HW3_Python

February 9, 2020

[1]:
```
%matplotlib inline
import pandas as pd
import seaborn as sns
import numpy as np
from math import sqrt
import itertools
import time
import statsmodels.api as sm
import matplotlib.pyplot as plt
from sklearn.linear_model import RidgeCV, LassoCV, ElasticNetCV, ElasticNet
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
import warnings
warnings.simplefilter("ignore")
```

[2]:
```
np.random.seed(0)
```

0.1 Conceptual Exercises

[3]:
```
# Data Generation
p, n = 20, 1000
X = np.random.rand(n, p)
beta = np.random.rand(p)
beta[[2,4,10,15]] = 0 ## setting 3 elements of beta to zero
e = np.random.normal(0, 0.5, 1000) ## setting error to follow a normal␣
 ↪distribution
Y = np.dot(X, beta) + e
```

[4]:
```
# Split dataset
data = pd.DataFrame(X)
data['Y'] = Y
train, test = train_test_split(data, shuffle=False, test_size = 0.9)
train.shape, test.shape
```

[4]:
```
((100, 21), (900, 21))
```

0.1.1 Find Best Models

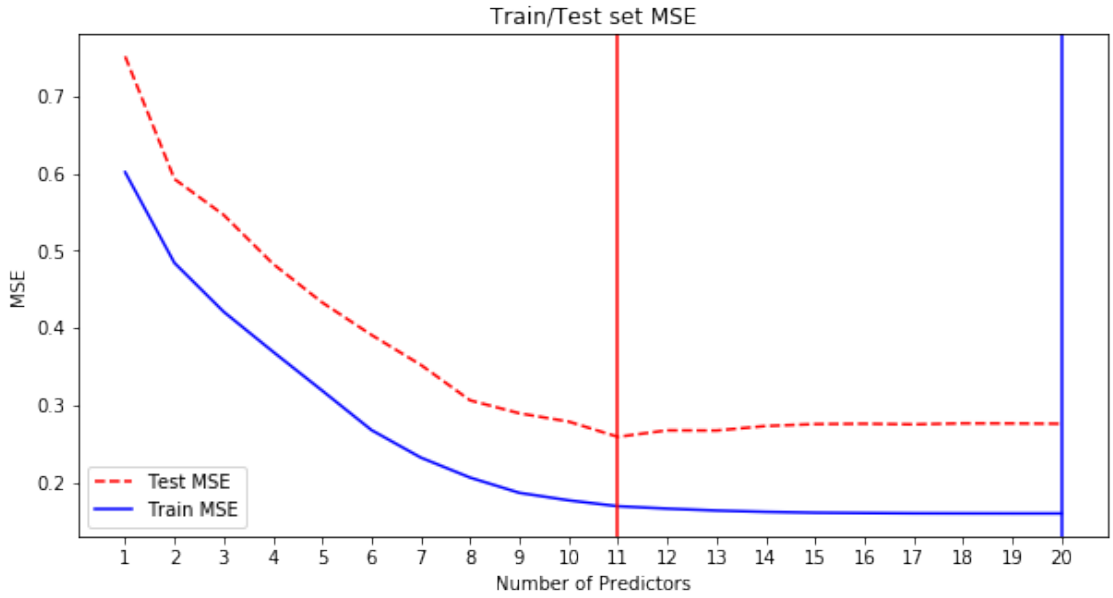
[5]:
```
def processSubset(feature_set, df):
    regr = LinearRegression().fit(df[list(feature_set)], df['Y'])
    MSE = ((regr.predict(df[list(feature_set)]) - df['Y']) ** 2).mean()
    return {"Model":regr, "MSE":MSE, 'Fset': list(feature_set)}

def getBest(k, df):
    results = []
    for ft_set in itertools.combinations(df.columns[:-1], k):
        results.append(processSubset(ft_set, df))
    models = pd.DataFrame(results)
    best_model = models.loc[models['MSE'].idxmin()]
    return best_model

def gentestMSE(result, test_df):
    test_mse = dict()
    for i in range(1, result.shape[0]+1):
        model = result.loc[i]['Model']
        Fset = result.loc[i]['Fset']
        MSE = ((model.predict(test_df[list(Fset)]) - test_df['Y']) ** 2).mean()
        test_mse[i] = MSE
    return test_mse
```

