Ridge, Lasso and Elastic net

February 17, 2020

1 RIDGE, LASSO AND ELASTIC NET

1.0.1 Adapted from Seth Flaxman

Questions to Mariana Clare (mc4117@ic.ac.uk) First, you will familiarize yourself with the maths underlying the ridge and lasso penalties. Then you will write code to investigate bias and variance in the context of ridge, lasso, and the elastic net. Finally you will investigate Principal Components Analysis (PCA) as an alternative dimensionality reduction method.

1.1 Ridge and lasso

Let us considered squared error loss, i.e. in linear regression. The ridge penalty adds an L_2 norm to a loss function: $\sum(y_i - x_i\beta)^2 + \lambda\|\beta\|_2^2$

The lasso penalty adds an L_1 norm to a loss function: $\sum(y_i - x_i\beta)^2 + \lambda|\beta|_1^1$

The loss function for ridge leads to the following optimization problem:

$\text{argmin}_\beta \sum(y_i - x_i\beta)^2 + \lambda\|\beta\|_2^2$

- Assume that $\beta \in R^p$ is a p-dimensional vector and rewrite the loss function with the L_2 norm using the vector elements $\beta_1, \beta_2, \ldots, \beta_p$.
- Similarly, write the loss function for the lasso using vector elements.
- To minimize these loss functions, we need to find their gradients. Find the gradient of the ridge penalized loss function. What parameter are you taking the gradient with respect to?

1.2 Bias vs. variance

Ridge and lasso introduce *bias* by shrinking the parameters in a model towards zero. Here is the code from the unbiased linear regression demo in class:

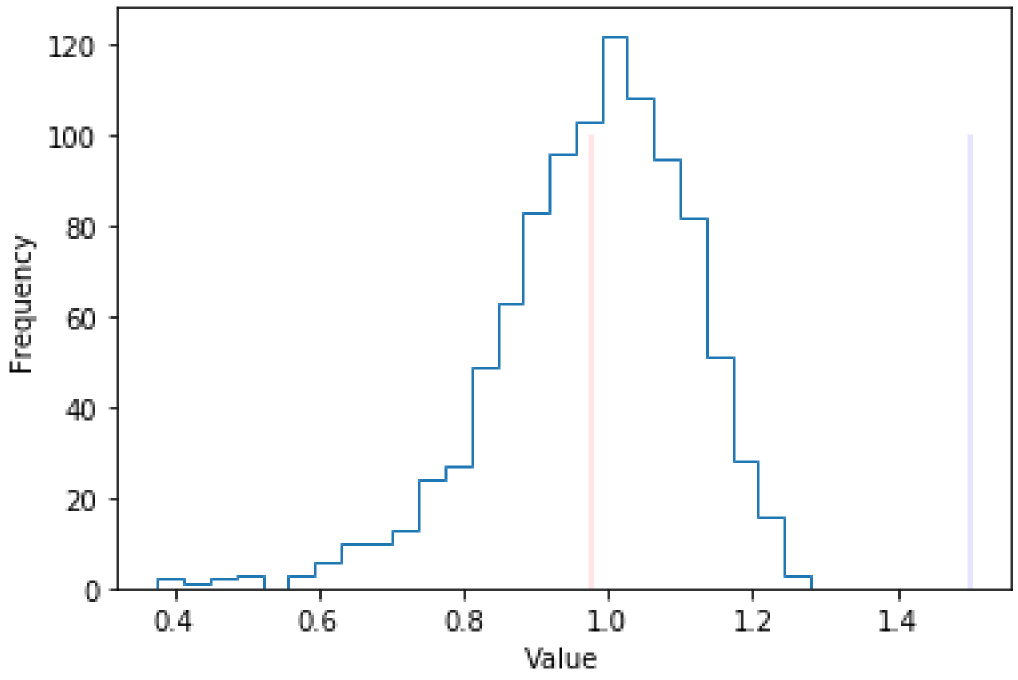
[2]:
```
import numpy as np
import pylab as plt
from scipy.optimize import minimize

def yhat(beta, x):
    return beta*x
```

```python
def squared_loss(beta):
        # standard OLS
    return sum((y - yhat(beta,x))**2)

result = []
fig = plt.figure()
for i in range(1000):
    n = 50
    x = np.random.randn(n)
    u = np.linspace(-5, 5, 11)
    error = np.random.randn(n) * .5
    y = 1.5 * x + error
    x_0 = -1
    ##### minimize square loss with beta bound between -1 and 3
    objective = lambda b: squared_loss(b)
    bnds = [(-1,3)]
    betahat = minimize(objective, x_0, bounds=bnds)

    result.append(betahat['x'][0])

plt.hist(result, bins = 25, histtype=u'step')
plt.vlines(np.mean(result), 0, 100, color = 'red', linewidth = 0.2)
plt.vlines(1.5, 0, 100, color = 'blue', linewidth = 0.2)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()

print(np.mean(result))
print(np.var(result))
```

```
<Figure size 640x480 with 1 Axes>
```

```
1.501404455896445
0.004802661025582587
```

