로봇신호처리 기말 프로젝트 발표

스마트IOT 학과 20235127 김민주

1

주제

머신러닝 회귀 모델을 활용한 비행체 Pitch 자세 추정

프로젝트 목표

Stanford Helicopter 데이터 셋을 활용해 비행체의 Pitch 자세를 추정하는 머신러닝 회귀 모델을 개발하고, 이를 칼만 필터와 비교해 성능을 평가.

- Pitch 자세 추정
- 성능 비교
- 비행 시나리오 분석
- 센서 융합 효과 검증
- 모델 최적화

설명

- 데이터셋
- 방법론

코드

```python
from google.colab import files
uploaded = files.upload()

import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.fft import fft, fftfreq
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.metrics import mean_squared_error, mean_absolute_error
from xgboost import XGBRegressor
```

파일 선택 파일 3개

- **smoother.txt**(text/plain) - 14064259 bytes, last modified: 2025. 5. 30. - 100% done
- **imugyro.txt**(text/plain) - 7715513 bytes, last modified: 2025. 5. 30. - 100% done
- **imuaccel.txt**(text/plain) - 7323979 bytes, last modified: 2025. 5. 30. - 100% done

```
Saving smoother.txt to smoother.txt
Saving imugyro.txt to imugyro.txt
Saving imuaccel.txt to imuaccel.txt
```

```python
# Ground Truth Pitch 보간 (degree 단위)
truth_time = truth_df['time'].values
pitch_gt_raw = truth_df['pitch'].values * 180 / np.pi
pitch_gt = np.interp(time, truth_time, pitch_gt_raw)
```

[8]
```python
# 데이터 분할
X_train, X_temp, y_train, y_temp = train_test_split(features, pitch_gt, test_size=0.3, random_state=42)
X_val, X_test, y_val, y_test = train_test_split(X_temp, y_temp, test_size=0.5, random_state=42)
```

[4]
```python
# 데이터 로드
accel_df = pd.read_csv("imuaccel.txt", sep=r'\s+', engine='python', header=None)
gyro_df = pd.read_csv("imugyro.txt", sep=r'\s+', engine='python', header=None)
truth_df = pd.read_csv("smoother.txt", sep=r'\s+', engine='python', header=None)

# 컬럼 이름 지정
accel_df.columns = ['type', 'time', 'acc_x', 'acc_y', 'acc_z']
gyro_df.columns = ['type', 'time', 'gyro_x', 'gyro_y', 'gyro_z']
truth_df.columns = ['type', 'time', 'pos_n', 'pos_e', 'pos_d',
                    'q_x', 'q_y', 'q_z', 'q_w',
                    'vel_n', 'vel_e', 'vel_d',
                    'w_n', 'w_e', 'w_d',
                    'vdot_n', 'vdot_e', 'vdot_d',
                    'wdot_n', 'wdot_e', 'wdot_d',
                    'roll', 'pitch', 'yaw']
```

[5]
```python
# 시간 및 dt 계산
time = accel_df['time'].values
dt = np.mean(np.diff(time))
```

[6]
```python
# 피처 엔지니어링
# 1. 가속도계 기반 Pitch 계산 (degree)
acc_x, acc_y, acc_z = accel_df['acc_x'], accel_df['acc_y'], accel_df['acc_z']
acc_pitch = np.arctan2(-acc_x, np.sqrt(acc_y**2 + acc_z**2)) * 180 / np.pi

# 2. 자이로스코프 gyro_y 적분 (Pitch 변화 추정)
gyro_y = gyro_df['gyro_y'].values
gyro_pitch = np.cumsum(gyro_y * dt) * 180 / np.pi  # rad/s -> degree

# 3. 입력 피처 데이터프레임 생성
features = pd.DataFrame({
    'acc_x': acc_x,
    'acc_y': acc_y,
    'acc_z': acc_z,
    'gyro_x': gyro_df['gyro_x'],
    'gyro_y': gyro_y,
    'gyro_z': gyro_df['gyro_z'],
    'acc_pitch': acc_pitch,
    'gyro_pitch': gyro_pitch
})
```

