조이기도 하며 허리를 굽히는 각도도 조금만 굽히기도 하고, 크게 굽히기도 하는 등 다양한 조건에서의 데이터를 수집할수 있도록 하였습니다. 머신러닝 모델의 특성상 다양한 조건의 데이터를 학습에 이용할 수만 있다면 해당 조건에서 나타나는 차이에도 잘 대처할 수 있다는 장점이 있습니다. 따라서 이렇게 다양한 조건의 학습 데이터를준비하여 모델이 다른 조건이 주어진 상황에서도 잘 작동할 수 있도록 하였습니다(해당 내용은 *3) 알고리즘 선정 및 설계*에서 자세하게 다룸).

3904	-13568	8512
3904	-13440	8704
3840	-13568	8704
3968	-13504	8576
4032	-13504	8768
3968	-13504	8576
4032	-13568	8576
4032	-13632	8384
3904	-13568	8512
4096	-13312	8768
3968	-13376	8960
3840	-13376	8576
3904	-13312	8768
4032	-13312	8704

데이터 예시

첫 번째 열은 x축 가속도, 두 번째 열은 y축 가속도, 세 번째 열은 z축 가속도를 나타냅니다. 다음과 같은 형식으로 실험 데이터를 준비를 마쳤습니다.

2) 데이터 전처리 및 분석

데이터 전처리

데이터 전처리의 목적은 다음과 같습니다.

- 데이터 파일이 9(방위) * 2(허리를 기울인 정도) * 3(벨트 조인 정도) * 6(실험자 수) = 324개로 생성되었습니다. 실제 학습에 이용하기 위해서는 하나의 데이터 파일로 정리되어 있어야 합니다.
- 데이터는 현재 x축, y축, z축으 총 3개의 column으로 구성되어 있고, 벨트의 조인 정도, 허리를 기울인 정도, 방위 값은 파일 제목에 정보가 담겨있습니다. 제목에서 정보를 추출하여 column에 추가해야 합니다. 데이터파일의 제목 형식은 다음과 같습니다.

(실험자이름)_(허리를 기울인 정도)_(벨트를 조인 정도)_(방위 값).csv
예시) 윤진성_1_2_8.csv

```
def CallFiles():
    #리스트, 집합 등의 자료구조를 통해 약 300개의 파일을 불러와 정리
    xyz_df_list = glob.glob('./datasets/*')
    return xyz_df_list
```

CallFiles 함수는 약 300여개의 데이터파일을 모두 리스트로 저장합니다. glob 메서드는 지정해준 디렉토리에 있는 파일을 한 번에 불러옵니다. Xyz_df_list는 길이 약 300개의 리스트가 되며 각각의 방에는 데이터 파일이DataFrame형태로 저장됩니다.

```
def AttachLabel(xyz_df_list):
    #현재 fileList에는 x,y,z가속도 변수만 존재. label과 꽉조인 정도, 각도 정도 총 3개의 column을 추가시켜야 함
    fileList=[]
    columns = ['x', 'y', 'z']
    for i, filename in enumerate(xyz_df_list):
        file = pd.read_csv(filename)
        file.columns = columns
        file['degree'] = 0
        file['belt'] = 0
        file['direction'] = 0
        addDegree(file, filename)
        addBelt(file, filename)
        addDirection(file, filename)
        fileList.append(file)

    return fileList
```

AttachLabel 함수는CallFiles()를 이용하여 만든 리스트에 저장된데이터프레임에, 3개의 column을 추가하는 함수입니다. 허리를 기울인 정도, 벨트를 조인 정도, 방위 값을 추가하여총 6개의 column을 가진 데이터프레임으로 변환합니다. addDegree, addBelt, addDirection 함수를 이용하여 해당 column을 생성하는작업을 수행합니다.

```python
def addDirection(file, filename):
    if filename[-5] == '0':
        file['direction'] = 0
    elif filename[-5] == '1':
        file['direction'] = 1
    elif filename[-5] == '2':
        file['direction'] = 2
    elif filename[-5] == '3':
        file['direction'] = 3
    elif filename[-5] == '4':
        file['direction'] = 4
    elif filename[-5] == '5':
        file['direction'] = 5
    elif filename[-5] == '6':
        file['direction'] = 6
    elif filename[-5] == '7':
        file['direction'] = 7
    elif filename[-5] == '8':
        file['direction'] = 8
    elif filename[-5] == '9':
        file['direction'] = 9
    else :
        file['direction'] = -1

    return file
```

```python
def addDegree(file, filename):
    if filename[-9] == '0':
        file['degree'] = 0
    elif filename[-9] == '1':
        file['degree'] = 1
    else :
        file['degree'] = -1

def addBelt(file, filename):
    if filename[-7] == '0':
        file['belt'] = 0
    elif filename[-7] == '1':
        file['belt'] = 1
    elif filename[-7] == '2':
        file['belt'] = 2
    else :
        file['belt'] = -1
```

제목에서 허리를 기울인 각도, 벨트를 조인 정도, 방위 값을 추출하여 column으로 저장하기 위해 다음 함수들을 이용했습니다. addDirection은 방위 값을 추출하는 함수입니다. 방위 값은 데이터 파일 제목에서 뒤에서5번째에 입력되어 있으므로 filename[-5]를 이용하여 인식합니다. 이렇게 얻은 정보를 'direction'column으로추가합니다.addDegree는 허리를 기울인 정도를 추출하는 함수이며, 데이터 파일 제목 뒤에서 9번째에 입력되어 있으므로 filename[-9]를 이용하여 인식합니다. 얻은 정보는 'degree' column으로 추가합니다. addBelt는 벨트를 조인 정도를 추출하는 함수입니다. 이는 파일 제목 뒤에서 7번째에 입력되어 있으므로 filename[-7]을 이용하여 인식합니다. 얻은 정보는 'belt' column으로 추가합니다.

```python
def Concatenation(fileList):
    #fileList안의 모든 파일을 df로 합치기
    df = pd.concat(fileList)
    return df
```