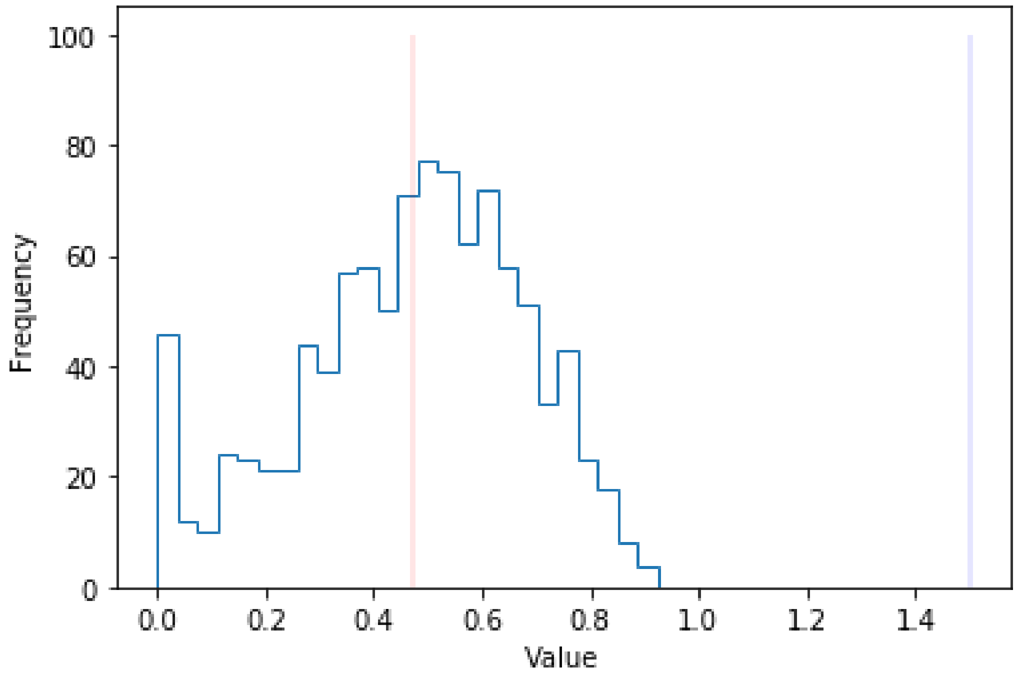
[3]:
```python
def lasso_loss(beta):
    return sum((y - yhat(beta,x))**2+lam*np.sum(np.abs(beta)))

result = []
fig = plt.figure()
for i in range(1000):
    lam = 1
    n = 50
    x = np.random.randn(n)
    u = np.linspace(-5, 5, 11)
    error = np.random.randn(n) * .5
    y = 1.5 * x + error
    x_0 = -1
    ##### minimize square loss with beta bound between -1 and 3
    objective = lambda b: lasso_loss(b)
    bnds = [(0, 10)]
    betahat = minimize(objective, x_0, bounds=bnds)

    result.append(betahat['x'][0])

plt.hist(result, bins = 25, histtype=u'step')
plt.vlines(np.mean(result), 0, 100, color = 'red', linewidth = 0.2)
plt.vlines(1.5, 0, 100, color = 'blue', linewidth = 0.2)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()

print(np.mean(result))
print(np.var(result))
```



```
0.9767782296452308
0.018050535830130995
```

