Xiong_Yinjiang_HW3

February 5, 2020

1 Linear Model Selection and Regularization

1.Generate a data set with $p = 20$ features, $n = 1000$ observations, and an associated quantitative response vector generated according to the model

$$Y = X\beta + \epsilon$$

where β has some elements that are exactly equal to zero.

[354]:
```
import numpy as np
import itertools
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
from mlxtend.feature_selection import SequentialFeatureSelector as SFS
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import Ridge
from sklearn.linear_model import Lasso
from sklearn.linear_model import ElasticNet
np.random.seed(24)
```

[102]:
```
# first, we randomly generate 20 features with 1000 observations
X = []
for _ in range(20):
    X.append(np.random.normal(np.random.choice(100,1), 1, 1000))
X = np.array(X)
```

[103]:
```
# then, we randomly generate 20 random betas
Beta = []
for _ in range(20):
    Beta.append(10 * np.random.random())

# let's generate 5 random indices and replace them with zero
zero_index = []
```

```
for n in np.random.choice(20,5):
    Beta[n] = 0
    zero_index.append(n)

# check beta
Beta = np.array(Beta)
Beta = Beta.reshape(-1,1)
Beta

# note that we have kept the beta = 0 information in zero_index
```

[103]:
```
array([[0.        ],
       [7.41212581],
       [8.09098401],
       [3.68270568],
       [0.        ],
       [4.53731565],
       [9.0219442 ],
       [0.71447831],
       [2.10138621],
       [6.86903874],
       [0.        ],
       [1.96584887],
       [9.85123182],
       [6.30804308],
       [0.42835617],
       [8.25425396],
       [0.        ],
       [8.11969394],
       [7.67387209],
       [0.        ]])
```

[104]:
```
# finally, create Y
# use broadcasting
Y = X * Beta
# generate error: with mean = 0, sd = 10
error = np.random.normal(0,10,1000)
Y = np.sum(Y, axis=0) + error
```

2.Split your data set into a training set containing 100 observations and a test set containing 900 observations.

[105]:
```
# create a dataframe containing Y and X
Xt = X.transpose()
df = pd.DataFrame({'Y':Y})
for n in range(0,20):
    df['X{}'.format(n)] = Xt[:,n]
```

```
df.head(5)
```

[105]:

```
             Y         X0         X1         X2         X3         X4  \
0  4707.633622  33.771182  95.932867  67.736543  52.220935  12.539658
1  4667.831605  33.894806  94.071022  66.757592  50.726081  12.705394
2  4709.137224  35.487894  95.383519  67.634304  51.974740  13.004285
3  4677.981952  34.046731  93.676851  68.747795  50.745773  11.456581
4  4708.148487  36.323789  93.901247  68.072361  50.108320  11.592697

          X5         X6         X7         X8  ...        X10        X11  \
0  20.860180  42.636533  12.114676  72.888851  ...  26.157619  75.434748
1  19.889749  40.640765  12.149863  70.991400  ...  27.466977  75.355861
2  21.134597  41.584655  12.629108  71.200583  ...  27.244833  73.746079
3  23.047611  41.068100  10.311666  73.056294  ...  28.034128  73.104750
4  22.659707  42.416513  11.340568  72.633341  ...  26.507698  74.296530

         X12        X13        X14        X15        X16        X17  \
0  54.385360  35.092312  62.661170  59.300054  28.214382  56.909681
1  54.587197  34.028896  60.410975  60.129402  26.831438  58.587753
2  54.553417  34.193719  61.728457  61.885028  27.077562  57.150222
3  53.587076  33.125989  62.532450  59.693219  27.240666  55.608992
4  55.103520  35.013668  61.938490  59.159034  27.542115  57.361038

         X18        X19
0  32.762930  11.025487
1  35.306703  11.370518
2  34.742751   9.815837
3  36.655743  12.631524
4  33.358290  11.258816

[5 rows x 21 columns]
```

[131]:

```
X = df.drop(['Y'], axis=1)
Y = df['Y']
X_train, X_test, y_train, y_test = train_test_split(X, Y, test_size = 0.1)
```

[134]:

```
# check X_train shape
X_train.shape
```

[134]: (900, 20)

3.Perform best subset selection on the training set, and plot the training set MSE associated with the best model of each size. For which model size does the training set MSE take on its minimum value?

