AICE\DNN_2025_10_25수정.py

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import random
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import RobustScaler, LabelEncoder
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score,
classification_report
from sklearn.metrics import roc_curve, roc_auc_score
import tensorflow as tf
import matplotlib as mpl
mpl.rcParams['font.family'] = 'Malgun Gothic'
mpl.rcParams['axes.unicode_minus'] = False
import warnings
warnings.filterwarnings('ignore')

random.seed(42)
np.random.seed(42)
tf.random.set_seed(42)

# load data, 컬럼명 소문자로 일괄 변경
df = pd.read_csv("./titanic.csv")
df.columns = df.columns.str.lower()

# 불필요한 컬럼 제거
df.drop(columns=["class", "who"], inplace=True)
'''
# 특정 컬럼 제거(.drop)
df.drop(columns=['col1', 'col2'], inplace=True)
df = df.drop(['col1', 'col2'], axis=1)
'''

# 결측치 처리(제거, 대치)
df.dropna(ignore_index=True, inplace=True)
'''
# axis = 0 or 1 (행기준, 열기준)
결측치 있는 모든 행 제거
df.dropna(axis=0, inplace=True)  ; how='any'(default) or 'all' 지정가능
# 특정 컬럼들만 선택해서 결측치 제거
df.dropna(subset=['col1', 'col2'], inplace=True)
# 결측치 대치
df.fillna({'col_name':df['col_name'].median()}, inplace=True)   # 결측치를 중앙값으로 대치.
또는
df['col_name'].fillna(df['col_name'].mean(), inplace=True)
'''

# 이상치 제거 removing outliers
num_cols = df.select_dtypes(include=[np.number]).columns.tolist()
cat_cols = [c for c in df.columns if c not in num_cols and c != "survived"]

for col in num_cols:
```

```
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1
    lower_bound = Q1 - 1.5 * IQR
    upper_bound = Q3 + 1.5 * IQR
    df = df[ (df[col]>=lower_bound) & (df[col]<=upper_bound) ]
'''
이상치 제거하는 다른 방법:
# index 사용해서 drop
df = df.drop(df[df['col_name'] >= 10].index, axis=0)
# .clip 사용해서 (lower, upper) 제한(행을 삭제하지 않고 대치)
df = df.clip(-10, 200) 또는
df['col'] = df['col'].clip(-10, 200) ; or .clip(lower=, upper=)
'''

# feature engineering
df['family_size'] = df['sibsp'] + df['parch'] + 1

# 인코딩 : 레이블 값 대치
df['adult_male'] = df['adult_male'].map({True:1, False:0}).astype(int)
df['alone'] = df['alone'].map({True:1, False:0}).astype(int)
df['sex'] = df['sex'].map({'male':1, 'female':0}).astype(int)
'''
.map() 과 .replace() 는 다르다. map은 key를 찾지 못하면 NaN 을 반환한다.
unit 컬럼에 metres, feet 두 종류의 값이 있다면,
df['unit'] = df['unit'].replace({'feet':'metres'})  # 기존 metres 는 유지함.
df['unit'] = df['unit'].map({'feet':'metres'})      # 기존 metres 는 NaN 으로 변경됨.

(연습)
# 'shift' 컬럼 직접 인코딩
shift_map = {'주간': 0, '야간': 1}
defect_pre['shift'] = defect_pre['shift'].map(shift_map)
'''

# 상관관계 출력
plt.figure(figsize=(5,5))
corr_mat = df[num_cols].corr()
sns.heatmap(corr_mat, annot=True)
plt.show()
'''
# heatmap 삼각형으로 출력
mask = np.zeros_like(corr_mat, dtype=np.bool)
mask[np.triu_indices_from(mask)] = True
sns.heatmap(corr_mat, mask=mask, annot=True)

# 혼동행렬을 히트맵으로 시각화(25년 10월 기출)
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_valid, y_pred)
sns.heatmap(cm, annot=True, fmt='d')

# lmplot(25년 10월 기출)
산점도+회귀직선

# 클러스터맵
```

```
sns.clustermap(corr_matrix, annot=True)

# boxplot : Show distributions with respect to categories.
(25년 8월 기출)
sns.boxplot(data=defect_df, x='defect_status', y='temperature')

# barplot : Show point estimates and errors as rectangular bars.
# 범주형 데이터별로 집계(statistical aggregate, default: mean) 값을 막대(bar)로 표현
# defaults: estimator='mean', errorbar=(ci, 95)
sns.barplot(data=df, x='class', y='fare', hue='sex', errorbar=None, estimator='median')
'''

