Homework 3 - Linear Model Selection and Regularization

February 9, 2020

Student: Dimitrios Tanoglidis

Due: Sunday, February 9, 2020

[137]:
```python
#Import stuff
import numpy as np
import pandas as pd
import itertools
from sklearn import linear_model
from sklearn.metrics import mean_squared_error
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt
%matplotlib inline
from matplotlib import rcParams
#import seaborn as sns
rcParams['font.family'] = 'serif'

# Adjust rc parameters to make plots pretty
def plot_pretty(dpi=200, fontsize=8):

    import matplotlib.pyplot as plt

    plt.rc("savefig", dpi=dpi)
    plt.rc('text', usetex=True)
    plt.rc('font', size=fontsize)
    plt.rc('xtick', direction='in')
    plt.rc('ytick', direction='in')
    plt.rc('xtick.major', pad=10)
    plt.rc('xtick.minor', pad=5)
    plt.rc('ytick.major', pad=10)
    plt.rc('ytick.minor', pad=5)
    plt.rc('lines', dotted_pattern = [0.5, 1.1])

    return
plot_pretty()
```

1 Training/test error for subset selection

1.1 Generating a dataset

Generate a dataset with $p = 20$ features, $n = 1000$ observations and quantitative responce vector $Y = X\beta + \epsilon$.

[138]:
```
# Set a random seed
np.random.seed(seed=42020)
```

Now, create a matrix of size $n \times p$ (That is 1000×20) with elements drawn from a normal distribution with $\mu = 0$ and $\sigma = 1.0$.

[139]:
```
mu = 0.0
sigma = 1.0

# Dimensions of the matrix
n = 1000
p = 20

X = np.random.normal(mu, sigma, [n,p])
```

Generate now en error vector ϵ (of dimension 1000) and and a vector β of dimension 20. Pick 5 of those in β and set them equal to zero.

[140]:
```
eps = np.random.normal(mu,sigma, 1000)
beta = np.random.normal(mu,sigma, 20)

beta[0] = 0.0
beta[1] = 0.0
beta[2] = 0.0
beta[3] = 0.0
beta[4] = 0.0

np.random.shuffle(beta)
```

Create the responce vector Y.

[141]:
```
Y = np.matmul(X,beta) + eps
```

1.2 Split in training / test set.

[142]:
```
X_train, X_test, y_train, y_test = train_test_split(X,Y,test_size=0.9,␣
↪random_state=42)
```

1.3 Best subset selection

Perform best subset selection on the training set and plot the training set MSE associated with the best model of each size.

Start by writing a function that performs the linear regression and returns the RSS of each model/fit.

Remember that the Residual Sum of Squares is defined as (for n instances):

$$RSS = \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{1}$$

The mean squared error is simply: $MSE = RSS/n$.

[143]:
```
def lin_reg_RSS(X_mat, y_vec):
    # Define linear model
    lin_mod = linear_model.LinearRegression(fit_intercept = True)
    lin_mod.fit(X_mat,y_vec) # Fit the training set
    y_pred = lin_mod.predict(X_mat) # Predict on the training set
    RSS = mean_squared_error(y_vec,y_pred) * len(y_vec) # Calculate RSS
    return RSS
```

Now perform the best subset selection.

[144]:
```
# Loop over the range 1, 2, ..., p (including p)
RSS_list = [] # List that stores the RSS values
features = [] # List that stores the features of each different model
numb_features = [] # List that stores the number of features of each model
p=20

for k in range(1,p+1):
    print("Running for k={}".format(k))
    # Then loop over all the (p,k) models
    for X_sub in itertools.combinations(X_train.T,k):
        X_sub = np.asarray(X_sub) #Submatrix that has features for the specific␣
 ↪case
        RSS_loc = lin_reg_RSS(X_sub.T,y_train) # Local RSS from the linear␣
 ↪regression
        # Now update the arrays - append
        RSS_list.append(RSS_loc)
        features.append(X_sub.T)
        numb_features.append(len(X_sub))
```

```
Running for k=1
Running for k=2
Running for k=3
Running for k=4
Running for k=5
```

```
Running for k=6
Running for k=7
Running for k=8
Running for k=9
Running for k=10
Running for k=11
Running for k=12
Running for k=13
Running for k=14
Running for k=15
Running for k=16
Running for k=17
Running for k=18
Running for k=19
Running for k=20
```

Now just save somewhere all the possible combinations. That's gonna be useful to build the model.

