환경 센서를 통한 동작 예측 모델

학번: 2218042

이름: 임경용

Github address:
https://github.com/yong2218042/Model-for-predicting-motion-through-environmental-sensors.git

1. 안전 관련 머신러닝 모델 개발의 목적

활용 대상은 다음과 같다.

첫 번째는 환경 센서 데이터를 기반으로 움직임, 연기, 이산화탄소 등을 감지하여 잠재적인 안전 문제를 식별할 수 있고, 이를 활용하여 Home Security System 또는 건물 내의 안전 시스템을 구축할 수 있다.

두 번째는 온도, 습도, 조도 등의 환경 센서 데이터를 분석하여 건물 또는 시설의 에너지 효율성을 평가하고 최적화할 수 있으며 에너지 소비를 예측하고 효율적인 에너지 사용을 권장하는 시스템 개발로 활용이 가능하다.

세 번째는 이산화탄소, 습도, 연기 등의 환경 센서 데이터를 사용하여 실내 환경의 공기 질을 모니터링하고 환기 시스템을 최적화하여 공기 질을 개선할 수 있다.

네 번째는 움직임, 연기 등의 데이터를 활용하여 재해를 예측하고 대응할 수 있는 시스템을 구축할 수 있다. 예시로 화재, 가스 누출, 동작 감지 등에 대한 대응이 가능하다.

다섯 번째는 환경 센서 데이터를 분석하여 건물 또는 시설의 운영을 최적화할 수 있다. 예시로 특정 조건에서 자동으로 에너지 소비를 최적화하거나, 특정 장소에서 활동에 따른 조명 및 환기를 제어할 수 있다.

여섯 번째는 환경 센서 데이터를 이용하여 사물인터넷 기술과 통합하여 smart home 또는 Smart Building System 을 개발할 수 있다.

사용한 데이터의 독립변수는 다음과 같다.

ts (Timestamp): 시간 정보이며 데이터가 기록된 시간이다.

device: 센서 장치 고유 식별자이고, 더미 변수로 변환되어 범주형 변수로 활용된다.

co (Carbon Monoxide): 이산화탄소 농도를 나타낸다.

humidity : 대기 중의 습도를 나타낸다.

light: 주변의 조명 수준을 나타낸다.
lpg: 액화 석유 가스(LPG) 농도를 나타낸다.
smoke: 연기 농도를 나타낸다.
temp (Temperature): 환경의 온도를 나타낸다.
‘device_00:0f:00:70:91:0a’, ‘device_1c:bf:ce:15:ec:4d’, ‘device_b8:27:eb:bf:9d:51’: 디바이스 열의 범주형 변수를 더미 변수로 변환한 결과이다.

사용한 데이터의 종속변수는 다음과 같다.
motion: 예측하고자 하는 대상 변수로, 환경에서의 움직임 등을 나타낸다.

해당 머신러닝 모델이 생성하는 가치 중에 가장 뛰어난 것은 ‘활용도’이다. 온도, 습도, 조도,이산화탄소, 습도, 연기를 분석하여 실내 환경 질을 개선할 수 있도록 있으며 이는 근로자가 건강하게 일할 수 있는 환경을 조성하도록 한다. 또한 움직임이나 연기 등을 활용한다면 화재나 가스 누출 등을 신속하게 대응할 수 있고 이는 중대재해로 번질 수 있는 사고를 예방할 수 있다. 마지막으로 사물인터넷과 통합하여 활용이 가능하다. 이 점은 해당 머신러닝 모델이 발전 가능성이 높다는 것을 강조하는 부분이다.

2. 안전 관련 머신러닝 모델의 네이밍의 의미

처음에는 ‘환경 센서를 통한 중대재해 예방 모델’로 정하였으나 해당 머신러닝 모델은 중대재해를 예방하는 것 이외에도 많은 역할을 수행할 수 있다는 점을 고려했고, 주어진 환경 센서 데이터로부터 어떤 조건에서 움직임이 발생할지를 학습하고 이를 토대로 안전 관련 예측을 수행하는 것이 목표라서 종속변수인 ‘motion’ 이 핵심이라고 생각했다. 그래서 최종적인 네이밍은 ‘환경 센서를 통한 동작 예측 모델’이다.

3. 개발 계획

데이터 주요 변수는 ts (Timestamp), device (센서 장치 ID), co (이산화탄소 농도), humidity (습도), light (조도), lpg (LPG 농도), smoke (연기 농도), temp (온도), 'device_00:0f:00:70:91:0a', 'device_1c:bf:ce:15:ec:4d', 'device_b8:27:eb:bf:9d:51' 등이 있다.

