과제 보고서

제목 : 당뇨병 & credit 보고서

과 목 명: 파이썬 통계분석
제출일자: 2022.12.11.
학 과: 정보통계학과
학 번: 2018015027
이 름: 김한탁

충북대학교
CHUNGBUK NATIONAL UNIVERSITY

*** 당뇨병 보고서(당뇨병 환자 데이터 예측 MSE 비교)**

1. 데이터 변환

1.1 당뇨병 데이터

1) 당뇨병 데이터

```
In [2]: diabetes = load_diabetes()
        diabetes
```

```
Out[2]: {'data': array([[ 0.03807591,  0.05068012,  0.06169621, ..., -0.00259226,
                  0.01990842, -0.01764613],
                [-0.00188202, -0.04464164, -0.05147406, ..., -0.03949338,
                 -0.06832974, -0.09220405],
                [ 0.08529891,  0.05068012,  0.04445121, ..., -0.00259226,
                  0.00286377, -0.02593034],
                ...,
                [ 0.04170844,  0.05068012, -0.01590626, ..., -0.01107952,
                 -0.04687948,  0.01549073],
                [-0.04547248, -0.04464164,  0.03906215, ...,  0.02655962,
                  0.04452837, -0.02593034],
                [-0.04547248, -0.04464164, -0.0730303 , ..., -0.03949338,
                 -0.00421986,  0.00306441]]),
         'target': array([151.,  75., 141., 206., 135.,  97., 138.,  63., 110., 310., 101.,
                 69., 179., 185., 118., 171., 166., 144.,  97., 168.,  68.,  49.,
                 68., 245., 184., 202., 137.,  85., 131., 283., 129.,  59., 341.,
                 87.,  65., 102., 265., 276., 252.,  90., 100.,  55.,  61.,  92.,
                259.,  53., 190., 142.,  75., 142., 155., 225.,  59., 104., 182.,
                128.,  52.,  37., 170., 170.,  61., 144.,  52., 128.,  71., 163.,
                150.,  97., 160., 178.,  48., 270., 202., 111.,  85.,  42., 170.,
                200., 252., 113., 143.,  51.,  52., 210.,  65., 141.,  55., 134.,
                 42., 111.,  98., 164.,  48.,  96.,  90., 162., 150., 279.,  92.,
                 83., 128., 102., 302., 198.,  95.,  53., 134., 144., 232.,  81.,
                104.,  59., 246., 297., 258., 229., 275., 281., 179., 200., 200.,
                173., 180.,  84., 121., 161.,  99., 109., 115., 268., 274., 158.,
                107.,  83., 103., 272.,  85., 280., 336., 281., 118., 317., 235.,
                 60., 174., 259., 178., 128.,  96., 126., 288.,  88., 292.,  71.,
                197., 186.,  25.,  84.,  96., 195.,  53., 217., 172., 131., 214.,
                 59.,  70., 220., 268., 152.,  47.,  74., 295., 101., 151., 127.,
                237., 225.,  81., 151., 107.,  64., 138., 185., 265., 101., 137.,
                143., 141.,  79., 292., 178.,  91., 116.,  86., 122.,  72., 129.,
                142.,  90., 158.,  39., 196., 222., 277.,  99., 196., 202., 155.,
                 77., 191.,  70.,  73.,  49.,  65., 263., 248., 296., 214., 185.,
                 78.,  93., 252., 150.,  77., 208.,  77., 108., 160.,  53., 220.,
                154., 259.,  90., 246., 124.,  67.,  72., 257., 262., 275., 177.,
                 71.,  47., 187., 125.,  78.,  51., 258., 215., 303., 243.,  91.,
                150., 310., 153., 346.,  63.,  89.,  50.,  39., 103., 308., 116.,
                145.,  74.,  45., 115., 264.,  87., 202., 127., 182., 241.,  66.,
                 94., 283.,  64., 102., 200., 265.,  94., 230., 181., 156., 233.,
                 60., 219.,  80.,  68., 332., 248.,  84., 200.,  55.,  85.,  89.,
                 31., 129.,  83., 275.,  65., 198., 236., 253., 124.,  44., 172.,
                114., 142., 109., 180., 144., 163., 147.,  97., 220., 190., 109.,
                191., 122., 230., 242., 248., 249., 192., 131., 237.,  78., 135.,
                244., 199., 270., 164.,  72.,  96., 306.,  91., 214.,  95., 216.,
                263., 178., 113., 200., 139., 139.,  88., 148.,  88., 243.,  71.,
                 77., 109., 272.,  60.,  54., 221.,  90., 311., 281., 182., 321.,
                 58., 262., 206., 233., 242., 123., 167.,  63., 197.,  71., 168.,
                140., 217., 121., 235., 245.,  40.,  52., 104., 132.,  88.,  69.,
                219.,  72., 201., 110.,  51., 277.,  63., 118.,  69., 273., 258.,
                 43., 198., 242., 232., 175.,  93., 168., 275., 293., 281.,  72.,
                140., 189., 181., 209., 136., 261., 113., 131., 174., 257.,  55.,
                 84.,  42., 146., 212., 233.,  91., 111., 152., 120.,  67., 310.,
                 94., 183.,  66., 173.,  72.,  49.,  64.,  48., 178., 104., 132.,
                220.,  57.]),
         'feature_names': ['age',
          'sex',
          'bmi',
          'bp',
          's1',
          's2',
          's3',
          's4',
          's5',
          's6'],
         'data_filename': 'diabetes_data.csv.gz',
         'target_filename': 'diabetes_target.csv.gz',
         'data_module': 'sklearn.datasets.data'}
```

2) 결과해석

'age', 'sex', 'bmi', 'bp', 's1', 's2', 's3', 's4', 's5', 's6' 10개의 독립변수가 존재하고, 종속변수로 'target' 존재하여 총 11개의 변수가 존재함을 알 수 있다.

1.2 데이터의 데이터프레임 변환, train-test 분류, 독립변수 표준화

1) 데이터 프레임 변환

```
In [3]: diabetes_df = pd.DataFrame(diabetes.data, columns = diabetes.feature_names)
        diabetes_df["diabetes value"]=diabetes.target
        diabetes_df.head()
```

Out[3]:

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	diabetes value
0	0.038076	0.050680	0.061696	0.021872	-0.044223	-0.034821	-0.043401	-0.002592	0.019908	-0.017646	151.0
1	-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	-0.068330	-0.092204	75.0
2	0.085299	0.050680	0.044451	-0.005671	-0.045599	-0.034194	-0.032356	-0.002592	0.002864	-0.025930	141.0
3	-0.089063	-0.044642	-0.011595	-0.036656	0.012191	0.024991	-0.036038	0.034309	0.022692	-0.009362	206.0
4	0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015596	0.008142	-0.002592	-0.031991	-0.046641	135.0

2) train-test 자료 분류

```
In [4]: ##훈련용, 테스트용 자료 분류 : fixed
        x_data=diabetes_df.iloc[:, 0:10]
        y_data=diabetes_df["diabetes value"]
        num_of_train = int(len(x_data) * 0.8) # 데이터의 전체 길이의 80%에 해당하는 길이값을 구한다.
        num_of_test = int(len(x_data) - num_of_train) # 전체 길이에서 80%에 해당하는 길이를 뺀다.
        print('훈련 데이터의 크기 :',num_of_train)
        print('테스트 데이터의 크기 :',num_of_test)
```

```
훈련 데이터의 크기 : 353
테스트 데이터의 크기 : 89
```

```
In [5]: x_test = x_data[num_of_train:] # 전체 데이터 중에서 20%만큼 뒤의 데이터 저장
        y_test = y_data[num_of_train:] # 전체 데이터 중에서 20%만큼 뒤의 데이터 저장
        x_train = x_data[:num_of_train] # 전체 데이터 중에서 80%만큼 앞의 데이터 저장
        y_train = y_data[:num_of_train] # 전체 데이터 중에서 80%만큼 앞의 데이터 저장
```