[201]:

```
def best_subset (X, y):
    res = {}
```

```
    for k in range (1,6):
        for each in itertools.combinations(X, k):
            X_full = sm.add_constant(X[list(each)])
            lm = sm.OLS(y, X_full).fit()
            mse = lm.mse_resid
            if k not in res.keys() or mse < res[k][1]:
                res[k] = [each, mse]
    return res
```

[202]:
```
result = best_subset(X_train, y_train)
```

[203]:
```
result
```

[203]:
```
{1: [('X12',), 605.3372714840621],
 2: [('X12', 'X17'), 539.0402189267678],
 3: [('X6', 'X12', 'X17'), 462.7796313447198],
 4: [('X6', 'X12', 'X15', 'X17'), 397.8476367584193],
 5: [('X2', 'X6', 'X12', 'X15', 'X17'), 340.4398083435847]}
```

[239]:
```
sfs = SFS(LinearRegression(), k_features=5, forward=True, scoring =
 ↪'neg_mean_squared_error', cv=0)
sfs.fit(X_train, y_train)
sfs.k_feature_names_
```

[239]:
```
('X2', 'X6', 'X12', 'X15', 'X17')
```

[309]:
```
# since the computational load for best subset selection is so high, we can use
 ↪forward selection to sustitute best subset
# this is due to we are convinced that all the features were independently and
 ↪randomly generated
result_n = {}
for n in range (1,21):
    sfs = SFS(LinearRegression(), k_features=n, forward=True, scoring =
 ↪'neg_mean_squared_error', cv=0)
    sfs.fit(X_train, y_train)
    result_n[len(sfs.k_feature_names_)] = - sfs.k_score_
result_n
```

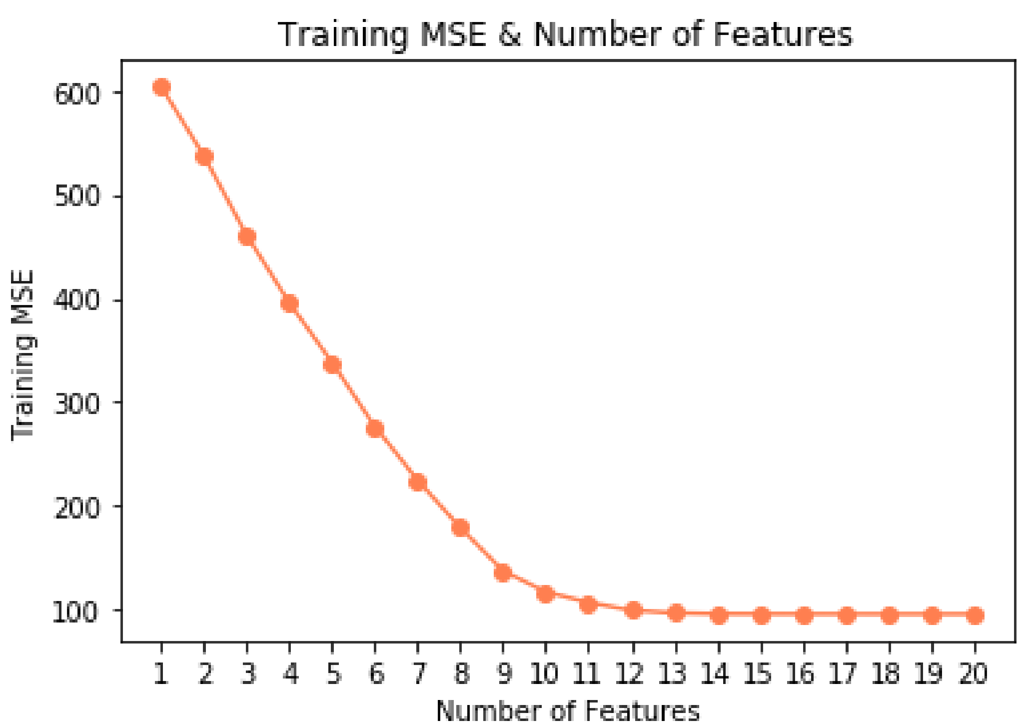
[309]:
```
{1: 603.9920775474318,
 2: 537.2434181970115,
 3: 460.7228329831881,
 4: 395.63737210976257,
 5: 338.1702096212938,
 6: 276.3064352884587,
 7: 224.5650484711889,
 8: 178.89500785481044,
 9: 137.17036132742422,
```

```
10: 116.82490517289672,
11: 106.62121141990936,
12: 99.04857146975054,
13: 96.49190629750615,
14: 95.60745808208749,
15: 95.47597200962412,
16: 95.40838739957732,
17: 95.34951343810877,
18: 95.30699428019211,
19: 95.29227944780285,
20: 95.28405524149898}
```