[4]:
```
def ridge_loss(beta):
        # Ridge OLS
    return sum((y - yhat(beta,x))**2+lam*np.sqrt(beta.T*beta))
```

```python
result = []
fig = plt.figure()
for i in range(1000):
    lam = 2
    n = 50
    x = np.random.randn(n)
    u = np.linspace(-5, 5, 11)
    error = np.random.randn(n) * .5
    y = 1.5 * x + error
    x_0 = -1
    ##### minimize square loss with beta bound between -1 and 3
    objective = lambda b: ridge_loss(b)
    bnds = [(0, 10)]
    betahat = minimize(objective, x_0, bounds=bnds)

    result.append(betahat['x'][0])

plt.hist(result, bins = 25, histtype=u'step')
plt.vlines(np.mean(result), 0, 100, color = 'red', linewidth = 0.2)
plt.vlines(1.5, 0, 100, color = 'blue', linewidth = 0.2)
plt.xlabel('Value')
plt.ylabel('Frequency')
plt.show()

print(np.mean(result))
print(np.var(result))
```



```
0.46986726238676063
0.043756446999277644
```

- Modify the code to include a ridge penalty. Consider a range of λ values from 0 to 10. What do you notice? **I notice a shrinkage of the coefficients**
- Modify the code to include a lasso penalty. Consider a range of λ values from 0 to 10. What do you notice?
- Earlier you found the derivatives of the ridge and lasso penalties. How does each vary with β? What does this mean in terms of how much each penalty penalizes large vs. small values of β?

1.3 Mouse genome dataset example with ridge, lasso, and the elastic net

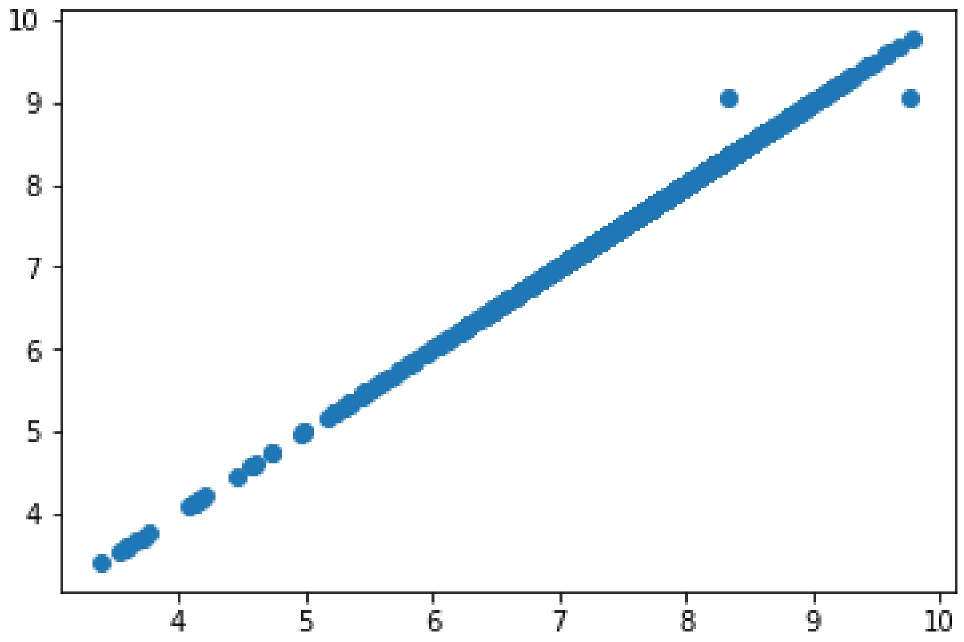
Ridge and lasso decrease *variance* by making a model less complex. We will consider a mouse genome dataset to predict red blood cell count from SNPs. You can read about the dataset at https://www.sanger.ac.uk/science/data/mouse-genomes-project