코드

```python
[12] # XGBoost 모델 학습
xgb = XGBRegressor(random_state=42)
xgb_param_grid = {
    'n_estimators': [50, 100],
    'max_depth': [3, 6, 10],
    'learning_rate': [0.01, 0.1]
}
xgb_grid = GridSearchCV(xgb, xgb_param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=-1)
xgb_grid.fit(X_train, y_train)

# 최적 모델로 예측
xgb_best = xgb_grid.best_estimator_
y_pred_xgb_val = xgb_best.predict(X_val)
y_pred_xgb_test = xgb_best.predict(X_test)
```

```python
# Random Forest 모델 학습 (최적화)
from tqdm import tqdm
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

rf = RandomForestRegressor(random_state=42)
rf_param_grid = {
    'n_estimators': [50],  # 단일 값으로 고정
    'max_depth': [10, 20],  # 최소화
    'min_samples_split': [2]
}
rf_grid = GridSearchCV(rf, rf_param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=1, ve
rf_grid.fit(X_train, y_train)

# 최적 모델로 예측
rf_best = rf_grid.best_estimator_
y_pred_rf_val = rf_best.predict(X_val)
y_pred_rf_test = rf_best.predict(X_test)

# 최적 하이퍼파라미터 출력
print("Best Parameters:", rf_grid.best_params_)
```

```python
# 칼만 필터 구현 (비교용)
def kalman_filter(acc_meas, gyro_rate, dt, Q=0.01, R=50.0):
    N = len(acc_meas)
    x_est = np.zeros(N)
    P = 1.0
    x = acc_meas[0]
    for k in range(1, N):
        x = x + gyro_rate[k] * dt
        P = P + Q
        K = P / (P + R)
        x = x + K * (acc_meas[k] - x)
        P = (1 - K) * P
        x_est[k] = x
    return x_est

pitch_est_kf = kalman_filter(acc_pitch, gyro_y, dt)
```

코드

```python
# Pitch 추정 비교 시각화
plt.figure(figsize=(12, 4))
plt.plot(time[:len(y_test)], y_test[:len(y_test)], label='Ground Truth', color='blue')
plt.plot(time[:len(y_test)], y_pred_rf_test[:len(y_test)], label='Random Forest', color='green'
plt.plot(time[:len(y_test)], y_pred_xgb_test[:len(y_test)], label='XGBoost', color='red')
plt.plot(time[:len(y_test)], pitch_est_kf[:len(y_test)], label='Kalman Filter', color='purple')
plt.title("Pitch Estimation Comparison (Test Set)")
plt.xlabel("Time [s]")
plt.ylabel("Pitch [deg]")
plt.grid()
plt.legend()
plt.show()
```

```python
# 성능 평가
def evaluate_model(y_true, y_pred, model_name):
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_true, y_pred)
    print(f"{model_name} - MSE: {mse:.4f}, RMSE: {rmse:.4f}, MAE: {mae:.4f}")
    return mse, rmse, mae
```

```python
# 검증 세트 결과
print("Validation Set Results:")
rf_mse_val, rf_rmse_val, rf_mae_val = evaluate_model(y_val, y_pred_rf_val, "Random Forest")
xgb_mse_val, xgb_rmse_val, xgb_mae_val = evaluate_model(y_val, y_pred_xgb_val, "XGBoost")
kf_mse_val = mean_squared_error(pitch_gt, pitch_est_kf)
print(f"Kalman Filter - MSE: {kf_mse_val:.4f}, RMSE: {np.sqrt(kf_mse_val):.4f}")

# 테스트 세트 결과
print("₩nTest Set Results:")
rf_mse_test, rf_rmse_test, rf_mae_test = evaluate_model(y_test, y_pred_rf_test, "Random Forest"
xgb_mse_test, xgb_rmse_test, xgb_mae_test = evaluate_model(y_test, y_pred_xgb_test, "XGBoost")
```