3) 독립변수(x) 표준화

```
In [6]: # x_train과 x_test를 StandardScaler를 이용해 정규화
        from sklearn.preprocessing import StandardScaler
        scaler= StandardScaler()
        scaler.fit(x_train)
        x_train = scaler.transform(x_train)
        x_test = scaler.transform(x_test)
        x_train=pd.DataFrame(x_train, columns=x_data.columns)
        x_test=pd.DataFrame(x_test, columns=x_data.columns)
        x_train.head()
```

Out[6]:

	age	sex	bmi	bp	s1	s2	s3	s4	s5	s6
0	0.767844	1.067424	1.347413	0.454905	-0.935888	-0.741500	-0.900815	-0.051951	0.440448	-0.359531
1	-0.057153	-0.936835	-1.100358	-0.560668	-0.178229	-0.410729	1.521083	-0.808181	-1.457857	-1.937199
2	1.742841	1.067424	0.974420	-0.125422	-0.965029	-0.728269	-0.673762	-0.051951	0.073759	-0.534828
3	-1.857146	-0.936835	-0.237810	-0.778291	0.258882	0.522048	-0.749446	0.704279	0.500333	-0.184235
4	0.092847	-0.936835	-0.773988	0.454905	0.084038	0.323585	0.158766	-0.051951	-0.676096	-0.973069

4) 결과해석

1)에서 배열 형태의 자료였던 당뇨병 데이터를 데이터프레임의 형태로 바꾸었다. 이 과정에서 target 데이터를 변수 “diabetes value”로 저장하여 데이터프레임에 추가하였다.

데이터프레임의 형태로 바꾼 데이터를 교차검증을 위해 train-test의 자료로 분류하였는데, 그 비율을 8:2로 하여 2)에서 분류하였음을 확인할 수 있다.

다음으로 분석 방법(다중회귀, 주성분 회귀, 신경망)에서 사용되기 위해, 3)에서 독립변수인 x 데이터(x_train: train 데이터, x_test: test 데이터)에 대하여 표준화를 시행하였음을 알 수 있다. 그뿐만 아니라 train 데이터의 평균, 분산을 기준으로 train, test 데이터의 표준화가 이루어진 것도 볼 수 있다.

2. 다중 회귀분석 및 예측 MSE

2.1 다중 회귀분석 결과

1) 다중 회귀분석 결과

```
In [7]: #1. 다중회귀모형 분석
        # 다중회귀모형분석(full-model)
        diabetes_train=x_train.join(y_train)
        fit1=smf.ols('y_train~x_train', diabetes_train).fit()
        print(fit1.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                y_train   R-squared:                       0.509
Model:                            OLS   Adj. R-squared:                  0.495
Method:                 Least Squares   F-statistic:                     35.44
Date:                Mon, 05 Dec 2022   Prob (F-statistic):           4.02e-47
Time:                        15:30:51   Log-Likelihood:                -1905.0
No. Observations:                 353   AIC:                             3832.
Df Residuals:                     342   BIC:                             3874.
Df Model:                          10
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    151.4788      2.887     52.471      0.000     145.800     157.157
x_train[0]    -0.3911      3.182     -0.123      0.902      -6.650       5.868
x_train[1]   -11.8353      3.291     -3.597      0.000     -18.308      -5.363
x_train[2]    23.7574      3.595      6.608      0.000      16.686      30.829
x_train[3]    13.9384      3.460      4.029      0.000       7.133      20.744
x_train[4]   -22.4557     23.432     -0.958      0.339     -68.544      23.633
x_train[5]    10.0876     19.135      0.527      0.598     -27.550      47.725
x_train[6]    -1.6351     12.086     -0.135      0.892     -25.407      22.136
x_train[7]     7.1825      8.655      0.830      0.407      -9.842      24.207
x_train[8]    29.7076      9.143      3.249      0.001      11.723      47.692
x_train[9]     4.0802      3.602      1.133      0.258      -3.004      11.165
==============================================================================
Omnibus:                        1.446   Durbin-Watson:                   1.998
Prob(Omnibus):                  0.485   Jarque-Bera (JB):                1.361
Skew:                           0.038   Prob(JB):                        0.506
Kurtosis:                       2.705   Cond. No.                         23.1
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

2) 결과해석

다중 회귀모형의 R^2는 0.509로, target 데이터에 대하여 약 51% 정도가 독립변수들에 의해 설명된다고 볼 수 있다.

또한 Durbin-Watson 통계량이 1.998인 점을 통해 오차 독립성 가정이 매우 잘 지켜진다고 판단할 수 있다. 그러나 모형에 독립변수 sex, bmi, bp, s5를 제외한 나머지 독립변수의 회귀계수가 유의하지 않다는 것과 다중공선성의 문제가 존재할 수 있다는 문제점을 가지고 있다.

2.2 다중 회귀모형의 예측 MSE

1) 예측 MSE & RMSE

```
In [8]: # Linear Regression OLS로 학습/예측평가 수행.
        reg= LinearRegression()
        reg.fit(x_train ,y_train)
        y_preds= reg.predict(x_test)
        mse = mean_squared_error(y_test, y_preds)
        rmse = np.sqrt(mse)
```

```
In [9]: # MSE, RMSE
        print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mse , rmse))
```

```
MSE : 2929.887 , RMSE : 54.128
```

2) 결과해석

다중 회귀모형에 대한 예측 MSE는 2929.887, 예측 RMSE는 54.128로 나타났다.

3 주성분 회귀 및 예측 MSE

3.1 주성분 분석(PCA)

1) 주성분 개수 결정

In [10]:
```
#2.주성분분석
# PCA모듈설치 및 PCA분석+주성분의 회귀계수를 데이터프레임으로 구성
import statsmodels.formula.api as smf
from sklearn.decomposition import PCA
pca = PCA(n_components=4)   # 주성분을 몇개로 할지 결정
printcipalComponents = pca.fit_transform(x_train)
principalDf = pd.DataFrame(data=printcipalComponents, columns =
                           ['pca1', 'pca2', 'pca3', 'pca4'])
```

2) 각 주성분의 설명력과 누적 설명력

In [11]:
```
# 각 주성분의 설명력
pca.explained_variance_ratio_
```

Out[11]:
```
array([0.40840746, 0.14966528, 0.11804921, 0.09612724])
```

In [12]:
```
sum(pca.explained_variance_ratio_)   # 누적설명력
```

Out[12]:
```
0.7722491877767218
```

3) 주성분 식의 고유벡터와 주성분의 데이터프레임 생성

In [13]:
```
# 주성분분석 : 고유벡터(주성분 식의 계수값)
pca.components=pd.DataFrame(pca.components_)
pca.components
```

Out[13]:

	0	1	2	3	4	5	6	7	8	9
0	0.210174	0.168510	0.316500	0.269375	0.346246	0.357016	-0.273829	0.424726	0.376457	0.332046
1	-0.138300	0.438194	0.087941	0.025278	-0.540536	-0.378793	-0.563160	0.153070	0.005876	0.064881
2	0.502106	0.059905	0.138816	0.535366	-0.166151	-0.351024	0.297858	-0.357257	0.071758	0.257077
3	-0.383641	-0.668344	0.476510	0.063620	-0.119878	-0.237042	-0.067010	-0.019469	0.305431	0.079430

In [15]:
```
# 주성분의 데이터프레임 생성
pca_xtrain=pd.DataFrame()
pca_xtrain['pca1']=principalDf['pca1']
pca_xtrain['pca2']=principalDf['pca2']
pca_xtrain['pca3']=principalDf['pca3']
pca_xtrain['pca4']=principalDf['pca4']
pca_xtrain.head()
```

Out[15]:

	pca1	pca2	pca3	pca4
0	0.572508	1.756905	0.985274	0.118318
1	-2.829351	-1.376205	-0.224767	-0.478562
2	0.239260	1.443937	1.108810	-0.611166
3	0.074432	-0.011093	-2.150851	1.195681
4	-0.759321	-0.812388	-0.233773	-0.129561

4) 결과해석

1)에서 train 데이터를 이용하여, 주성분의 개수를 임의로 4개로 정하여 주성분 분석을 시행하였다.

주성분 4개의 각 설명력은 약 0.41, 0.15, 0.12, 0.1 정도로 나타났고, 누적 설명력은 약 0.77로 전체 설명력의 77%에 해당하여 양호하다고 판단된다.

다음으로 3)을 통해, 각 주성분 식의 고유벡터를 확인할 수 있는데, 이것으로 주성분을 특징지어 새로운 이름(변수)을 붙일 수 있다. 그러나 기존의 몇몇 독립변수(s1, s2, …, s6)의 특징을 알 수 없으므로 4개의 주성분의 이름을 'pca1', 'pca2', 'pca3', 'pca4'로 정하였다. 그리고 주성분 회귀 분석에 이용될 주성분들로 구성된 데이터프레임 pca_xtrain을 생성하였다.

3.2 주성분 회귀 분석

1) 주성분 회귀 분석 결과