We can see that the MSE monotonically decreases as the number of predictors increases. The model that includes all the variables have the lowest MSE.

4.Plot the test set MSE associated with the best model of each size.

[230]:
```
feature_num =␣
↪['1','2','3','4','5','6','7','8','9','10','11','12','13','14','15','16','17','18','19','20']
MSE = []
for value in result_n.values():
    MSE.append(value)
plt.plot(feature_num, MSE, marker='o', color='coral', label='MSE')
plt.xlabel('Number of Features')
plt.ylabel('Training MSE')
plt.title('Training MSE & Number of Features');
```



5.For which model size does the test set MSE take on its minimum value? Comment on your results

[250]:
```python
result_test = {}
for n in range (1,21):
    sfs = SFS(LinearRegression(), k_features=n, forward=True, scoring =
 ↪'neg_mean_squared_error', cv=0)
    sfs.fit(X_train, y_train)
    lm = LinearRegression().fit(X_train[list(sfs.k_feature_names_)], y_train)
    result_test[sfs.k_feature_names_] = mean_squared_error(lm.
 ↪predict(X_test[list(sfs.k_feature_names_)]), y_test)
result_test
```

[250]:
```
{('X12',): 719.2575938194786,
 ('X12', 'X17'): 681.973017636605,
 ('X6', 'X12', 'X17'): 616.4695609909228,
 ('X6', 'X12', 'X15', 'X17'): 576.982262136989,
 ('X2', 'X6', 'X12', 'X15', 'X17'): 483.8935768903375,
 ('X2', 'X6', 'X9', 'X12', 'X15', 'X17'): 383.3741196195801,
 ('X1', 'X2', 'X6', 'X9', 'X12', 'X15', 'X17'): 285.52599281068683,
 ('X1', 'X2', 'X6', 'X9', 'X12', 'X15', 'X17', 'X18'): 199.244892681775,
 ('X1',
  'X2',
  'X6',
  'X9',
  'X12',
  'X13',
  'X15',
  'X17',
  'X18'): 142.8386046147517,
 ('X1',
  'X2',
  'X5',
  'X6',
  'X9',
  'X12',
  'X13',
  'X15',
  'X17',
  'X18'): 135.88442069878838,
 ('X1',
  'X2',
  'X3',
  'X5',
  'X6',
```

```
 'X9',
 'X12',
 'X13',
 'X15',
 'X17',
 'X18'): 110.99572535572248,
('X1',
 'X2',
 'X3',
 'X5',
 'X6',
 'X8',
 'X9',
 'X12',
 'X13',
 'X15',
 'X17',
 'X18'): 108.95311584909838,
('X1',
 'X2',
 'X3',
 'X5',
 'X6',
 'X8',
 'X9',
 'X11',
 'X12',
 'X13',
 'X15',
 'X17',
 'X18'): 107.52712069788699,
('X1',
 'X2',
 'X3',
 'X5',
 'X6',
 'X7',
 'X8',
 'X9',
 'X11',
 'X12',
 'X13',
 'X15',
 'X17',
 'X18'): 107.85595244497965,
('X1',
 'X2',
```

```
 'X3',
 'X5',
 'X6',
 'X7',
 'X8',
 'X9',
 'X10',
 'X11',
 'X12',
 'X13',
 'X15',
 'X17',
 'X18'): 108.65322989251703,
('X1',
 'X2',
 'X3',
 'X5',
 'X6',
 'X7',
 'X8',
 'X9',
 'X10',
 'X11',
 'X12',
 'X13',
 'X15',
 'X16',
 'X17',
 'X18'): 108.01356087793035,
('X1',
 'X2',
 'X3',
 'X4',
 'X5',
 'X6',
 'X7',
 'X8',
 'X9',
 'X10',
 'X11',
 'X12',
 'X13',
 'X15',
 'X16',
 'X17',
 'X18'): 107.66627726979024,
('X0',
```

```
 'X1',
 'X2',
 'X3',
 'X4',
 'X5',
 'X6',
 'X7',
 'X8',
 'X9',
 'X10',
 'X11',
 'X12',
 'X13',
 'X15',
 'X16',
 'X17',
 'X18'): 107.86770395863172,
('X0',
 'X1',
 'X2',
 'X3',
 'X4',
 'X5',
 'X6',
 'X7',
 'X8',
 'X9',
 'X10',
 'X11',
 'X12',
 'X13',
 'X14',
 'X15',
 'X16',
 'X17',
 'X18'): 107.62717677457347,
('X0',
 'X1',
 'X2',
 'X3',
 'X4',
 'X5',
 'X6',
 'X7',
 'X8',
 'X9',
 'X10',
```

```
 'X11',
 'X12',
 'X13',
 'X14',
 'X15',
 'X16',
 'X17',
 'X18',
 'X19'): 107.71148527265507}
```

