데이터사이언스 _ Classification 201911278 정경은

목차

- 모든 목차는 실행 화면 캡처를 포함

1. db_score 엑셀 파일을 읽어, DB 테이블로 구축

조건)

- homework, discussion, midterm, grade 만 포함
- grade: A -> A / B, C -> B / D, F -> C 변환하여 입력

```
import pandas as pd
from db_conn import *

def import_db_score_data():
    conn, cur = open_db()

    # classification table 생성
    sql = """create table classification(
                homework float(4,2),
                discussion int,
                midterm float(4,2),
                grade char(1)
            );"""

    cur.execute(sql)
    conn.commit()
```

사전에 작성한 db_conn.py

3. Classification > db_conn.py > ...

```
import pymysql

from pymysql.constants.CLIENT import MULTI_STATEMENTS

def open_db(dbname='university'):
    conn = pymysql.connect(host='localhost',
                           user='[redacted]',
                           passwd='[redacted]',
                           db=dbname,
                           client_flag=MULTI_STATEMENTS)

    cur = conn.cursor(pymysql.cursors.DictCursor)

    return conn, cur

def close_db(conn, cur):
    cur.close()
    conn.close()
```

university database 에 homework, discussion, midterm, grade 4 가지만 포함하는 classification 테이블을 생성한다.

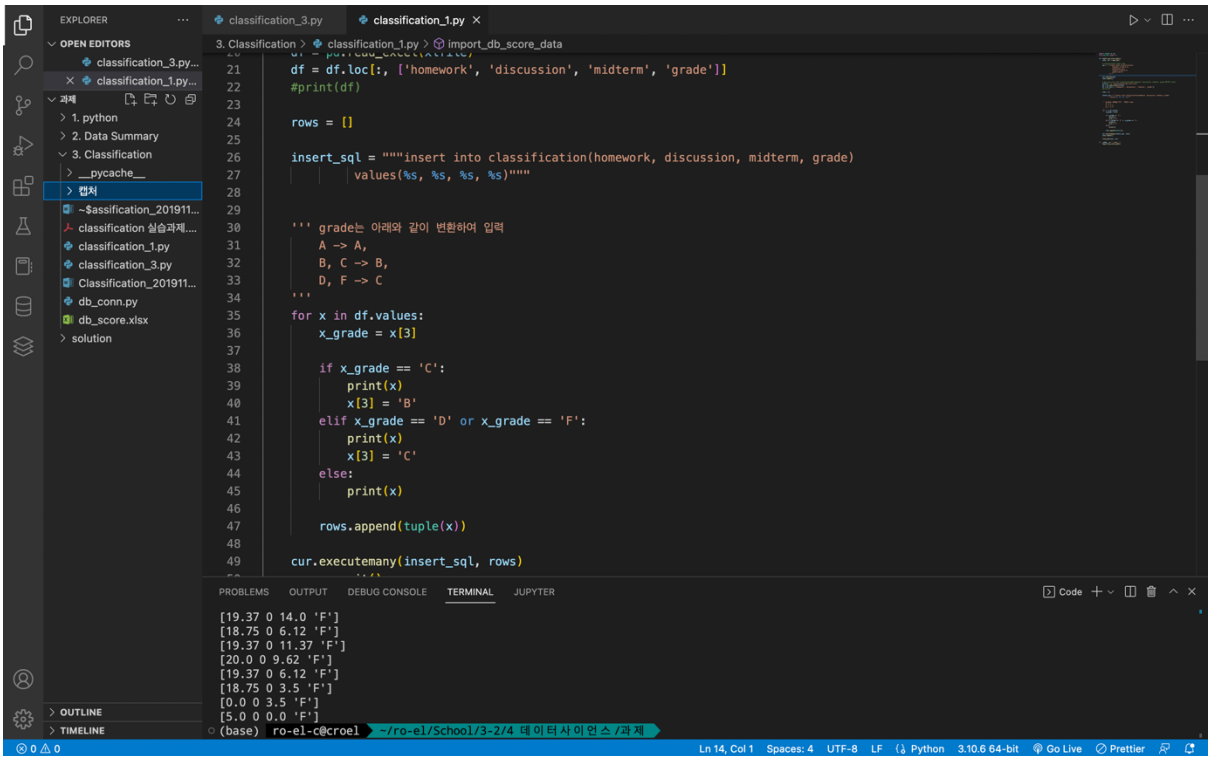
```
    # db_score.xlsx 읽어 classification 에 homework, discussion, midterm, grade 데이터 insert
    xlfile = '3. classification/db_score.xlsx'
    df = pd.read_excel(xlfile)
    df = df.loc[:, ['homework', 'discussion', 'midterm', 'grade']]
```

