Homework 3

February 9, 2020

1 Homework 3

```
In [14]: import numpy as np
         import scipy as sp
         import matplotlib.pyplot as plt
         from pylab import rcParams
         import seaborn as sns
         import pandas as pd
         import sklearn
         import warnings
         import itertools
         import sklearn.metrics
         from sklearn.linear_model import LinearRegression
         warnings.filterwarnings('ignore')
```

1.1 Conceptual exercises: Training/test error for subset selection

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

This artificial dataset includes 5 binary features, 5 multiple-value discrete features, and 10 continuous features.

```
In [2]: # feature generation
        feature_dict = {'f'+str(i):[] for i in range(1,21)}

        # categorical features
        for i in range(1,6):
            feature_dict['f'+str(i)] = np.random.randint(2, size=1000)

        # ordinal features
        for i in range(6,11):
            feature_dict['f'+str(i)] = np.random.randint(
            np.random.randint(5,10), size=1000)

        # numerical features
        for i in range(11,21):
            feature_dict['f'+str(i)] = np.random.random(1000)
```

```
my_df = pd.DataFrame(feature_dict)
```

```
In [7]: # response generation
        feature_selection_vector = np.random.randn(20)*np.random.randint(
        -2, 3, size=20)
        feature_selection_vector
```

```
Out[7]: array([ 0.        ,  0.29740592,  0.        ,  0.48273752, -0.        ,
               -0.        , -0.81562101,  0.09988678, -0.36438558, -0.04808328,
                0.        , -0.        ,  1.17299172,  3.32859281,  0.44176672,
               -0.67305308, -0.07130467, -1.75664688,  1.39356568, -0.        ])
```

```
In [ ]: my_df['response'] = my_df[:] @ feature_selection_vector + np.random.randn(1000)
```

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

```
In [9]: import sklearn.model_selection
        my_train, my_test = sklearn.model_selection.train_test_split(
        my_df, train_size=0.1,test_size=0.9)
```

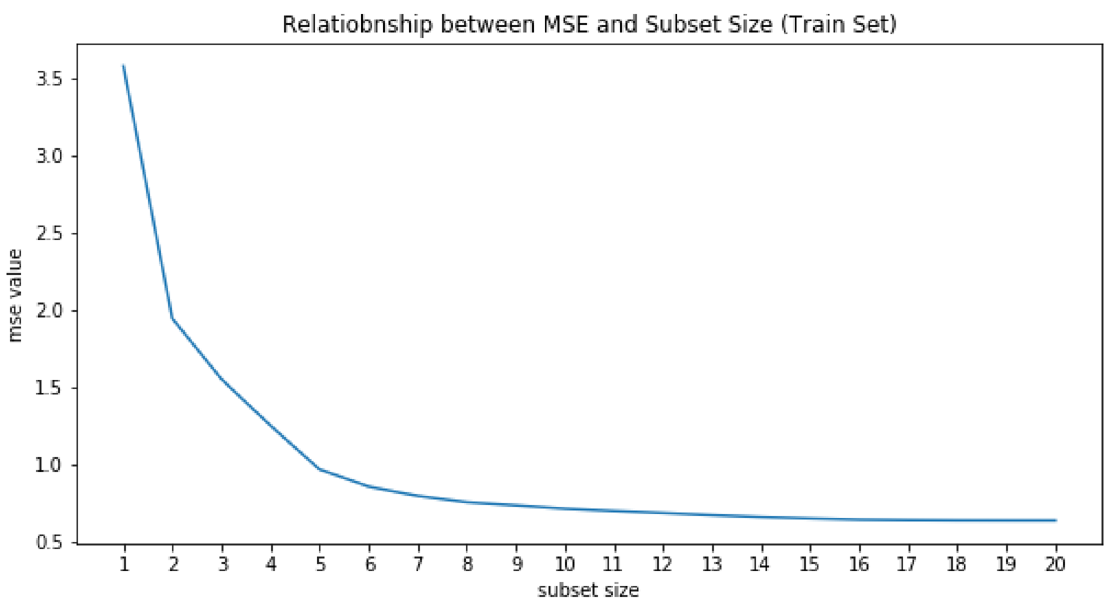
1.1.3 Perform best subset selection on the training set, and plot the training set MSE associated with the best model of each size. For which model size does the training set MSE take on its minimum value?

```
In [10]: %%writefile best_subset_select.py
         import itertools
         import sklearn.metrics
         from sklearn.linear_model import LinearRegression

         # The sklearn.feature_selection.SelectKBest is actually
         # an improved algorithm to perform feature selections,
         # but does not explore all the combinations as the best
         # selection method does

         def best_subset_k(i,df):
             features_list = list(itertools.combinations(range(0,20), i))
             mse_list = []
             for f in features_list:
                 new_features = df.iloc[:, list(f)]
                 model = LinearRegression().fit(
                 new_features, df['response'])
                 mse = sklearn.metrics.mean_squared_error(
                 df['response'], model.predict(new_features))
                 mse_list.append((list(f), mse))
             return sorted(mse_list, key=lambda x:x[1])[0]
```

```
Overwriting best_subset_select.py
```

```
In [11]: # multiprocessing
         import best_subset_select as best
         from multiprocessing import Pool

         p = Pool()
         subset_train_MSE = p.starmap(best.best_subset_k,
         [(i,my_train) for i in range(1,21)])
```

```
In [12]: # plot the relatiobnship between MSE and different model size
         plt.rcParams["figure.figsize"] = (10,5)
         plt.plot([len(x[0]) for x in subset_train_MSE],
                  [x[1] for x in subset_train_MSE])
         plt.xlabel('subset size')
         plt.xticks(range(1,21))
         plt.ylabel('mse value')
         plt.title('Relatiobnship between MSE and Subset Size (Train Set)')
         plt.show()
```