'X1','X2','X3','X5','X6','X8','X9','X11','X12','X13','X15','X17','X18' compose the model that produces the lowest MSE.

6.How does the model at which the test set MSE is minimized compare to the true model used to generate the data? Comment on the coefficient sizes.

[255]:
```python
# let's use the model with the smallest MSE to fit our entire data set
lm = LinearRegression().
 ↪fit(X[['X1','X2','X3','X5','X6','X8','X9','X11','X12','X13','X15','X17','X18']],␣
 ↪Y)
lm.coef_
```

[255]:
```
array([7.40443413, 7.97053512, 3.20909145, 4.2339433 , 9.39027123,
       2.73686812, 7.25486347, 1.60761402, 9.88848592, 6.75330859,
       8.56588326, 8.27709188, 7.50247345])
```

The model estimated using best subset selection is Y = 7.40X1 + 7.97X2 + 3.21X3 + 4.23X5 + 9.39X6 + 2.74X8 + 7.25X9 + 1.61X11 + 9.89X12 + 6.75X13 + 8.57X15 + 8.28X17 + 7.50X18 + error
The true model that we generate is: Y = 7.41X1 + 8.09X2 + 3.68X3 + 4.54X5 + 9.02X6 + 0.71X7 + 2.10X8 + 6.87X9 + 1.97X11 + 9.85X12 + 6.31X13 + 0.43X14 + 8.25X15 + 8.12X17 + 7.67X18
For the predictors missed by the estimated model, they all have small true coefficients. The model selection process judged that excluding them would improve overall performance in MSE. For the rest of the coefficients, their magnitudes are very similar between the estimated model and the true model.

7.Create a plot displaying

$$\sqrt{\sum_{j=1}^{p}(\beta_j - \hat{\beta}_j^r)^2}$$

for a range of values of r, where $\hat{\beta}_j^r$ is the jth coefficient estimate for the best model containing r coefficients. Comment on what you observe. How does this compare to the test MSE plot?