이렇게 column을 추가한 데이터프레임들을 Concatenation 함수를 통해 하나로 합치는 작업을 수행합니다. 300여개의 리스트에 담겨있는 모든 데이터 프레임을 행방향으로합치는 과정을 수행하여 하나의 큰 데이터프레임을 생성합니다.

```
        x      y     z  degree  belt  direction
0    3648 -13504  8640       0     0          0
1    3904 -13568  8512       0     0          0
2    3904 -13440  8704       0     0          0
3    3840 -13568  8704       0     0          0
4    3968 -13504  8576       0     0          0
..    ...    ...   ...     ...   ...        ...
150 -2496 -14848 -5120       1     2          8
151 -2560 -14912 -4864       1     2          8
152 -2560 -14848 -4992       1     2          8
153 -2560 -14848 -5120       1     2          8
154 -2496 -14976 -5056       1     2          8

[50987 rows x 6 columns]
```

다음은 생성된 데이터 프레임입니다. 총 50987개로 예상했던 48600개를 약간 초과하는 정도였습니다. 실험 시 수동으로 30초를 측정하였기 때문에 약간의 오차가발생하였습니다. 데이터프레임을 보면 허리를 기울인 정도, 벨트를 조인 정도, 방위 값이 잘 추가된 것으로 확인할 수 있습니다.

```
def SaveFile(df):
    #df를 csv파일로 저장하기
    df.to_csv("./data_set_ver2.csv", header = False, index = False)
    X = df.iloc[:,:-1]
    y = df.iloc[:,-1]
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2)
    train = pd.concat([X_train, y_train], axis = 1)
    test = pd.concat([X_test, y_test], axis = 1)
    train.to_csv("./data_set_train.csv", header =False, index = False)
    test.to_csv("./data_set_test.csv", header =False, index = False)
    return
```

SaveFile 함수는 모든 전처리 과정을 거친 데이터셋을 csv형태로 저장하는 역할을 수행합니다. 딥러닝의 경우train, test셋을 나누어 저장해 주어야 읽어올 때편리하므로 data_set_train.csv, data_set_test.csv로 나누어 저장하였고, ML모델에 이용하기 위해 sklearn으로 불러올 때는 train, test셋을 나누지 않아도 되므로 데이터 전체를한 번에 저장해 주었습니다.

데이터 분석

데이터 전처리 과정을 마친 후 데이터 분석을 진행하였습니다. 데이터 분석의 목적은 다음과 같습니다.

- 수집한 데이터의 경향성을 확인하고 feature와 label의 상관관계를 시각화 하여 데이터를 이해한다.

```
def Analysis(data_set):
    #합친 df를 이용하여 여러가지 데이터 분석을 통해 수치 확인하기
```

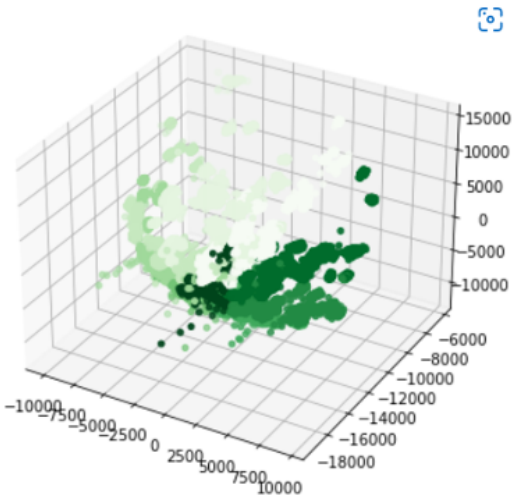
Analysis 함수는 데이터 분석을 수행해 주는 함수입니다. 후술할 데이터 분석 코드는 모두 Analysis 함수 안에 포함되어 있습니다.

```
n_value = data_set.isnull().sum().sum()
print('null 개수: ',n_value)
print(data_set.describe())
```

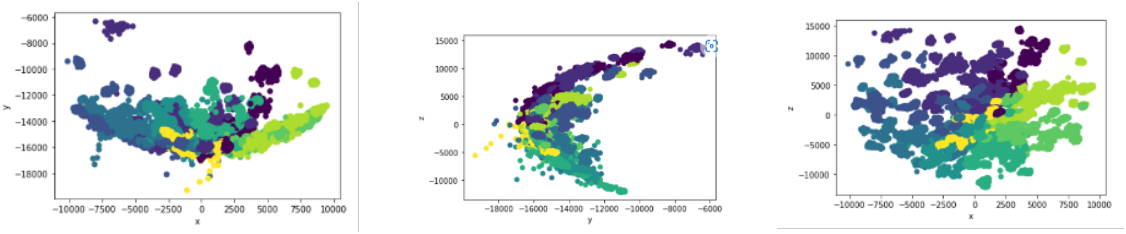
실험 데이터 수집 과정에서 예기치 못한 오류로 null값이 측정되었을 수 있습니다. 따라서null값의 수를 측정해 주었습니다. 또한 describe()를 통해 데이터셋의 통계학적 수치를 확인해보았습니다.

```
null 개수:  0
                 x             y             z        degree          belt  ₩
count  50987.000000  50987.000000  50987.000000  50987.000000  50987.000000
mean     276.003217 -14912.981584    -28.251280      0.500088      0.998156
std     3811.968337   1441.740547   4968.585771      0.500005      0.817327
min   -10112.000000 -19328.000000 -12160.000000      0.000000      0.000000
25%    -2176.000000 -15872.000000  -3648.000000      0.000000      0.000000
50%      576.000000 -15424.000000   -256.000000      1.000000      1.000000
75%     2816.000000 -14400.000000   3072.000000      1.000000      2.000000
max     9472.000000  -6336.000000  14528.000000      1.000000      2.000000

          direction
count  50987.000000
mean       4.009473
std        2.585199
min        0.000000
25%        2.000000
50%        4.000000
75%        6.000000
max        8.000000
```