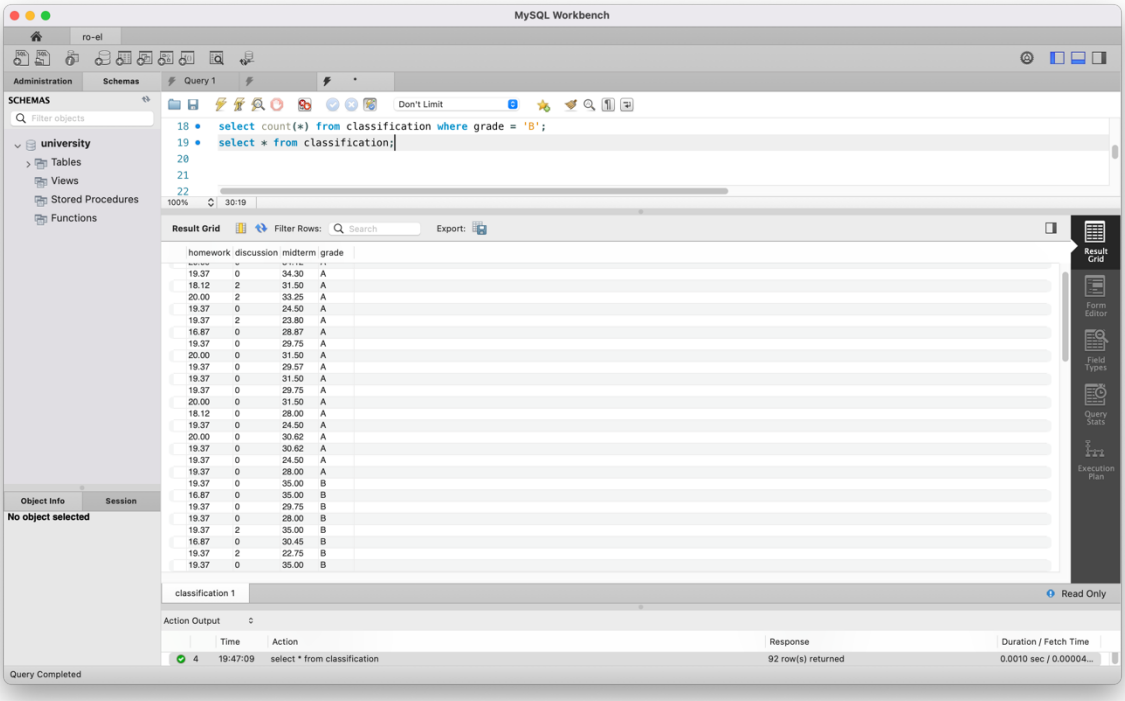
db_score.xlsx 파일을 dataframe 으로 읽어 들여서, 저장할 데이터인 homework, discussion, midterm, grade 만 추출한다.

```
    rows = []

    insert_sql = """insert into classification(homework, discussion,
midterm, grade)
            values(%s, %s, %s, %s);"""


    ''' grade 는 아래와 같이 변환하여 입력
        A -> A,
        B, C -> B,
        D, F -> C
    '''
    for x in df.values:
        x_grade = x[3]

        if x_grade == 'C':
            print(x)
            x[3] = 'B'
        elif x_grade == 'D' or x_grade == 'F':
            print(x)
            x[3] = 'C'
        else:
            print(x)

        rows.append(tuple(x))

    cur.executemany(insert_sql, rows)
    conn.commit()

    close_db(conn, cur)

if __name__ == "__main__":
    import_db_score_data()
```

주어진 조건에 따라, grade 가 A 와 B 인 경우는 그대로, C 는 B 로, D 와 F 는 C 로 변환하여 DB 입력한다.

