



AI 개발자 트랙 미니프로젝트 1차

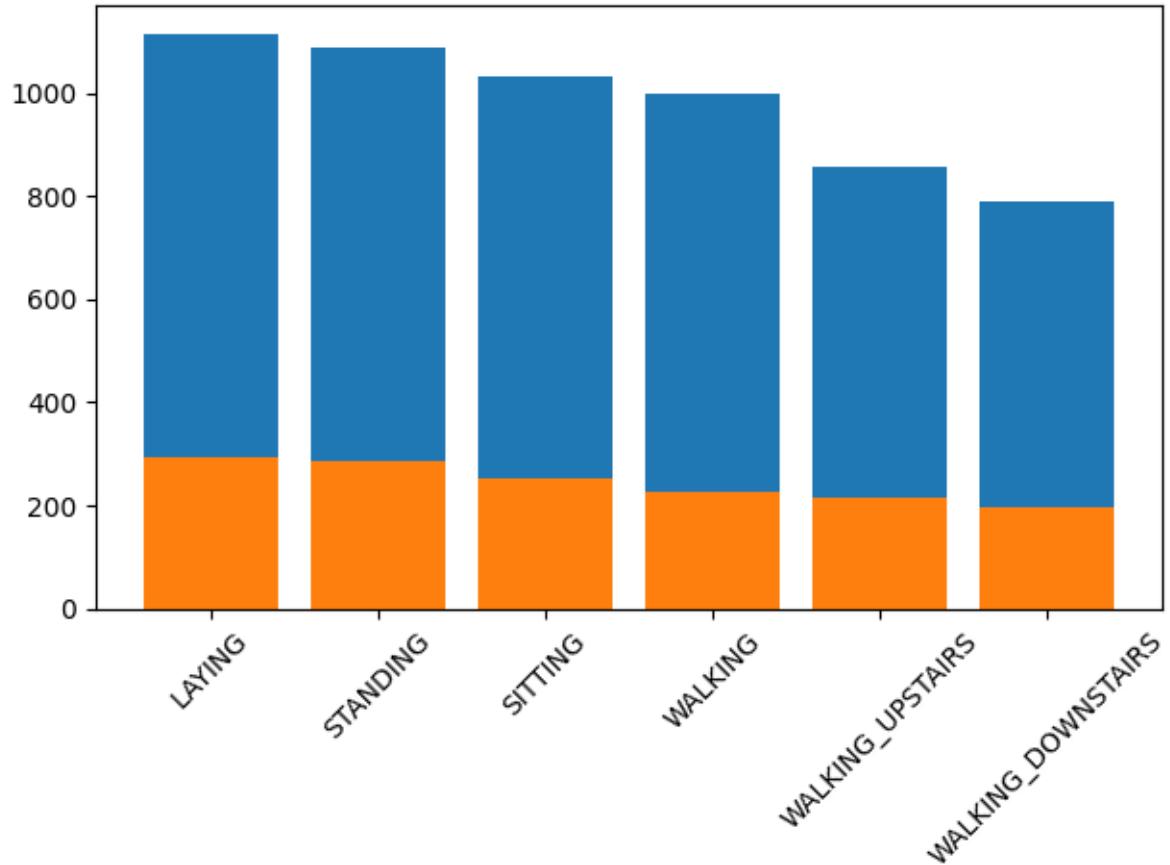
조별 발표 템플릿

AI 05반 10조



1. 탐색적 데이터 분석

- train, test 범주별 빈도수 시각화



Activity 범주별 빈도수 시각화

모델은 train 분포와 test 분포가 유사할 때,
안정적으로 작동합니다.

따라서, 두 데이터셋의 분포를 시각화 하여
차이를 한눈에 비교함으로써,
데이터의 불균형이나 편향을 사전에 확인했습니다.

