1 빅데이터 기반 AI 응용 솔루션 개발자 전문과정

1.1 교과목명 : 모델성능평가

- 평가일 : 22.9.19
- 성명 : 최애림
- 점수 : 70

Q1. iris data를 불러와서 붓꽃의 종류를 분류하는 모델링을 수행한 후 오차행렬과 정확도를 평가하세요.

- test_size = 0.2, 분류기는 DecisionTreeClassifier를 이용

In [6]:

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
import pandas as pd


iris = load_iris()
iris_df = pd.DataFrame(iris.data, columns = iris.feature_names)
iris_df['label'] = iris.target

x = iris_df.drop('label', axis = 1)
y = iris_df['label']


X_train, X_test, y_train, y_test = train_test_split(x, y, test_size= 0.2, random_state = 11

dtree = DecisionTreeClassifier()
dtree.fit(X_train, y_train)
pred = dtree.predict(X_test)

confusion = confusion_matrix(y_test, pred)
accuracy = accuracy_score(y_test , pred)

print(f'오차 행렬:\n {confusion}, \n\n정확도: {accuracy: .3f}')
```

```
오차 행렬:
 [[ 9  0  0]
 [ 0 10  0]
 [ 0  2  9]],

정확도:  0.933
```

Q2. 타이타닉 분석용 데이터세트인 tdf1.pkl를 불러와서 생존자 예측 모델을 만든 후 오차행렬, 정확도, 재현율, f1, AUC를 포함하는 사용자 함수를 활용하여 평가하세요.

- test_size = 0.2, 분류기는 RandomForestClassifier 이용

In [7]:

```
import pandas as pd
tdf = pd.read_pickle('tdf1.pkl')
tdf.head()
```

Out[7]:

	Survived	Sex	Town_0	Town_1	Town_2	Family_Big	Family_Single	Family_Small	Age_10s
0	0	1	0	0	1	0	0	1	0
1	1	0	1	0	0	0	0	1	0
2	1	0	0	0	1	0	1	0	0
3	1	0	0	0	1	0	0	1	0
4	0	1	0	0	1	0	1	0	0

In [13]:

```
from sklearn.metrics import confusion_matrix, accuracy_score,precision_score,recall_score,f

def get_clf_eval(y_test, pred, pred_proba):
    confusion = confusion_matrix(y_test,pred)
    accuracy = accuracy_score(y_test,pred)
    precision=precision_score(y_test,pred)
    recall=recall_score(y_test,pred)
    f1 = f1_score(y_test,pred)
    auc = roc_auc_score(y_test,pred_proba)

    print('오차행렬:\n', confusion)
    print('정확도: {:.4f}, 정밀도: {:.4f}, 재현율: {:.4f}, f1: {:.4f}, AUC: {:.4f}'.format(ac
```

In [15]:

```
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier

X = tdf.drop('Survived',axis=1)
y = tdf['Survived']

X_train, X_test, y_train, y_test = train_test_split(X,y, test_size=0.2, random_state=10)
rf_clf = RandomForestClassifier()
rf_clf.fit(X_train,y_train)
rf_pred = rf_clf.predict(X_test)
rf_pred_proba = rf_clf.predict_proba(X_test)[:,1]

get_clf_eval(y_test,rf_pred,rf_pred_proba)
```

```
오차행렬:
 [[104  13]
 [ 14  48]]
정확도: 0.8492, 정밀도: 0.7869, 재현율: 0.7742, f1: 0.7805, AUC: 0.8785
```

Q3. Q2에서 생성한 모델로 교차검증(cv=5)을 수행하고 평균 정확도를 출력하세요.

In [139]:

```
from sklearn.model_selection import cross_val_score, cross_validate
import numpy as np

accuracy = cross_val_score(rfc, x, y, cv = 5, scoring = 'accuracy')
mean_accuracy = np.mean(accuracy)

print(f'평균 정확도: {mean_accuracy: .4f}')
```

평균 정확도: 0.7969

Q4. Q2에서 생성한 예측모델에 대하여 교차 검증 및 성능 개선을 수행하세요.(GridSearchCV 활용)

In [140]:

```
from sklearn.model_selection import GridSearchCV


# 파라미터를 딕셔너리로 지정
params = {'max_depth': [1, 2, 3], 'min_samples_split': [2, 3]}

rfc_grid = GridSearchCV(rfc, param_grid = params, cv = 5, refit = True, error_score = 'rais


# param_grid의 하이퍼 파라미터를 순차적으로 학습/평가
rfc_grid.fit(X_train, y_train)


# GridSearchCV 결과를 추출해 DataFrame으로 변환
scores_df = pd.DataFrame(rfc_grid.cv_results_)
scores_df[['params', 'mean_test_score', 'rank_test_score',
           'split0_test_score','split1_test_score', 'split2_test_score']]
```

Out[140]:

	params	mean_test_score	rank_test_score	split0_test_score	split1_test_score	sp
0	{'max_depth': 1, 'min_samples_split': 2}	0.695184	6	0.727273	0.692308	
1	{'max_depth': 1, 'min_samples_split': 3}	0.714902	5	0.727273	0.692308	
2	{'max_depth': 2, 'min_samples_split': 2}	0.765439	3	0.790210	0.748252	
3	{'max_depth': 2, 'min_samples_split': 3}	0.751423	4	0.769231	0.720280	
4	{'max_depth': 3, 'min_samples_split': 2}	0.814685	1	0.790210	0.783217	
5	{'max_depth': 3, 'min_samples_split': 3}	0.811849	2	0.783217	0.804196	

In [144]:

```
rfc_grid.best_score_
```

Out[144]:

```
0.8146951639909386
```

Q5 ~ Q7. 'dataset/diabetes.csv'을 불러와서 아래사항을 수행하세요.

- 피마 인디언 당뇨병 예측을 로지스틱 회귀를 이용하여 수행하고 사용자 함수를 작성하여 평가(오차행렬, 정확도, 정밀도, 재현율, F1, ROC_AUC)
- 임곗값을 0.3에서 0.5까지 변화시키면서 정밀도와 재현율이 조정되는 과정을 시각화
- 재현율 기준의 성능을 개선하기 위하여 그 값이 0이 될 수 없는 각 칼럼을 탐색하여 적절한 처리를 한 후 로지스틱 회귀로 예측 및 평가 수행(오차행렬, 정확도, 정밀도, 재현율, F1, ROC_AUC)

In [145]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import roc_auc_score

# 데이터 불러오기
diabetes = pd.read_csv('diabetes.csv')
display(diabetes.head())
```

	Pregnancies	Glucose	BloodPressure	SkinThickness	Insulin	BMI	DiabetesPedigreeFunctio
0	6	148	72	35	0	33.6	0.62
1	1	85	66	29	0	26.6	0.35
2	8	183	64	0	0	23.3	0.67
3	1	89	66	23	94	28.1	0.16
4	0	137	40	35	168	43.1	2.28

In [147]:

```
# Q5_1: 피마 인디언 당뇨병 예측을 로지스틱 회귀를 이용하여 수행하고
# Q5_2: 사용자 함수를 작성하여 평가(오차행렬, 정확도, 정밀도, 재현율, F1, ROC_AUC)

# 사용자함수
def get_clf_eval(y_test , pred):
    confusion = confusion_matrix(y_test, pred)
    accuracy = accuracy_score(y_test , pred)
    precision = precision_score(y_test , pred)
    recall = recall_score(y_test , pred)
    f1 = f1_score(y_test, pred)
    roc_auc = roc_auc_score(y_test, pred)

    print('오차 행렬')
    print(confusion)
    print(f'정확도: {accuracy:.4f}, 정밀도: {precision: .4f}, 재현율: {recall:.4f}, f1 스코어


# 학습 및 예측
x = diabetes.drop('Outcome', axis = 1)
y = diabetes['Outcome']

lr_clf = LogisticRegression(solver = 'liblinear')
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 1


lr_clf.fit(X_train, y_train)
pred = lr_clf.predict(X_test)


get_clf_eval(y_test, pred)
```

```
오차 행렬
[[91  9]
 [20 34]]
정확도: 0.8117, 정밀도:  0.7907, 재현율: 0.6296, f1 스코어: 0.7010, ROC_AUC 스
코어: 0.7698
```