```
In [16]: pca_train=pca_xtrain.join(y_train)
         fit2=smf.ols('y_train~pca1+pca2+pca3+pca4', pca_train).fit()
         print(fit2.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                y_train   R-squared:                       0.498
Model:                            OLS   Adj. R-squared:                  0.493
Method:                 Least Squares   F-statistic:                     86.46
Date:                Mon, 05 Dec 2022   Prob (F-statistic):           6.27e-51
Time:                        15:30:57   Log-Likelihood:                -1908.7
No. Observations:                 353   AIC:                             3827.
Df Residuals:                     348   BIC:                             3847.
Df Model:                           4
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
Intercept    151.4788      2.892     52.376      0.000     145.790     157.167
pca1          21.0603      1.431     14.716      0.000      18.246      23.875
pca2           8.0861      2.364      3.420      0.001       3.436      12.736
pca3          10.1723      2.662      3.821      0.000       4.937      15.408
pca4          29.9357      2.950     10.148      0.000      24.134      35.737
==============================================================================
Omnibus:                        3.172   Durbin-Watson:                   1.938
Prob(Omnibus):                  0.205   Jarque-Bera (JB):                2.451
Skew:                           0.054   Prob(JB):                        0.294
Kurtosis:                       2.606   Cond. No.                         2.06
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

2) 결과해석

주성분 회귀모형의 R^2는 0.498로, target 데이터에 대하여 약 50% 정도가 주성분들에 의해 설명된다고 볼 수 있다.

또한 Durbin-Watson 통계량이 1.938인 점을 통해 오차 독립성 가정이 매우 잘 지켜진다고 판단할 수 있다.

마지막으로 모든 주성분의 회귀계수가 유의하고, 주성분 간에 다중공선성이 존재하지 않으므로 회귀모형에 큰 문제가 없는 것으로 보인다.

3.3 test 데이터 주성분 변환 및 주성분 회귀모형의 예측 MSE

1) test 데이터의 주성분 변환

```
In [17]: # test-set를 예측하기 위해 PCA점수로 변환 후 예측
         arr1=pca.components_
         arr2=np.array(x_test.T)
         result=arr1@arr2
```

```
In [18]: # 주성분의 이름부여 및 원자료에 주성분점수 추가
         cols = ["pca1", "pca2", "pca3", "pca4"]
         pca_xtest = pd.DataFrame(result.T, columns=cols)  # result.T 12*5
         pca_xtest.head()
```

2) 예측 MSE

```
In [19]: # Linear Regression OLS로 학습/예측평가 수행.
         from sklearn.metrics import mean_squared_error , r2_score
         y_preds= fit2.predict(pca_xtest)
         mse = mean_squared_error(y_test, y_preds)
         rmse = np.sqrt(mse)
         # MSE, RMSE, R^2, 설명분산점수(적합모형의 편향성 평가 : 0에 가까워야)
         print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mse , rmse))
```

```
MSE : 3104.153 , RMSE : 55.715
```

3) 결과해석

주성분 회귀모형의 예측 MSE를 구하기 위해, 예측값을 구해야 한다.

여기서 예측값은 test 데이터에 대한 예측값이므로 test 데이터와 train 데이터의 고유벡터 행렬 곱을 통해, 주성분 점수를 구할 수 있다. 이것은 1)을 통해 확인할 수 있다.

구해진 예측값을 이용해, 주성분 회귀모형의 예측 MSE는 3104.153, 예측 RMSE는 55.715로 얻어졌다.

4. 신경망의 예측 MSE

4.1 신경망 회귀모형(활성함수 - reiu), 예측 MSE

1) 신경망 회귀모형 생성(활성함수 - reiu)

In [20]:
```
#3. 신경망(예측)
#3.1 회귀 모델 생성 - reiu 활성함수
import tensorflow as tf
model_reiu = tf.keras.Sequential([
    tf.keras.layers.Dense(units=52, activation='relu', input_shape=(10,)),
    tf.keras.layers.Dense(units=39, activation='relu'),
    tf.keras.layers.Dense(units=26, activation='relu'),
    tf.keras.layers.Dense(units=1)
])
model_reiu.compile(optimizer=tf.keras.optimizers.Adam(lr=0.01), loss='mse')
model_reiu.summary()
```

```
WARNING:absl:`lr` is deprecated, please use `learning_rate` instead, or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.

Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 dense (Dense)               (None, 52)                572

 dense_1 (Dense)             (None, 39)                2067

 dense_2 (Dense)             (None, 26)                1040

 dense_3 (Dense)             (None, 1)                 27

=================================================================
Total params: 3,706
Trainable params: 3,706
Non-trainable params: 0
_________________________________________________________________
```

2) 회귀모형 학습

In [20]:
```
# 회귀 모델 학습 : epochs=신경망을 전체 한번 학습하여 나온 결과
# batch_size=train-set의 80%의 자료를 32개씩 소그룹(sub-group)으로 나누어 빠르게 학습
model_reiu.fit(x_train, y_train, epochs=100, batch_size=32, validation_split=0.2)
```

```
Epoch 92/100
9/9 [==============================] - 0s 5ms/step - loss: 2543.6902 - val_loss: 2817.6929
Epoch 93/100
9/9 [==============================] - 0s 4ms/step - loss: 2537.5647 - val_loss: 2809.0793
Epoch 94/100
9/9 [==============================] - 0s 5ms/step - loss: 2538.3069 - val_loss: 2799.7104
Epoch 95/100
9/9 [==============================] - 0s 5ms/step - loss: 2541.2915 - val_loss: 2797.1311
Epoch 96/100
9/9 [==============================] - 0s 5ms/step - loss: 2526.6650 - val_loss: 2796.6108
Epoch 97/100
9/9 [==============================] - 0s 5ms/step - loss: 2526.7119 - val_loss: 2800.1091
Epoch 98/100
9/9 [==============================] - 0s 5ms/step - loss: 2522.0005 - val_loss: 2812.4998
Epoch 99/100
9/9 [==============================] - 0s 5ms/step - loss: 2527.7200 - val_loss: 2804.3003
Epoch 100/100
9/9 [==============================] - 0s 5ms/step - loss: 2515.9607 - val_loss: 2796.7808
```

