Q3

April 18, 2024

In the previous question we observed some redundancy in the features. We would like to try some feature selection heuristic in this question. Consider the same dataset as question 2 (fat.csv), where **brozek** is the response variable and the other 17 columns are the model features. Follow this steps below.

– Form an extended version of the dataset, by appending two more columns. One column corresponding to $siri^2$ and one column corresponding to $\frac{1}{density}$. Your extended dataset should now have 20 columns, where the first column is brozek and used as the response variable, 17 columns identical to the original fat.csv data set, and columns 19 and 20 with the values $siri^2$ and $\frac{1}{density}$, respectively. density We will refer to this dataset as the *extended dataset*.

– In a similar way as question 2, split the extended dataset into two sets. Set 1 includes the first 200 rows of the data (do not count the row associated with the feature/response names), and set 2, which includes the last 52 rows of the data. Name the first set **train** and the second set **test**.

[1]:
```
import numpy as np
import pandas as pd
from matplotlib.pyplot import subplots
import statsmodels.api as sm
from ISLP import load_data
from ISLP.models import (ModelSpec as MS,
                         summarize)
```

[2]:
```
Fat = pd.read_csv("fat.csv")
train,test = np.split(Fat,[int(200)])
y = train['brozek']
X_train_full = pd.DataFrame({'intercept': np.ones(train.shape[0]),
                  'siri':         train['siri'],
                  'density':      train['density'],
                  'age':          train['age'],
                  'weight':       train['weight'],
                  'height':       train['height'],
                  'adipos':       train['adipos'],
                  'free':         train['free'],
                  'neck':         train['neck'],
                  'chest':        train['chest'],
                  'abdom':        train['abdom'],
                  'hip':          train['hip'],
                  'thigh':        train['thigh'],
```

```
                'knee':         train['knee'],
                'ankle':        train['ankle'],
                'biceps':       train['biceps'],
                'forearm':      train['forearm'],
                'wrist':        train['wrist'],
                'siri_squared': train['siri'] ** 2,
                'inv_density':   1/train['density']})
```

(a) Use the training data to fit a model of the following form brozek = $\beta_0 + \beta_1$ siri + ... + β_{17} wrist + β_{18} $siri^2$ + β_{19} $\frac{1}{density}$ report the fitted parameters, the 95% confidence interval for each estimated parameter and the p-values. What is the R2 value?

[3]:
```
fullFittedModel = sm.OLS(y, X_train_full).fit()
fullFittedModel.summary()
```

[3]:

Dep. Variable:	brozek	**R-squared:**	0.999
Model:	OLS	**Adj. R-squared:**	0.999
Method:	Least Squares	**F-statistic:**	1.703e+04
Date:	Thu, 18 Apr 2024	**Prob (F-statistic):**	6.09e-282
Time:	18:33:31	**Log-Likelihood:**	64.717
No. Observations:	200	**AIC:**	-89.43
Df Residuals:	180	**BIC:**	-23.47
Df Model:	19		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P>|t|**	**[0.025**	**0.975]**
intercept	-950.8057	580.610	-1.638	0.103	-2096.483	194.871
siri	0.9305	0.029	32.063	0.000	0.873	0.988
density	436.7337	269.081	1.623	0.106	-94.225	967.693
age	-0.0007	0.002	-0.414	0.680	-0.004	0.003
weight	0.0162	0.006	2.809	0.006	0.005	0.028
height	-0.0005	0.005	-0.096	0.924	-0.010	0.009
adipos	-0.0235	0.016	-1.505	0.134	-0.054	0.007
free	-0.0204	0.007	-2.780	0.006	-0.035	-0.006
neck	-0.0018	0.011	-0.163	0.870	-0.024	0.020
chest	0.0056	0.005	1.026	0.306	-0.005	0.016
abdom	-0.0006	0.005	-0.109	0.913	-0.011	0.010
hip	0.0008	0.008	0.106	0.916	-0.014	0.016
thigh	0.0174	0.008	2.264	0.025	0.002	0.033
knee	-0.0290	0.013	-2.296	0.023	-0.054	-0.004
ankle	0.0061	0.010	0.591	0.555	-0.014	0.026
biceps	-0.0169	0.008	-2.011	0.046	-0.034	-0.000
forearm	0.0219	0.010	2.110	0.036	0.001	0.042
wrist	0.0343	0.027	1.270	0.206	-0.019	0.088
siri_squared	-0.0026	0.001	-1.956	0.052	-0.005	2.26e-05
inv_density	518.6225	312.993	1.657	0.099	-98.985	1136.230

Omnibus:	98.592	**Durbin-Watson:**	1.907
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	6766.061
Skew:	0.902	**Prob(JB):**	0.00
Kurtosis:	31.437	**Cond. No.**	3.19e+07

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[2] The condition number is large, 3.19e+07. This might indicate that there are strong multicollinearity or other numerical problems.

