소음 발생 장소 예측기

학번: 1949099

이름: 박상현

Github address: snagheonpark

e-mail : gustid23@naver.com

1. 안전 관련 머신러닝 모델 개발의 목적

학습 모델 활용 대상: 장소 혹은 기계 등에서 소음이 어디서 많이 발생하는지 찾기위한 분야의 사람들 ex) 안전관리자, 기계설비사

실제(물리적) 위치라는 독립 변수를 입력하여 소음이 가장 많은 위치라는 종속변수를 예측한다.

개발의 의의 : 소음이 어디서 발생하는지 찾은 후 그 소음으로 인해 사람들에게 피해를 줄이기위한 안전대책을 세움으로써 소음으로 인한 피해를 방지할 수 있는 가치를 가져다 준다.

2. 안전 관련 머신러닝 모델의 네이밍의 의미

소음 발생 발생 장소 예측기는 말 그대로 소음이 어디서 발생하는지 알려주는 것이다.

3. 개발 계획

'측정된 위치'의 데이터와 , kalman Filter를 사용하여 추정된 위치 ('kalman position'), 그리고 실제위치 ('actual position') 를 구하여 이를 matplotlib을 이용해 시각화 한다. 이때 머시런닝 모델로 kalman Filter를 사용할 것인데 이는 시스템 상태를 추적하는 데 사용되는 상태 공간 모델이며, 이를 통해 시스템의 상태를 업데이트하고 노이즈를 제거할 수 있다. kalman Filte를 통해 예측된 위치 추정 결과를 시각화하고, 실제 위치와 비교하여 모델의 정확성을 확인할 수 있다. 이 모델은 노이즈를 최소화하여 실제 위치에 가까운 예측을 제공할 것으로 기대된다. 그럼 이를 이용하여 잡음을 줄이거나 노이즈를 처리하는 방법을 고려할 것이다. 이때 성능 지표는 모델의 성능을 평가하기 위해 예측된 위치와 실제 위치 간의 거리(오차) 측정, Mse(Mean squared Error), RMSE(Root Mean Squared Error)

등을 사용한다. 데이터를 훈련 세트와 테스트 세트로 분할하여 모델을 학습하고 검증한다. 테스트 세트에서 모델의 예측 성능을 확인하고, 가능하다면 교차 검증을 수행하여 모델의 일반화 능력을 평가할 수 있도록 진행할 것이다.

4. 개발 과정

```
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt

```

먼저 'numpy'는 행렬처리와 배열 처리를 해야하기 때문에 실행한다.
'numpy.linalg import inv'는 numpy의 선형 대수 모듈에 있는 함수로, 역행렬을 계산하는데 사용하는데 여기선 kalman Filter의 수학적 구현에 필요한 계산 중 하나인 행렬의 역행렬을 구하기위해 이 함수를 실행한다.
'matplotlib.pyplot'은 데이터의 값을 그래프를 통해 시각화 하기위해 실행한다.

```
import numpy as np
from numpy.linalg import inv
import matplotlib.pyplot as plt


n_iters = 50
```

1. 칼만 필터 알고리즘의 각 단계를 시뮬레이션하기에 필요한 코드이다.
여기선 반복횟수를 넣는다. 이때 50은 가상의 입력값이다. 그러면 50번의 반복 후 시스템의 초기상태에서부터 칼만 필터를 통해 50번의 위치 추정을 수행한다.

```
# 2. 대상의 실제 위치 정의
actual_x = 0
actual_y = 0
Q = np.full((2,2), 0.0001)
print('Q matrix:', Q.shape, '\n\n', Q)
```

2. 대상의 실제 위치를 넣어야 하므로 위 코드를 넣어 초기에 '실제 위치'를 나타내기 위한 변수를 설정한다.
이때 Q는 칼만 필터에서 프로세스 잡음의 공분산 행렬을 나타내는데 여기서 Q행렬은 작은 값인 0.0001로 초기화되었으며, 이는 칼만 필터에서의