Out[20]:
```
<keras.callbacks.History at 0x1ff0362ddf0>
```

3) 신경망 회귀모형의 예측 MSE

In [21]:
```
# 회귀 모델 평가
from sklearn.metrics import mean_squared_error
y_pred = model_reiu.predict(x_test)
mse = mean_squared_error(y_test, y_pred)
rmse = np.sqrt(mse)
print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mse , rmse))
```

```
3/3 [==============================] - 0s 2ms/step
MSE : 3369.631 , RMSE : 58.049
```

4) 결과해석

1)에서 활성함수를 'reiu'로 하고, 총 5개의 층(입력층: 1개, 은닉층: 3개, 출력층: 1개)을 가지는 회귀모형을 생성하였다.

생성한 모형을 토대로 train 데이터의 80%(validation_split=0.2)를 32개씩 소그룹으로 나누어 100번(epochs=100) 학습하였다.

따라서 학습된 신경망 회귀모형에 대한 예측 MSE는 3369.631, 예측 RMSE는 58.049로 나타났다.

4.2 신경망 회귀모형(활성함수 - sigmoid), 예측 MSE

1) 신경망 회귀모형 생성(활성함수 - sigmoid)

```
In [23]: #3. 신경망(예측)
         #3.2 회귀 모델 생성 - sigmoid 활성함수
         import tensorflow as tf
         model_sigmoid = tf.keras.Sequential([
             tf.keras.layers.Dense(units=52, activation='sigmoid', input_shape=(10,)),
             tf.keras.layers.Dense(units=39, activation='sigmoid'),
             tf.keras.layers.Dense(units=26, activation='sigmoid'),
             tf.keras.layers.Dense(units=1)
         ])
         model_sigmoid.compile(optimizer=tf.keras.optimizers.Adam(lr=0.01), loss='mse')
         model_sigmoid.summary()

         WARNING:absl:`lr` is deprecated, please use `learning_rate` instead, or use the legacy optimizer, e.g.,tf.keras.optimizers.legacy.Adam.

         Model: "sequential_1"
         _________________________________________________________________
          Layer (type)                Output Shape              Param #
         =================================================================
          dense_4 (Dense)             (None, 52)                572

          dense_5 (Dense)             (None, 39)                2067

          dense_6 (Dense)             (None, 26)                1040

          dense_7 (Dense)             (None, 1)                 27

         =================================================================
         Total params: 3,706
         Trainable params: 3,706
         Non-trainable params: 0
```

2) 회귀모형 학습

```
In [23]: # 회귀 모델 학습 : epochs=신경망을 전체 한번 학습하여 나온 결과
         # batch_size=train-set의 80%의 자료를 32개씩 소그룹(sub-group)으로 나누어 빠르게 학습
         model_sigmoid.fit(x_train, y_train, epochs=100, batch_size=40, validation_split=0.2)
         Epoch 92/100
         8/8 [==============================] - 0s 5ms/step - loss: 24971.4102 - val_loss: 26878.6523
         Epoch 93/100
         8/8 [==============================] - 0s 5ms/step - loss: 24946.2129 - val_loss: 26851.0703
         Epoch 94/100
         8/8 [==============================] - 0s 5ms/step - loss: 24920.2402 - val_loss: 26823.5039
         Epoch 95/100
         8/8 [==============================] - 0s 5ms/step - loss: 24894.6953 - val_loss: 26796.6895
         Epoch 96/100
         8/8 [==============================] - 0s 5ms/step - loss: 24869.7285 - val_loss: 26770.0059
         Epoch 97/100
         8/8 [==============================] - 0s 5ms/step - loss: 24845.1699 - val_loss: 26743.7852
         Epoch 98/100
         8/8 [==============================] - 0s 5ms/step - loss: 24820.5000 - val_loss: 26716.8145
         Epoch 99/100
         8/8 [==============================] - 0s 5ms/step - loss: 24794.4355 - val_loss: 26688.9043
         Epoch 100/100
         8/8 [==============================] - 0s 5ms/step - loss: 24768.7227 - val_loss: 26659.6699
Out[23]: <keras.callbacks.History at 0x1ff04d4b430>
```

3) 신경망 회귀모형의 예측 MSE

```
In [24]: # 회귀 모델 평가
         from sklearn.metrics import mean_squared_error
         y_pred = model_sigmoid.predict(x_test)
         mse = mean_squared_error(y_test, y_pred)
         rmse = np.sqrt(mse)
         print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mse , rmse))

         3/3 [==============================] - 0s 2ms/step
         MSE : 26668.096 , RMSE : 163.304
```

4) 결과해석

1)에서 활성함수를 'sigmoid'로 하고, 5개의 층(입력층: 1개, 은닉층: 3개, 출력층: 1개)을 가지는 회귀모형을 생성하였다.

생성한 모형을 토대로 train 데이터의 80%(validation_split=0.2)를 40개씩 소그룹으로 100번(epochs=100) 학습하였다.

따라서 학습된 신경망 회귀모형에 대한 예측 MSE는 26668.096, 예측 RMSE는 163.304로 나타났다. 예측 MSE의 값이 다른 모형에 비해 큰 것을 통해 활성함수가 예측에 적합하지 않은 것으로 판단된다.

5. 모든 모형의 예측 MSE 비교

1) 예측 MSE

1. 다중 회귀모형의 예측 MSE

```
In [8]: # Linear Regression OLS로 학습/예측평가 수행.
        reg= LinearRegression()
        reg.fit(x_train ,y_train)
        y_preds= reg.predict(x_test)
        mse = mean_squared_error(y_test, y_preds)
        rmse = np.sqrt(mse)

In [9]: # MSE, RMSE
        print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mse , rmse))

        MSE : 2929.887 , RMSE : 54.128
```

2. 주성분 회귀모형의 예측 MSE

```
In [19]: # Linear Regression OLS로 학습/예측평가 수행.
         from sklearn.metrics import mean_squared_error , r2_score
         y_preds= fit2.predict(pca_xtest)
         mse = mean_squared_error(y_test, y_preds)
         rmse = np.sqrt(mse)
         # MSE, RMSE, R^2, 설명분산점수(적합모형의 편향성 평가 : 0에 가까워야)
         print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mse , rmse))

         MSE : 3104.153 , RMSE : 55.715
```

3. 신경망 회귀모형(활성함수 - reiu)의 예측 MSE

```
In [21]: # 회귀 모델 평가
         from sklearn.metrics import mean_squared_error
         y_pred = model_reiu.predict(x_test)
         mse = mean_squared_error(y_test, y_pred)
         rmse = np.sqrt(mse)
         print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mse , rmse))

         3/3 [==============================] - 0s 2ms/step
         MSE : 3369.631 , RMSE : 58.049
```

4. 신경망 회귀모형(활성함수 - sigmoid)의 예측 MSE

```
In [24]: # 회귀 모델 평가
         from sklearn.metrics import mean_squared_error
         y_pred = model_sigmoid.predict(x_test)
         mse = mean_squared_error(y_test, y_pred)
         rmse = np.sqrt(mse)
         print('MSE : {0:.3f} , RMSE : {1:.3F}'.format(mse , rmse))

         3/3 [==============================] - 0s 2ms/step
         MSE : 26668.096 , RMSE : 163.304
```

2) 결과해석

사용한 모든 모형의 예측 MSE를 비교한 결과, 2929.87로 다중 회귀모형의 예측 MSE 값이 가장 작음을 확인할 수 있었다. 이를 통해 다중 회귀모형이 새로운 데이터에 대해 예측을 가장 잘 수행한다고 판단할 수 있을 것이다. 이어서 예측 MSE의 크기가 작은 순서대로, 주성분 회귀모형, 신경망 회귀모형(활성함수 - reiu), 신경망 회귀모형(활성함수 - sigmoid)이 예측을 잘 수행하는 것으로 보인다.