[6]:
```
models_best = pd.DataFrame(columns=["MSE", "Model", 'Fset'])
for i in range(1,p+1):
    print('start model of size', i)
    models_best.loc[i] = getBest(i, train)
```

```
start model of size 1
start model of size 2
start model of size 3
start model of size 4
start model of size 5
start model of size 6
start model of size 7
start model of size 8
start model of size 9
start model of size 10
start model of size 11
start model of size 12
start model of size 13
start model of size 14
start model of size 15
start model of size 16
start model of size 17
start model of size 18
```

```
start model of size 19
start model of size 20
```

[15]:
```
test_mse = list(gentestMSE(models_best, test).values())
mse = list(models_best["MSE"])

plt.figure(figsize=(10,5))
plt.rcParams.update({'font.size': 10, 'lines.markersize': 10})
plt.plot(range(1,21),test_mse, 'k--', color = 'red', label = 'Test MSE')
plt.plot(range(1,21),mse, color = 'blue', label='Train MSE')
plt.axvline(x=test_mse.index(min(test_mse))+1, color = 'red')
plt.axvline(x=mse.index(min(mse))+1, color = 'blue')
## +1 because the size of predictors starts from 1
plt.title('Train/Test set MSE')
plt.xlabel('Number of Predictors')
plt.ylabel('MSE')
plt.xticks(np.arange(1, 21, step=1))
plt.legend()
plt.show()
```