1. 탐색적 데이터 분석

- train과 validation 분할 및 모델 생성 시 random_state 활용

random_state 활용 장점

(3) 데이터분할2 : train, validation

- 세부 요구사항
 - train : val = 8 : 2 혹은 7 : 3
 - random_state 옵션을 사용하여 다른 모델과 비교를 위해 성능이 재현되도록 합니다.

```
1 #데이터 분할 진행(train:val = 8:2 혹은 7:3 권장)
2 X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)
```

1. 결과 재현성 확보

- 같은 코드, 같은 데이터라도 랜덤 시드가 다르면 학습 결과(정확도, loss 등)가 달라질 수 있습니다.
- random_state를 사용해 랜덤 과정이 매번 동일하게 일어나도록 통제하여 프로젝트 공유 시 동일 결과 재현이 가능하도록 했습니다.

2. 모델 비교의 공정성

- 두 개 이상의 모델을 비교할 때, 만약 데이터 분할이 매번 다르게 일어나면 모델 성능 차이가 랜덤 분할 탓인지 모델 차이 때문인지를 구분하기 어렵습니다.
- 따라서, random_state로 분할해서 완전히 동일한 데이터셋 기준으로 비교가 가능하도록 했습니다.

2. 기본 모델링

- train과 validation 분할 시 stratify 옵션 사용

```
X_train, X_val, y_train, y_val = train_test_split(  
    X, y_encoded,  
    test_size=0.2,  
    random_state=42,  
    stratify=y_encoded # stratify값을 target 값으로 지정해주면 target의 class 비율을 유지 한 채로 데이터 셋을 split  
)
```

stratify 사용 시 장점

- 일반적인 train_test_split은 랜덤하게 데이터를 나누기 때문에 클래스가 불균형한 경우 train/test 간 비율이 달라질 수 있습니다.
- stratify 사용 시 train과 val 모두 원본 데이터의 클래스 비율을 그대로 유지합니다.
- 이를 통해 학습 데이터와 평가 데이터가 같은 분포를 가지므로, 평가 결과의 안정성과 신뢰성을 높였습니다.

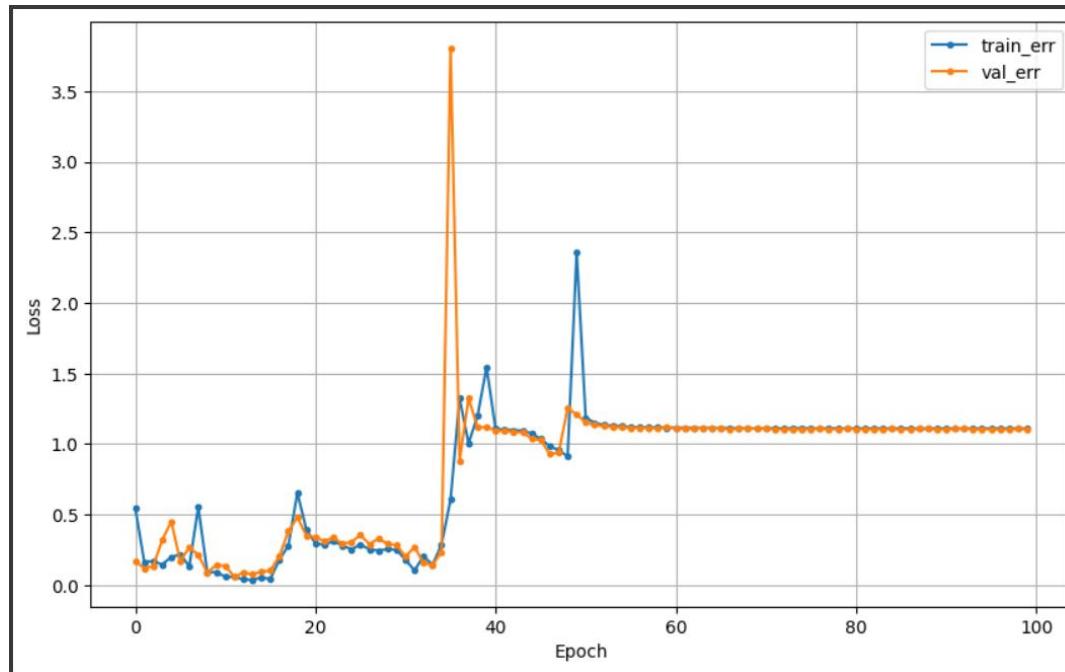
2. 기본 모델링

학습률 설정

학습률을 너무 크게 설정하면 Global minima를 건너뛰고 loss 가 튀어버리는 현상이 발생했습니다.

