당뇨 예측 모델 정확도 저하 원인 분석 및 해결 보고서

> 작성일: 2026-02-20

---

1. 증상

사용자가 Flutter 앱에서 당뇨 예측을 실행했을 때, **결과가 비정상적으로 낮게** 나오는 현상이 발생했다.

입력값	예상	실제 결과
혈당 148, BMI 33.6, 나이 50, 임신 6회	당뇨 위험 (높은 확률)	**정상 범위 (26.3%)**
혈당 185, BMI 40, 나이 55, 임신 8회	당뇨 위험 (매우 높은 확률)	**정상 범위 (낮은 확률)**

명백히 고위험군인 입력에도 "정상"이 나오고, 정상/고위험 구분이 거의 되지 않는 상태였다.

---

2. 원인 분석

노트북 `24-5.final.ipynb`을 분석한 결과, **두 가지 원인**이 겹쳐 있었다.

2-1. 전처리 파이프라인 불일치 (1차 원인)

노트북에서 모델을 학습할 때 사용한 **데이터 전처리 흐름**은 아래와 같다.

```
원본 수치 (혈당=148, BMI=33.6 ...)
    ↓
0값을 결측(NaN)으로 처리
    ↓
IQR 기반 이상치 클리핑
    ↓
StandardScaler 표준화 (z-score)   ← mean, std 기준으로 정규화
    ↓
KNN Imputer로 결측값 보간
    ↓
통계적 구간화 (1, 2, 3, 4)        ← pd.cut으로 4구간 분류
    ↓
모델 학습 (RandomForest)
```

그런데 **기존 FastAPI 서버**에서는 전처리를 전혀 하지 않고:

```
원본 수치 (혈당=148, BMI=33.6 ...)
    ↓
그대로 모델에 입력    ← 모델은 1~4 범위의 구간값을 기대하는데 148이 들어감
```

모델이 1~4 범위의 값을 기대하는데 148, 33.6 같은 원본 수치가 들어가니 예측이 완전히 틀어졌다.

2-2. 구간화 자체로 인한 성능 저하 (2차 원인)

1차 원인을 해결하여 구간화까지 적용하더라도, **노트북 실험 결과를 비교하면 구간화 자체가 성능을 크게 떨어뜨리고 있었다.**

노트북 내에서 동일한 데이터에 대해 두 가지 방식으로 실험한 결과:

StandardScaler만 적용한 경우 (Cell 6)

시나리오	최적 모델	Accuracy	F1 Score
A: 혈당 포함	**AdaBoost**	**0.8117**	**0.7184**
B: 혈당 미포함	**RandomForest**	**0.7143**	**0.5600**

StandardScaler + 구간화(1~4) 적용한 경우 (Cell 10)

시나리오	최적 모델	Accuracy	F1 Score
A: 혈당 포함	RF	**0.6494**	**0.0357**
B: 혈당 미포함	-	대폭 하락	대폭 하락

구간화를 하면 연속형 수치를 4개의 카테고리로 압축하므로 **정보 손실이 발생**한다.
예를 들어, 혈당 140과 199는 모두 "4"로 동일하게 취급되어 모델이 세밀한 차이를 학습할 수 없게 된다.

2-3. 잘못된 모델 저장 (근본 원인)

노트북의 마지막 모델 저장 셀(Cell 12)에서, **구간화된 데이터(`X_stat_cat`)로 학습한 모델**을 `joblib` 파일로 저장했다:

```
# Cell 12 (노트북 원본) — 문제의 코드
model_sugar = RandomForestClassifier(n_estimators=200, max_depth=5,
random_state=42)
model_sugar.fit(X_stat_cat[features_a], y_final)    # ← 구간화된 데이터로 학습
joblib.dump(model_sugar, '../Data/model_sugar.joblib')
```

즉, 성능이 0.81인 Cell 6의 모델이 아니라, **성능이 0.65인 Cell 10의 구간화 모델을 저장**해 버린 것이다.

---

3. 전체 원인 요약 (흐름도)

```
노트북에서의 실험 흐름

┌──────────────────────────────────────────────────────┐
│  Cell 6: StandardScaler만 적용 → Accuracy 0.81 (좋음)        │
│  Cell 10: StandardScaler + 구간화 → Accuracy 0.65 (나쁨)     │
│  Cell 12: Cell 10의 구간화 모델을 joblib으로 저장  ← 실수     │
└──────────────────────────────────────────────────────┘

                          ↓
                  저장된 joblib 파일
            (구간화 데이터 기대, 성능 0.65)
                          ↓
```

```
                FastAPI 서버에서 로드
                        ↓
원본 수치를 전처리 없이 직접 투입  ← 추가 실수
                        ↓
              예측 결과 완전히 엉망
```

실수 1: 성능이 낮은 구간화 모델을 저장
실수 2: 저장된 모델의 전처리(구간화)조차 API에 구현하지 않음
→ 두 실수가 겹쳐서 예측이 전혀 작동하지 않았다.

