Bihan Q

PCR, Stepwise, Model Comparison

Consider the Boston dataset, in R library MASS, on Housing Values in Suburbs of Boston, to fit a suitable model to predict medv (median value of owner-occupied homes in $1000s) using the following set of predictors:

- crim per capita crime rate by town.
- zn proportion of residential land zoned for lots over 25,000 sq.ft.
- indus proportion of non-retail business acres per town.
- nox nitrogen oxides concentration (parts per 10 million).
- rm average number of rooms per dwelling.
- age proportion of owner-occupied units built prior to 1940.
- tax full-value property-tax rate per $10,000.

Q1

Consider the problem of predicting predict ‘medv' using suitable variables.

1.Investigate whether there is any multicollinearity, and suggest remedial measures if appropriate.

In [72]:
```
# import
import statsmodels.api as sm
import statsmodels.stats.api as sms
import pandas as pd
from sklearn.linear_model import LinearRegression, Lasso
import scipy.stats as stats
import numpy as np
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error, r2_score
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
import matplotlib.pyplot as plt
```

In [73]:
```
# get data
Boston = sm.datasets.get_rdataset('Boston','MASS')
boston = pd.DataFrame(Boston.data)
boston.shape #enough number of data point to split train and test
```

Out[73]:
```
(506, 14)
```

In [83]:
```
# set up X and Y
X = boston[['crim', 'zn', 'indus', 'nox', 'rm', 'age', 'tax']]
```

```
y = boston['medv']
```

In [84]:

```
# VIF with 7
vif = pd.DataFrame()
vif["Variable"] = X.columns
vif["VIF"] = [variance_inflation_factor(X.values, i) for i in range(X.shape[1])]
print(vif)
# Condition number
X = sm.add_constant(X)
model = sm.OLS(y, X).fit()
X_matrix = model.model.exog
condition_number = np.linalg.cond(X_matrix)
print(f"condition number: {condition_number}")
```

```
  Variable        VIF
0     crim   1.807384
1       zn   2.068635
2    indus  12.603225
3      nox  65.943100
4       rm  29.690566
5      age  17.121147
6      tax  19.442239
condition number: 7723.055667833122
```

In [81]:

```
# VIF with all
X_all = boston[['crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax'
       'ptratio', 'black', 'lstat']]
vif = pd.DataFrame()
vif["Variable"] = X_all.columns
vif["VIF"] = [variance_inflation_factor(X_all.values, i) for i in range(X_all.shape[1]
print(vif)
# Condition number
X_all = sm.add_constant(X_all)
model_all = sm.OLS(y, X_all).fit()
X_matrix = model_all.model.exog
condition_number = np.linalg.cond(X_matrix)
print(f"condition number: {condition_number}")
```

```
   Variable        VIF
0      crim   2.100373
1        zn   2.844013
2     indus  14.485758
3      chas   1.152952
4       nox  73.894947
5        rm  77.948283
6       age  21.386850
7       dis  14.699652
8       rad  15.167725
9       tax  61.227274
10  ptratio  85.029547
11    black  20.104943
12    lstat  11.102025
condition number: 15113.517599134946
```

Answer

Investigation with 7 variables:

- Based on the condition number being greater than 1000 (specifically, 7723), the regression analysis suggests a significant issue with collinearity.
- Upon further examination using VIF for individual predictors, variables exhibit multicollinearity: ['indus', 'nox', 'rm', 'age', 'tax'].

Investigation with all variables:

- The condition number 15113 suggests a significant issue with collinearity.
- Upon further examination using VIF for individual predictors, besides 5 variables we mentioned above, newly added variables ['dis', 'rad', 'ptratio', 'black','lstat'] also exhibits multicollinearity. Meanwhile, ['chas'] is only one newly added variable without multicollinearity concerns.

Summary:

- multicollinearity: ['indus', 'nox', 'rm', 'age', 'tax'], ['dis', 'rad', 'ptratio', 'black', 'lstat'].
- without multicollinearity: ['crim', 'zn', 'chas'].

Remedies:

- PCA/weighted PCA to obtain uncorrelated predictors.
- Stepwise regression, Efroymson's method, and Exhaustive search.
- Lasso regresion, ridge regression, and elastic net by adding restrictions.
- Transformations including Box-Tidwell, Box-Cox, arcsin transformations.
- Nonlinear regression models like exponential models, GLM, or scatterplot smoothers when function form is unknown.