[4]:
```
X_test_full = pd.DataFrame({'intercept': np.ones(test.shape[0]),
                  'siri':    test['siri'],
                  'density': test['density'],
                  'age':     test['age'],
                  'weight':  test['weight'],
                  'height':  test['height'],
                  'adipos':  test['adipos'],
                  'free':    test['free'],
                  'neck':    test['neck'],
                  'chest':   test['chest'],
                  'abdom':   test['abdom'],
                  'hip':     test['hip'],
                  'thigh':   test['thigh'],
                  'knee':    test['knee'],
                  'ankle':   test['ankle'],
                  'biceps':  test['biceps'],
                  'forearm': test['forearm'],
                  'wrist':   test['wrist'],
                  'siri_squared': test['siri'] ** 2,
                  'inv_density':   1/test['density']})
```

(b) Use the test data to calculate the test error (similar to the formulation in part (c) of the previous question), and call it e_{full}.

[5]:
```
# Get prediction error by model
def get_prediction_error(model, X_pred):
    # the actual value from test dataset
    y_actual = np.array(test['brozek'].values)
    # the value predicted by the full-features model
    y_prediction = np.array(model.predict(X_pred))
    return np.sqrt(sum(np.square(y_actual - y_prediction)))
```

[6]:
```
# Calculate the prediction error of full-featured model
print('Prediction Error e_full =', get_prediction_error(fullFittedModel,␣
 ↪X_test_full))
```

```
Prediction Error e_full = 0.8565466791992765
```

(c) Let's run a heuristic scheme to perform feature selection (the method is called backward selection and described on page 79 of your textbook, also on the slides). Start with the full model (the model containing all 19 features of the extended dataset) and drop the feature with the highest p-value (or the second largest if the largest p-value is for the intercept), then redo the modeling and drop the next feature with the highest p-value, and continue dropping until all p-values are small and you are left with a set of important features. Implement this approach and stop when all p-values are below 0.03. Which features are selected as the most important ones when your code stops?

[7]:
```
# Extract the problematic features that has p-value greater than 0.03, among␣
↪all the p-values.
def get_problematic_pValues(model):
    pv = model.pvalues
    return pv[pv > 0.03]
```

[8]:
```
# Extract the problematic features
pv_problematic = get_problematic_pValues(fullFittedModel)
# print(pv_problematic,"\n")

# Training set for iteration
X_train_new = X_train_full

# Run the loop if there is any p-value greater than 0.03
while (pv_problematic > 0.03).any():

    # Find the most problematic feature
    fea_maxP = pv_problematic.idxmax()
    # print(fea_maxP,": ", pv_problematic[fea_maxP])

    # Drop that feature and get new training data
    X_train_new = X_train_new.drop(fea_maxP, axis = 1)

    # Retrain the model with new data
    newFittedModel = sm.OLS(y, X_train_new).fit()

    # Get new p-valuses from new model
    pv_problematic = get_problematic_pValues(newFittedModel)

newFittedModel.summary()
```

[8]:

Dep. Variable:	brozek	**R-squared (uncentered):**	1.000
Model:	OLS	**Adj. R-squared (uncentered):**	1.000
Method:	Least Squares	**F-statistic:**	5.569e+05
Date:	Thu, 18 Apr 2024	**Prob (F-statistic):**	0.00
Time:	18:33:31	**Log-Likelihood:**	51.369
No. Observations:	200	**AIC:**	-94.74
Df Residuals:	196	**BIC:**	-81.55
Df Model:	4		
Covariance Type:	nonrobust		

	coef	**std err**	**t**	**P> |t|**	**[0.025**	**0.975]**
siri	0.9222	0.002	474.600	0.000	0.918	0.926
thigh	0.0142	0.005	3.101	0.002	0.005	0.023
knee	-0.0262	0.010	-2.596	0.010	-0.046	-0.006
inv_density	1.5192	0.267	5.698	0.000	0.993	2.045

Omnibus:	187.007	**Durbin-Watson:**	1.919
Prob(Omnibus):	0.000	**Jarque-Bera (JB):**	13874.811
Skew:	2.978	**Prob(JB):**	0.00
Kurtosis:	43.367	**Cond. No.**	1.47e+03

Notes:
[1] R^2 is computed without centering (uncentered) since the model does not contain a constant.
[2] Standard Errors assume that the covariance matrix of the errors is correctly specified.
[3] The condition number is large, 1.47e+03. This might indicate that there are strong multicollinearity or other numerical problems.

***siri*, *thigh*, *knee*, $\frac{1}{density}$ are the most important features after backward selection.**

(d) Apply the model developed in part (c) to the test data and call the error e_{sel}.

[9]:
```
# Feature variables from test set
X_test_sel = pd.DataFrame({'siri': test['siri'], 'thigh': test['thigh'], 'knee':
  ↪ test['knee'], 'inv_density': 1/test['density']})

# Calculate the prediction error of feature-selection model
print('Prediction Error e_sel =', get_prediction_error(newFittedModel,␣
  ↪X_test_sel))
```

```
Prediction Error e_sel = 0.6670935385386173
```

(e) Compare e_{full} and e_{sel}. Does the feature selection scheme seem to reduce overfitting?

Yes, it does reduce overfitting. By removing less important features, backward selection helps focus the model on the most relevant predictors. This reduces the complexity of the model and decrease the chance of fitting too close to training data.

(f) Compare e_{sel} with e_3 from part (h) of question 2. In terms of the test accuracy does your feature selection scheme seem to find the best model?

e_{sel} (***0.6670935385386173***) **is greater than** e_3 (***0.563361670359713***)**, therefore, the feature selection scheme seems not to get the best model in this case. The reason might be some useful features that seemed not important were removed too early. This can lead to a loss of valuable information that could be useful in conjunction with other features.**