실행 결과)


EXPLORER
OPEN EDITORS
classification_3.py...
classification_1.py...
과제
1. python
2. Data Summary
3. Classification
__pycache__
캡처
~$assification_201911...
classification 실습과제....
classification_1.py
classification_3.py
Classification_201911...
db_conn.py
db_score.xlsx
solution
OUTLINE
TIMELINE
classification_3.py
classification_1.py
3. Classification > classification_1.py > import_db_score_data
df = df.loc[:, ['homework', 'discussion', 'midterm', 'grade']]
#print(df)
rows = []
insert_sql = """insert into classification(homework, discussion, midterm, grade)
values(%s, %s, %s, %s)"""
''' grade는 아래와 같이 변환하여 입력
A -> A,
B, C -> B,
D, F -> C
'''
for x in df.values:
x_grade = x[3]
if x_grade == 'C':
print(x)
x[3] = 'B'
elif x_grade == 'D' or x_grade == 'F':
print(x)
x[3] = 'C'
else:
print(x)
rows.append(tuple(x))
cur.executemany(insert_sql, rows)
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
JUPYTER
Code
[19.37 0 14.0 'F']
[18.75 0 6.12 'F']
[19.37 0 11.37 'F']
[20.0 0 9.62 'F']
[19.37 0 6.12 'F']
[18.75 0 3.5 'F']
[0.0 0 3.5 'F']
[5.0 0 0.0 'F']
(base) ro-el-c@croel ~/ro-el/School/3-2/4 데이터사이언스 /과제
Ln 14, Col 1
Spaces: 4
UTF-8
LF
Python
3.10.6 64-bit
Go Live
Prettier


MySQL Workbench
ro-el
Administration
Schemas
Query 1
SCHEMAS
university
Tables
Views
Stored Procedures
Functions
Object Info
Session
No object selected
Don't Limit
select count(*) from classification where grade = 'B';
select * from classification;
Result Grid
Filter Rows:
Export:
homework discussion midterm grade
19.37 0 34.30 A
18.12 2 31.50 A
20.00 2 33.25 A
19.37 0 24.50 A
19.37 2 23.80 A
16.87 0 28.87 A
19.37 0 29.75 A
20.00 0 31.50 A
19.37 0 29.57 A
19.37 0 31.50 A
19.37 0 29.75 A
20.00 0 31.50 A
18.12 0 28.00 A
19.37 0 24.50 A
20.00 0 30.62 A
19.37 0 30.62 A
19.37 0 24.50 A
19.37 0 28.00 A
19.37 0 35.00 B
16.87 0 35.00 B
19.37 0 29.75 B
19.37 0 28.00 B
19.37 2 35.00 B
16.87 0 30.45 B
19.37 2 22.75 B
19.37 0 35.00 B
classification 1
Read Only
Action Output
Time
Action
Response
Duration / Fetch Time
4 19:47:09 select * from classification
92 row(s) returned
0.0010 sec / 0.00004...
Query Completed

2. Logistic Regression, SVM 알고리즘 요약, scikit-learn 라이브러리에서 각각의 함수 사용 방법

2.1. Logistic Regression ; 로지스틱 회귀
: 이진 분류 문제를 해결하기 위한 대표적인 알고리즘.

회귀를 사용하여 데이터가 어떤 범주에 속할 확률을 0 과 1 사이의 값으로 예측하고, 이 값에 따라 가능성이 더 높은 범주에 속하는 것으로 분류해주는 알고리즘이다. 가장 대표적인 예로, spam 메일인 것과 아닌 메일을 구분하는 문제가 있으며, spam 메일일 확률이 0.5 이상이면 spam 메일, 0.5 보다 작은 경우 ham 메일인 것이다.
linear regression 모델은 classification 을 수행하는 데에 적합하지 못하다. 메일의 spam/ham 여부 문제를 Linear regression 방식으로 가정을 세우면, 데이터가 추가됨에 따라 직선이 계속 변하게 되고, 이는 정확한 분류를 할 수 없는 문제에 도달한다(이상치에 약함). 이러한 이유로, 우리는 결과 값을 0 과 1 사이로 얻을 수 있는 Logistic Regression 을 사용하게 된다.

- scikit-learn 라이브러리에서 Logistic Regression 함수 사용 방법

```
from sklearn.linear_model import LogisticRegression

logisticregression = LogisticRegression()
logisticregression_model = logisticregression.fit(X_train, y_train)
y_predict = logisticregression_model.predict(X_test)
```

logisticregression = LogisticRegression(): 모델 선언
logisticregression_model = logisticregression.fit(X_train, y_train): 모델 학습
y_predict = logisticregression_model.predict(X_test): 예측

➔ y_test 와 y_predict 값으로 성능 측정

2.2. SVM (Support Vector Machine)
: 데이터들을 고차원 공간으로 사상시킨 후, 선형 분류하는 마진 기반 기계 학습 모델이다.

데이터들을 고차원 공간으로 사상 시킬 때에는 커널 함수를 이용하며, 그 후에는 support vector 들로 이루어진 초평면을 이용하여 선형 분류를 한다. 대표적인 예로 XOR 문제를 들 수 있으며, 이는 직선 하나로 분리할 수 없는 문제이다.
커널 함수란 저차원의 데이터를 고차원 공간으로 사상시키는 함수이다. 이 커널 함수는 선형 분리가 불가능한 문제를 선형 분리 가능한 문제, 즉, 한 개의 직선(초평면)으로 분리할 수 있는 문제로 변환 시킨다. 예를 들어, 2 차원 평면 상에서 직선이 아닌 타원으로 분류되는 데이터들을 3 차원으로 사상 시킨 후, 평면 하나로 두 개로 분류하는 것이 있다.
Support vector 란 선형 분류의 경계 주변에 존재하는 데이터 포인터들을 의미한다. 바로 이 support vector 와 직선 사이의 관계를 가지고 선형 분류를 하는 최적의 직선을 찾게 된다. 선형 분류를 가능하게 하는 직선은 무수히 많이 존재하지만, 그 중 최적의 직선은, 마진을 최대로 하는 직선을 의미한다. 이는, support vector 들 사이의 거리가 가장 먼 직선을 의미한다.
데이터들을 저차원에서 고차원으로 사상하고 내적을 구하는 것을 매우 복잡하다. 따라서, 현재 차원에서 동일한 효과를 거두는 커널 함수를 사용하는데 이를 커널 트릭이라고 한다.

```
from sklearn.svm import SVC

svm = SVC(kernel = 'rbf')
svm_model = svm.fit(X_train, y_train)
y_predict = svm_model.predict(X_test)
```

svm = SVC(): 모델 선언
svm_model = svm.fit(X_train, y_train): 모델 학습
y_predict = svm_model.predict(X_test): 예측

➔ y_test 와 y_predict 값으로 성능 측정

3. grade B 에 대하여, 2 가지 알고리즘의 binary classification 성능 결과 얻을 수 있는 프로그램 작성, 결과 제시 및 분석

```
import pandas as pd
import numpy as np
from db_conn import *
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.svm import SVC

def load_score_data():
    conn, cur = open_db()

    sql = """select * from classification;"""
    cur.execute(sql)

    data = cur.fetchall()
    #print("data =", data)

    close_db(conn, cur)

    X = [(t['homework'], t['discussion'], t['midterm']) for t in data]
    X = np.array(X) # 2 차원 numpy array 로 변환

    y = [1 if t['grade'] == 'B' else -1 for t in data]
    y = np.array(y) # 2 차원 numpy array 로 변환

    return X, y
```

classification 테이블에서 모든 값을 가져와, X(feature variable), y(label attribute)로 데이터를 나눈다. 이 때, grade 가 B 이면 y 값이 1, 아니면(A 또는 C) y 값이 -1 로 2 개(binary)로 분류한다.

```


def score_classification_logistic_regression(X, y): # Logistic
Regression
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42) # train size : test size = 2 : 1 로
나눔 // random_state 는 42 로 쓰는 관습

    logisticregression = LogisticRegression(C=0.1, penalty='l1',
solver='saga', max_iter=10000)
```

```
    logisticregression_model = logisticregression.fit(X_train, y_train)
    y_predict = logisticregression_model.predict(X_test)
    #print("y_predict =", y_predict)

    acc, pre, rec, f1 = classification_performance_eval(y_test,
y_predict)
    print("Logistic Regressioin _ train_test_split:")
    print("accuracy = %f" %acc)
    print("precision = %f" %pre)
    print("recall = %f" %rec)
    print("f1_score = %f" %f1)

    print()

    kf = KFold(n_splits=5, random_state=42, shuffle=True)

    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

    logisticregression_model = logisticregression.fit(X_train, y_train)
    y_predict = logisticregression_model.predict(X_test)

    acc, pre, rec, f1 = classification_performance_eval(y_test,
y_predict)
    print("Logistic Regressioin _ K-Fold cross validation:")
    print("accuracy = %f" %acc)
    print("precision = %f" %pre)
    print("recall = %f" %rec)
    print("f1_score = %f" %f1)
```

sklearn 에서 제공하는 LogisticRegression 모델을 이용하여 학습한다. 단, train, test 분류 시 train_test_split, K-Fold cross validation 각각으로 데이터 나누었을 때의 성능을 따로 출력한다.

모델을 생성할 때, 파라미터에서 C 는 규제의 강도를 조절하는 역할을 한다. C 의 값이 작을수록 규제가 강해진다고 하는데, 그에 따라 0.1, 0.5, 1.0 으로 설정하여 실행해 본 결과, C 값의 크기가 작을수록 정확도가 올라갔다. 따라서, 0.1 로 설정하였다. 이와 같이 각 파라미터에 따라 여러 가지 값들을 넣어본 후, 정확도가 높게 측정되는 파라미터를 결과적으로 설정하였다.

참고: (https://stackoverflow.com/questions/38640109/logistic-regression-python-solvers-definitions)

```
def score_classification_svm(X, y): # SVM
```

```
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

    svm = SVC(kernel = 'rbf')
    svm_model = svm.fit(X_train, y_train)
    y_predict = svm_model.predict(X_test)

    acc, pre, rec, f1 = classification_performance_eval(y_test,
y_predict)
    print("SVM _ train_test_split:")
    print("accuracy = %f" %acc)
    print("precision = %f" %pre)
    print("recall = %f" %rec)
    print("f1_score = %f" %f1)

    print()

    kf = KFold(n_splits=5, random_state=42, shuffle=True)

    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

    svm_model = svm.fit(X_train, y_train)
    y_predict = svm_model.predict(X_test)

    acc, pre, rec, f1 = classification_performance_eval(y_test,
y_predict)
    print("SVM _ K-Fold cross validation:")
    print("accuracy = %f" %acc)
    print("precision = %f" %pre)
    print("recall = %f" %rec)
    print("f1_score = %f" %f1)
```

sklearn 에서 제공하는 SVM 모델을 이용하여 학습한다. 단, train, test 분류 시 train_test_split, K-Fold cross validation 각각으로 데이터 나누었을 때의 성능을 따로 출력한다. SVM 모델을 생성할 때, 파라미터의 값으로 kernel 함수를 지정하게 되는데, 기본값이 rbf 를 그대로 사용한다.

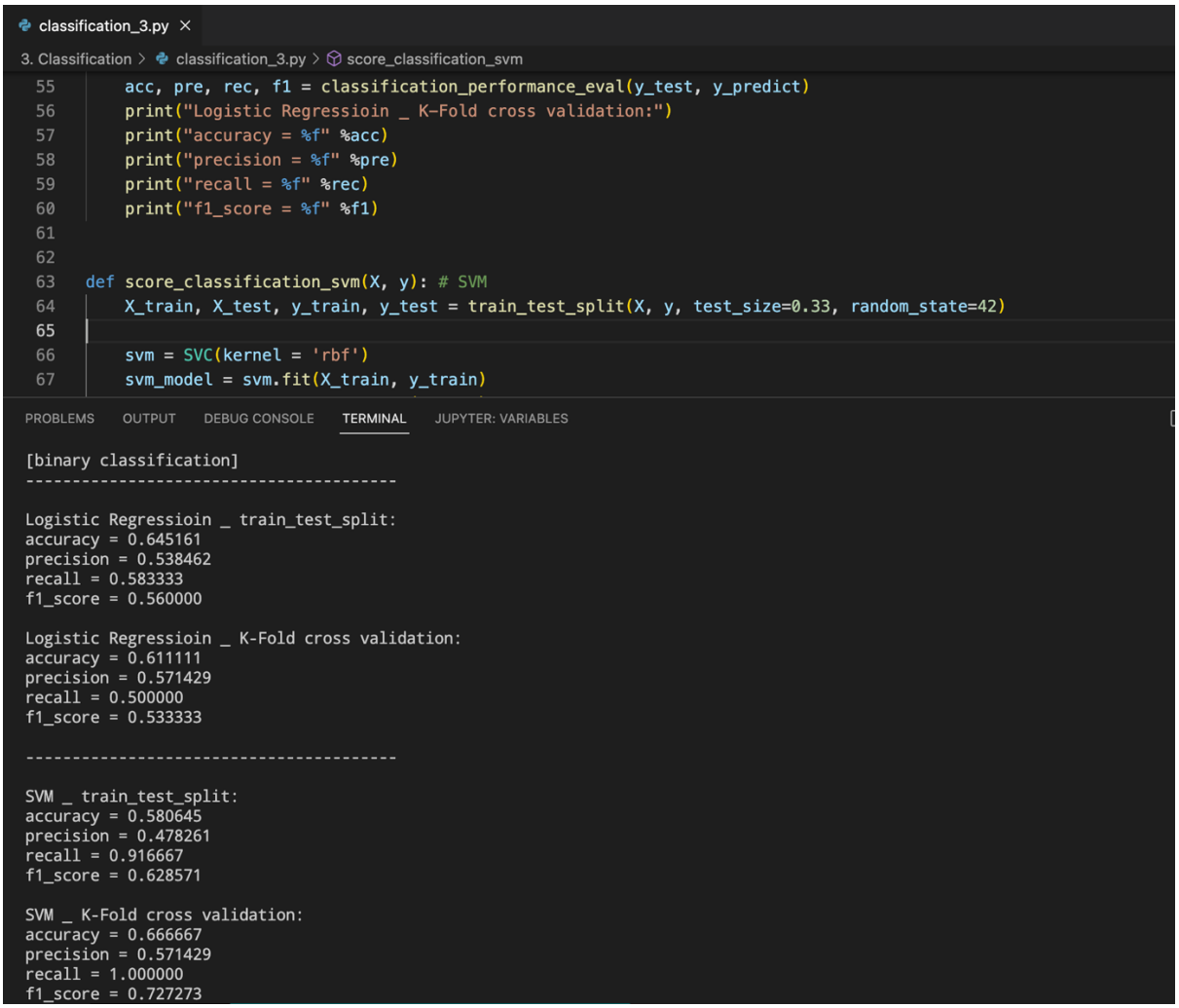
```
def classification_performance_eval(y_test, y_predict):
    tp, tn, fp, fn = 0, 0, 0, 0
```

```
    for y, yp in zip(y_test, y_predict): # zip: 2 개의 array 에서 동시
하나씩 가져옴
        if y == 1 and yp == 1:
            tp += 1
        elif y == 1 and yp == -1:
            fn += 1
        elif y == -1 and yp == 1:
            fp += 1
        else:
            tn += 1

    accuracy = (tp+tn)/(tp+tn+fp+fn)
    precision = tp/(tp+fp)
    recall = tp/(tp+fn)
    f1_score  = 2*precision*recall/(precision+recall)

    return accuracy, precision, recall, f1_score
```

실습 수업 시간에 학습했던 내용을 바탕으로 성능 4 가지 measure 사용하여 출력한다. accuracy 는 모든 값 중 올바르게 예측한 것의 비율, precision 은 positive 로 예측한 것 중 실제로 positive 인 것의 비율, recall 은 실제 positive 인 것 중 positive 로 예측한 것의 비율, f1_score 은 precision 과 recall 의 조화평균이다.

```
if __name__ == "__main__":
    print("[binary classification]")
    print("----------------------------------------")
    print()

    X, y = load_score_data()
    score_classification_logistic_regression(X, y)
    print()
    print("----------------------------------------")
    print()
    score_classification_svm(X, y)
```

실행 결과)



(모델 생성 시, 파라미터 값에 따른 성능 변화에 대한 설명은 소스 코드에서 함)

4. grade A, B, C 에 대하여, 2 가지 알고리즘의 multi-class classification 성능 결과 얻을 수 있는 프로그램 작성, 결과 제시 및 분석

```
import pandas as pd
import numpy as np
from db_conn import *
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import KFold
from sklearn.svm import SVC
from sklearn.metrics import *

def load_score_data():
    conn, cur = open_db()

    sql = """select * from classification;"""
    cur.execute(sql)

    data = cur.fetchall()
    #print("data =", data)

    close_db(conn, cur)

    X = [(t['homework'], t['discussion'], t['midterm']) for t in data]
    X = np.array(X) # 2 차원 numpy array 로 변환

    y = []
    for t in data:
        if t['grade'] == 'A':
            y.append(0)
        elif t['grade'] == 'B':
            y.append(1)
        else:
            y.append(2)

    y = np.array(y)

    return X, y
```

classification 테이블에서 모든 값을 가져와, X(feature variable), y(label attribute)로 데이터를 나눈다. 이 때, grade 가 A, B, C 인 경우를 모두 다르게 고려해야 하기 때문에, grader A 면 y 값이 0, B 면 y 값이 1, C 면 2 가 되도록 분류한다.

```
def score_classification_logistic_regression(X, y): # Logistic
Regression
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42) # train size : test size = 2 : 1 로
나눔 // random_state 는 42 로 쓰는 관습

    logisticregression = LogisticRegression(C=0.1, multi_class='ovr',
penalty='l1', solver='saga', max_iter=8000)
    logisticregression_model = logisticregression.fit(X_train, y_train)
    y_predict = logisticregression_model.predict(X_test)

    acc, pre, rec, f1 = classification_performance_eval(y_test,
y_predict)
    print("Logistic Regression _ train_test_split:")
    print("accuracy = %f" %acc)
    print("precision = %f" %pre)
    print("recall = %f" %rec)
    print("f1_score = %f" %f1)

    print()

    kf = KFold(n_splits=5, random_state=42, shuffle=True)

    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

    logisticregression_model = logisticregression.fit(X_train, y_train)
    y_predict = logisticregression_model.predict(X_test)

    acc, pre, rec, f1 = classification_performance_eval(y_test,
y_predict)
    print("Logistic Regression _ K-Fold cross validation:")
    print("accuracy = %f" %acc)
    print("precision = %f" %pre)
    print("recall = %f" %rec)
    print("f1_score = %f" %f1)
```

sklearn 에서 제공하는 LogisticRegression 모델을 이용하여 학습한다. 단, train, test 분류 시 train_test_split, K-Fold cross validation 각각으로 데이터 나누었을 때의 성능을 따로 출력한다.

모델 생성 시 입력하는 파라미터 값은, 3 번 문제와 동일하다.

```
def score_classification_svm(X, y): # SVM
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.33, random_state=42)

    svm = SVC(kernel = 'linear')
    svm_model = svm.fit(X_train, y_train)
    y_predict = svm_model.predict(X_test)

    acc, pre, rec, f1 = classification_performance_eval(y_test,
y_predict)
    print("SVM _ train_test_split:")
    print("accuracy = %f" %acc)
    print("precision = %f" %pre)
    print("recall = %f" %rec)
    print("f1_score = %f" %f1)

    print()

    kf = KFold(n_splits=5, random_state=42, shuffle=True)

    for train_index, test_index in kf.split(X):
        X_train, X_test = X[train_index], X[test_index]
        y_train, y_test = y[train_index], y[test_index]

    svm_model = svm.fit(X_train, y_train)
    y_predict = svm_model.predict(X_test)

    acc, pre, rec, f1 = classification_performance_eval(y_test,
y_predict)
    print("SVM _ K-Fold cross validation:")
    print("accuracy = %f" %acc)
    print("precision = %f" %pre)
    print("recall = %f" %rec)
    print("f1_score = %f" %f1)
```

sklearn 에서 제공하는 SVM 모델을 이용하여 학습한다. 단, train, test 분류 시 train_test_split, K-Fold cross validation 각각으로 데이터 나누었을 때의 성능을 따로 출력한다. SVM 모델을 생성할 때, 파라미터의 값으로 kernel 함수를 지정한다. 소스코드 실행 결과, binary classification 과 다르게 기본 값인 rbf 보다 linear 을 사용할 시 성능이 더 높았다. 따라서, linear 를 사용한다.