코드

```
# Pitch 추정 비교 시각화
plt.figure(figsize=(12, 4))
plt.plot(time[:len(y_test)], y_test[:len(y_test)], label='Ground Truth', color='blue')
plt.plot(time[:len(y_test)], y_pred_rf_test[:len(y_test)], label='Random Forest', color='green'
plt.plot(time[:len(y_test)], y_pred_xgb_test[:len(y_test)], label='XGBoost', color='red')
plt.plot(time[:len(y_test)], pitch_est_kf[:len(y_test)], label='Kalman Filter', color='purple')
plt.title("Pitch Estimation Comparison (Test Set)")
plt.xlabel("Time [s]")
plt.ylabel("Pitch [deg]")
plt.grid()
plt.legend()
plt.show()
```

```
# 피처 중요도 시각화 (Random Forest)
plt.figure(figsize=(10, 6))
feature_importance = pd.Series(rf_best.feature_importances_, index=features.columns)
feature_importance.sort_values().plot(kind='barh', color='green')
plt.title("Random Forest Feature Importance")
plt.xlabel("Importance")
plt.show()
```

실험 결과

```
Fitting 3 folds for each of 2 candidates, totalling 6 fits
[CV] END .max_depth=10, min_samples_split=2, n_estimators=50; total time=  24.8s
[CV] END .max_depth=10, min_samples_split=2, n_estimators=50; total time=  24.5s
[CV] END .max_depth=10, min_samples_split=2, n_estimators=50; total time=  24.6s
[CV] END .max_depth=20, min_samples_split=2, n_estimators=50; total time=  43.0s
[CV] END .max_depth=20, min_samples_split=2, n_estimators=50; total time=  43.8s
[CV] END .max_depth=20, min_samples_split=2, n_estimators=50; total time=  43.2s
Best Parameters: {'max_depth': 20, 'min_samples_split': 2, 'n_estimators': 50}
```

```python
# Random Forest 모델 학습 (최적화)
from tqdm import tqdm
from sklearn.ensemble import RandomForestRegressor
from sklearn.model_selection import GridSearchCV

rf = RandomForestRegressor(random_state=42)
rf_param_grid = {
    'n_estimators': [50],  # 단일 값으로 고정
    'max_depth': [10, 20],  # 최소화
    'min_samples_split': [2]
}
rf_grid = GridSearchCV(rf, rf_param_grid, cv=3, scoring='neg_mean_squared_error', n_jobs=1, verbose=2)
rf_grid.fit(X_train, y_train)

# 최적 모델로 예측
rf_best = rf_grid.best_estimator_
y_pred_rf_val = rf_best.predict(X_val)
y_pred_rf_test = rf_best.predict(X_test)

# 최적 하이퍼파라미터 출력
print("Best Parameters:", rf_grid.best_params_)
```