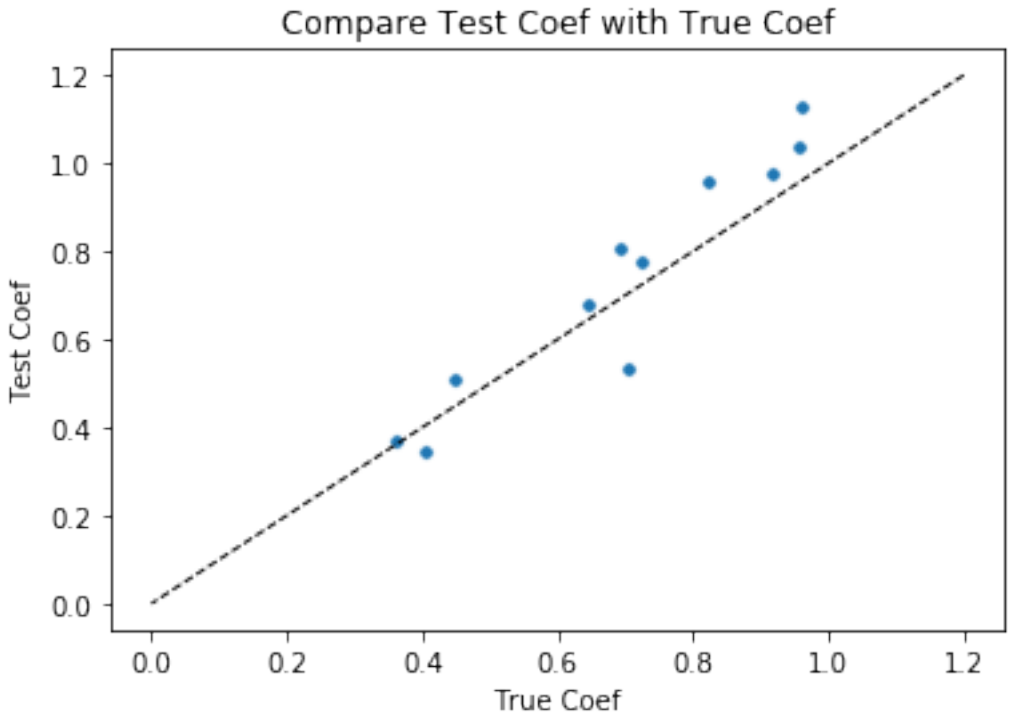


In terms of minimizing MSE, the best model for the train set has 20 predictors, and the best model for the test set has 12 predictors. Best subset selection loops through all possible combinations for `i` predictors for `i in range(1, all available predictors)`. Because adding features can never make worse predictions in the train set, we observe that the best model for the train set has all possible predictors. However, after having more than 7 predictors, adding new features can only reduces minimal MSE; that is, the curve for MSE gets flatten. Also, because more features add complexity and thus add variance, the best model for the train set is not the model with the most predictors. Instead, only 11 predictors are enough.

0.1.2 Coefficient Sizes

[19]:
```
best_test_model = models_best.loc[11]['Model']
test_coef = best_test_model.coef_
true_coef = beta[models_best.loc[11]['Fset']]
```

[20]:
```
sns.scatterplot(y = test_coef, x = true_coef, s = 30)
plt.plot([1.2, 0], [1.2, 0], 'k--', linewidth=1, color = 'black')
plt.title('Compare Test Coef with True Coef')
plt.xlabel('True Coef')
plt.ylabel('Test Coef')
plt.xticks(np.arange(0, 1.4, step=0.2))
plt.yticks(np.arange(0, 1.4, step=0.2))
plt.show()
```



[21]:
```
print('Predictors that should not be deleted but deleted by subset selection')
target = set(range(1,21)) - set(models_best.loc[12]['Fset']) - set([2,4,10,15])
print(sorted(target))
print('True coefs for the missing predictors:')
print(beta[np.array(sorted(target))-1])
```

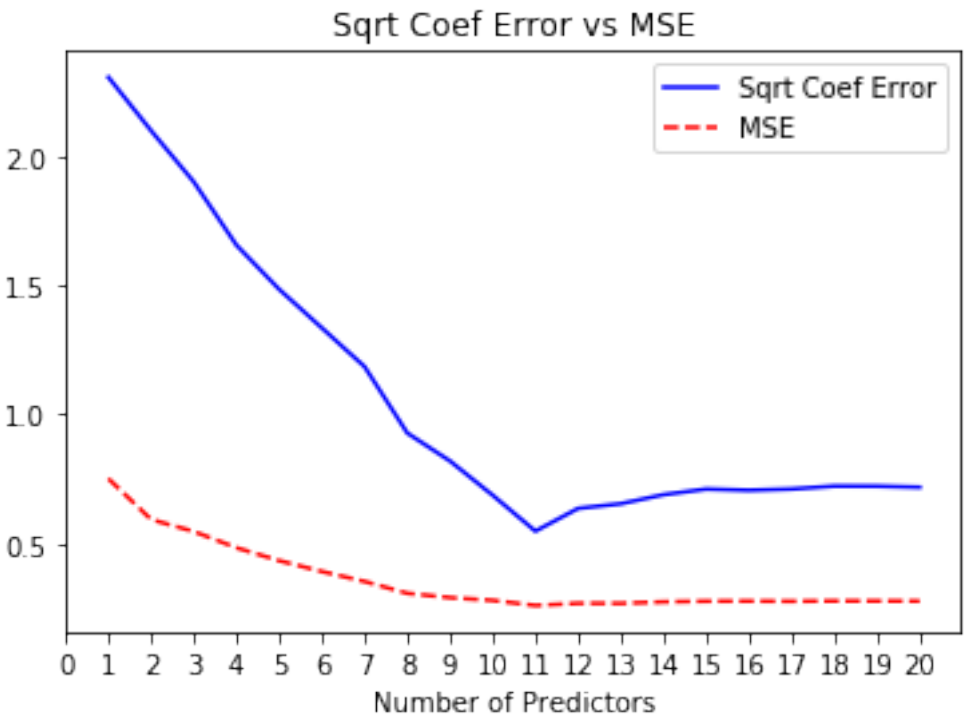
```
Predictors that should not be deleted but deleted by subset selection
[1, 9, 14, 20]
True coefs for the missing predictors:
```

```
[0.39217296 0.6468967  0.14222247 0.44599407]
```