프로세스 노이즈를 낮게 설정하여 시스템 모델의 예측 정확도를 과소평가하지 않고 낮은 불확실성을 나타낸다. 이 불확실성은 시스템의 상태 변화에 대한 예측 오차를 나타내며 시스템 모델의 정확도의 정보를 담기위해 사용한다.

```
# 2. 대상의 실제 위치 정의
actual_x = 0
actual_y = 0
Q = np.full((2,2), 0.0001)
print('Q matrix:', Q.shape, '\n\n', Q)

# 3. 소음 위치 측정기 만들기
# 50개의 랜덤 위치를 생성
x_pos = np.random.normal(0, 0.5, n_iters)
y_pos = np.random.normal(0, 0.5, n_iters)
```

3. 소음 위치 측정기를 만들기 위해 x 값과 y 값에 대한 정규 분포 소음 측정값을 여기에 생성하게 한다. 먼저 x_pos와 y_pos는 각각 평균이 0이고 표준편차가 0.5인 정규 분포에서 샘플을 생성한다.

```
# 3. 소음 위치 측정기 만들기
# 50개의 랜덤 위치를 생성
x_pos = np.random.normal(0, 0.5, n_iters)
y_pos = np.random.normal(0, 0.5, n_iters)

# 위치를 행렬로 표시
measurements = np.stack((x_pos, y_pos), axis=1).reshape((n_iters,2,1))

print('Measurements:', measurements.shape, '\n\n', measurements[0:5], '\n', '...')
```

그 후 이러한 위치 측정값들을 2차원 배열로 표현하기 위해 np.stack을 사용하여 하나의 배열로 합친다. 이 배열은 각 측정값을 가로로 늘어뜨린 후 다시 (n_iters, 2, 1)모양으로 변형하여 measurements변수에 저장한다. 이를 통해 위치를 행렬로 표시한다.

```
# 3. 소음 위치 측정기 만들기
# 50개의 랜덤 위치를 생성
x_pos = np.random.normal(0, 0.5, n_iters)
y_pos = np.random.normal(0, 0.5, n_iters)

# 위치를 행렬로 표시
measurements = np.stack((x_pos, y_pos), axis=1).reshape((n_iters,2,1))

print('Measurements:', measurements.shape, '\n\n', measurements[0:5], '\n', '...')

# 측정 오차의 공분산
R = np.diag([0.25,0.25])
print('R matrix: ', R.shape, '\n\n', R)
```

측정된 데이터가 실제 값에서 얼마나 벗어나 있는지를 나타내기 위해 'R'행렬을 이용한다. 이때 'R'은 대각행렬로 정의 하였고 '[0.25, 0.25]'라는 대각 성분을 가지게 설정하여 x와y의 각각의 측정 오차가 0.25로 가정하게 한다.

```
# 4. 빈 배열 만들기
# _hat --> 위치 추정 (칼만 필터의 경우 이 위치를 확률분포로 나타냄)
x_hat = np.zeros((n_iters, 2,1))
P = np.zeros((n_iters, 2,2))
x_hat_min = np.zeros((n_iters, 2,1))
P_min =np.zeros((n_iters, 2,2))
K = np.zeros((n_iters,2,2))
```

4. 칼만 필터가 적용된 후 예상 위치를 저장하기 위해 빈 배열이 필요하기 때문에 빈 배열을 만들어 줘야한다.

칼만 필터가 시스템의 상태를 추정하기 위해x_hat으로 나타내어 추정된 상태 값을 나타낸다.

추정된 상태 값에 대한 불확실성을 나타내는데 사용하기 위해 'P'라는 오차 공분산 행렬을 사용한다. 이때 대각 성분은 추정된 위치의 분산을 나타나게 하고, 비대각 성분은 위치들 간의 공분산을 의미하게 한다.

현재 상태 예측을 알기 위해 이전 상태의 추정치를 사용하여 예측하는 상태 예측값 'x_hat_min' 을 입력한다.

현재 상태의 예측에 대한 오차의 추정치를 구하기 위해 사전 오차 공분산 행렬을 하기위한 'P_min'을 입력한다.