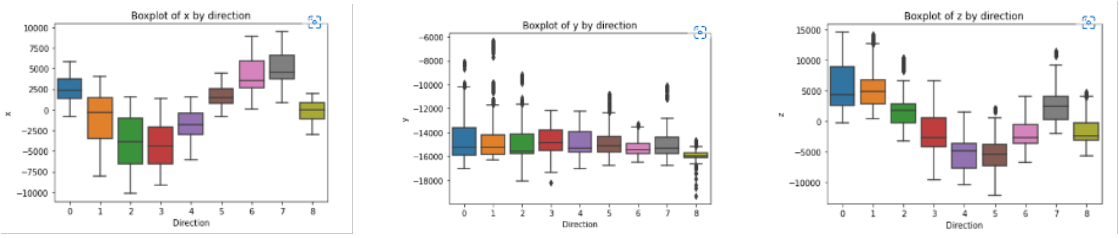
구동 결과 null 개수는 0개였습니다. 또한 x, y, z축의min_max값을 살펴보았을 때, x는 -10112~9472, z는 -12160~14528로 음수~양수의 넓은 범위의 값을 가지는 반면 y는 -19328~-6336으로 비교적 좁은 범위의 값을 가졌으며음수로만 분포되어 있다는 것을 확인할 수 있었습니다.



다음 그래프는 x, y, z의 값을 3차원 좌표평면에 점을 찍어 그린 산점도 그래프입니다. 이를 이용하여 방위 값(label)에 따라 다른 색깔로 표시하였습니다. 색깔별로 비슷한 위치에 분포해 있는 경향을확인할 수 있지만 3차원그래프를 2차원에 표현하다 보니 왜곡되는 부분이있어2차원 산점도 그래프로 표현해 보았습니다.

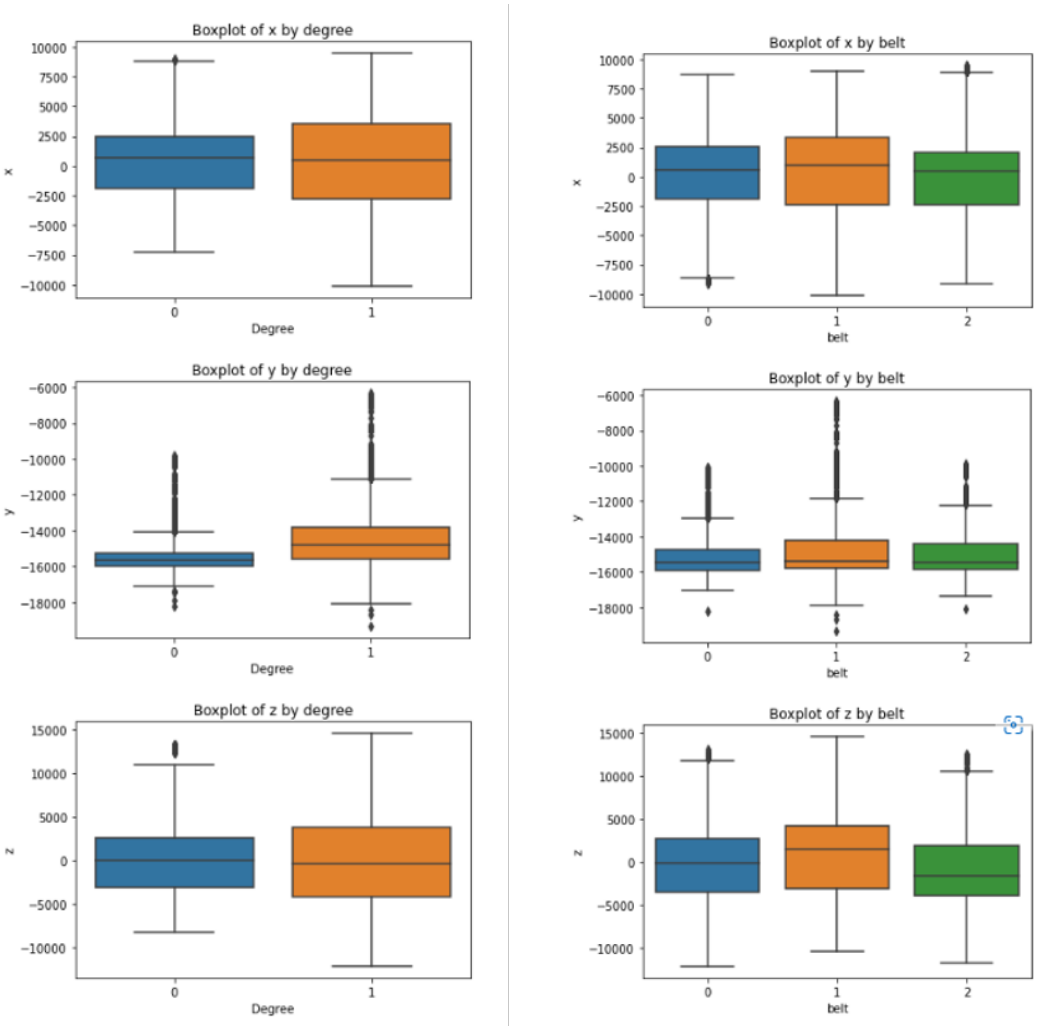


변수를 두개로 줄여 2차원 산점도 그래프로 나타낸 값입니다. 왼쪽 그래프는 x, y를 이용하였고, 중간 그래프는 y, z, 오른쪽 그래프는 x, z를 이용하여그린 그래프입니다. 방위 값(label)에 따라 다른 색깔로 표현하였습니다. 왼쪽 그래프를 보면 좌, 우로는 색깔이 잘 구분되고 있으나 상, 하로는 색깔이잘 구분되지 않고 있습니다. 마찬가지로 중간 그래프에서는 상, 하로는 색깔이잘 구분되고 있으나 좌 우로는 색깔이 잘 구분되지 않습니다. 오른쪽 그래프를 보면 색깔이 영역별로 잘 구분되는것을 볼 수 있습니다. 따라서 x, z값이 자세의 방위 값을 결정짓는 중요한 변수이며y또한 분류에 도움은 주지만 x, z에 비해잘 구분하지 못한다는 것을 알 수 있었습니다.



다음은 각 방위 값 별로 x, y, z값이 어떻게 분포하고 있는를 나타낸 boxplot입니

다. 왼쪽부터 x, y, z축 가속도 입니다. x, z축 boxplot은 방위값에 따라 분포하고 있는 모양이 다릅니다. 주목할만한 점은방위 값 0(후방)과 가까운 1(우측 후방)은 비슷한 값의 분포를 가지며, 반대 방향인 4(전방)과는 큰 차이를 가지는 것을 알 수 있습니다.그러나y축 boxplot은 모두 비슷한 값의 분포를 가지며, 특히 이상치로추정되는 값들이 다른 그래프에 비해 월등히 많은 것을 알 수 있습니다.



다음으로 허리를 굽힌 정도와, 벨트를 조인 정도에 따라 x, y, z값이 어떻게 분포하는 지 나타낸boxplot입니다. 허리를 굽힌 정도를 크게할수록(1) 더 넓은 범위에 값이 분포한다는 것을 알 수 있고, belt를 조인 정도는 이렇다할 경향성을 보이지는 않았습니다. 두feature 모두 자세 분류에 큰 역할을 하지는 않는다는 것을 알 수 있었습니다.

3) 알고리즘 선정 및 설계

자세의 방위 값을 분류하는 모델을 설계하기 위해 저희는 다음과 같은 알고리즘을 후보로 선정하였습니다.

- Multi LayerPerceptron(다층 퍼셉트론)

- K Nearest Neighbor(KNN)
- LightGBM

회귀 방식과의 비교

저희가 개발하려는 모델은 방향만을 기준으로9개의 자세를 구분하는 알고리즘입니다. 따라서 회귀가 아닌 분류 문제로 생각할 수 있으나 방향을 정의하는 것은 회귀 방식으로 접근할 수도 있습니다. 9개의 방향을 분류하는 것이 아닌 좌우방향과 전후 방향으로 회귀식을 만들어 내면 9개의 방위 값뿐만 아니라 각도가 얼마나 기울어졌는지 까지도 측정할 수 있습니다. 그럼에도 저희가 분류기를 이용한 이유는다음과 같습니다.

물론 회귀 방식이 언급한 것처럼 각도까지 측정할 수 있다는 장점이 있습니다. 그러나 회귀 방식의 문제점은 이상치와 조건에 민감하다는심각한 문제가 있습니다. 회귀식을 이용하는 방법을 사용한다면, 데이터셋을 이용해서 전후 방향의 선형회귀식 1개, 좌우 방향의 선형회귀식 1개로 총 2개의 회귀식이 생성될 것입니다. 그런데 저희가 개발할 웨어러블 벨트를이용한 자세 측정에서 가속도 센서의 가속도 값은 모두가 같은 값을 띄지 않습니다. 개개인은 몸무게, 키, 벨트를 조이는 정도, 벨트를 메는 위치 등 다른 조건에서 자세를측정하게 될 것입니다. 그런데이러한 다른 조건의 측정값을 두 개의 회귀식으로 표현한다면 개개인에 따라 정확한 값을 산출하기 어려울 것입니다. 데이터셋을 생성한 6명 실험자를 예로 들어보면, 6명이 모은 전체 데이터를 이용하여 회귀식을만들고,또 각각 한 명한 명의 데이터로만 회귀식을 만들어 비교해보면 6명의 조건이 모두 다르기 때문에 서로 다른 양상을 띄고 있을 것입니다. 즉 개인의 조건을 전혀 반영하지 못하는것입니다.

그러나 분류기는 동작 방식의 하나의 식을 생성하는 것이 아니라, 피처를 무수히 많은 조건으로 나누어 분류해냅니다. 따라서 회귀가 여러 조건의 데이터셋을융합하여 하나의 식을 만든다면, 분류는 데이터셋 자체를 분류 기준으로 사용하여 분류합니다. 즉, 다양한 조건의 실험자를 대상으로 데이터를 수집할 수있을 경우 다양한 조건에서도 잘 동작하는 분류기를 만들 수 있습니다. 따라서 저희는 분류기를 사용하고, 최대한 다른 조건(성별, 몸무게 등)을 가진 여러 명의 실험자를 대상으로 데이터를 수집하여 조건과 환경에 따라 성능이 떨어지는 현상을 방지하였습니다.

Multi Layer Perceptron

가장 먼저 시도했던 알고리즘은 MLP입니다. MLP는 현재 가장 각광받고 있는 분류기이기 때문에 저희 프로젝트에서도 좋은 성능을 나타낼 수 있을 것이라고 생각하여 먼저 시도하였습니다.

```
BATCH_SIZE = 32
EPOCHS = 10

class CustomDataset(Dataset):
  def __init__(self, csv_path):
    df = pd.read_csv(csv_path)

    self.inp = df.iloc[:, :3].values
    self.outp = df.iloc[:,-1].values.reshape(-1,1)

  def __len__(self):
    # 가지고 있는 데이터셋의 길이를 반환한다.
    return len(self.inp) # 1314

  def __getitem__(self,idx):
    inp = torch.FloatTensor(self.inp[idx])
    outp = torch.LongTensor(self.outp[idx])
    return inp, outp # 해당하는 idx(인덱스)의 input과 output 데이터를 반환한다.

''' 3. MNIST 데이터 다운로드 (Train set, Test set 분리하기) '''
PATH_TRAIN = './drive/MyDrive/WELT/data_set_train.csv'
PATH_TEST = './drive/MyDrive/WELT/data_set_test.csv'

train_dataset = CustomDataset(PATH_TRAIN)
test_dataset = CustomDataset(PATH_TEST)

train_loader = torch.utils.data.DataLoader(dataset = train_dataset,
                                           batch_size = BATCH_SIZE,
                                           shuffle = True)

test_loader = torch.utils.data.DataLoader(dataset = test_dataset,
                                          batch_size = BATCH_SIZE,
                                          shuffle = False)
```

Batch size는 32 Epochs는 10으로 설정하였으며, 정형데이터를 이용하였기 때문에 CustomDataset클래스을 만들어 이용했습니다.

```
[ ]  for (X_train, y_train) in train_loader:
        print('X_train:', X_train.size(), 'type:', X_train.type())
        print('y_train:', y_train.size(), 'type:', y_train.type())
        break
```

```
X_train: torch.Size([32, 3]) type: torch.FloatTensor
y_train: torch.Size([32, 1]) type: torch.LongTensor
```

Train_loader의 X_train은32(배치 수)*3(피처x, y, z)이고 y_train은 32*1(방위 값)입니다.

```
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(3, 512)
        self.fc2 = nn.Linear(512, 256)
        self.fc3 = nn.Linear(256, 9)
        self.dropout_prob = 0.5
        self.batch_norm1 = nn.BatchNorm1d(512)
        self.batch_norm2 = nn.BatchNorm1d(256)

    def forward(self, x):
        x = x.view(-1, 3)
        x = self.fc1(x)
        x = self.batch_norm1(x)
        x = F.relu(x)
        x = F.dropout(x, training = self.training, p = self.dropout_prob)
        x = self.fc2(x)
        x = self.batch_norm2(x)
        x = F.relu(x)
        x = F.dropout(x, training = self.training, p = self.dropout_prob)
        x = self.fc3(x)
        x = F.log_softmax(x, dim = 1)
        return x
```

네트워크의 구조는 3개의 fully connected layer를 가지도록 구성하였습니다. Activation function으로는 ReLU를 사용하였으며, 이를 통과하기 전에 batchnormalization을 수행했습니다. Activation function을 거친 이후에는 dropout을 0.5로 사용하였습니다. 레이어 마지막엔 log_softmax를 이용하여 확률 값을 나타내도록 하였습니다.

```
import torch.nn.init as init
def weight_init(m):
    if isinstance(m, nn.Linear):
        init.kaiming_uniform_(m.weight.data)

model = Net().to(DEVICE)
model.apply(weight_init)
optimizer = torch.optim.Adam(model.parameters(), lr = 0.01)
criterion = nn.CrossEntropyLoss()
```

Kaiming_uniform_을 통해parameter를 초기화 시켰고, optimizer로는 Adam을 사용하였습니다. Learning rate는 0.01, loss function은 분류기에서 많이 사용하는 CrossEntropyLoss를 이용하였습니다.

```
Train Epoch: 10 [0/27288 (0%)]  Train Loss: 0.595847
Train Epoch: 10 [6400/27288 (23%)]      Train Loss: 0.812984
Train Epoch: 10 [12800/27288 (47%)]     Train Loss: 0.797070
Train Epoch: 10 [19200/27288 (70%)]     Train Loss: 0.739474
Train Epoch: 10 [25600/27288 (94%)]     Train Loss: 0.637642

[EPOCH: 10],    Test Loss: 0.5476,      Test Accuracy: 77.70 %
```

10 Epoch까지 학습한 결과 accuracy는 77.70%로 예상보다 훨씬 못 미치는 수준이 나타났습니다. 원인으로는 여러가지가 있을 수 있겠지만 저희가 생각한가장 큰 문제점은 MLP가 정형 데이터에 있어서는 성능이 비교적 떨어지기 때문이라고 생각하였습니다. 퍼셉트론을 기반으로 한 신경망 분류기는 이미지 분류에뛰어난 CNN,자연어처리에뛰어난 RNN등 복잡한 데이터를input으로 가지는 분류에 있어서는 뛰어난 성능을보이지만 정형 데이터에 대해서는 다른 ML 알고리즘에 비해 좋은 성능을 나타내지 못합니다. 따라서 다음 접근은 신경망이 아닌 ML 알고리즘으로 시도하였습니다.

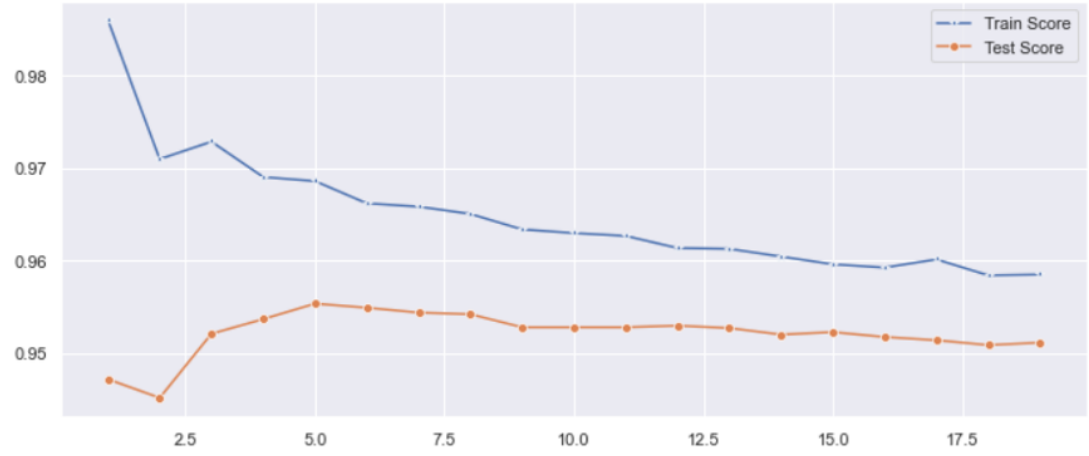
K Nearest Neighbor

다음으로 선정한 알고리즘은 군집화(clustering) 알고리즘 중 하나인 KNN입니다. KNN은 예측할 데이터가 들어왔을 때 그 데이터와 가장 가까이 있는 학습 데이터를 k개 만큼 고르고 그 중 가장 많은 비율을가지는 label을 선택하는 알고리즘입니다.

```
test_scores = []
train_scores = []

for i in range(1,20):

    knn = KNeighborsClassifier(i)
    '''
    i는 판단할 가장 가까운 이웃 점의 개수입니다.
    성능이 가장 좋은 best i는 데이터마다 다르기 때문에 탐욕적인 방식으로 찾아야 하므로 점 1개부터 20개까지
    for문을 돌려 제일 좋은 성능을 보여주는 점의 개수를 찾습니다.
    '''
    knn.fit(X_train,y_train)

    train_scores.append(knn.score(X_train,y_train))
    test_scores.append(knn.score(X_test,y_test))

max_train_score = max(train_scores)
train_scores_ind = [i for i, v in enumerate(train_scores) if v == max_train_score]
print('Max train score {} % and k = {}'.format(max_train_score*100,list(map(lambda x: x+1, train_scores_ind))))

max_test_score = max(test_scores)
test_scores_ind = [i for i, v in enumerate(test_scores) if v == max_test_score]
print('Max test score {} % and k = {}'.format(max_test_score*100,list(map(lambda x: x+1, test_scores_ind))))
```

k값을 1부터 20까지 돌아가며train_scores와 test_scores를 산출하였고, 이중 가장 높은 값을 가진knn과 그 값을 출력하였습니다.

```
Max train score 98.6060419506618 % and k = [1]
Max test score 95.53249494327676 % and k = [5]
```



Max test score는 k=5일 때 95.532%로 가장 높은 값을 나타냈습니다. 정확도가 95% 이상으로 신경망 기반의 딥 러닝보다 훨씬안정적이고 좋은 성능을 보였습니다.

LightGBM

LightGBM은 결정 트리 기반 앙상블 모델 중 boosting 방식의 알고리즘 중 하나입니다. 결정 트리 기반 앙상블 모델은 데이터셋을 이용하여 무수히많은 결정 트리를 생성하고 예측해야 할 데이터가 들어왔을 때 각각의 결정 트리의 결과 중 가장 많은 비율을 가진 label을 예측 값으로 산출하는 알고리즘입니다. 특히 boosting 방식은 결정 트리를 무작위로 생성하는것이 아닌,이전에 생성 결정트리를 기반으로 피드백하여 새로운 결정 트리를 생성하고 가중치를 다르게 부여하는 방식입니다. LightGBM은 정형 데이터 분류 문제에 있어XGBoost와 함께 가장 사랑받는 알고리즘이며XGBoost

에 비해 학습 시간이 적게 소요되면서비슷한 성능을 나타내기 때문에 LightGBM을 이용해 보게 되었습니다.

```
from lightgbm import LGBMClassifier

lgbm_clf = LGBMClassifier(n_estimators=500)

eval_set = [(X_tr, y_tr), (X_val, y_val)]
lgbm_clf.fit(X_tr, y_tr, early_stopping_rounds = 100, eval_metric = "auc_mu", eval_set = eval_set)
lgbm_pred = lgbm_clf.predict(X_test)
lgbm_accuracy_score = accuracy_score(y_test, lgbm_pred)
print("LGBM 정확도: ", lgbm_accuracy_score)
```

```
ss: 0.139777
[187]   training's auc_mu: 0.999946     training's multi_logloss: 0.0440951     valid_1's auc_mu: 0.999481      valid_1's multi_loglo
ss: 0.139916
[188]   training's auc_mu: 0.999946     training's multi_logloss: 0.043888      valid_1's auc_mu: 0.99948       valid_1's multi_loglo
ss: 0.139955
[189]   training's auc_mu: 0.999947     training's multi_logloss: 0.0436866     valid_1's auc_mu: 0.999481      valid_1's multi_loglo
ss: 0.140051
[190]   training's auc_mu: 0.999948     training's multi_logloss: 0.0435245     valid_1's auc_mu: 0.99948       valid_1's multi_loglo
ss: 0.140395
[191]   training's auc_mu: 0.999948     training's multi_logloss: 0.0433753     valid_1's auc_mu: 0.999479      valid_1's multi_loglo
ss: 0.140478
[192]   training's auc_mu: 0.999948     training's multi_logloss: 0.0432542     valid_1's auc_mu: 0.999477      valid_1's multi_loglo
ss: 0.140674

[193]   training's auc_mu: 0.999949     training's multi_logloss: 0.043052      valid_1's auc_mu: 0.999478      valid_1's multi_loglo
ss: 0.140943
[194]   training's auc_mu: 0.999949     training's multi_logloss: 0.0429351     valid_1's auc_mu: 0.999474      valid_1's multi_loglo
ss: 0.14124
LGBM 정확도:  0.953814473426162
```

총 생성할 결정 트리 개수 n_estimators = 500으로 설정하였고 early stopping rounds를100으로 설정하여 학습에 진전이 없을 시 조기종료 하도록하였습니다.이렇게 생성한LightGBM모델의 정확도는 95.381%로KNN방식과 비슷하게 좋은 성능을 나타냈습니다.

```
lgbm_search_space = {'num_leaves': hp.quniform('num_leaves', 32, 64, 2),
                     'max_depth': hp.quniform('max_depth', 100, 200, 2),
                     'min_child_samples': hp.quniform('min_child_samples', 20,60,2),
                     'subsample': hp.uniform('subsample', 0.7,1),
                     'learning_rate': hp.uniform('learning_rate', 0.01,0.2)
                     }
```

```
def objective_func(search_space):
    lgbm_clf =  LGBMClassifier(n_estimators=100, num_leaves=int(search_space['num_leaves']),
                               max_depth=int(search_space['max_depth']),
                               min_child_samples=int(search_space['min_child_samples']),
                               subsample=search_space['subsample'],
                               learning_rate=search_space['learning_rate'])
    # 3개 k-fold 방식으로 평가된 roc_auc 지표를 담는 list
    accuracy_list = []

    # 3개 k-fold방식 적용
    kf = KFold(n_splits=3)
    # X_train을 다시 학습과 검증용 데이터로 분리
    for tr_index, val_index in kf.split(X_train):
        # kf.split(X_train)으로 추출된 학습과 검증 index값으로 학습과 검증 데이터 세트 분리
        X_tr, y_tr = X_train.iloc[tr_index], y_train.iloc[tr_index]
        X_val, y_val = X_train.iloc[val_index], y_train.iloc[val_index]

        # early stopping은 30회로 설정하고 추출된 학습과 검증 데이터로 XGBClassifier 학습 수행.
        lgbm_clf.fit(X_tr, y_tr, early_stopping_rounds=30, eval_metric="auc_mu",
           eval_set=[(X_tr, y_tr), (X_val, y_val)])

        # 1로 예측한 확률값 추출후 roc auc 계산하고 평균 roc auc 계산을 위해 list에 결과값 담음.
        score = accuracy_score(y_val, lgbm_clf.predict(X_val))
        accuracy_list.append(score)

    # 3개 k-fold로 계산된 roc_auc값의 평균값을 반환하되,
    # HyperOpt는 목적함수의 최소값을 위한 입력값을 찾으므로 -1을 곱한 뒤 반환.
    return -1*np.mean(accuracy_list)
```

다음으로는 hyper optimization을 통해 LightGBM의 여러 hyper parameter를 튜

닝하였습니다. num_leaves, max_depth, min_child_samples,subsamples, learning_rate등의 파라미터를 튜닝했습니다.

```
lgbm_clf =  LGBMClassifier(n_estimators=500, num_leaves=int(best['num_leaves']),
                           max_depth=int(best['max_depth']),
                           min_child_samples=int(best['min_child_samples']),
                           subsample=round(best['subsample'], 5),
                           learning_rate=round(best['learning_rate'], 5)
                          )

# evaluation metric을 auc로, early stopping은 100 으로 설정하고 학습 수행.
lgbm_clf.fit(X_tr, y_tr, early_stopping_rounds=100,
             eval_metric="auc_mu",eval_set=[(X_tr, y_tr), (X_val, y_val)])

lgbm_accuracy_score = accuracy_score(y_test, lgbm_clf.predict(X_test))
print('Accuracy: {0:.4f}'.format(lgbm_accuracy_score))
```

```
[138]   training's auc_mu: 0.999962     training's multi_logloss: 0.0337079     valid_1's auc_mu: 0.999406      valid_1's multi_loglo
ss: 0.168177
[139]   training's auc_mu: 0.999962     training's multi_logloss: 0.0335652     valid_1's auc_mu: 0.999408      valid_1's multi_loglo
ss: 0.16855
[140]   training's auc_mu: 0.999962     training's multi_logloss: 0.0334129     valid_1's auc_mu: 0.999406      valid_1's multi_loglo
ss: 0.169038
[141]   training's auc_mu: 0.999963     training's multi_logloss: 0.0332717     valid_1's auc_mu: 0.999408      valid_1's multi_loglo
ss: 0.169436
[142]   training's auc_mu: 0.999963     training's multi_logloss: 0.0331345     valid_1's auc_mu: 0.999407      valid_1's multi_loglo
ss: 0.169931
[143]   training's auc_mu: 0.999963     training's multi_logloss: 0.0329977     valid_1's auc_mu: 0.999406      valid_1's multi_loglo
ss: 0.170433
[144]   training's auc_mu: 0.999963     training's multi_logloss: 0.0328774     valid_1's auc_mu: 0.999401      valid_1's multi_loglo
ss: 0.171105
[145]   training's auc_mu: 0.999963     training's multi_logloss: 0.0327364     valid_1's auc_mu: 0.999396      valid_1's multi_loglo
ss: 0.172131
[146]   training's auc_mu: 0.999964     training's multi_logloss: 0.0326286     valid_1's auc_mu: 0.999395      valid_1's multi_loglo
ss: 0.172693
Accuracy: 0.9511
```

Hyper parameter optimization의 결과로 가장 좋은 성능을 나타내는 hyper parameter를 이용하여 다시 정확도를 측정하였습니다. 결과는 95.11%로 기존 성능보다 근소하게 떨어진 것을확인할 수 있었으나, 측정마다정확도 값이 근소하게 다르게 나타날 수 있는 점과 하이퍼 파라미터 최적화 이후 과적합이 개선되었을 수 있을 것이라 판단하여 optimization이 적용된 모델을 최종 분류 모델로 선정하였습니다.

4 효과

사용자는 자신의 자세에 대해 정확한 수치의 데이터를 얻습니다. 이를 통해 객관적인 진단과 처방이 이루어질 수 있습니다. 또한 기록된 데이터를 보며 사용자는 확인 가능한 성취감을 느끼며 자세 개선에 힘쓸 수 있습니다.

5 결과물

현재까지 Weltried의 결과물이자 사용자에게 서비스 가능한 제품은 안드로이드 어플리케이션입니다. 사용자의 자세는 아홉 가지 방향으로 분류하며, 각 방위에 따른 값은 다음과 같습니다.

5	4	3
6	8	2
7	0	1

2: 동, 6: 서, 0: 남, 4: 북

6 참여인원

클라이언트 파트

김영길 (201811165) (파트장)
김주혜 (202011274)
신지수 (201911186)

서버 파트

박원준 (201811182) (팀장, 파트장)

AI 파트

윤진성 (201912550) (파트장)
차준혁 (201711430)

7 평가 방법

평가는 다음의 2가지를 시행했습니다. 프로젝트 Weltried의 자체 평가와 어플 사용성 평가입니다.

1) Weltried 자체 평가

평가는 다음의 3가지 질문을 이용했습니다.

1번. Weltried가 사용자의 앉은 자세 개선에 도움이 되었는 지
2번. 앉은 자세 분석을 위해 앱을 재사용 할 것인지
3번. 실시간 앉은 자세 분석 기능이 정확한 지

프로젝트 구성원이 아닌 Weltried 사용자 6명에게 구글 폼을 이용해 설문을 진행했으며, 결과는 다음과 같습니다.

1: 매우 불만족, 2: 불만족, 3: 보통, 4: 만족, 5: 매우 만족

	1번	2번	3번
사용자 A	4	4	4
사용자 B	2	4	3
사용자 C	5	4	4
사용자 D	4	4	4
사용자 E	4	3	3
사용자 F	4	4	4
평균	3.83	3.83	3.67

2) 어플 사용성 평가

평가에 이용한 질문은 다음의 6가지입니다.

1번. 앱을 쉽게 조작할 수 있는지

2번. 메뉴/버튼 등의 기능을 찾기 쉬운 지

3번. 자세 분석 결과를 쉽게 이해할 수 있는 지

4번. 앱의 시각화 정보 (그림, 그래프)가 정보 해석에 도움이 되는 지

5번. 어플리케이션의 글씨 크기나 그림의 크기는 적당한 지

6번. 어플리케이션이 시각적으로 보기 좋은 지

프로젝트 구성원이 아닌 Weltried 사용자 4명에게 구글 폼을 이용해 설문을 진행했으며, 결과는 다음과 같습니다.

1: 매우 불만족, 2: 불만족, 3: 보통, 4: 만족, 5: 매우 만족

	1번	2번	3번	4번	5번	6번
사용자 A	5	4	5	4	4	5
사용자 B	4	3	4	4	4	4
사용자 C	5	5	5	5	5	5
사용자 D	4	3	3	4	4	4
평균	4.5	3.75	4.25	4.25	4	4.5

한편 비교 대상으로는 PostureReminder라는 시중의 어플을 사용하였습니다. App Store에 출시된 어플로 Iurii Kriuchkov가 제작하였습니다. Weltried와 동일한 질문을 사용했으며, 위의 설문을 진행한 이와 동일한 사용자 4명에게 답변을 받았습니다.

1: 매우 불만족, 2: 불만족, 3: 보통, 4: 만족, 5: 매우 만족

	1번	2번	3번	4번	5번	6번
사용자 A	4	3	4	3	3	3
사용자 B	4	4	3	4	4	4
사용자 C	5	5	2	2	4	2
사용자 D	4	3	1	3	5	3
평균	4.25	4.25	2.5	3	4	3