[256]:
```python
Beta
```

[256]:
```
array([[0.        ],
       [7.41212581],
```

```
       [8.09098401],
       [3.68270568],
       [0.        ],
       [4.53731565],
       [9.0219442 ],
       [0.71447831],
       [2.10138621],
       [6.86903874],
       [0.        ],
       [1.96584887],
       [9.85123182],
       [6.30804308],
       [0.42835617],
       [8.25425396],
       [0.        ],
       [8.11969394],
       [7.67387209],
       [0.        ]])
```

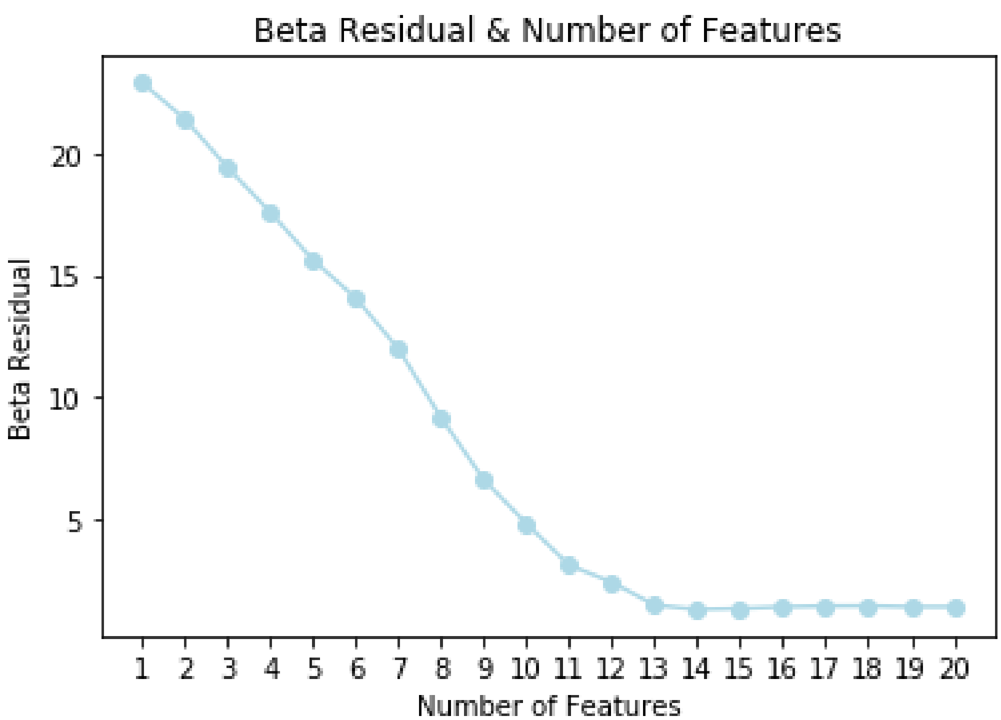
[308]:
```
# get the coefficient for each r
Beta_n = Beta.ravel()
coef_hat = {}
for key, value in result_test.items():
    lm = LinearRegression().fit(X[list(key)], Y)
    coef_hat[key] = lm.coef_

# compute the square root of sum of beta residual squared for each model
Beta_res = {}
for key, value in coef_hat.items():
    zero = np.zeros(20)
    for n in range(len(value)):
        zero[int(df.columns.get_loc(key[n])) - 1] = value[n]
    Beta_res[len(key)] = (((Beta_n - zero) ** 2).sum()) ** 0.5
Beta_res
# the key represents how many features are included in the model
```

[308]:
```
{1: 22.910996082520086,
 2: 21.426696266575984,
 3: 19.437567526979205,
 4: 17.598222488975477,
 5: 15.638668571664878,
 6: 14.078921850832995,
 7: 11.994342258097692,
 8: 9.176623860557708,
 9: 6.632625661480814,
 10: 4.826779550924385,
 11: 3.1190758735904387,
```

```
12: 2.4225711007077653,
13: 1.4807056566700816,
14: 1.3046840409967855,
15: 1.3432017834262682,
16: 1.4090081169805273,
17: 1.4414522647129009,
18: 1.448289611890343,
19: 1.4114401386097108,
20: 1.4105674478420807}
```

[307]:
```
Beta_rss = []
for value in Beta_res.values():
    Beta_rss.append(value)
plt.plot(feature_num, Beta_rss, marker='o', color='lightblue')
plt.xlabel('Number of Features')
plt.ylabel('Beta Residual')
plt.title('Beta Residual & Number of Features');
```



The graph suggests an overall downward trend in Beta residuals, but if we check the data computed above, we can find that 14 features yield the lowest Beta residual. Note that the true model contains 15 features, and the model with 15 features has indeed very low Beta residual (second lowest). The reason why the 14-feature model is the lowest is possibly due to the irreducible error.