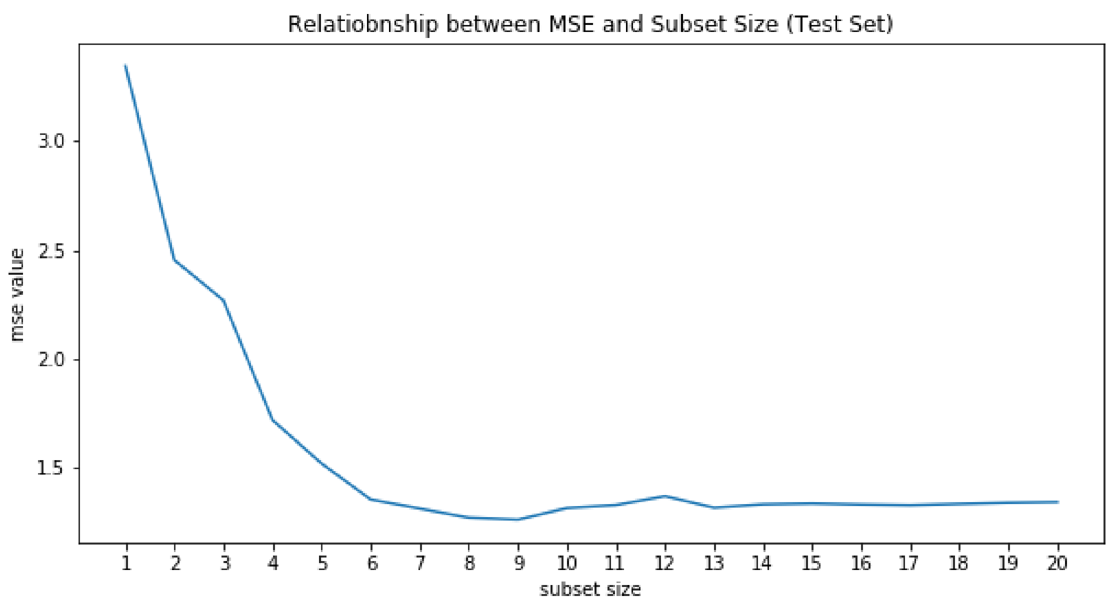
For the training set, the MSE value decreases as the size of feature subset increases. Therefore, the model size that yields the best result is the total number of original features. This is expected because more features generally increase the capability of the model to capture variance of the training data.

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

```
In [15]: subset_test_MSE = []
         for t in subset_train_MSE:
             model = LinearRegression().fit(
```

```
            my_train.iloc[:, t[0]], my_train['response'])
        mse = sklearn.metrics.mean_squared_error(my_test['response'],
        model.predict(my_test.iloc[:, t[0]]))
        subset_test_MSE.append((t[0], mse))
```

In [17]:
```
# plot the relatiobnship between MSE and different model size
plt.rcParams["figure.figsize"] = (10,5)
plt.plot([len(x[0]) for x in subset_test_MSE],
         [x[1] for x in subset_test_MSE])
plt.xlabel('subset size')
plt.xticks(range(1,21))
plt.ylabel('mse value')
#plt.ylim([0,5])
plt.title('Relatiobnship between MSE and Subset Size (Test Set)')
plt.show()
```



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

In [22]:
```
best_subset = sorted(subset_test_MSE, key=lambda x:x[1])[0]
print('Best subset size {}'.format(len(best_subset[0])), best_subset)
```

```
Best subset size 9 ([3, 6, 7, 8, 12, 13, 14, 17, 18], 1.261039835291449)
```

From the plot and the minimized test MSE value for subset models, we know that the best model size is 9 instead of 20. From the MSE curve, we see that the MSE value drops at the beginning as the feature number increases. But after the point of 9, the value starts increasing rising slowly and finally fluctuates around a certain level higher than the minimized value. This suggests that when the feature number becomes too large, the model would already overfit the data, making it hard to precisely capture data variance outside of the training set.

1.1.6 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.

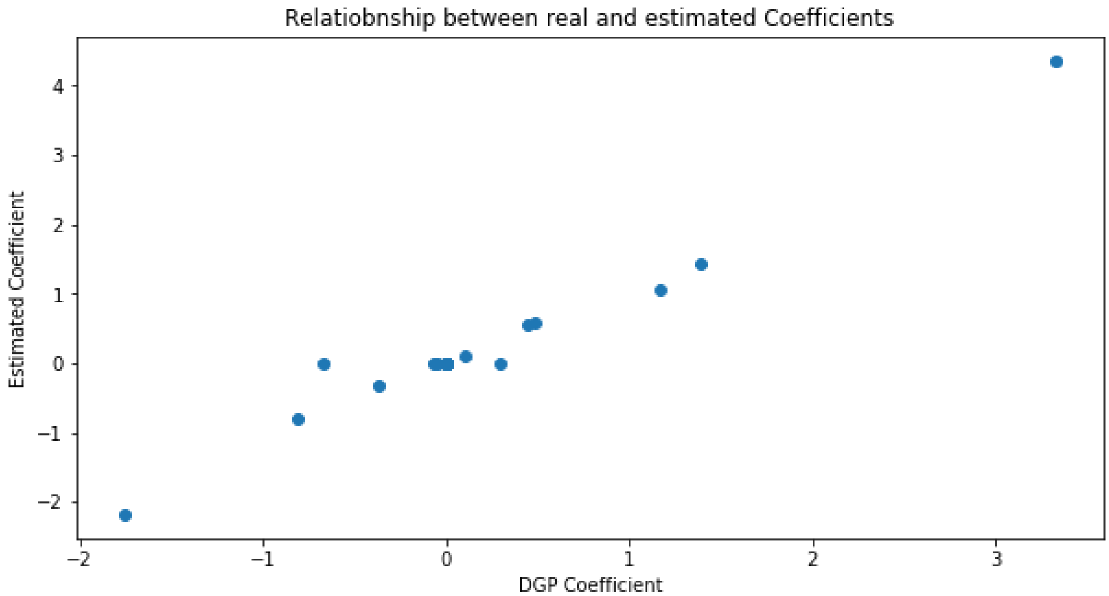
In [34]:
```
# calculate the coefficients for the best model
best_model = LinearRegression().fit(
my_train.iloc[:, best_subset[0]], my_train['response'])
best_coef = [0]*20
for index in best_subset[0]:
    best_coef[index] = best_model.coef_[best_subset[0].index(index)]
np.array(best_coef)
```

Out[34]:
```
array([ 0.        ,  0.        ,  0.        ,  0.58042661,  0.        ,
        0.        , -0.79501153,  0.10676815, -0.31489293,  0.        ,
        0.        ,  0.        ,  1.06628287,  4.36436521,  0.55205404,
        0.        ,  0.        , -2.18193901,  1.43996555,  0.        ])
```

In [33]:
```
feature_selection_vector
```

Out[33]:
```
array([ 0.        ,  0.29740592,  0.        ,  0.48273752, -0.        ,
       -0.        , -0.81562101,  0.09988678, -0.36438558, -0.04808328,
        0.        , -0.        ,  1.17299172,  3.32859281,  0.44176672,
       -0.67305308, -0.07130467, -1.75664688,  1.39356568, -0.        ])
```

In [35]:
```
plt.rcParams["figure.figsize"] = (10,5)
plt.scatter(feature_selection_vector, best_coef)
plt.xlabel('DGP Coefficient')
plt.ylabel('Estimated Coefficient')
plt.title('Relatiobnship between real and estimated Coefficients')
plt.show()
```




```
In [37]: feature_selection_vector - best_coef
```

```
Out[37]: array([ 0.        ,  0.29740592,  0.        , -0.09768909, -0.        ,
                -0.        , -0.02060947, -0.00688137, -0.04949264, -0.04808328,
                 0.        , -0.        ,  0.10670884, -1.0357724 , -0.11028732,
                -0.67305308, -0.07130467,  0.42529214, -0.04639987, -0.        ])
```

```
In [36]: np.corrcoef(best_coef,feature_selection_vector)
```

```
Out[36]: array([[1.        , 0.97909313],
                [0.97909313, 1.        ]])
```

From the scatter plot and the correlation test, it is clear to see that the estimated coefficients and the real coefficients have a strong positive relation. For most of the coefficients, the differences of the real and the estimated are less than 0.1, which indicates the reliability of this model. Several features like f14 and f16 have greater difference, which might be resulted from the disturbance of the ϵ and the setting of intercept.

1.1.7 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?

```
In [44]: coef_distances = []
         for t in subset_test_MSE:
```

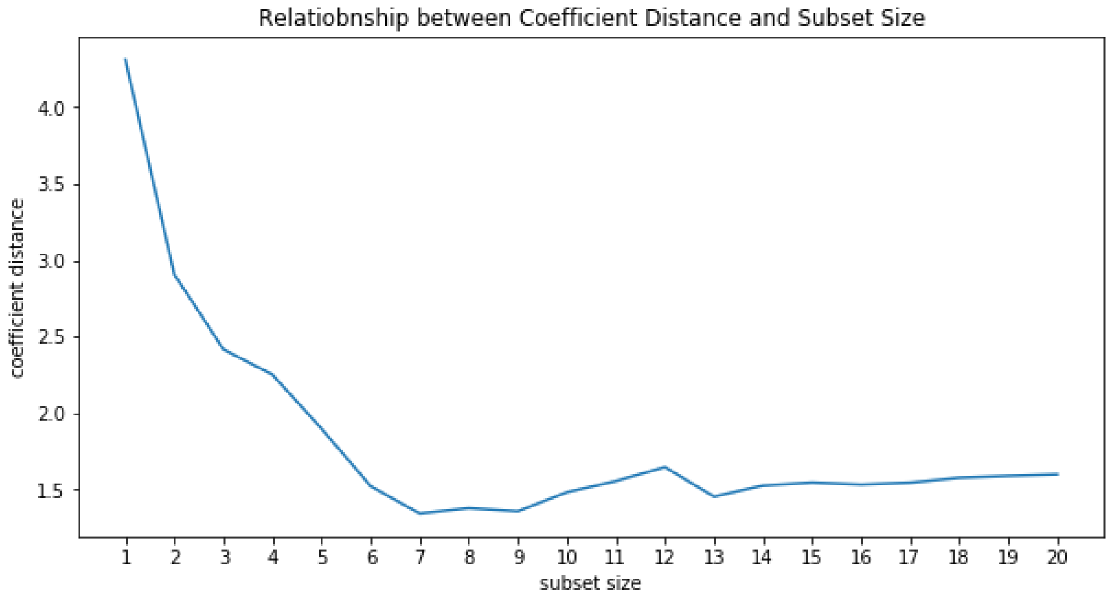
```
        selected = t[0]
        model = LinearRegression().fit(
        my_train.iloc[:, selected], my_train['response'])
        estimated_coef = [0]*20
        for i in range(len(selected)):
            estimated_coef[selected[i]] = model.coef_[i]
        distance = (((estimated_coef-feature_selection_vector)**2).sum())**0.5
        coef_distances.append((len(selected), distance))
    coef_distances
```

Out[44]:

```
[(1, 4.313185950796175),
 (2, 2.9070141521615316),
 (3, 2.4163434061372873),
 (4, 2.2505749650157525),
 (5, 1.8977285147489176),
 (6, 1.5218827472966603),
 (7, 1.3421509296576688),
 (8, 1.377489892266472),
 (9, 1.3567286984222318),
 (10, 1.4817136694657005),
 (11, 1.5547193398283088),
 (12, 1.6456342043679697),
 (13, 1.4535798829770201),
 (14, 1.525807484316243),
 (15, 1.5450848389937628),
 (16, 1.5322100892209722),
 (17, 1.5445592143338924),
 (18, 1.5761959580072793),
 (19, 1.5880928331997313),
 (20, 1.5979037178683242)]
```

In [45]:

```
# plot the relatiobnship between coef_distances and different model size
plt.rcParams["figure.figsize"] = (10,5)
plt.plot([x[0] for x in coef_distances],
         [x[1] for x in coef_distances])
plt.xlabel('subset size')
plt.xticks(range(1,21))
plt.ylabel('coefficient distance')
plt.title('Relatiobnship between Coefficient Distance and Subset Size')
plt.show()
```




```
In [46]: np.corrcoef([x[1] for x in coef_distances],[x[1] for x in subset_test_MSE])

Out[46]: array([[1.        , 0.98612853],
                [0.98612853, 1.        ]])
```

This plot shows the same trend as the test MSE plot, though not exactly the same. The stages of underfitting and overfitting as the number of included features increases are also reflected in this plot. The point at which the distance value is the smallest is 7, and 9 (the point for the smallest test MSE) is the second smallest. The high correlation between these two types of values can also be verified with the correlation test result above.

1.2 Application exercises

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

```
In [53]: # loading train and test set
         GSS_train = pd.read_csv('data/gss_train.csv')
         GSS_test = pd.read_csv('data/gss_test.csv')
         GSS_train_features = GSS_train.drop(columns='egalit_scale')
         GSS_test_features = GSS_test.drop(columns='egalit_scale')

In [67]: # fitting the least squares linear model and calculate test MSE
         OLS_model = LinearRegression().fit(
         GSS_train_features, GSS_train['egalit_scale'])
         print("OLS test MSE:", sklearn.metrics.mean_squared_error(
         GSS_test['egalit_scale'], OLS_model.predict(GSS_test_features)))

OLS test MSE: 63.213629623014995
```

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

In [94]:
```
from sklearn.linear_model import RidgeCV
# here the lambda is determined by the lowest MSE among k-folds
# fitting the Ridge model and calculate test MSE
neg_mse = sklearn.metrics.make_scorer(
sklearn.metrics.mean_squared_error, greater_is_better = False)
Ridge_model = RidgeCV(alphas=np.linspace(0.1,1.0,30), scoring = neg_mse,
normalize = True, cv = 10).fit(GSS_train_features, GSS_train['egalit_scale'])
print("RidgeCV test MSE:", sklearn.metrics.mean_squared_error(
GSS_test['egalit_scale'], Ridge_model.predict(GSS_test_features)))
```

```
RidgeCV test MSE: 60.955798347261194
```

1.2.3 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.

In [96]:
```
from sklearn.linear_model import LassoCV
# fitting the Lasso model and calculate test MSE
Lasso_model = LassoCV(normalize = True, cv = 10, n_jobs = -1).fit(
GSS_train_features, GSS_train['egalit_scale'])
print("LassoCV test MSE:", sklearn.metrics.mean_squared_error(
GSS_test['egalit_scale'], Lasso_model.predict(GSS_test_features)))
```

```
LassoCV test MSE: 61.30153540292484
```

In [77]:
```
# checking the coefficients
Lasso_model.coef_
```

Out[77]:
```
array([-0.03905169, -0.        ,  0.        ,  0.95419783,  0.        ,
        0.15079592,  0.        ,  0.        , -0.        , -0.        ,
        0.04359243,  0.        , -0.        , -0.        , -1.24762186,
        0.20698539,  0.        ,  0.        , -0.11176059,  0.        ,
        0.7098221 , -1.55429862, -0.        ,  0.        , -4.32653709,
        0.        , -0.05345982,  0.95427603,  0.09590645, -0.        ,
       -0.        , -0.        , -0.19627341,  0.17641766, -0.06465349,
       -0.        ,  0.        , -0.3368883 , -1.64294643, -0.        ,
       -0.03431446,  0.        , -0.        ,  0.        , -0.        ,
        0.        , -0.        ,  0.0830095 , -0.86899771, -2.45473177,
       -0.        ,  0.        , -0.        ,  0.        ,  0.        ,
       -1.36451583,  0.        ,  0.        , -0.        , -0.        ,
       -0.        ,  0.        ,  0.        ,  0.        ,  0.04399991,
        1.22664377,  0.        , -0.        ,  0.        ,  0.        ,
        0.30844883, -0.        , -0.14708518, -0.        ,  0.        ,
        0.13100665, -0.        ])
```