# 범주형 변수는 자동으로 1/0으로 인코딩된다.
df = pd.get_dummies(data=df, drop_first=True).astype(int)

'''
(연습)
# 교차표 생성 (비율 기준)
cross_table_ratio = pd.crosstab(defect_df['production_line'], defect_df['defect_status'],
normalize='index')

# 'shift'로 그룹화하여 주요 측정값들의 평균을 계산합니다.
shift_avg = defect_df.groupby('shift')[['measurement_A', 'measurement_B', 'measurement_C']].mean()
'''



# 종속변수(범주형) 수치형으로 인코딩
le = LabelEncoder()
df['survived'] = le.fit_transform(df['survived'])
'''
# .map({ key:value,... }) 로 값 대치
df['col_name'] = df['col_name'].map({"Y":1, "N":0})

# to_categorical() 함수는 정수형(integer) 클래스 레이블(label)을 원-핫 인코딩(one-hot encoding) 벡터
로 변환
from keras.utils import to_categorical
df[col] = to_categorical(df[col], num_classes=2) 또는 y = to_categorical(y, num_classes=2)
로 이진 분류도 출력층을 2개로 만들 수 있다.
--> 이 경우 model.add(Dense(2, activation='softmax')) 로 해야한다.

ex)
    labels = [0, 1, 2, 1, 0]
    one_hot_labels = to_categorical(labels)
    print(one_hot_labels)

    Out[]:
    [[1. 0. 0.]
    [0. 1. 0.]
    [0. 0. 1.]
    [0. 1. 0.]
    [1. 0. 0.]]
'''

y = df.pop('survived')
```

```
X = df

# split dataset
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# 데이터셋 분리 후 스케일링
scaler = RobustScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)

'''
다중클래스 분류 문제라면 출력층을 softmax 함수로 변경하고,
y_train, y_test 를 to_categorical() 로 인코딩 해야한다.
'''

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Dropout, Input
from keras.utils import to_categorical

model = Sequential()
model.add(Input((X_train.shape[1],)))   # keras recommend this
# model.add(Dense(32, activation='relu', input_shape=(X_train.shape[1],))), # input_dim = X_train.shape[1]
model.add(Dense(32, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dropout(rate=0.2))
model.add(Dense(8, activation='relu'))
model.add(Dropout(rate=0.2))
model.add(Dense(4, activation='relu'))
model.add(Dropout(rate=0.2))
# 출력층 시그모이드 함수 사용(이진분류문제)
model.add(Dense(1, activation='sigmoid'))
# 출력층이 2개이상인 경우 model.add(Dense(n, activation='softmax'))
'''
또는
model = Sequential([
    Dense(32, activation='relu', input_shape=(X_train.shape[1],)),
    Dense(16, activation='relu'),
    Dropout(rate=0.2),
    ...
])
로도 가능

keras에서 activation 의 default 값은 없다.  activation 함수의 종류:
    linear :
        입력과 동일한 값을 출력합니다. 결과적으로 입력에 가중치를 곱하여 모두 더한 값을 그대로 출력합니다.
    sigmoid :
        시그모이드 함수를 사용하여 출력값을 0과 1 사이의 값으로 변환합니다. 이진 분류 모델 출력층에 많이 사용됩니다.
    tanh :
        하이퍼볼릭 탄젠트 함수를 사용하여 출력값을 -1과 1 사이의 값으로 변환합니다. 은닉층에 많이 사용됩니다.
    relu :
```

```
        Rectified Linear Unit 함수를 사용하여 출력값을 0 이상의 값으로 변환합니다. 은닉층에 많이 사용됩니다.
    softmax :
        softmax 함수를 사용하여 출력값을 다중 클래스 분류에 적합한 확률 값으로 변환합니다. 다중 클래스 분류 모델 출력층에 많이 사용됩니다.
    selu :
        Scaled Exponential Linear Unit 함수를 사용하여 출력값을 0과 1 사이로 스케일링합니다. 자기 정규화(Self-Normalizing) 효과로 인해 딥러닝 모델의 성능을 향상시킬 수 있습니다.
    leaky_relu :
        ReLU 함수의 단점을 보완하기 위해 제안된 활성화 함수입니다. ReLU 함수는 음수 입력에 대해 항상 0을 출력하는데, 이는 뉴런이 죽는 문제(dying ReLU)를 야기할 수 있습니다. Leaky ReLU는 이러한 문제를 해결하기 위해 음수 입력에 대해 아주 작은 기울기 (보통 0.01)를 부여합니다.
'''