시각화 부분은 'co'와 'humidity'의 분포를 시각화한 히스토그램을 통해 이상치를 발견할 수 있으며 'temp'와 'humidity' 간의 산점도를 통해 음의 상관관계를 확인할 수 있다.

데이터 전처리 계획은 데이터가 위치한 경로를 지정하고, print(data.columns) 으로 데이터의 열 정보를 확인하여 각 열이 어떤 정보를 담고 있는지 파악한다.
다음으로 데이터에서 누락된 값이 있는 경우, 해당 값을 처리한 뒤 'device'와 같은 범주형 변수는 더미 변수로 변환하여 모델 학습에 활용하기 위해 pd.get_dummies 함수를 사용한다. 이후 데이터를 학습용과 테스트용으로 나누어 모델의 성능을 평가하고, 필요한 경우 추가적인 전처리 작업을 한다.

머신러닝 모델은 RandomForestClassifier 를 사용하여 'motion'을 예측할 것이다.
RandomForest 는 여러 결정 트리를 구성하고, 각 트리의 예측을 기반으로 최종 예측을 수행한다. 각 트리는 부트스트랩 샘플링으로 생성되며, 노드 분할 시 최적의 특성을 선택한다. 이를 통해 과적합을 줄이고 안정적인 예측을 가능하게 한다. 장점으로는 안정적인 모델이고, 다양한 유형의 데이터에 적용 가능하다.
또한 특성 중요도를 제공하여 모델의 설명력을 높인다.

RandomForest 모델이 주어진 환경 데이터에서 'motion'을 예측하는 정확도는 높을 것으로 예상한다. 특히, 'temp', 'humidity', 'co' 등이 중요한 변수로 드러날 것으로 예상한다.

사용할 성능 지표는 '정확도'이다.
모델 학습 후 테스트 데이터를 사용하여 정확도를 계산하고, 혼동 행렬을 출력하여 모델의 성능을 평가하고 시각화한다. 정확도는 전체 예측 중 올바르게 분류된 샘플의 비율을 나타내며, 혼동 행렬은 모델이 각 클래스를 얼마나 정확하게 예측했는지를 보여준다.
성능 검증 방법 계획은 데이터를 학습용과 테스트용으로 단순히 나누는 방식으로 성능 검증을 진행한다. train_test_split 함수를 사용하여 전체 데이터를 학습 데이터와 테스트 데이터로 나누고 나눈 데이터를 사용하여 랜덤 포레스트 모델을 학습하고, 테스트 데이터를 활용하여 정확도를 계산하고 혼동 행렬을 출력한다.

4. 개발 과정

주요 작업 개발 과정은 다음과 같다.
필요한 라이브러리를 임포트하고, 데이터를 로드한다.

```
import pandas as pd

data_path = r"C:₩Users₩user₩OneDrive₩바탕 화면₩iot_telemetry_data.csv"
data = pd.read_csv(data_path)
```

데이터의 누락된 값 처리, 범주형 변수 처리, 필요한 특성 선택 등을 수행한다.

```
# 누락된 값 처리 (예시)
data = data.dropna()

# 범주형 변수 처리 (예시)
categorical_columns = ['device']
for column in categorical_columns:
    if column in data.columns:
        data = pd.get_dummies(data, columns=[column])
```

데이터의 분포를 이해하기 위해 시각화를 수행한다.

```
# 히스토그램 그리기
data.hist(figsize=(10, 10))
plt.show()
```

전체 데이터를 학습 데이터와 테스트 데이터로 나눈다.

```
from sklearn.model_selection import train_test_split

X = data.drop('motion', axis=1)
y = data['motion']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

사용할 모델을 선택하고 학습용 데이터에 모델을 학습시킨다.