As shown above, the coefficient sizes for the model with least test set MSE are slightly larger than the true coefficient sizes. This is because our subset selection led us exclude four predictors (1, 9, 13, 20) with true nonzero coefficients. Because all of these missing coefficients are positive, the rest of the coefficients in the best test model had to make this up by getting larger. Thus, all coefficient estimates are biased up.

[23]:
```
coef_err = []
for i in range(1, 21):
    coef = models_best.loc[i]['Model'].coef_
    test_coef = np.zeros(20)
    test_coef[models_best.loc[i]['Fset']] = coef
    true_coef = np.array(beta)
    error = sqrt(((test_coef - true_coef) ** 2).sum())
    coef_err.append(error)

coef_err_2 = []
for i in range(1, 21):
    test_coef = models_best.loc[i]['Model'].coef_
    true_coef = beta[models_best.loc[i]['Fset']]
    error = sqrt(((test_coef - true_coef) ** 2).sum())
    coef_err_2.append(error)
```

[40]:
```
plt.plot(range(1,21), coef_err, label = 'Sqrt Coef Error', color = 'blue')
#plt.plot(coef_err_2, label = 'Sqrt Coef Error')
plt.plot(range(1,21), test_mse, 'k--', color = 'red', label = 'MSE')
plt.title('Sqrt Coef Error vs MSE')
plt.xlabel('Number of Predictors')
plt.xticks(np.arange(0, 21, step=1))
plt.legend()
plt.show()
```



Here we focus on the squared root of the sum of the coefficient errors for `r in range(1, 20)`. Generally speaking, if only a small amount of predictors are allowed, the coefficients in our model need to be biased in order to make up the missing predictors, and therefore yeild higher coefficient errors. Therefore, as we increase the number of predictors, the sqrt coefficient error reduces. But this pattern ends when the number of predictors is over 11. The model with lowest test MSE is exactly the model with 11 predictors. Although the absolute value is different, `Sqrt Coef Error` and `test MSE` increases and decreases at the same time. This confirms the intuition that, increasing the number of predictors reduces model biases but increases model variances.

0.2 Application Exercises

[27]:
```
gss_test = pd.read_csv('gss_test.csv')
gss_train = pd.read_csv('gss_train.csv')
```

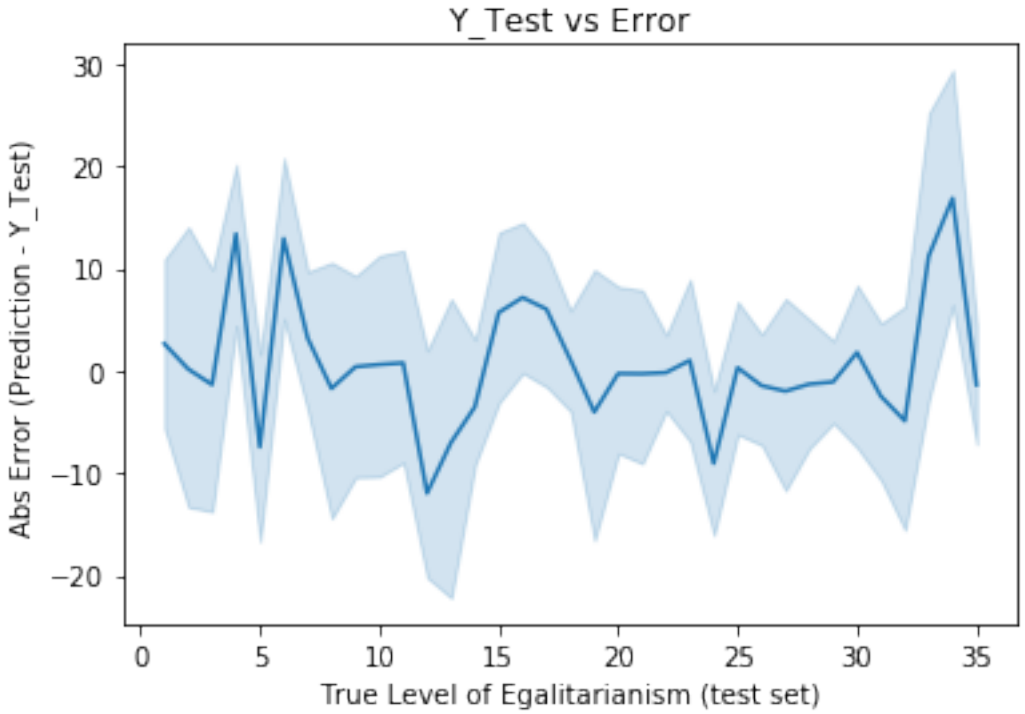
[28]:
```
Y_train = gss_train['egalit_scale']
X_train = gss_train[gss_train.columns[~gss_train.columns.
 ↪isin(['egalit_scale'])]]

Y_test = gss_test['egalit_scale']
X_test = gss_test[gss_test.columns[~gss_test.columns.isin(['egalit_scale'])]]
```