# 모델 요약
model.summary()

# 모델 컴파일 (클래스가 2개인 2진 분류: 0 or 1)
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
'''
# loss = 'name' 으로 지정하는 기본값
# 회귀문제 : 'mean_squared_error'
# 2진 분류 : 'binary_crossentropy'
# 다중 클래스 분류
    - 'categorical_crossentropy' : 원핫인코딩 했을 경우 (pd.get_dummies)
    - 'sparse_categorical_crossentropy' : 원핫인코딩 안했을 경우 (LabelEncoder 정수 인코딩)

# metrics 기본 옵션 : 'accuracy', 'binary_accuracy', 'mse'
    다중 레이블 분류에서는 일반적인 accuracy가 성능을 과소평가 할 수 있음
        - binary_accuracy: 레이블별 정확도를 평가
        - precision, recall, f1_score: 클래스 불균형이나 특정 클래스(양성)에 초점을 맞출 때 유용
        - auc: 클래스 분리 능력을 평가
'''

# 모델 학습
# model.fit(X_train, y_train, epochs=30, batch_size=16, verbose=1)

# 모델 학습을 history 로 저장하려면,
from tensorflow.keras.callbacks import EarlyStopping, ModelCheckpoint    # point 는 소문자임에 주의

es = EarlyStopping(monitor='val_loss', patience=9, verbose=1)
mc = ModelCheckpoint('best_model.keras', monitor='val_loss', save_best_only=True, verbose=1)

history = model.fit(
    X_train,
    y_train,
    validation_split=0.2,
    # validation_data=(X_test, y_test),
    epochs=30,
    batch_size=16,
    callbacks=[es, mc]
    )

```

```
y_pred = (model.predict(X_test) > 0.5).astype(int)
'''
시그모이드 출력층과 binary_crossentropy 손실 함수를 사용하는 이진 분류 DNN 모델이다.
model.predict(X_test) > 0.5는 각 샘플의 예측 확률이 0.5보다 크면 True
.astype(int)는 boolean 값을 0 or 1 로 변환
예를 들어 predict 값이 [0.85, 0.11] 이면 [0] 이 반환됨.  즉, Negative 라는 의미로 해석

softmax 함수로 2개 output 출력되었을 경우,
y_pred = np.argmax(model.predict(X_test), axis=1) 로 큰 값을 저장한다.  axis=1 은 컬럼방향 최대값 선택의 의미
np.argmax() : 행(axis=0) 또는 열(axis=1)을 따라 가장 큰 값(높은 확률)의 index 반환
예를 들어 [[0.15, 0.85], [0.89, 0.11] ] 이면 [1, 0] 이 반환됨.  즉, [Positive, Negative] 라는 의미로 해석

다중 클래스 분류(하나의 클래스 선택)와 달리, 다중 레이블 분류(여러 클래스 동시 선택 가능)에서는
softmax 대신 클래스별로 독립적인 sigmoid를 사용하고, binary_crossentropy를 적용
'''

# 점수 출력

print("Accuracy:", accuracy_score(y_test, y_pred))
print("Precision:", precision_score(y_test, y_pred))
print("Recall:", recall_score(y_test, y_pred))
print("F1 Score:", f1_score(y_test, y_pred))
print("Classification Report", classification_report(y_test, y_pred)) # 모아서 출력

# 모델 성능 시각화
def plot_training_history(history):
    plt.figure(figsize=(10, 4))

    # 손실 값 (Loss)
    plt.subplot(1, 2, 1)
    plt.plot(history.history['loss'], label='Training Loss', marker='o')
    plt.plot(history.history['val_loss'], label='Validation Loss', marker='o')
    plt.title('Loss Over Epochs')
    plt.xlabel('Epochs')
    plt.ylabel('Loss')
    plt.legend()
    plt.grid(True)
    # 정확도 (Accuracy)
    plt.subplot(1, 2, 2)
    plt.plot(history.history['accuracy'], label='Training Accuracy', marker='o')
    plt.plot(history.history['val_accuracy'], label='Validation Accuracy', marker='o')
    plt.title('Accuracy Over Epochs')
    plt.xlabel('Epochs')
    plt.ylabel('Accuracy')
    plt.legend()
    plt.grid(True)

    plt.tight_layout()
    plt.show()

def plot_roc_curve(y_test, y_pred_proba_pos):
   fpr, tpr, _ = roc_curve(y_test, y_pred_proba_pos)
```