2 Application Exercises

[313]:
```
gss_train = pd.read_csv('gss_train.csv')
gss_test = pd.read_csv('gss_test.csv')
gss_train.head(5)
```

[313]:
```
   age  attend  authoritarianism  black  born  childs  colath  colrac  colcom  \
0   21       0                 4      0     0       0       1       1       0
1   42       0                 4      0     0       2       0       1       1
2   70       1                 1      1     0       3       0       1       1
3   35       3                 2      0     0       2       0       1       0
4   24       3                 6      0     1       3       1       1       0

   colmil  ...  zodiac_GEMINI  zodiac_CANCER  zodiac_LEO  zodiac_VIRGO  \
0       1  ...              0              0           0             0
1       0  ...              0              0           0             0
2       0  ...              0              0           0             0
3       1  ...              0              0           0             0
4       0  ...              0              0           0             0

   zodiac_LIBRA  zodiac_SCORPIO  zodiac_SAGITTARIUS  zodiac_CAPRICORN  \
0             0               0                   0                 0
1             0               0                   0                 0
2             0               0                   0                 0
3             0               1                   0                 0
4             0               1                   0                 0

   zodiac_AQUARIUS  zodiac_PISCES
0                0              0
1                0              0
2                0              0
3                0              0
4                0              0

[5 rows x 78 columns]
```

1.Fit a least squares linear model on the training set, and report the test MSE.

[318]:
```
X_train = gss_train.drop(['egalit_scale'], axis=1)
X_test = gss_test.drop(['egalit_scale'], axis=1)
y_train = gss_train['egalit_scale']
y_test = gss_test['egalit_scale']
lm = LinearRegression().fit(X_train, y_train)
MSE = mean_squared_error(lm.predict(X_test), y_test)
print('The test MSE for linear model is', MSE)
```

```
The test MSE for linear model is 63.213629623014995
```

2.Fit a ridge regression model on the training set, with λ chosen by 10-fold cross-validation. Report the test MSE.

[327]:
```
parameters = {'alpha':[1e-15, 1e-10, 1e-8, 1e-4, 1e-3, 1e-2, 1, 5, 10, 20]}
ridge_regressor = GridSearchCV(Ridge(), parameters,␣
 ↪scoring='neg_mean_squared_error', cv=10)
ridge_regressor.fit(X_train, y_train)
print('The best alpha value is', ridge_regressor.best_params_)
print('The training MSE is', -ridge_regressor.best_score_)
```

```
The best alpha value is {'alpha': 20}
The training MSE is 61.148463225383004
```

[325]:
```
MSE = mean_squared_error(ridge_regressor.predict(X_test), y_test)
print('The test MSE for ridge regression is', MSE)
```

```
The test MSE for ridge regression is 62.298944127974785
```

3.Fit a lasso regression on the training set, with λ chosen by 10-fold cross-validation. Report the test MSE, along with the number of non-zero coefficient estimates.

[334]:
```
parameters = {'alpha':[1e-15, 1e-10, 1e-8, 1e-4, 1e-3, 1e-2, 1, 5, 10, 20]}
lasso_regressor = GridSearchCV(Lasso(), parameters,␣
 ↪scoring='neg_mean_squared_error', cv=10)
lasso_regressor.fit(X_train, y_train)
print('The best alpha value is', lasso_regressor.best_params_)
print('The training MSE is', -lasso_regressor.best_score_)
```

```
The best alpha value is {'alpha': 0.01}
The training MSE is 61.37196490960359
```

[329]:
```
MSE = mean_squared_error(lasso_regressor.predict(X_test), y_test)
print('The test MSE for lasso regression is', MSE)
```

```
The test MSE for lasso regression is 62.37909680839509
```

[341]:
```
lasso = Lasso(alpha=0.01)
lasso.fit(X_train, y_train)
print('There are',(lasso.coef_ != 0).sum(),'non-zero coefficient estimates in␣
 ↪the lasso regression.')
```

```
There are 65 non-zero coefficient estimates in the lasso regression.
```

4.Fit an elastic net regression model on the training set, with α and λ chosen by 10-fold cross-validation. That is, estimate models with $\alpha = 0, 0.1, 0.2, \ldots, 1$ using the same values for λ across each model. Select the combination of α and λ with the lowest cross-validation MSE. For that combination, report the test MSE along with the number of non-zero coefficient estimates.