Least Square Linear Model Reference: https://towardsdatascience.com/linear-regression-using-least-squares-a4c3456e8570

[41]:
```
ols = LinearRegression().fit(X_train, Y_train)
Y_pred = ols.predict(X_test)
MSE = ((Y_pred - Y_test) ** 2).mean()
print('Least Squares Linear Model MSE:', MSE)
```

```
Least Squares Linear Model MSE: 63.213629623014995
```

[43]:
```
error = Y_pred - Y_test
sns.lineplot(x = Y_train, y = error)
plt.title('Y_Test vs Error')
plt.xlabel('True Level of Egalitarianism (test set)')
plt.ylabel('Abs Error (Prediction - Y_Test)')
plt.show()
```



Ridge, Lasso, and Elastic Net Regression Model Reference: * http://www.science.smith.edu/~jcrouser/SDS293/labs/lab10-py.html * https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.ElasticNetCV.html

[32]:
```
len(X_train.columns)
```

[32]: 77

[49]:
```
ridge = RidgeCV(cv = 10)
ridge.fit(X_train, Y_train)
pred = ridge.predict(X_test)
print('Ridge Regression with 10-fold CV MSE:\n', ((pred - Y_test) ** 2).mean())
```

```
Ridge Regression with 10-fold CV MSE:
 62.499202439578085
```

[50]:
```
lasso = LassoCV(cv = 10)
lasso.fit(X_train, Y_train)
pred = lasso.predict(X_test)
print('Lasso Regression with 10-fold CV MSE:\n', ((pred - Y_test) ** 2).mean())
nonzero_coef = list(X_train.columns[lasso.coef_!=0])
print(f'There are {len(nonzero_coef)} non-zero coefficient estimates')
```

```
Lasso Regression with 10-fold CV MSE:
 62.7780157899344
There are 24 non-zero coefficient estimates
```

[51]:
```
options = np.linspace(0,1,11,endpoint=True)
elasticNet = ElasticNetCV(alphas = options, l1_ratio = options, cv = 10)
elasticNet.fit(X_train, Y_train)
pred = elasticNet.predict(X_test)
print('Elastic Net Regression with 10-fold CV MSE:\n', ((pred - Y_test) ** 2).
 ↪mean())
nonzero_coef = list(X_train.columns[elasticNet.coef_!=0])
print(f'There are {len(nonzero_coef)} non-zero coefficient estimates')
print(f'alpha is {elasticNet.l1_ratio_}, lambda is {elasticNet.alpha_}')
```

```
Elastic Net Regression with 10-fold CV MSE:
 62.7784155547739
There are 24 non-zero coefficient estimates
alpha is 1.0, lambda is 0.1
```