In [135]:

```
# Q6: 임곗값을 0.3에서 0.5까지 변화시키면서 정밀도와 재현율이 조정되는 과정을 시각화

from sklearn.preprocessing import Binarizer
import matplotlib.pyplot as plt

pp1 = lr_clf.predict_proba(X_test)[:, 1].reshape(-1,1)
thresholds = np.linspace(0.3, 0.5, 10)

precision = []
recall = []
for threshold in thresholds:
    binarizer = Binarizer(threshold = threshold).fit(pp1)
    pred = binarizer.transform(pp1)

    precision.append(precision_score(y_test, pred))
    recall.append(recall_score(y_test, pred))

    print(f'\n임곗값: {threshold}')
    get_clf_eval(y_test, pred)

fig = plt.figure(figsize = (10,6))
fig.add_subplot(111)

plt.plot(thresholds, precision, color = 'black')
plt.plot(thresholds, recall, color = 'gray')
```

```
임곗값: 0.3
오차 행렬
[[71 29]
 [ 9 45]]
정확도: 0.7532, 정밀도:  0.6081, 재현율: 0.8333, f1 스코어: 0.7031, ROC_AUC 스
코어: 0.7717

임곗값: 0.3222222222222222
오차 행렬
[[75 25]
 [ 9 45]]
정확도: 0.7792, 정밀도:  0.6429, 재현율: 0.8333, f1 스코어: 0.7258, ROC_AUC 스
코어: 0.7917

임곗값: 0.34444444444444444
오차 행렬
[[82 18]
 [ 9 45]]
정확도: 0.8247, 정밀도:  0.7143, 재현율: 0.8333, f1 스코어: 0.7692, ROC_AUC 스
코어: 0.8267

임곗값: 0.36666666666666664
오차 행렬
[[85 15]
 [11 43]]
정확도: 0.8312, 정밀도:  0.7414, 재현율: 0.7963, f1 스코어: 0.7679, ROC_AUC 스
코어: 0.8231

임곗값: 0.3888888888888889
오차 행렬
[[89 11]
```

```
 [12 42]]
정확도: 0.8506, 정밀도:  0.7925, 재현율: 0.7778, f1 스코어: 0.7850, ROC_AUC 스
코어: 0.8339

임곗값: 0.4111111111111111
오차 행렬
[[89 11]
 [16 38]]
정확도: 0.8247, 정밀도:  0.7755, 재현율: 0.7037, f1 스코어: 0.7379, ROC_AUC 스
코어: 0.7969

임곗값: 0.43333333333333335
오차 행렬
[[90 10]
 [17 37]]
정확도: 0.8247, 정밀도:  0.7872, 재현율: 0.6852, f1 스코어: 0.7327, ROC_AUC 스
코어: 0.7926

임곗값: 0.45555555555555555
오차 행렬
[[91  9]
 [18 36]]
정확도: 0.8247, 정밀도:  0.8000, 재현율: 0.6667, f1 스코어: 0.7273, ROC_AUC 스
코어: 0.7883

임곗값: 0.47777777777777775
오차 행렬
[[91  9]
 [19 35]]
정확도: 0.8182, 정밀도:  0.7955, 재현율: 0.6481, f1 스코어: 0.7143, ROC_AUC 스
코어: 0.7791

임곗값: 0.5
오차 행렬
[[91  9]
 [20 34]]
정확도: 0.8117, 정밀도:  0.7907, 재현율: 0.6296, f1 스코어: 0.7010, ROC_AUC 스
코어: 0.7698
```

Out[135]:

```
[<matplotlib.lines.Line2D at 0x1d75f31a5e0>]
```

임곗값은 0.38 정도가 적당해보임

In [ ]:

```
# Q7_1: 재현율 기준의 성능을 개선하기 위하여 그 값이 0이 될 수 없는 각 칼럼을 탐색하여 적절한 처리
# Q7_2: 로지스틱 회귀로 예측 및 평가 수행(오차행렬, 정확도, 정밀도, 재현율, F1, ROC_AUC)
```

Q8. "dataset/auto-mpg.xlsx"을 불러와서 회귀 모델을 생성하고 MSE, RMSE, R2로 평가를 수행하세요.