[145]:
```python
iterable = np.arange(1,21)
combinations  = []

for k in range(1,p+1):
    for combs in itertools.combinations(iterable,k):
        combs = np.asarray(combs)
        combinations.append(combs)
```

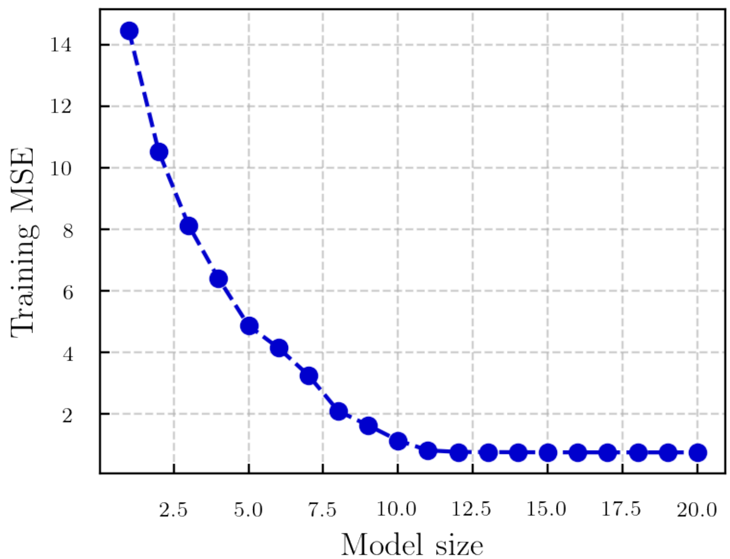
[146]:
```python
# Now create a data frame that contains the three lists
df = pd.DataFrame({'RSS_list':RSS_list, 'features':features,
                   'combinations':combinations, 'numb_features': numb_features})
#df = pd.DataFrame({'features':features, 'numb_features': numb_features})
```

[147]:
```python
Min_RSSs = np.zeros(p) # This is an array where we will store the minimum RSS␣
 ↪values

for i in range(1,p+1):
    # Create a local data frame that keeps all models with number of features i
    df_loc =df.loc[df['numb_features'] == i]
    # Create an array that contain the RSS values of all models with
    # i number of features
    RSS_loc_list = np.asarray(df_loc['RSS_list'])
    # Keep the minimum of the model with i elements
    Min_RSSs[i-1] = np.min(RSS_loc_list)
```

We have the minimum RSS values for each number of features values. We estimate the MSE values as RSS/n, where $n = 100$ in our case.

[148]:
```python
MSE_train = Min_RSSs/100.
feat_arr = np.arange(1,21)
```

```python
plt.figure(figsize=(4.2,3.2))

plt.plot(feat_arr, MSE_train, c='mediumblue', ls='--', marker='o')

plt.xlabel('Model size', fontsize=12);plt.ylabel('Training MSE', fontsize=12)
plt.grid(ls='--', alpha=0.6)
plt.show()
```



[149]:
```python
print(feat_arr[MSE_train==np.min(MSE_train)])
```

```
[20]
```

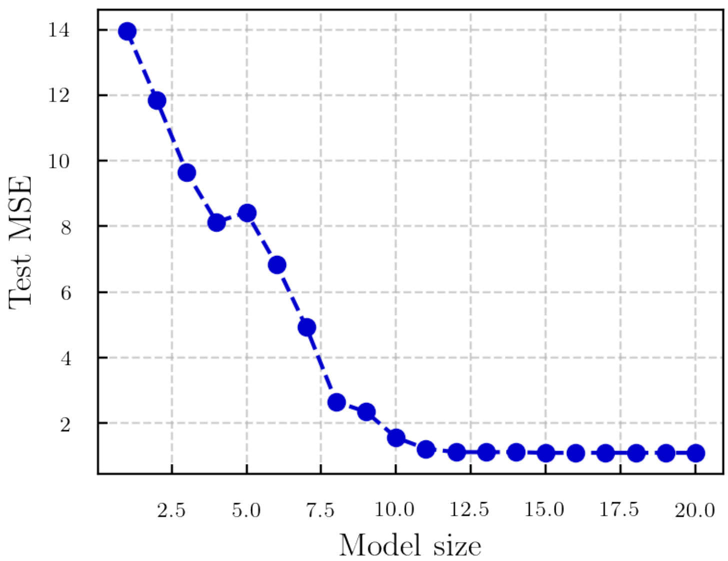
The minimum is achieved for the model with all predictors ($p = 20$).