For an elastic net regression model as such:

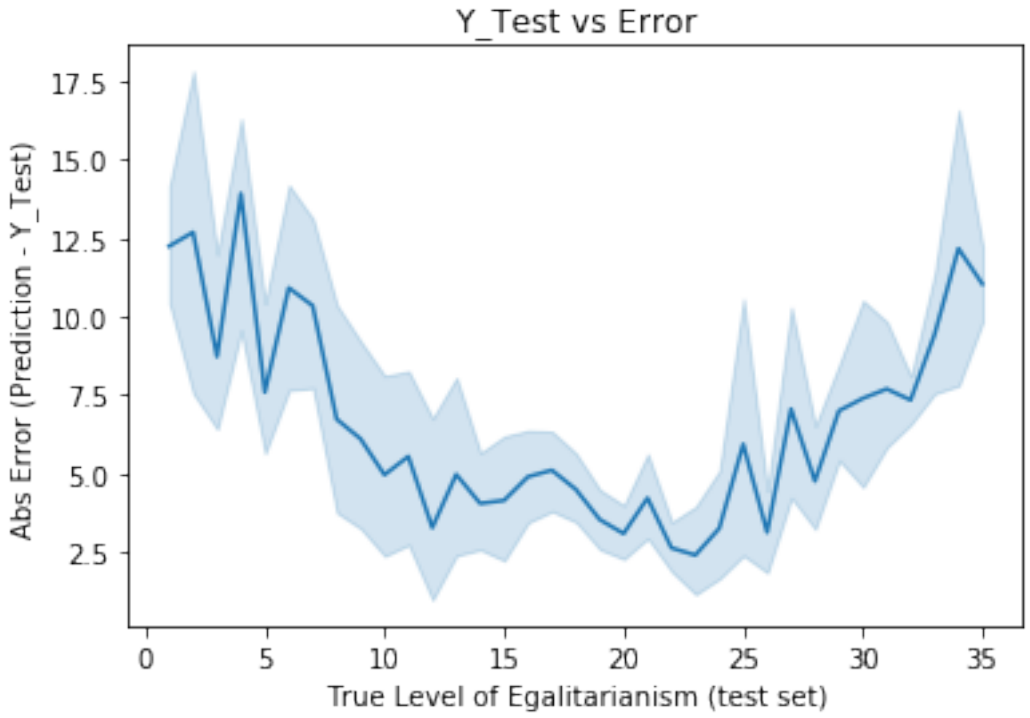
$$RSS + \lambda(\frac{1-\alpha}{2}\sum_{j=1}^{m}\hat{\beta}_j^2 + \alpha\sum_{j=1}^{m}|\hat{\beta}_j|) \tag{1}$$

the best model combination is: $\alpha = 1.0, \lambda = 0.1$

[36]:
```
elasticNet = ElasticNet(alpha = lasso.alpha_, l1_ratio = 1)
elasticNet.fit(X_train, Y_train)
pred = elasticNet.predict(X_test)
print('Elastic Net Regression with best lambda in Lasso CV and alpha = 1.0:\n',
      ((pred - Y_test) ** 2).mean())
```