추가로 신경망 회귀모형에 대하여 반복(학습) 횟수(epochs)를 크게 할수록 예측 MSE가 작아지는 형태를 발견할 수 있는데, 이를 통해 신경망 회귀모형의 반복 횟수를 충분히 크게 하면 소그룹을 통한 학습이 더 잘 이루어져 위의 모형 중에서 예측을 가장 잘 수행하는 예도 나타날 수 있을 것으로 생각된다.

그러나 반대로 반복 횟수를 너무 늘리게 되면, train 데이터에 대하여 과적합이 이루어져 train 데이터에 대한 MSE는 크게 감소하지만, 오히려 예측 MSE는 증가하는 형태도 나타날 수 있을 것으로 생각된다.

* credit 보고서(default(target 데이터) 분류)

1. 데이터 변환

1.1 credit 데이터

1) credit 데이터

```
In [1]: #한글폰트 파일실행
        !python han-font.py
        exec(open('han-font.py').read())
        import numpy as np
        import pandas as pd

        credit=pd.read_csv("credit.csv", encoding="CP949")
        credit.head()
```

Out[1]:

	checking_balance	months_loan_duration	credit_history	amount	savings_balance	employment_duration	percent_of_income	years_at_residence	age	I
0	< 0 DM	6	critical	1169	unknown	> 7 years	4	4	67	
1	1 - 200 DM	48	good	5951	< 100 DM	1 - 4 years	2	2	22	
2	unknown	12	critical	2096	< 100 DM	4 - 7 years	2	3	49	
3	< 0 DM	42	good	7882	< 100 DM	4 - 7 years	2	4	45	
4	< 0 DM	24	poor	4870	< 100 DM	1 - 4 years	3	4	53	

```
In [2]: print('credit 데이터셋 크기: ', credit.shape)
        credit.info()
```

```
credit 데이터셋 크기:  (1000, 13)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1000 entries, 0 to 999
Data columns (total 13 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   checking_balance      1000 non-null   object
 1   months_loan_duration  1000 non-null   int64
 2   credit_history        1000 non-null   object
 3   amount                1000 non-null   int64
 4   savings_balance       1000 non-null   object
 5   employment_duration   1000 non-null   object
 6   percent_of_income     1000 non-null   int64
 7   years_at_residence    1000 non-null   int64
 8   age                   1000 non-null   int64
 9   housing               1000 non-null   object
 10  existing_loans_count  1000 non-null   int64
 11  job                   1000 non-null   object
 12  default               1000 non-null   object
dtypes: int64(6), object(7)
memory usage: 101.7+ KB
```

2) 결과해석

credit 데이터의 크기는 1000개임을 알 수 있다.
또한 12개의 독립변수와 target 데이터인 default가 종속변수로 존재함을 확인할 수 있다. 이들 중 7개는 범주형 자료이고, 나머지는 수치형 자료임을 확인할 수 있다. 범주형 자료에 대한 자세한 설명은 아래의 표를 통해 확인할 수 있다.

*범주형 자료(default 변수는 yes, no로 구성)

범주형 자료	checking_balance	credit_history	saving_balance	employment_duration	housing	job
범주	1. unknown	1. critical	1. unknown	1. <1year	1. own	1. skilled
	2. 0<DM	2. poor	2. <100DM	2. 1-4years	2. rent	2. unskilled
	3. 1-200DM	3. good	3. 100-500DM	3. 4-7years	3. other	3. management
	4. >200DM	4. very good	4. 500-1000DM	4. >7years		4. unemployed
		5. perfect	5. >1000DM			

1.2 범주형 변수를 더미변수로 변환, train-test 분류, 독립변수 표준화

1) 범주형 변수를 더미변수로 변환

In [3]:
```
# 질적변수를 더미변수로 변환
# 질적변수 -> 더미변수
credit_data1 = pd.get_dummies(credit.loc[:,['checking_balance', 'credit_history', 'savings_balance',
                                            'employment_duration', 'housing', 'job']])
# 양적변수 & target
credit_data2=credit.loc[:, ['months_loan_duration', 'amount', 'percent_of_income',
                            'years_at_residence', 'age', 'existing_loans_count', 'default']]
# 분석에 사용되는 데이터셋
credit_data=credit_data1.join(credit_data2)
credit_data.head()
```

Out[3]:

	checking_balance_1 - 200 DM	checking_balance_< 0 DM	checking_balance_> 200 DM	checking_balance_unknown	credit_history_critical	credit_history_good	credit_history_per
0	0	1	0	0	1	0	
1	1	0	0	0	0	1	
2	0	0	0	1	1	0	
3	0	1	0	0	0	1	
4	0	1	0	0	0	0	

5 rows × 33 columns

2) train-test 자료 분류

In [4]:
```
# 독립변수(수치형 자료)와 종속변수(target)
# 독립변수 6개 이름과 회귀계수 값을 매칭시켜 출력(zip - 매칭하여 묶어줌)
x_data=credit_data.iloc[:,:-1]
y_data=credit_data["default"]
```

In [5]:
```
#훈련용 데이터와 평가용 데이터 분할하기
from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(x_data, y_data, test_size = 0.2, random_state = 0)
```

3) 독립변수(x) 표준화

In [6]:
```
# x_train과 x_test를 StandardScaler를 이용해 정규화
from sklearn.preprocessing import StandardScaler
scaler= StandardScaler()
scaler.fit(x_train)
x_train = scaler.transform(x_train)
x_test = scaler.transform(x_test)
x_train=pd.DataFrame(x_train, columns=x_data.columns)
x_test=pd.DataFrame(x_test, columns=x_data.columns)
x_train.head()
```

Out[6]:

	checking_balance_1 - 200 DM	checking_balance_< 0 DM	checking_balance_> 200 DM	checking_balance_unknown	credit_history_critical	credit_history_good	credit_history_per
0	1.654786	-0.619744	-0.260942	-0.801692	-0.652706	-1.048684	-0.210
1	-0.604308	1.613569	-0.260942	-0.801692	-0.652706	0.953576	-0.210
2	1.654786	-0.619744	-0.260942	-0.801692	-0.652706	0.953576	-0.210
3	1.654786	-0.619744	-0.260942	-0.801692	-0.652706	-1.048684	-0.210
4	-0.604308	-0.619744	-0.260942	1.247361	1.532084	-1.048684	-0.210

5 rows × 32 columns

4) 결과해석

1)에서 분류분석(로지스틱 회귀, SVM, 판별분석, 신경망)에서 사용하기 위해 범주형 자료를 가변수를 이용하여 변경하였음을 확인할 수 있다.

변형된 범주형 데이터와 기존의 수치형 데이터를 교차검증을 위해 train-test의 자료로 분류하였는데, 그 비율을 8:2로 하여 2)에서 분류하였음을 확인할 수 있다.