실험 결과

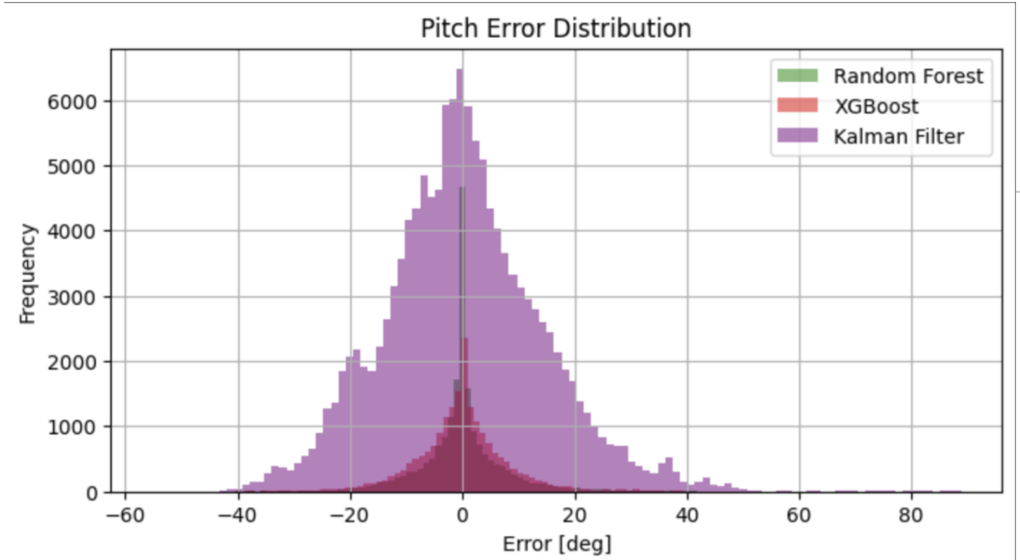
```
# 검증 세트 결과
print("Validation Set Results:")
rf_mse_val, rf_rmse_val, rf_mae_val = evaluate_model(y_val, y_pred_rf_val, "Random Forest")
xgb_mse_val, xgb_rmse_val, xgb_mae_val = evaluate_model(y_val, y_pred_xgb_val, "XGBoost")
kf_mse_val = mean_squared_error(pitch_gt, pitch_est_kf)
print(f"Kalman Filter - MSE: {kf_mse_val:.4f}, RMSE: {np.sqrt(kf_mse_val):.4f}")

# 테스트 세트 결과
print("\nTest Set Results:")
rf_mse_test, rf_rmse_test, rf_mae_test = evaluate_model(y_test, y_pred_rf_test, "Random Forest")
xgb_mse_test, xgb_rmse_test, xgb_mae_test = evaluate_model(y_test, y_pred_xgb_test, "XGBoost")
```

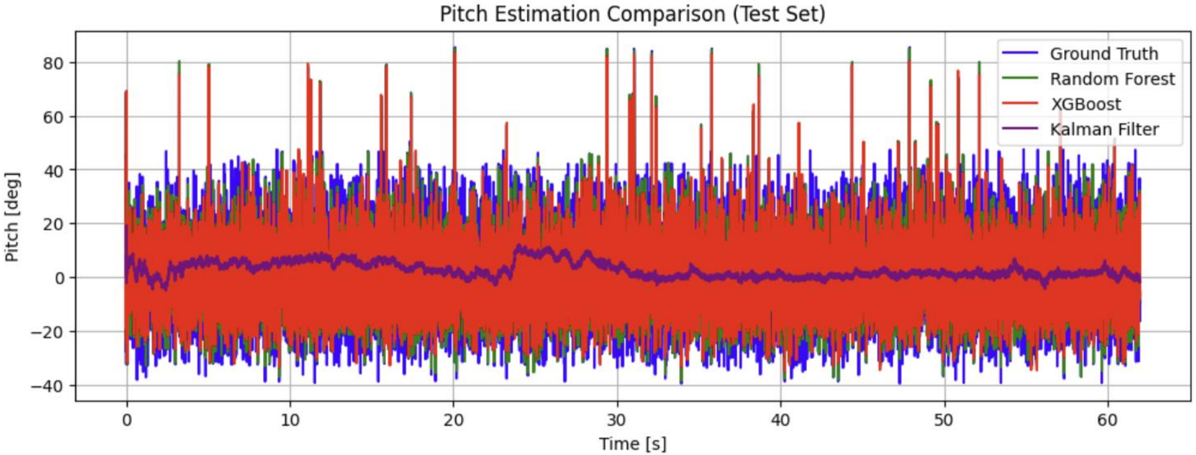
```
Validation Set Results:
Random Forest - MSE: 64.9065, RMSE: 8.0565, MAE: 4.8831
XGBoost - MSE: 76.5677, RMSE: 8.7503, MAE: 5.9410
Kalman Filter - MSE: 206.7860, RMSE: 14.3801

Test Set Results:
Random Forest - MSE: 62.6451, RMSE: 7.9149, MAE: 4.8464
XGBoost - MSE: 73.1853, RMSE: 8.5548, MAE: 5.8846
```