---

4. 해결 방법

4-1. 모델 재학습 (성능 좋은 방식으로)

Cell 6의 결과를 기반으로, **구간화 없이 StandardScaler 표준화만 적용**한 데이터로 모델을 재학습했다.

```
# 데이터 전처리 (구간화 없음)
scaler = StandardScaler()
imputer = KNNImputer(n_neighbors=5)
X_all = imputer.fit_transform(scaler.fit_transform(X))

# 시나리오 A: 혈당 포함 → AdaBoost (Cell 6에서 최고 성능)
model_sugar = AdaBoostClassifier(n_estimators=100, learning_rate=1.0,
random_state=42)
model_sugar.fit(X_all[features_a], y)

# 시나리오 B: 혈당 미포함 → RandomForest (Cell 6에서 최고 성능)
model_no_sugar = RandomForestClassifier(n_estimators=100, max_depth=5,
random_state=42)
model_no_sugar.fit(X_all[features_b], y)

joblib.dump(model_sugar, 'model_sugar.joblib')
joblib.dump(model_no_sugar, 'model_no_sugar.joblib')
```

4-2. API에 StandardScaler 전처리 추가

`model_loader.py`에 학습 시 사용된 StandardScaler의 통계값(mean, scale)을 하드코딩하고, 표준화 함수를 추가했다.

```
# 당뇨.csv 전체 데이터에 IQR Clipping 후 fit한 통계값
SCALER_STATS = {
    "pregnancies": (3.837240, 3.341979),   # (mean, std)
    "glucose":     (121.686763, 30.515624),
    "bmi":         (32.394716, 6.711356),
    "age":         (33.199870, 11.620831),
}

def standardize(feature: str, raw_value: float) -> float:
```

```
    mean, scale = SCALER_STATS[feature]
    return (raw_value - mean) / scale
```

`predictor.py`에서 모델 입력 전에 표준화를 적용하도록 수정했다.

```
# 수정 전 (원본 수치 직접 입력)
x_values = [user_provided.get(f, 0.0) for f in feature_names]

# 수정 후 (표준화 후 입력)
x_values = [standardize(f, user_provided.get(f, 0.0)) for f in
feature_names]
```

---

5. 수정된 파일 목록

파일	변경 내용
`fastapi/app/model_sugar.joblib`	AdaBoost로 재학습 (기존: 구간화 RF)
`fastapi/app/model_no_sugar.joblib`	RF로 재학습 (기존: 구간화 RF)
`fastapi/app/model_loader.py`	StandardScaler 통계값 및 `standardize()` 함수 추가, 구간화 로직 제거
`fastapi/app/predictor.py`	예측 전 `standardize()` 호출 추가, 모델명 "AdaBoost" 반영

Flutter 측 코드는 변경 불필요 (원본 수치를 서버에 전송하고, 전처리는 서버 담당).

---

6. 해결 후 테스트 결과

API 테스트

입력값	결과	당뇨 확률	판정
혈당 148, BMI 33.6, 나이 50, 임신 6	당뇨 위험	**54.6%**	합리적
혈당 85, BMI 22, 나이 25, 임신 0	정상 범위	**23.6%**	합리적
혈당 185, BMI 40, 나이 55, 임신 8	당뇨 위험	**62.4%**	합리적
BMI 35, 나이 55, 임신 5 (혈당 없음)	당뇨 위험	**55.6%**	합리적
BMI 22, 나이 25, 임신 0 (혈당 없음)	정상 범위	**7.6%**	합리적

모델 성능 비교 (수정 전 vs 후)

항목	수정 전	수정 후
혈당 포함 모델	RF (구간화, Acc 0.65)	**AdaBoost (표준화, Acc 0.81)**

항목	수정 전	수정 후
혈당 미포함 모델	RF (구간화, Acc < 0.65)	**RF (표준화, Acc 0.71)**
전처리	없음 (원본 수치 직접 투입)	**StandardScaler 표준화 적용**
당뇨 고위험 판정	거의 불가	정상 작동

---

7. 교훈 및 주의사항

1. **모델 저장 시 어떤 전처리를 거친 데이터로 학습했는지 반드시 기록**해야 한다.
   joblib 파일만으로는 학습 시 전처리 파이프라인을 알 수 없다.

2. **scaler, imputer 등 전처리 객체도 함께 저장**하는 것이 베스트 프랙티스이다.
   (이번에는 통계값을 하드코딩했지만, `scaler.joblib`로 함께 저장해두는 것이 이상적)

3. **노트북의 여러 실험 셀 중 최종 저장 셀이 최적의 모델을 사용하는지 반드시 확인**해야 한다.
   이번 케이스에서는 성능이 좋은 Cell 6의 결과가 아니라, 성능이 나쁜 Cell 10의 결과를 저장했다.

4. **API에서 전처리를 빠뜨리면 모델이 무의미**해진다.
   학습 시와 추론 시 동일한 전처리 파이프라인을 반드시 적용해야 한다.

---

8. 현재 데이터 흐름 (수정 후)

```
Flutter 앱
    │
    │  POST /predict  { "혈당": 148, "BMI": 33.6, "나이": 50, "임신횟수": 6 }
    │
    ▼
FastAPI 서버 (predictor.py)
    │
    ├─ 1. 입력값 수집 및 범위 검증
    │
    ├─ 2. 혈당 유무에 따라 모델 분기
    │      혈당 있음 → AdaBoost (model_sugar.joblib)
    │      혈당 없음 → RandomForest (model_no_sugar.joblib)
    │
    ├─ 3. StandardScaler 표준화 (model_loader.standardize)
    │      혈당 148 → (148 - 121.69) / 30.52 = +0.862
    │      BMI 33.6 → (33.6 - 32.39) / 6.71 = +0.180
    │      나이  50 → (50 - 33.20) / 11.62  = +1.446
    │      임신   6 → (6 - 3.84) / 3.34     = +0.647
    │
    ├─ 4. 모델 예측 (predict_proba)
    │      입력: [+0.862, +0.180, +1.446, +0.647]
    │      출력: 당뇨 확률 54.6%
    │
    ├─ 5. 차트 이미지 생성 (matplotlib → Base64)
    │
    └─ 6. 응답 반환
```

```
            { "prediction": 1, "probability": 0.546, "label": "당뇨 위험", ...
}
```