```
    auc_score = roc_auc_score(y_test, y_pred_proba_pos)  # ← 추가
    plt.plot(fpr, tpr, label=f'ROC curve (AUC = {auc_score:.2f})')
    plt.plot([0, 1], [0, 1], 'k--', label='Classifier')
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver Operating Characteristic (ROC) Curve')
    plt.legend(loc='lower right')
    plt.show()


# 모델 학습 결과 시각화
plot_training_history(history)

# plot ROC curve with auc score
y_prob = model.predict(X_test).ravel()
plot_roc_curve(y_test, y_prob)
"""
sklearn의 분류기(LogisticRegression, RandomForestClassifier 등)에는 .predict_proba()가 있지만,
Keras 모델에는 .predict_proba() 메서드는 존재하지 않음.  tf.keras 모델은 model.predict가
“마지막 층의 출력값”을 그대로 반환한다.
    y_prob = model.predict(X_test)           # 각 클래스별 확률 (softmax 출력)
    y_pred = np.argmax(y_prob, axis=1)       # 확률이 가장 큰 클래스 선택

마지막 층이 sigmoid(이진 분류)면 predict 결과가 곧 양성 확률.
마지막 층이 softmax(다중 분류)면 predict 결과가 각 클래스 확률 분포(합=1.0).

numpy.ravel(a, order='C') : Return a contiguous flattened array.
numpy.array.ravel()은 모양(차원)을 (N,1) → (N,)으로 축소함.
이진 분류에서 predict가 (샘플수, 1) 형태로 나오므로,
y_prob = model.predict(X_test).ravel()로 1차원 벡터로 바꿔
roc_auc_score, roc_curve 같은 sklearn 지표에 맞춥니다.
"""

###### y값 Imbalance Handling with SMOTE ######
# pip install -U imbalanced-learn
from imblearn.over_sampling import SMOTE
smote = SMOTE(random_state=42)
X_train_ovr, y_train_ovr = smote.fit_resample(X_train, y_train)

print("SMOTE 적용 전 train/test 데이터셋",  X_train.shape, y_train.shape)
print("SMOTE 적용 후 train/test 데이터셋",  X_train_ovr.shape, y_train_ovr.shape)

# SMOTE 적용 후 레이블값 분포 확인
pd.Series(y_train_ovr).value_counts()

# 다중클래스로 인코딩(softmax 함수를 출력층에 사용하려고)
y_train_ovr = to_categorical(y_train_ovr)

# build DNN model
model = Sequential()
model.add(Input((X_train_ovr.shape[1],)))   # Keras recommend this
# model.add(Dense(32, activation='relu', input_shape=(X_train.shape[1],)), # input_dim =
X_train.shape[1]
model.add(Dense(32, activation='relu')),
```

```
362 model.add(Dropout(rate=0.2)),
363 model.add(Dense(16, activation='relu')),
364 model.add(Dropout(rate=0.2)),
365 model.add(Dense(8, activation='relu')),
366 model.add(Dropout(rate=0.2)),
367 model.add(Dense(2, activation='softmax'))
368
369 # model compile
370 model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
371 es = EarlyStopping(monitor='val_loss', patience=9, verbose=1)
372 mc = ModelCheckpoint('best_model_smote.keras', monitor='val_loss', save_best_only=True, verbose=1)
373
374 history = model.fit(
375     X_train_ovr,
376     y_train_ovr,
377     validation_split=0.2,
378     # validation_data=(X_test, y_test),
379     epochs=30,
380     batch_size=16,
381     callbacks=[es, mc]
382     )
383
384 # np.argmax() : 행(axis=0) 또는 열(axis=1)을 따라 가장 큰 값(높은 확률)의 index 반환
385 y_pred = np.argmax(model.predict(X_test), axis=1)
386
387 print("Accuracy:", accuracy_score(y_test, y_pred))
388 print("Precision:", precision_score(y_test, y_pred))
389 print("Recall:", recall_score(y_test, y_pred))
390 print("F1 Score:", f1_score(y_test, y_pred))
391 print("Classification Report", classification_report(y_test, y_pred)) # 모아서 출력
392
393 # 모델 학습 결과 시각화
394 plot_training_history(history)
395
396 # plot ROC curve with auc score
397 y_prob_pos = model.predict(X_test)[:,1]
398 plot_roc_curve(y_test, y_prob_pos)
```