학습 곡선의 그래프에 중간중간 Loss값이 튀는 것을 보아 학습률이 너무 커서 발생하는 문제로 예상되었습니다.

learning rate 를 0.01 에서 0.001 로 수정하였습니다.



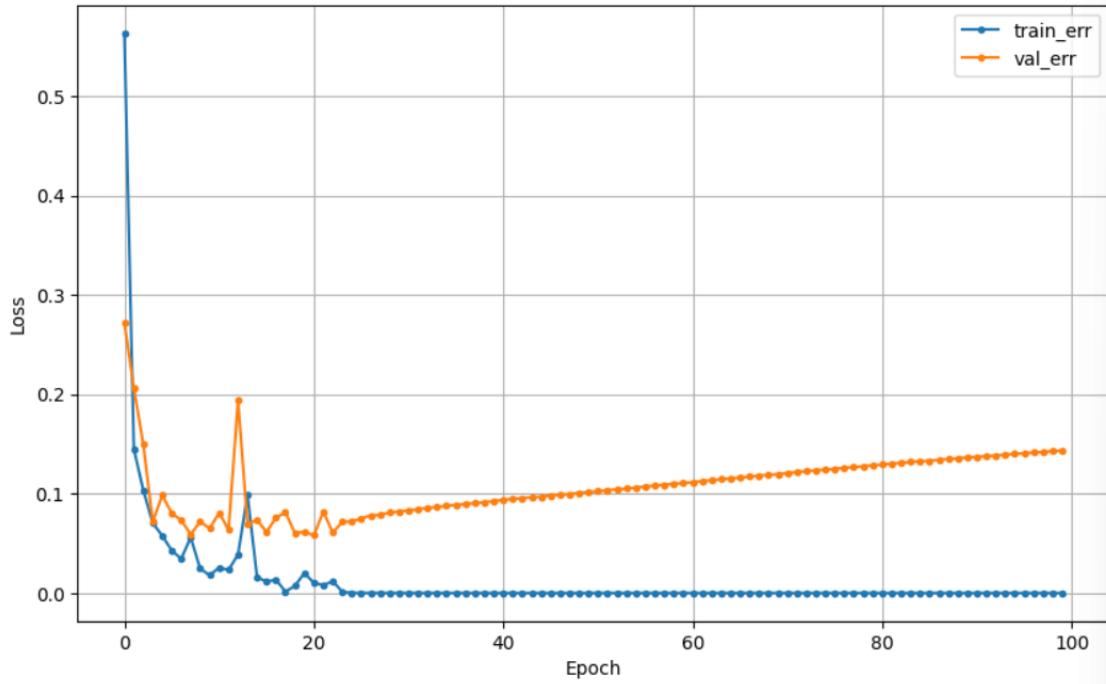
[[222 0 0 1 0 0]	precision	recall	f1-score	support
[190 0 0 1 0 0]	0	0.34	1.00	223
[242 0 0 1 0 0]	1	0.00	0.00	191
[0 0 0 188 0 0]	2	0.00	0.00	243
[0 0 0 159 0 0]	3	0.36	1.00	188
[0 0 0 173 0 0]]	4	0.00	0.00	159
	5	0.00	0.00	173
	accuracy		0.35	1177
	macro avg	0.12	0.33	1177
	weighted avg	0.12	0.35	0.18 1177

2. 기본 모델링

학습률 조정

```
# 컴파일 및 학습 학습률 0.01->0.001로 수정
model_3.compile(optimizer=Adam(learning_rate=0.001), loss='sparse_categorical_crossentropy')

hist_3 = model_3.fit(x_train, y_train, epochs=100, validation_split=0.2, verbose=0).history
```