다음으로 분류분석에서 사용되기 위해, 3)에서 독립변수인 x 데이터(x_train: train 데이터, x_test: test 데이터)에 대하여 표준화를 시행하였음을 알 수 있다. 그뿐만 아니라 train 데이터의 평균, 분산을 기준으로 train, test 데이터의 표준화가 이루어진 것도 볼 수 있다.

2. 로지스틱 회귀분석 및 분류정확도

2.1 로지스틱 회귀분석 결과

1) 로지스틱 회귀모형 생성

In [7]:
```
#1. 로지스틱 회귀
#로지스틱 회귀 분석: (1) 모델 생성
from sklearn.linear_model import LogisticRegression
credit_LR = LogisticRegression()
```

2.2 로지스틱 회귀모형의 오차행렬 및 분류정확도, 정밀도, 재현성

1) test 데이터에 대한 예측 및 오차행렬

In [10]:
```
#로지스틱 회귀 분석: (3) 평가자료에 대한 예측 수행 -> 예측 결과 Y_predict 구하기
from sklearn.metrics import confusion_matrix, accuracy_score
from sklearn.metrics import precision_score, recall_score
y_pred = credit_LR.predict(x_test)
print(confusion_matrix(y_test, y_pred))
```

```
[[123  19]
 [ 32  26]]
```

2) 예측에 대한 (분류)정확도, 정밀도, 재현성

In [11]:
```
# 정확도, 정밀도, 재현성 -> 정확도 중요
acccuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, pos_label="yes")
recall = recall_score(y_test, y_pred, pos_label="yes")
print('정확도: {0:.3f}, 정밀도: {1:.3f}, 재현율: {2:.3f}'.format(acccuracy,precision,recall))
# 모형평가(분류정확도_보고서) 종합정리
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
정확도: 0.745, 정밀도: 0.578, 재현율: 0.448
              precision    recall  f1-score   support

          no       0.79      0.87      0.83       142
         yes       0.58      0.45      0.50        58

    accuracy                           0.74       200
   macro avg       0.69      0.66      0.67       200
weighted avg       0.73      0.74      0.73       200
```

3) 결과해석

1)에서 수행한 예측에 대하여, 오차 행렬을 확인할 수 있다. 전체 데이터 200개 중 데이터를 실제 값과 동일하게 잘 예측한 데이터는 149개로 (분류)정확도는 74%(149/200) 정도 된다는 것을 알 수 있다.

또한 'no'에 대한 정밀도(precision)가 0.79, 'yes'에 대한 정밀도(precision)가 0.58로 예측한 것 중 정답의 비율이 각각 79%, 58% 정도로 나타났다고 볼 수 있다.

재현율에 대해선 'no', 'yes' 각각 0.87, 0.45로 실제 목표로 찾아야 하는 데이터를 정답의 비율이 각각 87%, 45%임을 확인할 수 있다.

정밀도와 재현율은 높을수록 좋으므로, 정밀도와 재현율의 평균인 f1-score가 높을수록 성능이 좋다고 할 수 있을 것이다. 위의 표에서 'no'에 대한 f1-score가 0.83으로 가장 높으므로, 따라서 위의 모형 default의 범주인 'no'를 분류하는 성능이 가장 좋다고 볼 수 있다.

이것은 대출받은 사람에게 돈을 돌려받아야 하는 은행의 입장에서 좋은 모형이라고 할 수 있을 것으로 판단된다.

3. SVM 분류 및 분류정확도

3.1 의미있는 양적 변수 선택

1) 양적 변수 선택

```
In [20]:
# svm - 의미있는 양적자료 사용( )
# 'months_loan_duration', 'amount', 'percent_of_income', 'age', 'existing_loans_count
numeric_xtrain=x_train.loc[:, ['months_loan_duration', 'amount', 'percent_of_income',
                               'age', 'existing_loans_count']]
numeric_xtest=x_test.loc[:, ['months_loan_duration', 'amount', 'percent_of_income',
                             'age', 'existing_loans_count']]
numeric_xtrain.head()
```

Out[20]:

	months_loan_duration	amount	percent_of_income	age	existing_loans_count
0	1.266142	-0.133708	0.903680	-0.508707	-0.715967
1	0.270242	-0.041477	0.903680	-0.767043	-0.715967
2	3.257942	1.472742	0.903680	-1.025380	-0.715967
3	-0.476683	-0.698405	-0.896927	-0.939268	-0.715967
4	-1.223608	-0.595925	-1.797231	-1.025380	1.041007

2) 결과해석

SVM 분류는 의미있다고 판단되는 양적 자료만을 이용하여 시행할 것이다.
따라서 6개의 양적 변수 중 'months_loan_duration'(대출기간), 'amount'(전체 빚), 'percent_of_income'(수입 백분율), 'age'(나이), 'existing_loans_count'(기존 대출 건수)를 독립변수로 선택하였다.

3.2 SVM(커널함수 - linear) 분류 결과 및 분류 정확도

1) test 데이터에 대한 예측 및 오차행렬

```
In [22]:
# SVM - 분류정확도 분석(선형)
svclassifier = SVC(kernel='linear')
svclassifier.fit(numeric_xtrain, y_train)
y_pred = svclassifier.predict(numeric_xtest)
print('score : ',svclassifier.score(numeric_xtest,y_test))
print(confusion_matrix(y_test, y_pred))
```

```
score :  0.71
[[142   0]
 [ 58   0]]
```

2) 예측에 대한 (분류)정확도, 정밀도, 재현성

```
In [23]:
# 정확도, 정밀도, 재현성 -> 정확도 중요
acccuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, pos_label="yes")
recall = recall_score(y_test, y_pred, pos_label="yes")
print('정확도: {0:.3f}, 정밀도: {1:.3f}, 재현율: {2:.3f}'.format(acccuracy,precision,recall))
# 모형평가(분류정확도_보고서) 종합정리
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
정확도: 0.710, 정밀도: 0.000, 재현율: 0.000
              precision    recall  f1-score   support

          no       0.71      1.00      0.83       142
         yes       0.00      0.00      0.00        58

    accuracy                           0.71       200
   macro avg       0.35      0.50      0.42       200
weighted avg       0.50      0.71      0.59       200
```

3) 결과해석

1)에서 수행한 예측에 대하여, 오차 행렬을 확인할 수 있다. 전체 데이터 200개 중 데이터를 실제 값과 동일하게 잘 예측한 데이터는 142개로 (분류)정확도는 71%(142/200) 정도 된다는 것을 알 수 있다.
그러나 모든 자료가 'no'로 분류되었음을 확인할 수 있는데, 이러한 결과가 나타나는 이유는 독립변수가 데이터를 분류하는데 적절하지 않았거나, 커널함수가 분류에 적합하지 않았기 때문이라고 생각한다.
따라서 모든 자료를 'no'로 분류하여 나타내기 때문에 분류정확도 또한 신뢰성이 높다고 판단할 수 없을 것이다.