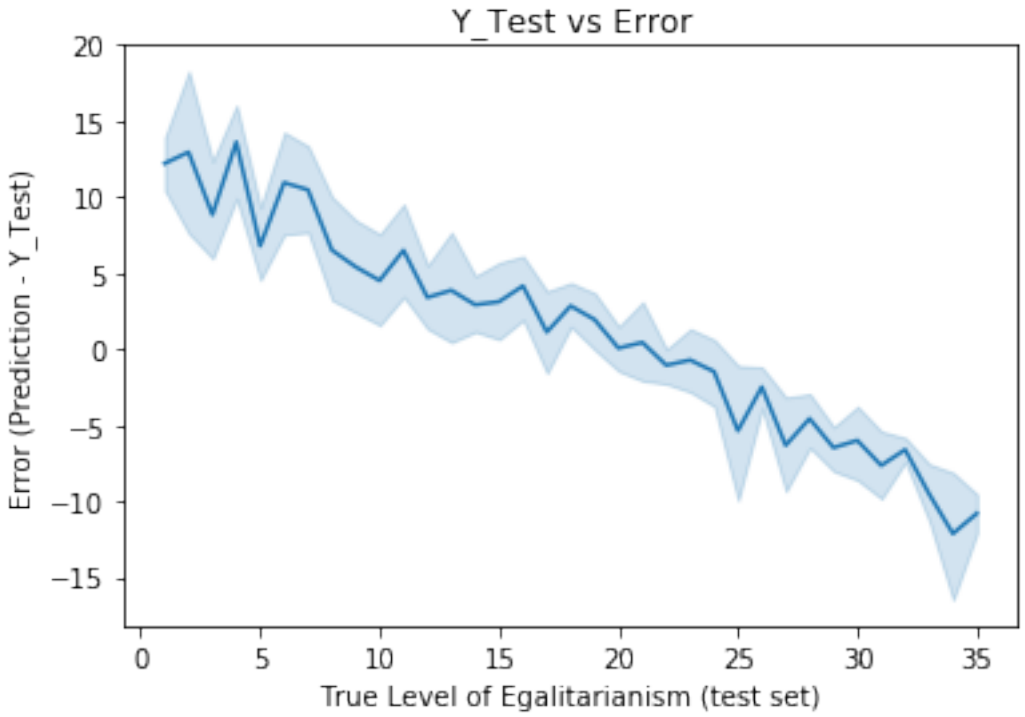
```
Elastic Net Regression with best lambda in Lasso CV and alpha = 1.0:
 62.7780157899344
```

[37]:
```
pred = lasso.predict(X_test)
error = abs(pred - Y_test)
sns.lineplot(x = Y_test, y = error)
plt.title('Y_Test vs Error')
plt.xlabel('True Level of Egalitarianism (test set)')
plt.ylabel('Abs Error (Prediction - Y_Test)')
plt.show()
print(lasso.alpha_)
```




```
0.09991906177130906
```

[39]:
```
pred = ridge.predict(X_test)
#error = abs(pred - Y_test)
error = pred - Y_test
sns.lineplot(x = Y_test, y = error)
plt.title('Y_Test vs Error')
plt.xlabel('True Level of Egalitarianism (test set)')
plt.ylabel('Error (Prediction - Y_Test)')
plt.show()
print(ridge.alpha_)
```




```
0.1
```

In summary, Ordinal Least Squares Linear Model performs the best with a MSE of 63.21. The best models selected by Ridge, Lasso, and Elastic Net Regression Methods have about the same MSE at 62-63. Particularly, Elastic Net Regression suggests an alpha of 1.0, meaning that it ends up selecting a Lasso model. The slight difference in MSE between ElasticNetCV snd LassoCV is because the best lambda selected by LassoCV is about 0.99, which is not an option we tried in ElasticNetCV. I showed above that if using the best lambda selected by LassoCV and alpha = 1.0, the MSE for the best Elastic Net model is the same as the best Lasso model.

As shown in the graph (Y-test vs Error) above, Ridge, Lasso, and Elastic Net Regression Methods seem to have strong biases. As the true level of egalitarianism increases, error increases at first and then decreases. In other words, our models tend to overestimate the egalitarianism level of those who care less about egalitarianism, and underestimate the egalitarianism level of those who care less more egalitarianism. This suggests systemetic bias in our models.

As a comparison, least squares linear model performs the best and it does not have such a systemetic bias–as defined by the nature of least squares linear models. This systemetic bias implies that either some non-zero coefficients are omitted (overly minimized) by the Ridge, Lasso, and Elastic Net Regressions, or there are still some other variables missing in the whole data that affect the level of egalitarianism.

[ ]: