# Linear Regression as a Classifier

20213088 임세윤

1. 선형 회귀를 분류기로 써 봅시다.
- y는 0과 1이므로 (사망/생존) 우리는 y값을 예측하는 "선형 회귀" 식을 만들 수 있습니다.
- 데이터의 전처리는 강의시간에 나온 방식을 참조하세요
- 먼저 train set으로 선형회귀를 학습합시다. 그럼 임의의 입력에 대해서 0과 1 사이의 값을 내 주는 회귀 모델이 만들어 집니다
- 만든 모델에서 어떤 y값을 분류의 cutoff로 삼으면 가장 좋은 성능의 모델이 되나요, Accuracy, Precision, Recall, F1 score 등을 종합적으로 판단해서 최적 모델을 찾아보세요.

```python
import pandas as pd
import numpy as np

# 데이터 전처리 및 모델링 도구 import
from sklearn.model_selection import train_test_split
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, classification_report

# ----------------------------------------------------------------------
# 데이터 로드 및 분리
# ----------------------------------------------------------------------
try:
    # CSV 파일 로드
    df = pd.read_csv('train.csv')

    # PassengerId, Name, Ticket, Cabin은 분석에서 제외
    df = df.drop(['PassengerId', 'Name', 'Ticket', 'Cabin'], axis=1)

    # X (특성)과 y (타겟) 분리
    X = df.drop('Survived', axis=1)
    y = df['Survived']

    # 훈련/검증 데이터 분리
    X_train, X_val, y_train, y_val = train_test_split(X, y, test_size=0.2, random_state=42)

    print(f"훈련 데이터: {X_train.shape[0]}건, 검증 데이터: {X_val.shape[0]}건\n")
```

```python
# ------------------------------------------------------------------------
# 전처리 파이프라인 설정
# ------------------------------------------------------------------------

# 숫자형 특성: Age, SibSp, Parch, Fare
# 결측치(Age, Fare)는 중앙값(median)으로 채우고, 스케일링(StandardScaler) 적용
numeric_features = ['Age', 'Fare', 'SibSp', 'Parch']
numeric_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# 범주형 특성: Pclass, Sex, Embarked
# 결측치(Embarked)는 최빈값(most_frequent)으로 채우고, 원-핫 인코딩(OneHotEncoder) 적용
categorical_features = ['Pclass', 'Sex', 'Embarked']
categorical_transformer = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])

# ColumnTransformer로 두 파이프라인 통합
preprocessor = ColumnTransformer(
    transformers=[
        ('num', numeric_transformer, numeric_features),
        ('cat', categorical_transformer, categorical_features)
    ])
```

```python
# ------------------------------------------------------------------
# 선형 회귀 (Linear Regression)
# ------------------------------------------------------------------
print("--- 1. 선형 회귀(Linear Regression) 모델 학습 ---")

# 선형 회귀 파이프라인 생성
lin_reg_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                                   ('regressor', LinearRegression())])

# 모델 학습
lin_reg_pipeline.fit(X_train, y_train)

# 검증 데이터로 예측 (결과가 -0.1, 0.8, 1.1 등 연속적인 값으로 나옴)
y_pred_linear_raw = lin_reg_pipeline.predict(X_val)

# 최적의 Cutoff 찾기 (F1 Score 기준)
best_f1 = -1
best_cutoff = 0

# 0.01부터 0.99까지 탐색
thresholds = np.arange(0.01, 1.0, 0.01)

for cutoff in thresholds:
    # Cutoff를 기준으로 0 또는 1로 분류
    y_pred_classified = (y_pred_linear_raw > cutoff).astype(int)

    # F1 Score 계산
    current_f1 = f1_score(y_val, y_pred_classified)

    if current_f1 > best_f1:
        best_f1 = current_f1
        best_cutoff = cutoff

print(f"\n[선형 회귀 최적 Cutoff 탐색 (F1 Score 기준)]")
print(f"최적 Cutoff: {best_cutoff:.2f}")
print(f"최대 F1 Score: {best_f1:.4f}")

# 최적 Cutoff로 최종 성능 평가
y_pred_linear_best = (y_pred_linear_raw > best_cutoff).astype(int)

print("\n[선형 회귀 최종 성능 (Cutoff = {best_cutoff:.2f})]".format(best_cutoff=best_cutoff))
print(classification_report(y_val, y_pred_linear_best))
print(f"Accuracy: {accuracy_score(y_val, y_pred_linear_best):.4f}")
```

```
    # ------------------------------------------------------------------
    # 로지스틱 회귀 (Logistic Regression)
    # ------------------------------------------------------------------
    print("\n--- 2. 로지스틱 회귀(Logistic Regression) 모델 학습 ---")

    # 로지스틱 회귀 파이프라인 생성
    log_reg_pipeline = Pipeline(steps=[('preprocessor', preprocessor),
                                       ('classifier', LogisticRegression(random_state=42))])

    # 모델 학습
    log_reg_pipeline.fit(X_train, y_train)

    # 검증 데이터로 예측 (결과는 0 또는 1)
    y_pred_logistic = log_reg_pipeline.predict(X_val)

    print("\n[로지스틱 회귀 최종 성능 (기본 Cutoff = 0.5)]")
    print(classification_report(y_val, y_pred_logistic))
    print(f"Accuracy: {accuracy_score(y_val, y_pred_logistic):.4f}")

except FileNotFoundError:
    print("오류: 'train.csv' 파일을 찾을 수 없습니다. 파일을 현재 디렉토리에 업로드했는지 확인해주세요.")
except Exception as e:
    print(f"코드 실행 중 오류가 발생했습니다: {e}")
```