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

[189]:
```python
MSE_test = np.zeros(p) #Store the MSE of the test set

for i in range(1,p+1):
    # Create a local data frame that keeps all models with number of features i
    df_loc =df.loc[df['numb_features'] == i]
    # Create a matrix that contains the feature matrices of all models with
    # i number of features
    X_mat_loc = np.asarray(df_loc['features'])
    # Create a list that contains the RSS of all models with i number of features
    RSS_loc_list = np.asarray(df_loc['RSS_list'])
    # Find the minimum
```

```
    min_RSS_loc = np.min(RSS_loc_list)
    a = np.arange(len(X_mat_loc))
    a_up = a[RSS_loc_list==min_RSS_loc]
    X_train_loc = X_mat_loc[a_up[0]]
    #print(np.shape(X_train_loc))
    # ================================
    # Keep the test feature matrix that corresponds to the minimum RSS
    # Get the combinations of all models with i number of features
    combinations_loc = np.asarray(df_loc['combinations'])
    # Combination of features corresponding to the minimum
    combs_min = combinations_loc[RSS_loc_list==min_RSS_loc]
    combs_min = combs_min[0]-1




    # Create the feature matrix
    X_test_loc = X_test.T[combs_min].T



    # Fit a linear model on the training set
    lin_mod = linear_model.LinearRegression(fit_intercept = True)
    lin_mod.fit(X_train_loc,y_train) # Fit the training set
    y_pred_loc = lin_mod.predict(X_test_loc) # Predict on the test set
    MSE_loc = mean_squared_error(y_test,y_pred_loc) # Calculate MSE
    MSE_test[i-1] = MSE_loc
```

[190]:
```
plt.figure(figsize=(4.2,3.2))

plt.plot(feat_arr, MSE_test, c='mediumblue', ls='--', marker='o')

plt.xlabel('Model size', fontsize=12);plt.ylabel('Test MSE', fontsize=12)
plt.grid(ls='--', alpha=0.6)
plt.show()
```



1.5 Best Model size

[192]:
```
print(feat_arr[MSE_test==np.min(MSE_test)])
```

```
[15]
```

The model fro which minimum MSE is achieved is for 15 features.

1.6 True and best model

[194]:
```
coeffs = []
for i in range(15,16): #Keep only the model 15
    # Create a local data frame that keeps all models with number of features i
    df_loc =df.loc[df['numb_features'] == i]
    # Create a matrix that contains the feature matrices of all models with
    # i number of features
    X_mat_loc = np.asarray(df_loc['features'])
    # Create a list that contains the RSS of all models with i number of features
    RSS_loc_list = np.asarray(df_loc['RSS_list'])
    # Find the minimum
    min_RSS_loc = np.min(RSS_loc_list)
    a = np.arange(len(X_mat_loc))
    a_up = a[RSS_loc_list==min_RSS_loc]
    X_train_loc = X_mat_loc[a_up[0]]
    #print(np.shape(X_train_loc))
    # ===============================
```

```
    # Keep the test feature matrix that corresponds to the minimum RSS
    # Get the combinations of all models with i number of features
    combinations_loc = np.asarray(df_loc['combinations'])
    # Combination of features corresponding to the minimum
    combs_min = combinations_loc[RSS_loc_list==min_RSS_loc]
    combs_min = combs_min[0]-1



    # Create the feature matrix
    X_test_loc = X_test.T[combs_min].T


    # Fit a linear model on the training set
    lin_mod = linear_model.LinearRegression(fit_intercept = True)
    lin_mod.fit(X_train_loc,y_train) # Fit the training set
    coef  = lin_mod.coef_
    coeffs.append(coef)
```

[195]:
```
# Coefficients of the best model
print(coeffs)
```

```
[array([ 0.08381228,  1.26974877,  0.8060982 ,  0.59523776, -1.18213344,
        0.07783395, -0.55277929,  2.15440778, -1.24488641,  1.05978406,
        0.05355805, -1.07334115,  1.35713046, -1.42591147,  0.27746017])]
```

[197]:
```
# Real coefficients
print(beta)
```

```
[ 0.          1.24887504  1.02291413  0.          0.54608262  0.05866911
 -1.12012887  0.0091939  -0.56141357  0.          2.05638529  0.
 -1.32818211  0.96565878  0.14881125 -1.16400088  1.18491951 -1.38323677
  0.26502957  0.        ]
```

We see that the coeffiecients that are actually 0 tend to be smaller, and in some cases there is some agreement between the coefficients, but overall it is not the case.

1.7 Plot

[ ]:
```

```

2 Application Excersize - General Social Survey analysis

Let's first read the data.

[66]:
```python
df_train = pd.read_csv('gss_train.csv') #Train data
df_test = pd.read_csv('gss_test.csv') #Test data

# Get feature matrices
X_train = np.asarray(df_train.loc[:, df_train.columns != 'egalit_scale'])
X_test = np.asarray(df_test.loc[:, df_test.columns != 'egalit_scale'])
# Get responses
y_train = np.asarray(df_train['egalit_scale'])
y_test = np.asarray(df_test['egalit_scale'])
df_train.head()
```

[66]:
```
   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]
```

2.1 Least squares fit

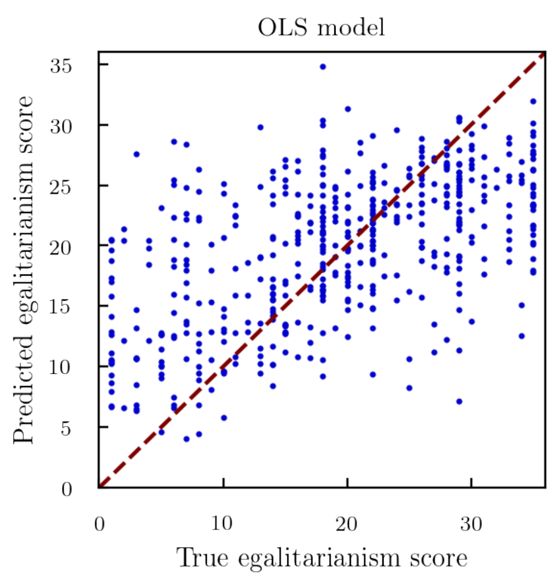
[31]:
```
OLS = linear_model.LinearRegression(fit_intercept = True)
OLS.fit(X_train,y_train)
y_pred_ols = OLS.predict(X_test)
# ==========================
MSE_ols = mean_squared_error(y_test,y_pred_ols)

print(MSE_ols)
```

```
63.213629623014995
```

The test MSE of the ordinarey least squares linear model is ~ 63.2.

Let's plot the predicted versus the true value, to see how our model performs.

[47]:
```
x_lin = np.linspace(0,36,200)
plt.figure(figsize=(3.0,3.0))

plt.scatter(y_test,y_pred_ols,color='mediumblue', s=1.5)
plt.plot(x_lin,x_lin,c='maroon',ls='--')

plt.xlabel('True egalitarianism score',fontsize=10)
plt.ylabel('Predicted egalitarianism score',fontsize=10)
plt.title('OLS model')
plt.xlim(0,36);plt.ylim(0,36)
plt.show()
```



2.2 Ridge regression model

Now fit a ridge regression model on the training set, with the λ parameter chosen by 10-fold cross-validation.

For that reason I will use the the GridSearchCV class from scikit-learn.

[52]:
```
from sklearn.model_selection import GridSearchCV
param_grid = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1.0, 10.0,100,1000]}
Ridge = linear_model.Ridge(alpha=1.0, fit_intercept=True)

grid = GridSearchCV(Ridge, param_grid,cv=10, scoring='neg_mean_squared_error')
grid.fit(X_train, y_train)
print(grid.best_params_)
```

```
{'alpha': 100}
```

So, the best parameter, in the way we defined the grid, is $\lambda = 100$ (or α as named in the scikit-learn Ridge regression class).