마지막으로 측정값과 사전 예측값 간의 오차를 토대로 상태 업데이트에 사용하기 위해 칼만 이득 행렬 'K'를 입력한다.
이 코드들을 이용해 필터가 측정값과 예측값 사이의 불확실성을 고려하여 시스템의 상태를 추정하게 사용한다.

```
# 초기 상태
x_hat[0] = [[0], [0]]

# 초기 오차 공분산
P[0] = np.diag([1000.0, 1000.0])

print('x_hat[0]:\n\n', x_hat[0], '\n\n P[0] matrix: ', P[0].shape, '\n\n', P[0])
```

초기 위치와 초기 오차 공분산을 설정해야 한다. 'x_hat[0]'을 이용해 초기 위치 값을 0,0으로 설정하고 'P[0]'을 이용해 초기 오차 공분산 행렬을 설정한다.
이때 초기 공분산 행렬을 1000.0값을 주어 대각선 원소를 채운 2x2 행렬로 만든다. 이를 통해 이전 상태를 초기화하는데도 사용한다.

```
# 5. 수학적 계산을 위한 관련 행렬 정의
A = np.array([[1.0, 0.0],
              [0.0, 1.0]])

# H 행렬
H= np.eye(2)

# Kalman gain을 위한 단위 행렬
I = np.eye(2)

print('A matrix \n\n', A, '\n\n Hmatrix \n\n', H, '\n\n I matrix \n\n', I)
```

5. 수학적 계산을 위한 관련 행렬 정의
칼만 필터의 수학적 계산에 사용되는 행렬들을 정의 해준다.
먼저 시스템의 상태 변화를 나태내기 위해 A행렬을 이용한다. 변화가 없는 경우에 대해서 2차원 항등 행렬로 정의한다.
그리고 측정값과 상태의 관계를 정의하기 위해 H행렬 이용하고 여기소도 2차원 항등 행렬로 정의 한다.
마지막으로 칼만gain을 위한 단위 행렬인 I 행렬을 사용하여 칼만 Gain을 계산하거나 행렬 연상에서 항등 원소로 사용하게 한다.

```
#6. 계산 및 오차 감소 측정
for k in range(1, n_iters):
    # 구할수 있는 추정치 및 오차 공분산 계산
    x_hat_min[k] = A.dot(x_hat[k-1])
    P_min[k] = A.dot(P[k-1]).dot(A.T) + Q

    # Kalman gain 계산
    S = H.dot(P_min[k]).dot(H.T) + R
    K[k] = P_min[k].dot(H.T).dot(inv(S))

    # 사후 추정치 및 오차 공분산 계산
    x_hat[k] = x_hat_min[k] + K[k].dot(measurements[k] - H.dot(x_hat_min[k]))
    P[k] = (I-K[k]).dot(P_min[k])
```

6. 계산 및 오차 감소 측정

현재 추정치를 업데이트 하는 과정이 있어여 하므로 이전 추정치와 측정값을 이용한다. 이때 칼만 필터의 반복적인 단계를 이용한다. 'x_hat_min[k]'와 'P_min[k]' 둘 다 상태 변환 행렬 'A'를 사용해 현재 상태에서 다음 상태를 예측한다.

kalman gain계산을 위해 먼저 측정 행렬 'H'를 이용해 예측된 오차 공분산과 측정 노이즈 공분산 'R'을 합산하여 측정 소음의 총 공분산을 계산한다. 그리고 칼만 이득 행렬을 계산하기 위해 'K[K]'를 계산한다.

마지막으로 예측된 상태와 측정값 사이의 오차를 보정해야 하므로 측정값을 사용한다. 측정값 measurements[k]와 예측된 상태 x_hat_min[k]사이의 오차를 보정하여 실제 추정치 x_hat[k]를 계산한다. 그 후 'P[K]'를 이용해 측정 후 오차 공분산을 업데이트 한다.

```
# 7. 실제 오차 측정 및 추정 위치 시각화
plt.figure(figsize=(7,7))
for n in range(n_iters):
    plt.scatter(float(measurements[n][0]), float(measurements[n][1]),
                color='orange', label='measured position', alpha=0.7)
    plt.scatter(float(x_hat[n][0]), float(x_hat[n][1]),
                color='blue', label='kalman position', alpha=0.3)

    plt.scatter(actual_x, actual_y, color='red', s=100, label='actual position')
    plt.title('Actual Measured and Estimated Positions')
    plt.xlabel('X axis')
    plt.ylabel('y axis')
plt. show()
```