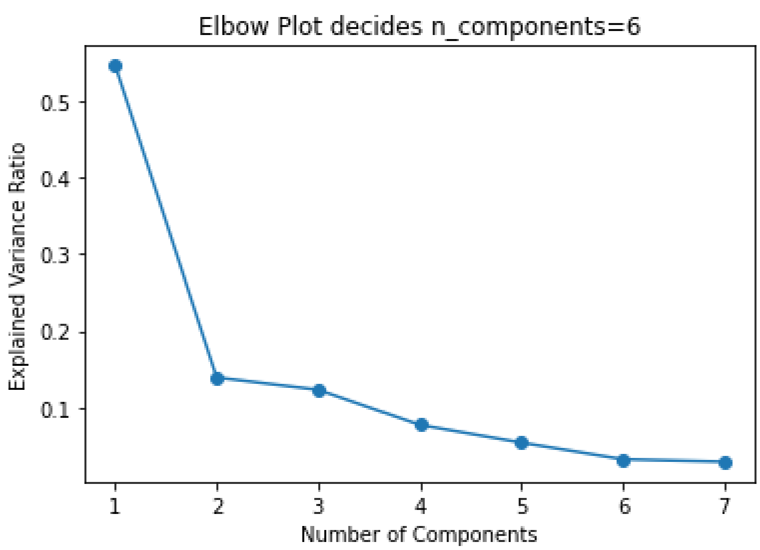
Q2

2.Compare results based on the usual linear regression and Principal Component Regression to predict ‘medv' using the available variables.

In [157…

```
# PC regression
X = boston[['crim', 'zn', 'indus', 'nox', 'rm', 'age', 'tax']]
y = boston['medv']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

# Standardize
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

pca = PCA()
X_train_pca = pca.fit_transform(X_train_scaled)
X_test_pca = pca.transform(X_test_scaled)

# Choose the number of principal components
explained_variance_ratio = pca.explained_variance_ratio_
cumulative_variance_ratio = np.cumsum(explained_variance_ratio)
```

```python
plt.plot(range(1, len(pca.explained_variance_ratio_) + 1), pca.explained_variance_rati
plt.xlabel('Number of Components')
plt.ylabel('Explained Variance Ratio')
plt.title('Elbow Plot decides n_components=6')
plt.show()

n_components=6
original_feature_names = X_train.columns.tolist()
pca_feature_names = original_feature_names[:num_components_to_retain]
print(f"Use 95% as threshold, we use lefted with {n_components} predictors to refit. T

# summary table
summary_table = pd.DataFrame({
    'Variable': X.columns,
    'Explained Variance Ratio': explained_variance_ratio,
    'Cumulative Variance Ratio': cumulative_variance_ratio,
})
print(summary_table)

# Refit
pca_new = PCA(n_components)
original_feature_names = X_train.columns.tolist()
pca_feature_names = original_feature_names[:num_components_to_retain]


X_train_pca = pca_new.fit_transform(X_train_scaled)
X_test_pca = pca_new.transform(X_test_scaled)

model_pca = LinearRegression()
model_pca.fit(X_train_pca, y_train)
```




```
Use 95% as threshold, we use lefted with 6 predictors to refit. They are ['crim', 'z
n', 'indus', 'nox', 'rm', 'age'].
  Variable  Explained Variance Ratio  Cumulative Variance Ratio
0     crim                  0.546607                   0.546607
1       zn                  0.139050                   0.685657
2    indus                  0.122930                   0.808587
3      nox                  0.076978                   0.885565
4       rm                  0.053759                   0.939325
5      age                  0.031982                   0.971307
6      tax                  0.028693                   1.000000
```

Out[157]:

```
LinearRegression()
```

In [129…

```
# Linear
X = boston[['crim', 'zn', 'indus', 'nox', 'rm', 'age', 'tax']]
y = boston['medv']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

X_train = sm.add_constant(X_train)
model = sm.OLS(y_train, X_train).fit()
print(model.summary())
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   medv   R-squared:                       0.597
Model:                            OLS   Adj. R-squared:                  0.590
Method:                 Least Squares   F-statistic:                     83.82
Date:                Thu, 05 Oct 2023   Prob (F-statistic):           3.26e-74
Time:                        15:29:59   Log-Likelihood:                -1291.5
No. Observations:                 404   AIC:                             2599.
Df Residuals:                     396   BIC:                             2631.
Df Model:                           7
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const        -21.9794      3.593     -6.117      0.000     -29.043     -14.916
crim          -0.1350      0.041     -3.263      0.001      -0.216      -0.054
zn             0.0096      0.017      0.573      0.567      -0.023       0.042
indus          0.0322      0.081      0.399      0.690      -0.127       0.191
nox           -1.6441      4.660     -0.353      0.724     -10.805       7.517
rm             8.1235      0.462     17.580      0.000       7.215       9.032
age           -0.0237      0.017     -1.439      0.151      -0.056       0.009
tax           -0.0098      0.003     -3.341      0.001      -0.016      -0.004
==============================================================================
Omnibus:                      220.416   Durbin-Watson:                   2.145
Prob(Omnibus):                  0.000   Jarque-Bera (JB):             2208.978
Skew:                           2.116   Prob(JB):                         0.00
Kurtosis:                      13.645   Cond. No.                     7.56e+03
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly spec
ified.
[2] The condition number is large, 7.56e+03. This might indicate that there are
strong multicollinearity or other numerical problems.
```