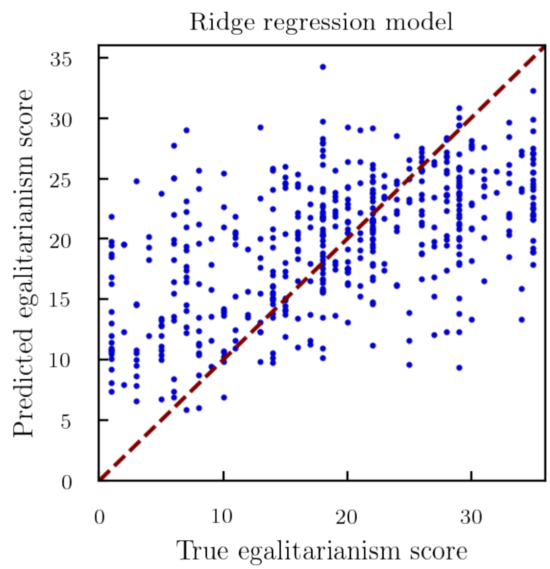
Let's run again the model, using this regularization parameter.

[60]:
```
Ridge = linear_model.Ridge(alpha=100, fit_intercept=True)
Ridge.fit(X_train,y_train)
y_pred_Ridge = Ridge.predict(X_test)
# ==========================
MSE_Ridge = mean_squared_error(y_test,y_pred_Ridge)

print(MSE_Ridge)
```

```
62.256410822269125
```

The Ridge regression model (with $\lambda = 100$ chosen from a 10-fold CV process) gives a MSE on the test set MSE ~ 62.25. That means, not so great.

[68]:
```
plt.figure(figsize=(3.0,3.0))

plt.scatter(y_test,y_pred_Ridge,color='mediumblue', s=1.5)
plt.plot(x_lin,x_lin,c='maroon',ls='--')

plt.xlabel('True egalitarianism score',fontsize=10)
plt.ylabel('Predicted egalitarianism score',fontsize=10)
plt.title('Ridge regression model')
plt.xlim(0,36);plt.ylim(0,36)
plt.show()
```



2.3 Lasso regression model

Now fit a lasso regression model on the training set, with the λ parameter chosen by 10-fold cross-validation.

[61]:
```
param_grid = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1.0, 10.0,100,1000]}
Lasso = linear_model.Lasso(alpha=1.0, fit_intercept=True)

grid = GridSearchCV(Lasso, param_grid,cv=10, scoring='neg_mean_squared_error')
grid.fit(X_train, y_train)
print(grid.best_params_)
```

```
{'alpha': 0.1}
```

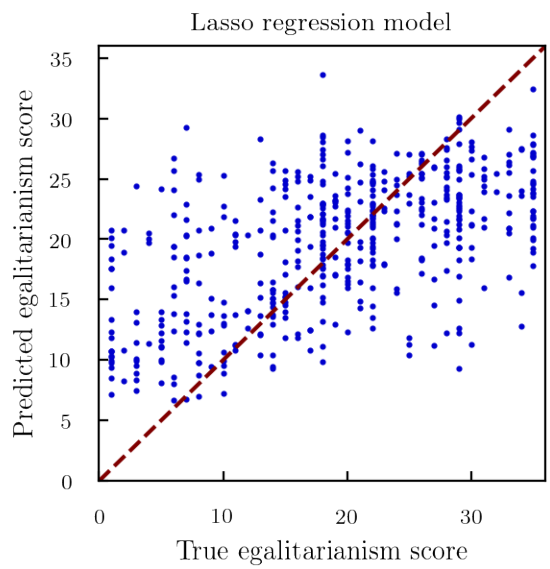
The best λ parameter (or α, as called here) is $\lambda = 0.1$. Re-fit the model with this parameter and re-calculate.

[71]:
```
Lasso = linear_model.Lasso(alpha=0.1, fit_intercept=True)
Lasso.fit(X_train,y_train)
y_pred_Lasso = Lasso.predict(X_test)
# ==========================
MSE_Lasso = mean_squared_error(y_test,y_pred_Lasso)

print(MSE_Lasso)
```

```
62.77841555477389
```

The MSE is high; higher than the one obtained using the Ridge regression method, but less than that of the OLS model. Let's plot to see.

[72]:
```
plt.figure(figsize=(3.0,3.0))

plt.scatter(y_test,y_pred_Lasso,color='mediumblue', s=1.5)
plt.plot(x_lin,x_lin,c='maroon',ls='--')

plt.xlabel('True egalitarianism score',fontsize=10)
plt.ylabel('Predicted egalitarianism score',fontsize=10)
plt.title('Lasso regression model')
plt.xlim(0,36);plt.ylim(0,36)
plt.show()
```



2.4 Elastic net regression

Again, select α and λ parameters using 10-fold cross validation.