In [148]:

```
# 데이터 불러오기
auto_mpg = pd.read_excel('auto-mpg.xlsx')
auto_mpg = auto_mpg[['mpg','cylinders','weight','horsepower']]
auto_mpg.head(5)
```

Out[148]:

	mpg	cylinders	weight	horsepower
0	18.0	8	3504	130
1	15.0	8	3693	165
2	18.0	8	3436	150
3	16.0	8	3433	150
4	17.0	8	3449	140

In [149]:

```
auto_mpg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 398 entries, 0 to 397
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   mpg         398 non-null    float64
 1   cylinders   398 non-null    int64  
 2   weight      398 non-null    int64  
 3   horsepower  398 non-null    object 
dtypes: float64(1), int64(2), object(1)
memory usage: 12.6+ KB
```

In [150]:

```
auto_mpg['horsepower'].unique()
```

Out[150]:

```
array([130, 165, 150, 140, 198, 220, 215, 225, 190, 170, 160, 95, 97, 85,
       88, 46, 87, 90, 113, 200, 210, 193, '?', 100, 105, 175, 153, 180,
       110, 72, 86, 70, 76, 65, 69, 60, 80, 54, 208, 155, 112, 92, 145,
       137, 158, 167, 94, 107, 230, 49, 75, 91, 122, 67, 83, 78, 52, 61,
       93, 148, 129, 96, 71, 98, 115, 53, 81, 79, 120, 152, 102, 108, 68,
       58, 149, 89, 63, 48, 66, 139, 103, 125, 133, 138, 135, 142, 77, 62,
       132, 84, 64, 74, 116, 82], dtype=object)
```

In [151]:

```
auto_mpg['horsepower'] = auto_mpg['horsepower'].replace('?', np.nan)
```

In [152]:

```
auto_mpg.dropna(subset = 'horsepower', axis = 0, inplace = True)
```

In [153]:

```
auto_mpg.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 392 entries, 0 to 397
Data columns (total 4 columns):
 #   Column      Non-Null Count  Dtype  
---  ------      --------------  -----  
 0   mpg         392 non-null    float64
 1   cylinders   392 non-null    int64  
 2   weight      392 non-null    int64  
 3   horsepower  392 non-null    float64
dtypes: float64(2), int64(2)
memory usage: 15.3 KB
```

In [154]:

```
from sklearn.linear_model import LinearRegression

# 데이터 학습 및 예측
x = auto_mpg.drop('mpg', axis = 1)
y = auto_mpg['mpg']

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 5

lr_clf = LinearRegression()
lr_clf.fit(X_train, y_train)
pred = lr_clf.predict(X_test)
```

In [155]:

```
from sklearn.metrics import mean_squared_error, r2_score

# 평가
mse =  mean_squared_error(y_test, pred)
rmse = np.sqrt(mse)
r2 = r2_score(y_test, pred)
r2_2 = lr_clf.score(X_test, y_test)

print(f'MSE: {mse:.3f}, RMSE: {rmse:.3f}, r2_score: {r2_2:.3f}')
```

```
MSE: 25.753, RMSE: 5.075, r2_score: 0.654
```

Q9. 'load_boston' 을 불러와서 cross_val_score를 이용한 cv=5인 교차검증을 수행 후 MSE, RMSE를 출력하세요.(LineaRegression)

In [95]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.datasets import load_boston
import warnings
warnings.filterwarnings('ignore')

house = load_boston()
boston = pd.DataFrame(house.data, columns = house.feature_names)
boston['PRICE'] = house.target

display(boston.head())
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LS
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	

In [111]:

```
x = boston.drop('PRICE', axis = 1)
y = boston['PRICE']

X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2, random_state = 5

lr_clf = LinearRegression()
lr_clf.fit(X_train, y_train)
pred = lr_clf.predict(X_test)


cvs = cross_val_score(lr_clf, x, y, scoring = 'neg_mean_squared_error', cv = 5)
print(f'MSE: {np.round((-1) * cvs, 3)}')

print(f'RMSE: {np.round(np.sqrt((-1) * cvs), 3)}')
```

```
MSE: [12.46  26.049 33.074 80.762 33.314]
RMSE: [3.53  5.104 5.751 8.987 5.772]
```

Q10. 'Q9에 대하여 R2 Score를 구하세요.(k=5)

In [112]:

```
r2_score = lr_clf.score(X_test, y_test)
print(f'R2_score: {r2_score: .3f}')
```

```
R2_score:  0.733
```