3.3 SVM(커널함수 - 다항식) 분류 결과 및 분류 정확도

1) test 데이터에 대한 예측 및 오차행렬

In [24]:
```
# SVM - 커널함수= 다항식(poly)
svclassifier = SVC(kernel='poly', degree=8)
svclassifier.fit(numeric_xtrain, y_train)
y_pred = svclassifier.predict(numeric_xtest)
print('score : ',svclassifier.score(numeric_xtest,y_test))
print(confusion_matrix(y_test, y_pred))
```

```
score :  0.7
[[131  11]
 [ 49   9]]
```

2) 예측에 대한 (분류)정확도, 정밀도, 재현성

In [25]:
```
# 정확도, 정밀도, 재현성 -> 정확도 중요
acccuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, pos_label="yes")
recall = recall_score(y_test, y_pred, pos_label="yes")
print('정확도: {0:.3f}, 정밀도: {1:.3f}, 재현율: {2:.3f}'.format(acccuracy,precision,recall))
# 모형평가(분류정확도_보고서) 종합정리
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
정확도: 0.700, 정밀도: 0.450, 재현율: 0.155
              precision    recall  f1-score   support

          no       0.73      0.92      0.81       142
         yes       0.45      0.16      0.23        58

    accuracy                           0.70       200
   macro avg       0.59      0.54      0.52       200
weighted avg       0.65      0.70      0.64       200
```

3) 결과해석

1)에서 수행한 예측에 대하여, 오차 행렬을 확인할 수 있다. 전체 데이터 200개 중 데이터를 실제 값과 동일하게 잘 예측한 데이터는 140개로 (분류)정확도는 70%(140/200) 정도 된다는 것을 알 수 있다.

또한 'no'에 대한 정밀도(precision)가 0.73, 'yes'에 대한 정밀도(precision)가 0.45로 예측한 것 중 정답의 비율이 각각 73%, 45% 정도로 나타났다고 볼 수 있다.

재현율에 대해선 'no', 'yes' 각각 0.92, 0.16로 실제 목표로 찾아야 하는 데이터를 정답의 비율이 각각 92%, 16%임을 확인할 수 있다.

위의 표에서 'no'에 대한 f1-score가 0.81로 가장 높으므로, 따라서 위의 모형이 default의 범주인 'no'를 분류하는 성능이 가장 좋다고 볼 수 있다. 그러므로 대출받은 사람에게 돈을 돌려받아야 하는 은행의 입장에서 좋은 모형이라고 할 수 있을 것으로 판단된다.

3.4 SVM(커널함수 - 가우시안) 분류 결과 및 분류 정확도

1) test 데이터에 대한 예측 및 오차행렬

In [26]:
```
# SVM - 커널함수= 가우시안(정규분포함수)
svclassifier = SVC(kernel='rbf')
svclassifier.fit(numeric_xtrain, y_train)
y_pred = svclassifier.predict(numeric_xtest)
print('score : ',svclassifier.score(numeric_xtest,y_test))
print(confusion_matrix(y_test, y_pred))
```

```
score :  0.72
[[137   5]
 [ 51   7]]
```

2) 예측에 대한 (분류)정확도, 정밀도, 재현성

In [27]:
```
# 정확도, 정밀도, 재현성 -> 정확도 중요
acccuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, pos_label="yes")
recall = recall_score(y_test, y_pred, pos_label="yes")
print('정확도: {0:.3f}, 정밀도: {1:.3f}, 재현율: {2:.3f}'.format(acccuracy,precision,recall))
# 모형평가(분류정확도_보고서) 종합정리
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
정확도: 0.720, 정밀도: 0.583, 재현율: 0.121
              precision    recall  f1-score   support

          no       0.73      0.96      0.83       142
         yes       0.58      0.12      0.20        58

    accuracy                           0.72       200
   macro avg       0.66      0.54      0.52       200
weighted avg       0.69      0.72      0.65       200
```

3) 결과해석

1)에서 수행한 예측에 대하여, 오차 행렬을 확인할 수 있다. 전체 데이터 200개 중 데이터를 실제 값과 동일하게 잘 예측한 데이터는 144개로 (분류)정확도는 72%(144/200) 정도 된다는 것을 알 수 있다.
또한 'no'에 대한 정밀도(precision)가 0.73, 'yes'에 대한 정밀도(precision)가 0.58로 예측한 것 중 정답의 비율이 각각 73%, 58% 정도로 나타났다고 볼 수 있다.
재현율에 대해선 'no', 'yes' 각각 0.96, 0.12로 실제 목표로 찾아야 하는 데이터를 정답의 비율이 각각 96%, 12%임을 확인할 수 있다.
위의 표에서 'no'에 대한 f1-score가 0.83으로 가장 높으므로, 따라서 위의 모형이 default의 범주인 'no'를 분류하는 성능이 가장 좋다고 볼 수 있다. 그러므로 대출받은 사람에게 돈을 돌려받아야 하는 은행의 입장에서 좋은 모형이라고 할 수 있을 것으로 판단된다.

3.4 SVM(커널함수 - 시그모이드) 분류 결과 및 분류 정확도

1) test 데이터에 대한 예측 및 오차행렬

```
In [28]:
# SVM - 커널함수= sigmoid
svclassifier = SVC(kernel='sigmoid')
svclassifier.fit(numeric_xtrain, y_train)
y_pred = svclassifier.predict(numeric_xtest)
print('score : ',svclassifier.score(numeric_xtest,y_test))
print(confusion_matrix(y_test, y_pred))
```

```
score :  0.655
[[115  27]
 [ 42  16]]
```

2) 예측에 대한 (분류)정확도, 정밀도, 재현성

```
In [29]:
# 정확도, 정밀도, 재현성 -> 정확도 중요
acccuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, pos_label="yes")
recall = recall_score(y_test, y_pred, pos_label="yes")
print('정확도: {0:.3f}, 정밀도: {1:.3f}, 재현율: {2:.3f}'.format(acccuracy,precision,recall))
# 모형평가(분류정확도_보고서) 종합정리
from sklearn.metrics import classification_report
print(classification_report(y_test, y_pred))
```

```
정확도: 0.655, 정밀도: 0.372, 재현율: 0.276
              precision    recall  f1-score   support

          no       0.73      0.81      0.77       142
         yes       0.37      0.28      0.32        58

    accuracy                           0.66       200
   macro avg       0.55      0.54      0.54       200
weighted avg       0.63      0.66      0.64       200
```

3. 결과해석

1)에서 수행한 예측에 대하여, 오차 행렬을 확인할 수 있다. 전체 데이터 200개 중 데이터를 실제 값과 동일하게 잘 예측한 데이터는 131개로 (분류)정확도는 65.5%(131/200) 정도 된다는 것을 알 수 있다.
또한 'no'에 대한 정밀도(precision)가 0.73, 'yes'에 대한 정밀도(precision)가 0.37로 예측한 것 중 정답의 비율이 각각 73%, 37% 정도로 나타났다고 볼 수 있다.
재현율에 대해선 'no', 'yes' 각각 0.81, 0.28로 실제 목표로 찾아야 하는 데이터를 정답의 비율이 각각 81%, 28%임을 확인할 수 있다.
위의 표에서 'no'에 대한 f1-score가 0.77로 가장 높으므로, 따라서 위의 모형이 default의 범주인 'no'를 분류하는 성능이 가장 좋다고 볼 수 있다. 그러므로 대출받은 사람에게 돈을 돌려받아야 하는 은행의 입장에서 좋은 모형이라고 할 수 있을 것으로 판단된다.