Note that sklearn has a strange implementation of the elastic net regression, with the parameters `alpha`, `l1_ratio`. These are connected to the parameters α, λ as:

`alpha` $= \alpha + \lambda$

`l1_ratio` $= \frac{\alpha}{\alpha+\lambda}$.

`l1_ratio` get's values in the range $[0, 1]$. Here I will peak different values in alpha, as before, and values in l1 ratio in the above range.

[70]:
```
param_grid = {'alpha':[0.0001, 0.001, 0.01, 0.1, 1.0, 10.0,100,1000],
              'l1_ratio': [ 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]}
ElasticNet = linear_model.ElasticNet(alpha=1.0, l1_ratio=0.5,␣
 ↪fit_intercept=True)
```

```
grid = GridSearchCV(ElasticNet, param_grid,cv=10,␣
↪scoring='neg_mean_squared_error')
grid.fit(X_train, y_train)
print(grid.best_params_)
```

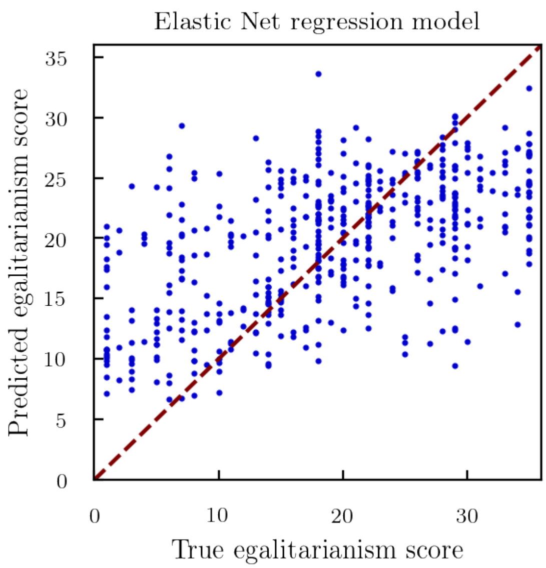
```
{'alpha': 0.1, 'l1_ratio': 0.9}
```

[75]:
```
ElasticNet = linear_model.ElasticNet(alpha=0.1,l1_ratio=0.9, fit_intercept=True)
ElasticNet.fit(X_train,y_train)
y_pred_Elastic = ElasticNet.predict(X_test)
# ==========================
MSE_Elastic = mean_squared_error(y_test,y_pred_Elastic)

print(MSE_Elastic)
```

```
62.73277839229572
```

The MSE of the elastic net is ~ 62.73, similar to that of the Lasso regression. Let's plot true and predicted egalitarianism score.

[76]:
```
plt.figure(figsize=(3.0,3.0))

plt.scatter(y_test,y_pred_Elastic,color='mediumblue', s=1.5)
plt.plot(x_lin,x_lin,c='maroon',ls='--')

plt.xlabel('True egalitarianism score',fontsize=10)
plt.ylabel('Predicted egalitarianism score',fontsize=10)
plt.title('Elastic Net regression model')
plt.xlim(0,36);plt.ylim(0,36)
plt.show()
```



Let's also report the non-zero coefficient estimates.

[79]:
```
coeffs = ElasticNet.coef_

print(len(coeffs[coeffs!=0]))
```

```
27
```

We see that we have 27 non-zero coefficient estimates (from the 77 predictors). Hopefully, the zodiacs give zero estimates. Let's print the non zero coefficients.

[80]:
```
print((coeffs[coeffs!=0]))
```

```
[-0.06471542 -0.00520684  0.41619963  0.33500247 -0.2793271  -0.10907867
 -1.49518207  0.36476309 -0.13146534  0.66171997 -1.73634255  0.06524264
 -3.99011224 -0.14037874  0.91743958  0.15836637 -0.02264782 -0.30664922
  0.26449128 -0.12806461  0.01342779 -1.13925132  0.01102697 -1.17268022
 -2.46829444 -0.0224217   1.17533535]
```

2.5 Comments

In all cases the MSE was large (~ 63) and the errors were similar in all models (OLS, Ridge, Lasso, Elastic Net). It seems that it is very hard to predict a person's egalitarianism.

[ ]:
```

```