In [158…

```
# Calculate MSE+RMSE for LR
X_test = sm.add_constant(X_test)
y_pred_lr = model.predict(X_test)
mse_lr = mean_squared_error(y_test, y_pred_lr)
rmse_lr = np.sqrt(mse_lr)
rss_lr = np.sum((y_test - y_pred_lr) ** 2)


# Calculate MSE+RMSE for PCR
y_pred_pca = model_pca.predict(X_test_pca)
mse_pca = mean_squared_error(y_test, y_pred_pca)
rmse_pca = np.sqrt(mse_pca)
rss_pca = np.sum((y_test - y_pred_pca) ** 2)

r2_pca = r2_score(y_test, y_pred_pca)
```

```
num_predictors_pca = num_components_to_retain
n = len(y_test)
adjusted_r2_pca = 1 - ((1 - r2_pca) * (n - 1) / (n - num_predictors_pca - 1))
```

In [159…

```
comparison_df = pd.DataFrame({
    'Metric': ['MSE', 'RMSE', 'RSS','Adjusted R-squared'],
    'Linear Regression': [mse_lr, rmse_lr, rss_lr, model.rsquared_adj],
    'PCR': [mse_pca, rmse_pca, rss_pca, adjusted_r2_pca]
})
print(comparison_df)
```

```
               Metric  Linear Regression          PCR
0                 MSE          37.386878    36.779146
1                RMSE           6.114481     6.064581
2                 RSS        3813.461570  3751.472865
3  Adjusted R-squared           0.589924     0.466794
```

Answer

PCR conclusion:

- ['crim', 'zn', 'indus'] explain variance in 'medv' about 55%, 14%, and 12%. By choosing 6 components, we left with ['crim', 'zn', 'indus', 'nox', 'rm', 'age'], which is one variable less than Linear.

Comparison:

- PCR mitigates multicollinearity by transforming the original features into uncorrelated principal components. The result of pcr is not straight forward like linear regression. In the contrast, LR assumes that predictors are not highly correlated. Due to collinearity, the results of our linear model lacks of confidence.

- PCR have Lower MSE, RMSE, and RSS, indicating better predictive accuracy. In terms of adjusted R-squared, multicollinearity can lead to an inflated R-squared value. Although adjusted R-squared of LR is larger than PCR, we should not rely on this statistics.

Conclusion:

- We will prefer PCR in terms of collinearity and better predictive accuracy toward 'medv'.

Q3

3.Compare models selected using lasso vs a stepwise procedure to predict ‘medv' using all available variables.

In [131…

```
# set up all predictors
X_all = boston[['crim', 'zn', 'indus', 'chas', 'nox', 'rm', 'age', 'dis', 'rad', 'tax'
       'ptratio', 'black', 'lstat']]
y = boston['medv']
X_train, X_test, y_train, y_test = train_test_split(X_all, y, test_size=0.2, random_st
```

In [138…

```
# lasso
lasso_model = Lasso(alpha=1.0)
lasso_model.fit(X_train, y_train)

coefficients = lasso_model.coef_
intercept = lasso_model.intercept_

# summary table
summary_table = pd.DataFrame({
    'Feature': ['Intercept'] + list(X_train.columns),
    'Coefficient': [intercept] + list(coefficients)
})
print(summary_table)
selected_variables = X_all.columns[lasso_model.coef_ != 0]
print("Selected Variables:", selected_variables)
selected_lasso = len(selected_variables)


# metrics
y_pred_lasso = lasso_model.predict(X_test)
rss_lasso = np.sum((y_test - y_pred_lasso) ** 2)
mse_lasso = mean_squared_error(y_test, y_pred_lasso)

num_predictors = X_train.shape[1]
num_observations = len(y_test)
df = num_observations - num_predictors -1
aic_lasso = (num_observations * np.log(rss_lasso / num_observations)) + (2 * num_predi
bic_lasso = (num_observations * np.log(rss_lasso / num_observations)) + (num_predictor

r2_lasso = r2_score(y_test, y_pred_lasso)
adj_r2_lasso = 1 - ((1 - r2_lasso) * (num_observations - 1) / df)
```

```
      Feature  Coefficient
0   Intercept    44.085168
1        crim    -0.066797
2          zn     0.061603
3       indus    -0.012100
4        chas     0.000000
5         nox    -0.000000
6          rm     0.391248
7         age     0.029483
8         dis    -0.611593
9         rad     0.307379
10        tax    -0.016251
11    ptratio    -0.725637
12      black     0.008805
13      lstat    -0.795830
Selected Variables: Index(['crim', 'zn', 'indus', 'rm', 'age', 'dis', 'rad', 'tax',
'ptratio',
       'black', 'lstat'],
      dtype='object')
```