```
from sklearn.ensemble import RandomForestClassifier

model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

테스트 데이터를 사용하여 모델의 성능을 평가한다.

```
from sklearn.metrics import accuracy_score, confusion_matrix

y_pred = model.predict(X_test)

# 정확도 출력
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')

# 혼동 행렬 출력
conf_matrix = confusion_matrix(y_test, y_pred)
print(f'Confusion Matrix:₩n{conf_matrix}')
```

최종 코드는 다음과 같다.

```
# 필요한 라이브러리 임포트
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# 데이터 불러오기
data_path = r"C:₩Users₩user₩OneDrive₩바탕 화면₩iot_telemetry_data.csv"
data = pd.read_csv(data_path)

# 데이터 전처리
# (필요에 따라 추가 전처리 수행)
# 예시: 누락된 값 처리, 범주형 변수 처리, 특성 선택 등

# 데이터 열 확인
print(data.columns)

# 범주형 변수 처리
categorical_columns = ['device']  # 다른 범주형 변수가 있는 경우 이곳에 추가

for column in categorical_columns:
    if column in data.columns:
        data = pd.get_dummies(data, columns=[column])

# 데이터 요약 및 시각화
data_summary = data.describe()
print(data_summary)

# 히스토그램 그리기
data.hist(figsize=(10, 10))
plt.show()

# 머신러닝 모델 학습
X = data.drop('motion', axis=1)  # 'motion'을 예측하려면 해당 열을 지정해야 합니다.
y = data['motion']

# 데이터를 학습용과 테스트용으로 나눕니다.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 모델 생성 및 학습
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)

# 모델 성능 평가
y_pred = model.predict(X_test)
```

```
# 정확도 출력
accuracy = accuracy_score(y_test, y_pred)
print(f'Accuracy: {accuracy}')

# 혼동 행렬 출력
conf_matrix = confusion_matrix(y_test, y_pred)
print(f'Confusion Matrix:₩n{conf_matrix}')

# 정확도를 시각화
plt.bar(['Accuracy'], [accuracy])
plt.ylim(0, 1)
plt.show()

# 추가 결과 확인 여부 입력 받기
while True:
    try:
        user_input = input("더 많은 결과를 확인하시겠습니까? (y/n): ").lower()
        if user_input == 'y':
            # 모델 성능 평가 (다시 예측 및 출력)
            y_pred = model.predict(X_test)
            accuracy = accuracy_score(y_test, y_pred)
            print(f'Accuracy: {accuracy}')

            # 혼동 행렬 출력
            conf_matrix = confusion_matrix(y_test, y_pred)
            print(f'Confusion Matrix:₩n{conf_matrix}')

            # 정확도를 시각화
            plt.bar(['Accuracy'], [accuracy])
            plt.ylim(0, 1)
            plt.show()
        elif user_input == 'n':
            print("프로그램을 종료합니다.")
            break
        else:
            print("올바른 입력이 아닙니다. 'y' 또는 'n' 중 하나를 입력하세요.")
    except Exception as e:
        print(f"오류 발생: {e}")
```

각 함수의 동작은 다음과 같다.

train_test_split(X, y, test_size=0.2, random_state=42):

주어진 독립 변수 X 와 종속 변수 y 를 학습용과 테스트용으로 나누어준다.

test_size 파라미터는 테스트 데이터의 비율을 나타낸다. 여기서는 0.2 로 설정되어 있으므로 전체 데이터의 20%가 테스트 데이터로 사용된다.

random_state 파라미터는 난수 발생 시드를 지정한다. 동일한 시드를 사용하면 매번 데이터를 분할할 때 동일한 결과를 얻을 수 있다.

RandomForestClassifier(random_state=42):

랜덤 포레스트 분류기를 생성한다. 랜덤 포레스트는 여러 의사결정 트리를 앙상블하여 생성되는 모델로, 각 트리의 예측을 결합하여 높은 정확도를 제공한다.

random_state 파라미터는 모델의 무작위성을 제어하는데 사용되는 시드를 지정한다.

model.fit(X_train, y_train):
랜덤 포레스트 모델을 학습용 데이터에 맞춘다. 모델은 주어진 데이터를 사용하여 특징과 클래스 간의 관계를 학습한다.
model.predict(X_test):
학습된 모델을 사용하여 테스트 데이터에 대한 예측을 수행한다. 각 샘플에 대해 모델은 예측된 클래스를 반환한다.
accuracy_score(y_test, y_pred):
실제 값(y_test)과 모델의 예측 값(y_pred)을 비교하여 정확도를 계산한다. 정확도는 전체 예측 중 올바르게 분류된 샘플의 비율을 나타낸다.
confusion_matrix(y_test, y_pred):
혼동 행렬을 생성한다. 혼동 행렬은 각 클래스에 대한 실제 및 예측 값의 개수를 보여준다.

수많은 에러 중 해결하기 위해 시간이 가장 오래 걸려서 기억에 남는 에러의 발생 지점과 해결 과정은 다음과 같다.