[[223 0 0 0 0 0]					
[1 180 9 0 0 1]					
[0 11 232 0 0 0]					
[0 0 0 188 0 0]					
[0 0 0 0 159 0]					
[0 0 0 1 1 171]]					
precision	recall	f1-score	support		
0	1.00	1.00	1.00	223	
1	0.94	0.94	0.94	191	
2	0.96	0.95	0.96	243	
3	0.99	1.00	1.00	188	
4	0.99	1.00	1.00	159	
5	0.99	0.99	0.99	173	
accuracy			0.98	1177	
macro avg	0.98	0.98	0.98	1177	
weighted avg	0.98	0.98	0.98	1177	

2. 기본 모델링

mapping vs LabelEncoding

LabelEncoding 방식은 클래스 값에 임의의 정수를 자동으로 부여하기 때문에 인코딩 값의 의미나 순서를 명확하게 통제하기 어려웠습니다.

때문에 각 클래스에 대한 인코딩 값을 직접 정의하는 매핑 방식을 사용했습니다.
이를 통해 인코딩 과정의 일관성을 확보했습니다.

```
# 정수 인코딩 변환(LabelEncoder)
encoder = LabelEncoder()
activity_map = {
    'LAYING': 0,
    'SITTING': 1,
    'STANDING': 2,
    'WALKING': 3,
    'WALKING_UPSTAIRS': 4,
    'WALKING_DOWNSTAIRS': 5
}
y2 = y.map(activity_map)
```

데이터 분할 8:2

데이터 분할은 8:2로 진행하였습니다.

- train data가 부족할 경우 모델의 성능을 판단하기 어려울 것이라고 판단하여 validation 셋을 더 작게 설정하였습니다.

Dense

Dense - 은닉층에는 ReLU 활성화 함수를 사용하였습니다.

3. 단계별 모델링

모델링 수

100으로 시작했는데, 과적합 현상이 보였습니다. 따라서, early stopping (혹은 Dropout)을 주어서 해결했습니다.

```

▶ 1 # 컴파일 및 학습
2 es = EarlyStopping(monitor='val_loss', min_delta=0.005, patience=10, verbose=1)
3 model1.compile(optimizer=Adam(learning_rate=0.01), loss = 'sparse_categorical_crossentropy')
4 hist = model1.fit(X_train, y_train, epochs=100, validation_split=0.2, verbose=0, callbacks=[es]).history

```

→ Epoch 13: early stopping

최종적으로 Dropout이 아닌, EarlyStopping 사용 이유

상황	이유
데이터셋이 작거나 중간 규모	Dropout을 적용하면 데이터 손실 효과가 커져 학습 불안정성이 증가함. EarlyStopping은 불필요한 epoch를 방지하면서 과적합을 줄임.
모델이 비교적 단순함	Dropout의 효과가 크지 않음. 대신 EarlyStopping으로 "언제 멈출지" 제어하는 게 더 효율적.
정확한 일반화 성능이 중요	EarlyStopping은 "val_loss가 최소일 때의 모델"을 자동 선택하므로, 과적합 방지와 일반화 사이의 균형을 가장 잘 맞춤.

3. 종합 결과

- Standard와 MinMax 스케일러

초기에는 데이터가 이상치에 덜 민감하다는 이유로 StandardScaler를 사용하여 실험을 진행했습니다.
그러나 다양한 모델을 반복적으로 실험한 결과, MinMaxScaler 방식이 더 우수한 성능을 보였습니다.

이를 통해, 데이터 처리나 모델링에 대한 이론적 선택도 중요하지만,
궁극적으로는 지속적인 실험과 비교를 통해 직접 최적의 방법을 찾아가는 과정이
더 중요하다는 점을 깨달았습니다.

구분	StandardScaler	MinMaxScaler
스케일링 방식	평균 0, 표준편차 1로 변환	최소값 0, 최대값 1로 변환
이상치 영향	이상치(outlier)에 덜 민감	이상치에 매우민감
분포 유지	정규분포에 가까운 경우 유리	원래 분포 비율 유지, 정규분포와 관계 없음