📌 실험 결과

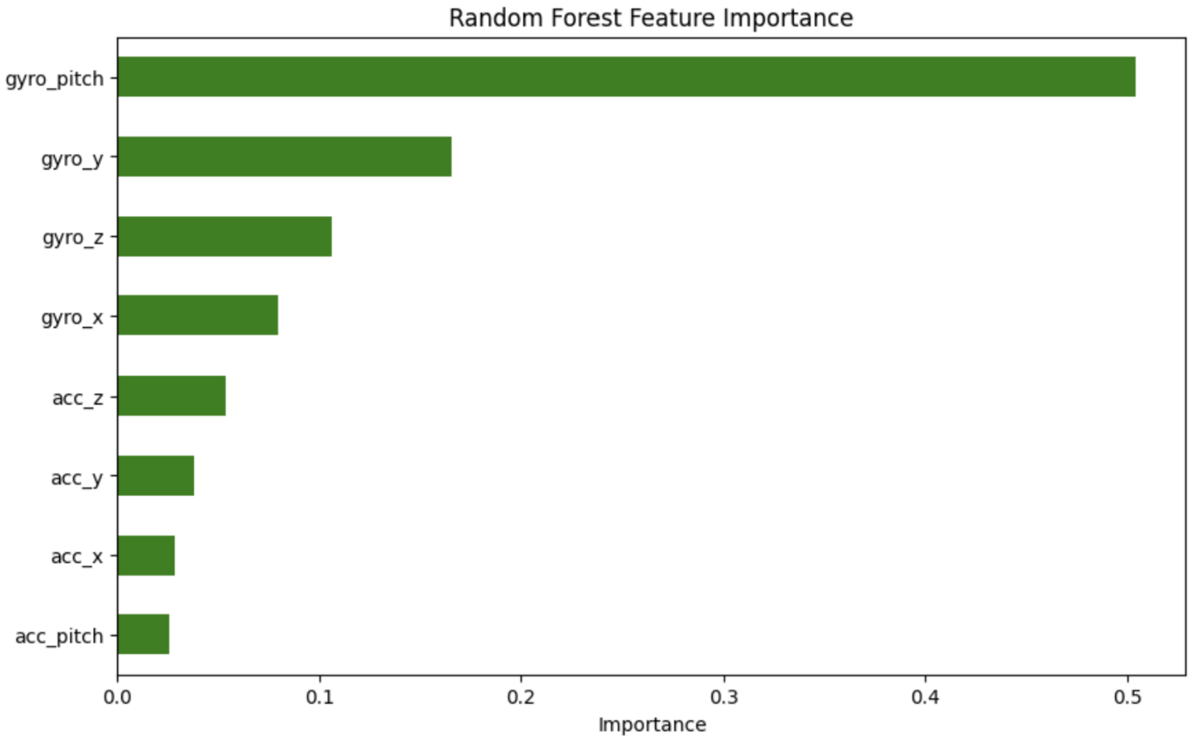


Pitch Error Distribution
Random Forest
XGBoost
Kalman Filter
Frequency
6000
5000
4000
3000
2000
1000
0
−60
−40
−20
0
20
40
60
80
Error [deg]



Pitch Estimation Comparison (Test Set)
Ground Truth
Random Forest
XGBoost
Kalman Filter
Pitch [deg]
80
60
40
20
0
−20
−40
0
10
20
30
40
50
60
Time [s]

실험 결과



Random Forest Feature Importance
gyro_pitch
gyro_y
gyro_z
gyro_x
acc_z
acc_y
acc_x
acc_pitch
0.0
0.1
0.2
0.3
0.4
0.5
Importance

Thanks