In [141…

```
#stepwise backward
lr = LinearRegression()

sfs_backward = SFS(lr,
                   k_features='best',
                   forward=False,
                   scoring=None,
                   cv=5)
sfs_backward = sfs_backward.fit(X_train, y_train)
selected_feature_indices = sfs_backward.k_feature_idx_
selected_features = X_all.columns[list(selected_feature_indices)]
selected_backward = len(selected_features)

# rebuild model testing for RSS and MSE
final_X_train = X_train[selected_features]
final_X_test = X_test[selected_features]
final_model = LinearRegression().fit(final_X_train, y_train)

n = len(y_test)
num_predictors = len(selected_features)
rss_step = np.sum((y_test - y_pred_step) ** 2)

# AIC and BIC formulas
aic_step = (n * np.log(rss_step / n)) + (2 * num_predictors)
bic_step = (n * np.log(rss_step / n)) + (num_predictors * np.log(n))

# Adjusted R-squared
n = len(y_train)
k = len(selected_features)
r_squared = final_model.score(final_X_train, y_train)
adj_r2_step = 1 - (1 - r_squared) * (n - 1) / (n - k - 1)

print("Selected Features:", selected_features)
print("Final Model Summary:")
print("Intercept:", final_model.intercept_)
print("Coefficients:", final_model.coef_)
```

```
Selected Features: Index(['crim', 'zn', 'nox', 'rm', 'dis', 'rad', 'tax', 'ptratio',
'black',
       'lstat'],
      dtype='object')
Final Model Summary:
Intercept: 40.76583789189259
Coefficients: [-1.09319249e-01  5.41542211e-02 -1.72424443e+01  3.42450276e+00
 -1.53212077e+00  3.38173324e-01 -1.32556927e-02 -1.01594929e+00
  9.11527482e-03 -5.44142355e-01]
```

In [149…

```
#stepwise backward
lr = LinearRegression()

sfs_forward = SFS(lr,
                   k_features='best',
                   forward=True,
                   scoring=None,
                   cv=5)
sfs_forward = sfs_forward.fit(X_train, y_train)
selected_feature_indices = sfs_forward.k_feature_idx_
selected_features_for = X_all.columns[list(selected_feature_indices)]
if list(selected_features_for) == list(selected_features):
```

```
    print(f"Forward stepwise have same result as backward.")
else:
    print(f"Forward stepwise have different result as backward.")
```

```
Forward stepwise have same result as backward.
```

In [151…

```
comparison_table = pd.DataFrame({
    'Metric': ['RSS', 'MSE', 'AIC', 'BIC', 'Adjusted R-squared','Number selected'],
    'Stepwise Regression': [rss_step, mse_step, aic_step, bic_step, adj_r2_step, selec
    'Lasso Regression': [rss_lasso, mse_lasso, aic_lasso, bic_lasso, adj_r2_lasso, sel
    })

print(comparison_table)
```

```
               Metric  Stepwise Regression  Lasso Regression
0                 RSS         14092.892531       2949.504253
1                 MSE            37.606326         28.916708
2                 AIC           522.702212        369.170796
3                 BIC           548.951940        403.295443
4  Adjusted R-squared             0.724028          0.586108
5     Number selected            10.000000         11.000000
```

Answer

- Selected variables of **lasso**: 11 variables. ['crim', 'zn', 'indus', 'rm', 'age', 'dis', 'rad', 'tax', 'ptratio','black', 'lstat'].
- Selected variables of two-directions **stepwise**: 10 varibles. ['crim', 'zn', 'nox', 'rm', 'dis', 'rad', 'tax', 'ptratio', 'black','lstat']. 'indus' is excluded.
- Compare two models: Surprisingly, although lasso includes one more variable than stepwise, **lasso have lower RSS, MSE, AIC, and BIC.** Although stepwise have higher adjusted R-squared, without collinearity, adjusted R-squared cannot be a reliable metrics. Thus, lasso is better in terms of model simplicity (AIC/BIC), predictive accuracy (MSE/RMSE), and overall assessment of model fit (RSS).