In [102]:
```python
non_zero_Lasso = [x for x in Lasso_model.coef_ if x != 0]
print('Number of non-zero coefficients in Lasso model:',
      len(non_zero_Lasso))
```

```
Number of non-zero coefficients in Lasso model: 28
```

1.2.4 Fit an elastic net regression model on the training set, with α and λ chosen by 10-fold cross-validation.

In [98]:
```python
from sklearn.linear_model import ElasticNetCV
# fitting the ElasticNet model and calculate test MSE
ElasticNet_model = ElasticNetCV(l1_ratio = np.linspace(0.1,1.0,10),
                   normalize = True, cv = 10, n_jobs = -1).fit(
                   GSS_train_features, GSS_train['egalit_scale'])
print("ElasticNetCV test MSE:", sklearn.metrics.mean_squared_error(
GSS_test['egalit_scale'], ElasticNet_model.predict(GSS_test_features)))
```

```
ElasticNetCV test MSE: 61.30153540292484
```

In [88]:
```python
# checking the coefficients
ElasticNet_model.coef_
```

Out[88]:
```
array([-0.03905169, -0.        ,  0.        ,  0.95419783,  0.        ,
        0.15079592,  0.        ,  0.        , -0.        , -0.        ,
        0.04359243,  0.        , -0.        , -0.        , -1.24762186,
        0.20698539,  0.        ,  0.        , -0.11176059,  0.        ,
        0.7098221 , -1.55429862, -0.        ,  0.        , -4.32653709,
        0.        , -0.05345982,  0.95427603,  0.09590645, -0.        ,
       -0.        , -0.        , -0.19627341,  0.17641766, -0.06465349,
       -0.        ,  0.        , -0.3368883 , -1.64294643, -0.        ,
       -0.03431446,  0.        , -0.        ,  0.        , -0.        ,
        0.        , -0.        ,  0.0830095 , -0.86899771, -2.45473177,
       -0.        ,  0.        , -0.        ,  0.        ,  0.        ,
       -1.36451583,  0.        ,  0.        , -0.        , -0.        ,
       -0.        ,  0.        ,  0.        ,  0.        ,  0.04399991,
        1.22664377,  0.        , -0.        ,  0.        ,  0.        ,
        0.30844883, -0.        , -0.14708518, -0.        ,  0.        ,
        0.13100665, -0.        ])
```

In [101]:
```python
non_zero_EN = [x for x in ElasticNet_model.coef_ if x != 0]
print('Number of non-zero coefficients in ElasticNet model:',
      len(non_zero_EN))
```

```
Number of non-zero coefficients in ElasticNet model: 28
```

1.2.5 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?

```
In [105]: print("OLS r2 score:", sklearn.metrics.r2_score(
          GSS_test['egalit_scale'],OLS_model.predict(GSS_test_features)))
          print("RidgeCV r2 score:", sklearn.metrics.r2_score(
          GSS_test['egalit_scale'],Ridge_model.predict(GSS_test_features)))
          print("LassoCV r2 score:", sklearn.metrics.r2_score(
          GSS_test['egalit_scale'],Lasso_model.predict(GSS_test_features)))
          print("ElasticNetCV r2 score:", sklearn.metrics.r2_score(
          GSS_test['egalit_scale'],ElasticNet_model.predict(GSS_test_features)))
```

```
OLS r2 score: 0.30045854148317575
RidgeCV r2 score: 0.3254443964822481
LassoCV r2 score: 0.32161836393792786
ElasticNetCV r2 score: 0.32161836393792786
```

I used the goodness of fit to evaluate the results of the regression models described above. The results show that the accuracy of these regression models for individual egalitarianism prediction is quite low, reaching only about the level of 0.3. In these models, the ridge regression model performs slightly better than other models, but the test MSE values are quite close among all the approaches.

I think an important reason for the poor prediction performance of these models is that they do not capture the point that most variables are ordinal or nominal variables, and our response label is also not continuous. We just simply view them as the same as the continuous numeric variables here, which could bring more errors in prediction. I think maybe classification methods would be better than these regressors for this prediction work within the GSS dataset.