4. 판별분석 및 분류정확도

1) 판별분석 모형 생성

```
In [22]: # 3. 판별분석(DA)
         import pandas as pd
         import numpy as np
         import matplotlib.pyplot as plt
         import seaborn as sns
         from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
         from sklearn.preprocessing import StandardScaler
         from sklearn.metrics import confusion_matrix

In [23]: # 전체 자료에 대한 판별분석
         lda = LinearDiscriminantAnalysis()
         lda.fit(x_train, y_train)

Out[23]: LinearDiscriminantAnalysis()
```

2) test 데이터에 대한 예측 및 오차행렬

```
In [36]: y_pred = mlp.predict(x_test)
         print(confusion_matrix(y_test, y_pred))

[[109  33]
 [ 30  28]]
```

3) 예측에 대한 (분류)정확도, 정밀도, 재현성

```
In [37]: # 정확도, 정밀도, 재현성 -> 정확도 중요
         acccuracy = accuracy_score(y_test, y_pred)
         precision = precision_score(y_test, y_pred, pos_label="yes")
         recall = recall_score(y_test, y_pred, pos_label="yes")
         print('정확도: {0:.3f}, 정밀도: {1:.3f}, 재현율: {2:.3f}'.format(acccuracy,precision,recall))
         # classification_report를 이용해 정확률, 재현율 등을 확인
         print(classification_report(y_test, y_pred))

정확도: 0.685, 정밀도: 0.459, 재현율: 0.483
              precision    recall  f1-score   support

          no       0.78      0.77      0.78       142
         yes       0.46      0.48      0.47        58

    accuracy                           0.69       200
   macro avg       0.62      0.63      0.62       200
weighted avg       0.69      0.69      0.69       200
```

4) 결과해석

1)에서 선형 판별분석 모형을 생성하였다.

다음으로 2)에서 수행한 예측에 대하여, 오차 행렬을 확인할 수 있다. 전체 데이터 200개 중 데이터를 실제 값과 동일하게 잘 예측한 데이터는 148개로 (분류)정확도는 0.74%(148/200) 정도 된다는 것을 알 수 있다.

또한 'no'에 대한 정밀도(precision)가 0.78, 'yes'에 대한 정밀도(precision)가 0.57로 예측한 것 중 정답의 비율이 각각 78%, 57% 정도로 나타났다고 볼 수 있다.

재현율에 대해선 'no', 'yes' 각각 0.87, 0.41로 실제 목표로 찾아야 하는 데이터를 정답의 비율이 각각 87%, 41%임을 확인할 수 있다.

위의 표에서 'no'에 대한 f1-score가 0.83으로 가장 높으므로, 따라서 위의 모형이 default의 범주인 'no'를 분류하는 성능이 가장 좋다고 볼 수 있다. 그러므로 대출받은 사람에게 돈을 돌려받아야 하는 은행의 입장에서 좋은 모형이라고 할 수 있을 것으로 판단된다.

5. 신경망 분류 및 분류정확도

1) 신경망 모형 생성

```
In [35]: # 4. 신경망
         # 다층인공신경망(MLP) 분류 알고리즘을 sklearn의 neural_network에서 로드
         from sklearn.neural_network import MLPClassifier
         # MLP 알고리즘의 히든레이어를 3계층(10,10,10)으로 할당
         mlp = MLPClassifier(hidden_layer_sizes=(10,10,10))

In [36]: # 위에서 분류한 X_train과 y_train을 MLP를 이용해 학습
         mlp.fit(x_train, y_train)

C:\Users\gksxk\anaconda3\lib\site-packages\sklearn\neural_network\_multilayer_perceptron.py:692: ConvergenceWarning: Stochastic Opt
imizer: Maximum iterations (200) reached and the optimization hasn't converged yet.
  warnings.warn(

Out[36]: MLPClassifier(hidden_layer_sizes=(10, 10, 10))
```

2) test 데이터에 대한 예측 및 오차행렬

```
In [36]: y_pred = mlp.predict(x_test)
         print(confusion_matrix(y_test, y_pred))

[[109  33]
 [ 30  28]]
```

3) 예측에 대한 (분류)정확도, 정밀도, 재현성

```
In [37]: # 정확도, 정밀도, 재현성 -> 정확도 중요
         acccuracy = accuracy_score(y_test, y_pred)
         precision = precision_score(y_test, y_pred, pos_label="yes")
         recall = recall_score(y_test, y_pred, pos_label="yes")
         print('정확도: {0:.3f}, 정밀도: {1:.3f}, 재현율: {2:.3f}'.format(acccuracy,precision,recall))
         # classification_report를 이용해 정확률, 재현율 등을 확인
         print(classification_report(y_test, y_pred))

정확도: 0.685, 정밀도: 0.459, 재현율: 0.483
              precision    recall  f1-score   support

          no       0.78      0.77      0.78       142
         yes       0.46      0.48      0.47        58

    accuracy                           0.69       200
   macro avg       0.62      0.63      0.62       200
weighted avg       0.69      0.69      0.69       200
```

4) 결과해석

1)에서 3개의 은닉층으로 이루어졌으며 각 층에는 10개의 노드를 가지고 있는 신경망 모형을 생성하였다.

다음으로 2)에서 수행한 예측에 대하여, 오차 행렬을 확인할 수 있다. 전체 데이터 200개 중 데이터를 실제 값과 동일하게 잘 예측한 데이터는 137개로 (분류)정확도는 0.685%(137/200) 정도 된다는 것을 알 수 있다.

또한 'no'에 대한 정밀도(precision)가 0.78, 'yes'에 대한 정밀도(precision)가 0.46로 예측한 것 중 정답의 비율이 각각 78%, 46% 정도로 나타났다고 볼 수 있다.

재현율에 대해선 'no', 'yes' 각각 0.77, 0.48로 실제 목표로 찾아야 하는 데이터를 정답의 비율이 각각 77%, 48%임을 확인할 수 있다.

위의 표에서 'no'에 대한 f1-score가 0.78으로 가장 높으므로, 따라서 위의 모형이 default의 범주인 'no'를 분류하는 성능이 가장 좋다고 볼 수 있다. 그러므로 대출받은 사람에게 돈을 돌려받아야 하는 은행의 입장에서 좋은 모형이라고 할 수 있을 것으로 판단된다.

6. 모든 모형의 분류정확도 비교

1) 분류정확도

1. 로지스틱 회귀 분류정확도

```
In [56]: # 로지스틱 회귀 분류 정확도
         print( "score =", reg.score(x_test, y_test))
```

```
score = 0.745
```

2.1 SVM(커널함수 - 선형) 분류정확도

```
In [44]: print('score : ',svclassifier.score(numeric_xtest,y_test))
```

```
score :  0.71
```

2.2 SVM(커널함수 - 다항식) 분류정확도

```
In [46]: print('score : ',svclassifier.score(numeric_xtest,y_test))
```

```
score :  0.7
```

2.3 SVM(커널함수 - 가우시안) 분류정확도

```
In [48]: print('score : ',svclassifier.score(numeric_xtest,y_test))
```

```
score :  0.72
```

2.4 SVM(커널함수 - 시그모이드) 분류정확도

```
In [50]: print('score : ',svclassifier.score(numeric_xtest,y_test))
```

```
score :  0.655
```

3. 선형 판별분석 분류정확도

```
In [53]: print('score : ',lda.score(x_test,y_test))
```

```
score :  0.74
```

4. 신경망 분류정확도

```
In [55]: print('score : ',mlp.score(x_test,y_test))
```

```
score :  0.685
```

2) 결과해석

사용한 모든 모형의 분류정확도를 비교한 결과, 0.745로 로지스틱 회귀모형의 분류정확도 값이 가장 큼을 확인할 수 있었다. 이를 통해 로지스틱 회귀모형이 새로운 데이터에 대해 분류를 가장 잘 수행한다고 판단할 수 있을 것이다. 이어서 분류정확도의 크기가 큰 순서대로 선형 판별분석, SVM(커널함수 - 가우시안), SVM(커널함수 - 선형), SVM(커널함수 - 다항식), 신경망, SVM(커널함수 - 시그모이드)이 분류를 잘 수행하는 것으로 보인다.