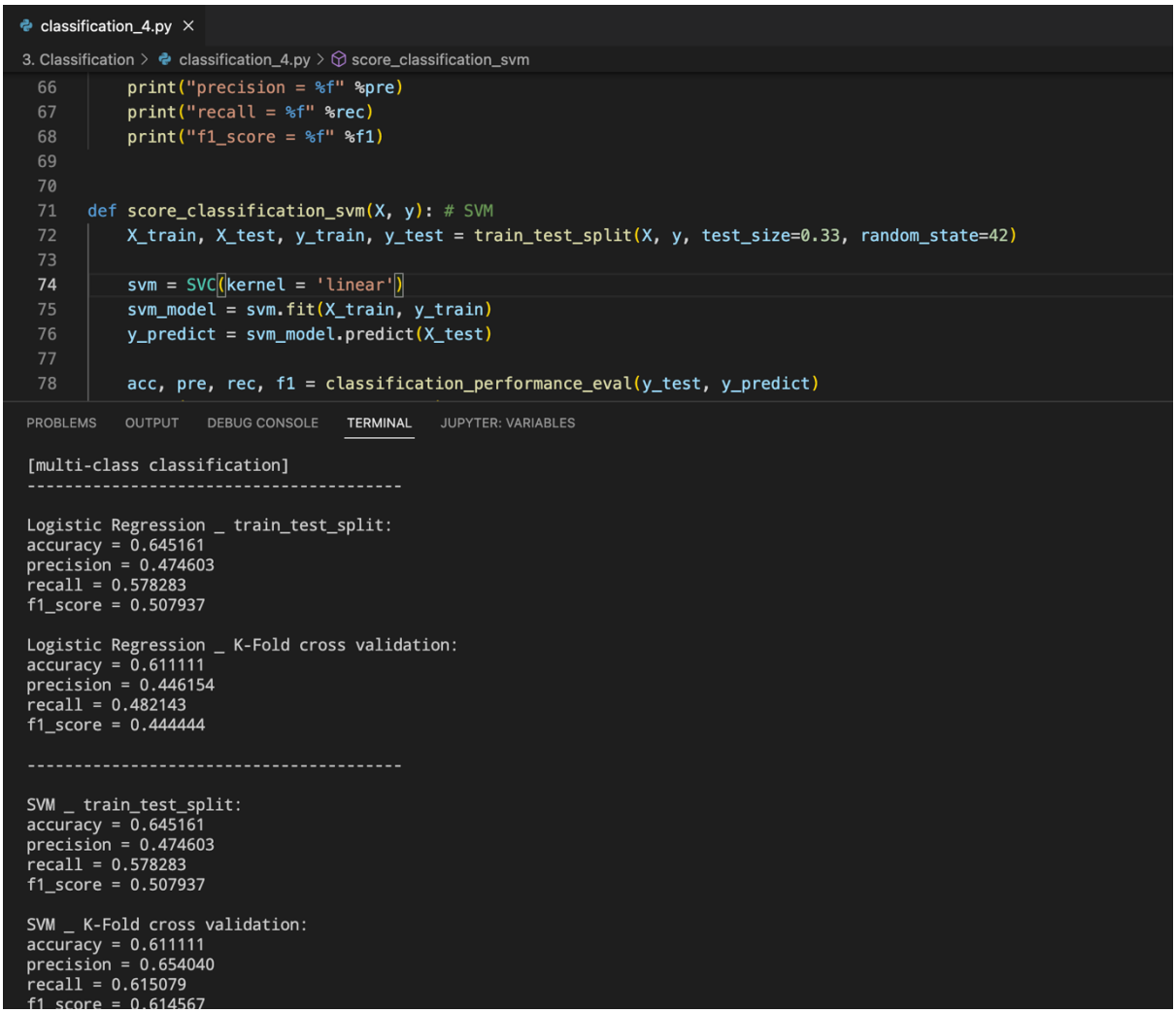
```
```

```
def classification_performance_eval(y_test, y_predict):
    accuracy = accuracy_score(y_test, y_predict)
    precision = precision_score(y_test, y_predict, average='macro',
zero_division=False)
    recall = recall_score(y_test, y_predict, average='macro',
zero_division=False)
    f1 = f1_score(y_test, y_predict, average='macro',
zero_division=False)

    # 성능 지표 한 번에 확인할 수 있는 함수
    # report = classification_report(y_test, y_predict,
zero_division=False)

    return accuracy, precision, recall, f1
```

실습 수업 시간에 학습했던 내용을 바탕으로, 성능 4 가지 measure(accuracy, precision, recall, f1_score)를 sklearn 에서 제공하는 각각의 함수 accuracy_score, precision_score, recall_score, f1_score 함수를 사용하여 계산 및 출력한다.

```
if __name__ == "__main__":
    print("[multi-class classification]")
    print("---------------------------------------")
    print()

    X, y = load_score_data()
    score_classification_logistic_regression(X, y)
    print()
    print("---------------------------------------")
    print()
    score_classification_svm(X, y)
```

실행 결과)



(모델 생성 시, 파라미터 값에 따른 성능 변화에 대한 설명은 소스 코드에서 함)