[349]:
```
alpha = [1e-15, 1e-10, 1e-8, 1e-4, 1e-3, 1e-2, 1, 5, 10, 20]
for n in alpha:
    parameters = {'l1_ratio':[0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1],␣
 ↪'alpha':[n]}
    en_regressor = GridSearchCV(ElasticNet(), parameters,␣
 ↪scoring='neg_mean_squared_error', cv=10)
    en_regressor.fit(X_train, y_train)
    print('The best alpha value is', en_regressor.best_params_)
    print('The training MSE at l1_ratio =', en_regressor.
 ↪best_params_['l1_ratio'], 'and alpha(lambda) =',
          en_regressor.best_params_['alpha'], 'is', -en_regressor.best_score_)
```

```
The best alpha value is {'alpha': 1e-15, 'l1_ratio': 0.1}
The training MSE at l1_ratio = 0.1 and alpha(lambda) = 1e-15 is
62.357446358210225
The best alpha value is {'alpha': 1e-10, 'l1_ratio': 0.1}
The training MSE at l1_ratio = 0.1 and alpha(lambda) = 1e-10 is
62.357446293984346
The best alpha value is {'alpha': 1e-08, 'l1_ratio': 0.1}
The training MSE at l1_ratio = 0.1 and alpha(lambda) = 1e-08 is
62.357439935622324
The best alpha value is {'alpha': 0.0001, 'l1_ratio': 0.1}
The training MSE at l1_ratio = 0.1 and alpha(lambda) = 0.0001 is
62.298858480534875
The best alpha value is {'alpha': 0.001, 'l1_ratio': 0.1}
The training MSE at l1_ratio = 0.1 and alpha(lambda) = 0.001 is
62.00028310967855
The best alpha value is {'alpha': 0.01, 'l1_ratio': 0.1}
The training MSE at l1_ratio = 0.1 and alpha(lambda) = 0.01 is 61.30512590261391
The best alpha value is {'alpha': 1, 'l1_ratio': 0.1}
The training MSE at l1_ratio = 0.1 and alpha(lambda) = 1 is 65.98844258107927
The best alpha value is {'alpha': 5, 'l1_ratio': 0.1}
The training MSE at l1_ratio = 0.1 and alpha(lambda) = 5 is 76.22947380485463
The best alpha value is {'alpha': 10, 'l1_ratio': 0.1}
The training MSE at l1_ratio = 0.1 and alpha(lambda) = 10 is 81.05727865324738
The best alpha value is {'alpha': 20, 'l1_ratio': 0.1}
The training MSE at l1_ratio = 0.1 and alpha(lambda) = 20 is 85.19674518831349
```

[353]:
```
# from the above CV training MSE, we get the lowest MSE at l1 ration = 0.1 and␣
 ↪alpha(lambda) = 0.01
# now we can fit the model to the test set
en = ElasticNet(l1_ratio=0.1, alpha=0.01)
en.fit(X_train, y_train)
MSE = mean_squared_error(en.predict(X_test), y_test)
print('The test MSE for elastic net regression is', MSE)
print('There are',(en.coef_ != 0).sum(),'non-zero coefficient estimates in the␣
 ↪elastic net regression.')
```

```
The test MSE for elastic net regression is 62.3801287740633
There are 76 non-zero coefficient estimates in the elastic net regression.
```

5.Comment on the results obtained. How accurately can we predict an individual's egalitarianism? Is there much difference among the test errors resulting from these approaches?

All models achieved similar test MSE, with ridge being the best and OLS being the worst, but the margin is within 1. Generally, we get a MSE at around 62, which is not so far from the training data, showing that we are not overfitting. Whether there exist better models deserves a further discussion.