7. 실제 오차 측정 및 추정 위치 시각화

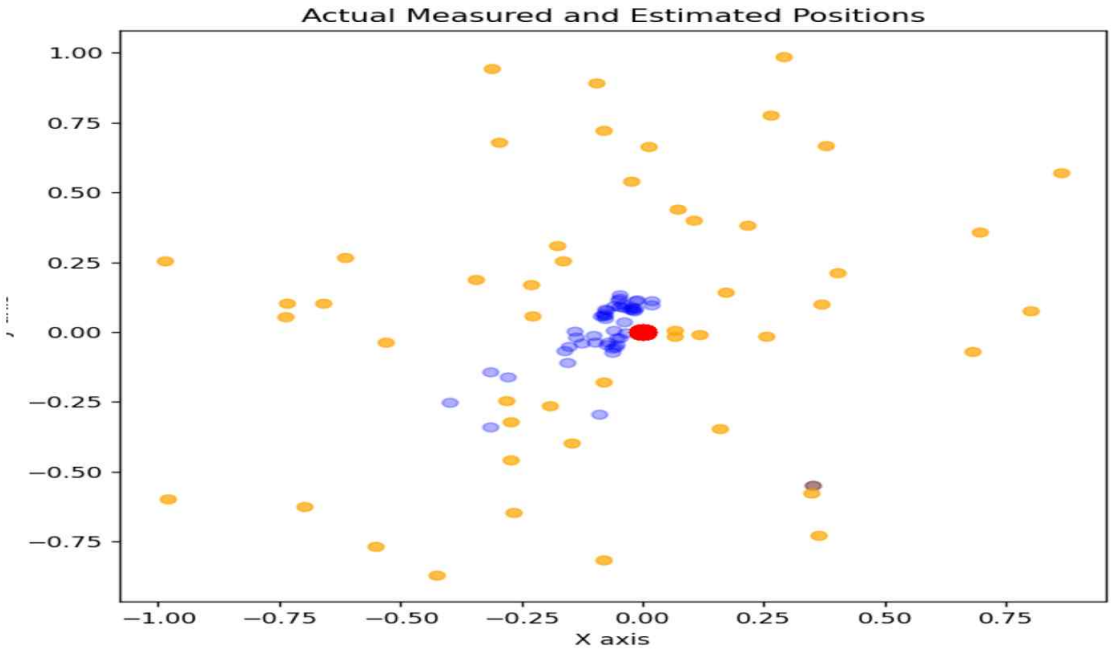
마지막으로 이를 그래프로 시각화를 해준다.

먼저 새로운 Figure를 생성하고, 크기를 7x7로 설정한다.

각각의 반복에서 측정값, 칼만 필터의 추정값, 실제 위치를 나타내기 위해 n_iters만큼 반복한다.

측정된 위치는 주황색, 칼만 필터가 추정한 위치는 파란색, 실제 위치는 빨간색으로 표시했다.

제목은 실제 측정 위치와 추정 위치로 'Actual Measured and Estimated Positions'으로 설정하였다. x축의 라벨과 y축의 라벨을 설정하며 그래프를 완성한다.



마지막으로 결과를 시각화하면 이런 그래프가 나타나게 된다.

5. 개발 후기

시작이 가장 어려웠던것 같다. 주제를 무엇을 잡을지 방향성을 어떻게 나아갈건지 가장 어려웠다. 주제를 안전 쪽을 생각하며 내가 여태껏 배운 안전관리 지식을 되돌아 볼 수 있었던 기회가 되었고 이 중에서 무엇이 내 진로와 관련이 있는지도 다시 돌아볼 수 있었다. 그 중에서도 나는 건설쪽을 희망하기 때문에 건설 분야에서 많이 발생하는 문제인 소음을 생각하게 되었고 이 소음들을 해결하기 위해서는 어디서 소음이 발생하는지가 가장 중요하다고 생각하여 이러한 소음을 추측할 수 있는 코드를 만들기로 하였다. 코드를 만들면서 컴퓨터 프로그래밍 실력 향상은 물론 파이썬을 통해 정말 다양한 것을 만들고 어떠한 분야에도 접목할 수 있다고 느꼈다. 안전관리를 하기위해서 컴퓨터 프로그래밍은 필수라 생각하며 이를 통해 일의 효율성을 높이고 일의 능률도 올라가 좀 더 전문적인 프로그래밍 능력이 필요할 것 같다고 느꼈다. 내가 안전관리자가 되었을때 어떤 현장을 가던 이런 코드들을 직접 만들어 현장에 필요한 머신러닝들을 개발할 수 있는 안전관리자가 될 수있도록 더 발전해야겠다고 느꼈다.