## 2. 비교를 위해 Logistic Regression으로 동일한 데이터를 분류해 봅시다

- 두 모델을 비교해 봅시다. 어느 모델이 성능이 나은가요?

```
--- 데이터 로드 및 분리 완료 ---
훈련 데이터: 712건, 검증 데이터: 179건

--- 1. 선형 회귀(Linear Regression) 모델 학습 ---

[선형 회귀 최적 Cutoff 탐색 (F1 Score 기준)]
최적 Cutoff: 0.35
최대 F1 Score: 0.7950

[선형 회귀 최종 성능 (Cutoff = 0.35)]
              precision    recall  f1-score   support

           0       0.89      0.78      0.83       105
           1       0.74      0.86      0.80        74

    accuracy                           0.82       179
   macro avg       0.81      0.82      0.81       179
weighted avg       0.83      0.82      0.82       179

Accuracy: 0.8156

--- 2. 로지스틱 회귀(Logistic Regression) 모델 학습 ---

[로지스틱 회귀 최종 성능 (기본 Cutoff = 0.5)]
              precision    recall  f1-score   support

           0       0.81      0.86      0.83       105
           1       0.78      0.72      0.75        74

    accuracy                           0.80       179
   macro avg       0.80      0.79      0.79       179
weighted avg       0.80      0.80      0.80       179

Accuracy: 0.7989
```

로지스틱 회귀(Logistic Regression) 모델이 Accuracy, Precision, Recall, F1-score 등 모든 분류 지표에서 선형 회귀 모델보다 더 나은 성능을 보인다.

- 왜 성능 차이가 날까요?

모델의 목적이 다르다. 선형 회귀는 데이터의 선형적 관계(직선)를 찾습니다. y = Wx + b 처럼 특성(X)이 변할 때 타겟(y)이 얼마나 변하는지 예측한다.
반면, 로지스틱 회귀는 데이터가 특정 클래스에 속할 확률(Probability)을 찾습니다. 이를 위해 '시그모이드(Sigmoid)' 함수를 사용하여 결과를 항상 0과 1 사이의 S-커브 형태(비선형)로 변환한다.
따라서 '사망(0)/생존(1)'과 같은 이진 분류 문제는 직선 관계가 아닌, 확률적(S-커브) 관계를 가집니다. 로지스틱 회귀는 이 S-커브 관계를 수학적으로 잘 모델링하도록 설계되었기 때문에, 억지로 직선을 맞추려는 선형 회귀보다 성능이 좋다.

- 왜 선형회귀를 분류기로 쓰면 안 될까요?

첫번째로, 출력값이 0과 1을 벗어난다. 선형 회귀는 예측값의 범위에 제한이 없다. 학습 데이터에 따라 '생존 확률'이 1(100%)를 초과하거나 0 (0%) 미만의 값처럼 해석이 불가능한 값을 출력할 수 있다. 이에 반해, 로지스틱 회귀는 항상 0~1 사이의 확률값만 반환한다.

둘째로, 출력값이 '확률'이 아니다. 선형 회귀의 예측값 0.8은 "생존 확률 80%"를 의미하지 않는다.

마지막으로 데이터의 특성에 민감하다. 만약 요금(Fare)을 엄청나게 많이 낸 생존자(극단치, Outlier)가 데이터에 추가되면, 선형 회귀는 그 점까지 포함하는 '최적의 직선'을 찾기 위해 분류 경계선이 크게 변동된다. 로지스틱 회귀는 S-커브의 양쪽 끝으로 데이터를 밀어내므로 극단치에 훨씬 덜 민감하다.