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
~₩AppData₩Local₩Temp₩ipykernel_23804₩1586940290.py in <module>
     31 # 모델 생성 및 학습
     32 model = RandomForestClassifier(random_state=42)
---> 33 model.fit(X_train, y_train)
     34
     35 # 모델 성능 평가

~₩anaconda3₩lib₩site-packages₩sklearn₩ensemble₩_forest.py in fit(self, X, y, sample_weight)
    325         if issparse(y):
    326             raise ValueError("sparse multilabel-indicator for y is not supported.")
--> 327         X, y = self._validate_data(
    328             X, y, multi_output=True, accept_sparse="csc", dtype=DTYPE
    329         )

~₩anaconda3₩lib₩site-packages₩sklearn₩base.py in _validate_data(self, X, y, reset, validate_separately, **check_para
ms)
    579                 y = check_array(y, **check_y_params)
    580             else:
--> 581                 X, y = check_X_y(X, y, **check_params)
    582             out = X, y
    583

~₩anaconda3₩lib₩site-packages₩sklearn₩utils₩validation.py in check_X_y(X, y, accept_sparse, accept_large_sparse, dt
ype, order, copy, force_all_finite, ensure_2d, allow_nd, multi_output, ensure_min_samples, ensure_min_features, y
_numeric, estimator)
    962         raise ValueError("y cannot be None")
    963
--> 964     X = check_array(
    965         X,
    966         accept_sparse=accept_sparse,

~₩anaconda3₩lib₩site-packages₩sklearn₩utils₩validation.py in check_array(array, accept_sparse, accept_large_sparse,
dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator)
    744                     array = array.astype(dtype, casting="unsafe", copy=False)
    745                 else:


~₩anaconda3₩lib₩site-packages₩sklearn₩utils₩validation.py in check_array(array, accept_sparse, accept_large_sparse,
dtype, order, copy, force_all_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, estimator)
    744                     array = array.astype(dtype, casting="unsafe", copy=False)
    745                 else:
--> 746                     array = np.asarray(array, order=order, dtype=dtype)
    747             except ComplexWarning as complex_warning:
    748                 raise ValueError(

~₩anaconda3₩lib₩site-packages₩pandas₩core₩generic.py in __array__(self, dtype)
   2062
   2063     def __array__(self, dtype: npt.DTypeLike | None = None) -> np.ndarray:
-> 2064         return np.asarray(self._values, dtype=dtype)
   2065
   2066     def __array_wrap__(

ValueError: could not convert string to float: 'b8:27:eb:bf:9d:51'
```

해당 에러는 RandomForestClassifier 모델이 문자열 타입의 데이터를 처리하지 못해서 발생한 것이다. 따라서 문자열을 숫자로 변환해야 한다. 하지만 바로 변환이 가능한 것이 아닌 범주형 데이터에 대해 인코딩을 수행해야 가능하다. 범주형 데이터를 인코딩한 후에 모델을 다시 학습시켜 해결이 가능하다.

```
# 범주형 변수 처리
categorical_columns = ['device']  # 다른 범주형 변수가 있는 경우 이곳에 추가

for column in categorical_columns:
    if column in data.columns:
        data = pd.get_dummies(data, columns=[column])

# 머신러닝 모델 학습
X = data.drop('motion', axis=1)  # 'motion'을 예측하려면 해당 열을 지정해야 합니다.
y = data['motion']

# 데이터를 학습용과 테스트용으로 나눕니다.
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 모델 생성 및 학습
model = RandomForestClassifier(random_state=42)
model.fit(X_train, y_train)
```

학습 모델의 성능 평가는 다음과 같다.

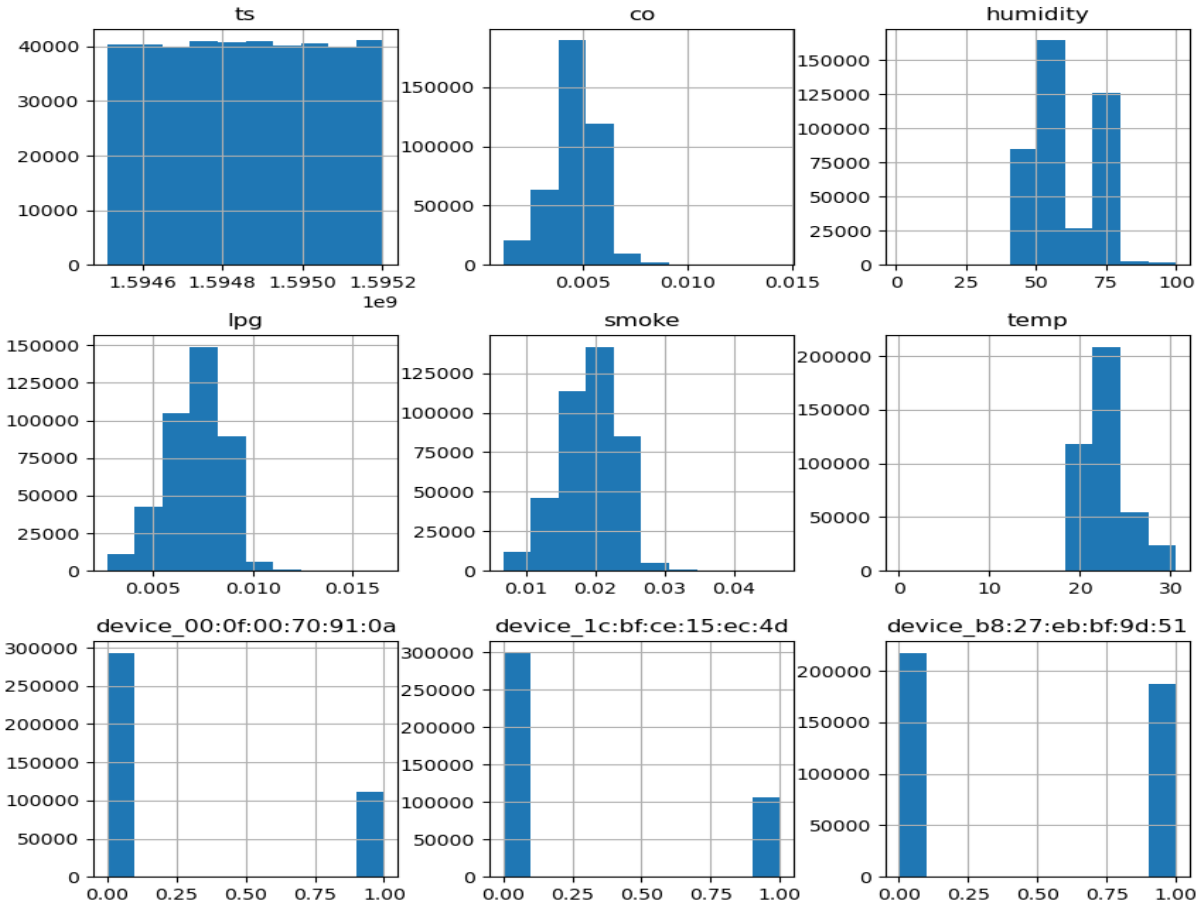
```
Index(['ts', 'device', 'co', 'humidity', 'light', 'lpg', 'motion', 'smoke',
       'temp'],
      dtype='object')
                 ts             co       humidity            lpg  ₩
count  4.051840e+05  405184.000000  405184.000000  405184.000000
mean   1.594858e+09       0.004639      60.511694       0.007237
std    1.994984e+05       0.001250      11.366489       0.001444
min    1.594512e+09       0.001171       1.100000       0.002693
25%    1.594686e+09       0.003919      51.000000       0.006456
50%    1.594858e+09       0.004812      54.900000       0.007489
75%    1.595031e+09       0.005409      74.300003       0.008150
max    1.595203e+09       0.014420      99.900002       0.016567

               smoke           temp  device_00:0f:00:70:91:0a  ₩
count  405184.000000  405184.000000                405184.000000
mean        0.019264      22.453987                     0.275961
std         0.004086       2.698347                     0.446998
min         0.006692       0.000000                     0.000000
25%         0.017024      19.900000                     0.000000
50%         0.019950      22.200000                     0.000000
75%         0.021838      23.600000                     1.000000
max         0.046590      30.600000                     1.000000

       device_1c:bf:ce:15:ec:4d  device_b8:27:eb:bf:9d:51
count             405184.000000             405184.000000
mean                   0.261407                  0.462632
std                    0.439402                  0.498602
min                    0.000000                  0.000000
25%                    0.000000                  0.000000
50%                    0.000000                  0.000000
75%                    1.000000                  1.000000
max                    1.000000                  1.000000
```

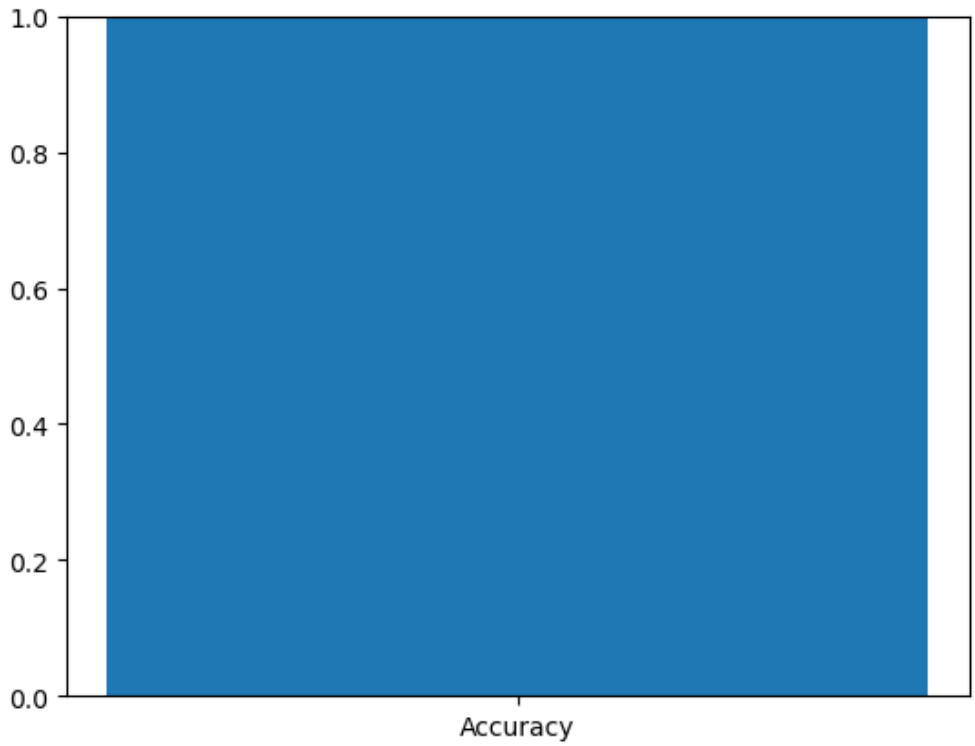
```
Accuracy: 0.9986425953576761
Confusion Matrix:
[[80924    27]
 [   83     3]]
```

총 데이터의 개수는 405,184 개이며, 'motion'을 예측하기 위해 사용된 독립 변수는 'co', 'humidity', 'light', 'lpg', 'smoke', 'temp', 'device'이다. 이 중 'device'는 범주형 변수로 처리되었고, 각 범주에 대해 더미 변수가 생성되었다.
모델의 성능은 정확도로 측정되었으며, 정확도는 약 99.86%이다. 혼동 행렬을 통해 예측 결과를 분석해보면 다음과 같다.

True Negative (TN): 80,924
False Positive (FP): 27
False Negative (FN): 83
True Positive (TP): 3

결과 시각화는 다음과 같다.





5. 개발 후기

제가 개발한 '환경 센서를 통한 동작 예측 모델'은 전체적으로 만족하지만, 아쉬운 부분이 있었습니다. 정확도는 높게 나타났지만, 클래스 불균형으로 인해 모델이 'motion' 클래스를 제대로 예측하지 못하는 경우가 발생했으며 특히, True Positive (실제 'motion'이면서 예측도 'motion'인 경우)가 매우 낮게 나타났습니다. 이러한 클래스 불균형을 다루기 위해서는 추가적인 평가 지표나 클래스 가중치를 고려해야 할 것으로 생각했습니다.

또한 개발 과정에 있어서 강의를 수강하며 들은 내용을 바탕으로 적용하는 것이 가능하다는 것을 느꼈고, 그것이 수치로 명확하게 표현되었을 때 만족감도 높았던 것 같습니다.

마지막으로 기억에 남았던 개발 과정은 '환경 센서 데이터' 외에도 '날씨 예측 데이터', '대한민국 지리 정보 데이터'를 추가로 병합하여 조금 더 모델을 강화하려는 시도를 했던 것입니다. 다만 병합하는 과정이 너무 복잡했고, 대응하지 않는 데이터가 너무 많아서 아직은 구현하기에 실력적인 부분이 부족했지만 추후 연구를 통해 강화된 모델을 개발할 예정입니다.