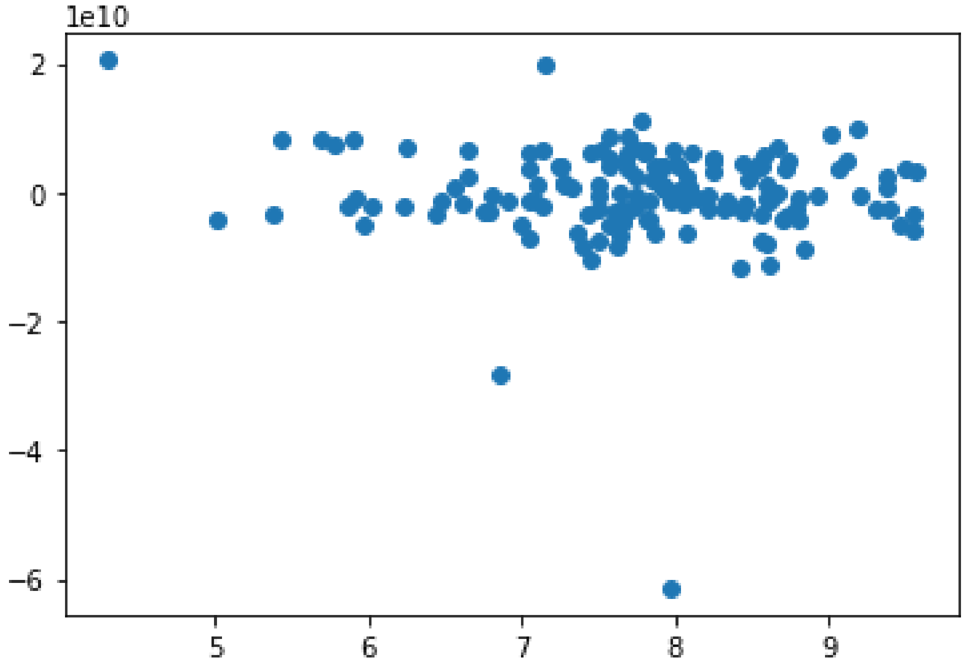
The dataset has 1522 observations and 10346 predictors. We've provided you with an 80% train and 10% test dataset. If we consider just linear regression, the problem is overdetermined, and we can find an infinite number of models which perfectly fit the data:

[5]:
```
import pyreadr # you will need to pip install this
result = pyreadr.read_r('mice.rdata')
```

[6]:
```
from sklearn.linear_model import LinearRegression
import pylab as plt

X_train = result['X_train']

# there are computational difficulties in addition to theoretical ones with␣
 ↪fitting a model with so many predictors, so we randomly sample 1300
ii = X_train.sample(1300, axis = 1)

y_train = result['y_train']
X_test = result['X_test']
y_test = result['y_test']

reg = LinearRegression().fit(ii, y_train)

plt.scatter(y_train, reg.predict(ii)) # almost perfect
plt.show()


plt.scatter(y_test, reg.predict(X_test[list(ii.columns)])) # very bad␣
 ↪fit---this is because our model has high variance
```

```
plt.show()
```

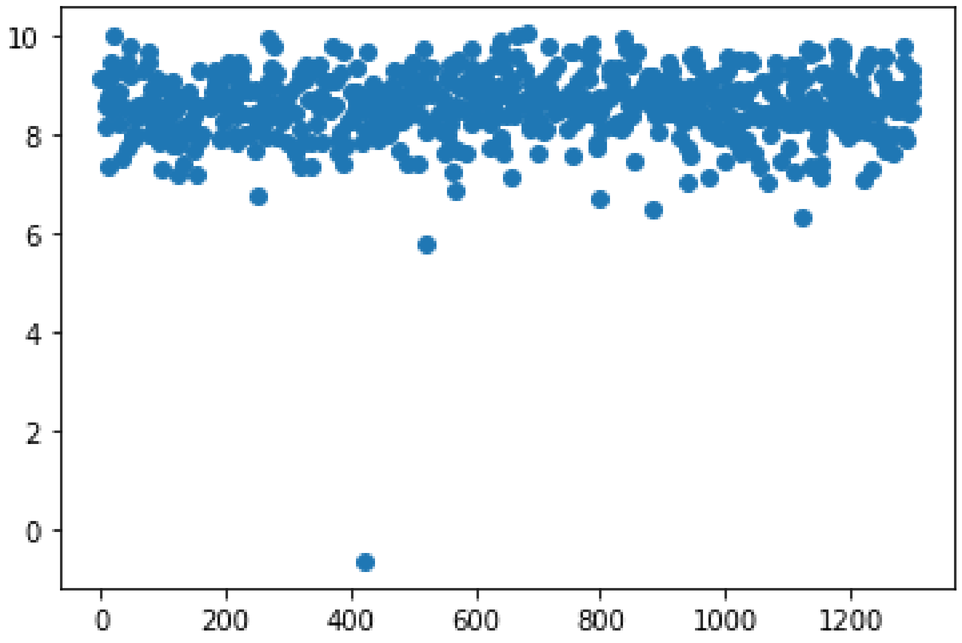




- Inspect the coefficients in the linear model. You can find their values using ''reg.coef_''. Do you notice any that are very large in magnitude?

[7]:
```
plt.scatter(range(1300),np.log10(reg.coef_))
plt.show()
```

```
/Users/luischavesrodriguez/opt/anaconda3/lib/python3.7/site-
packages/ipykernel_launcher.py:1: RuntimeWarning: invalid value encountered in
log10
  """Entry point for launching an IPython kernel.
```

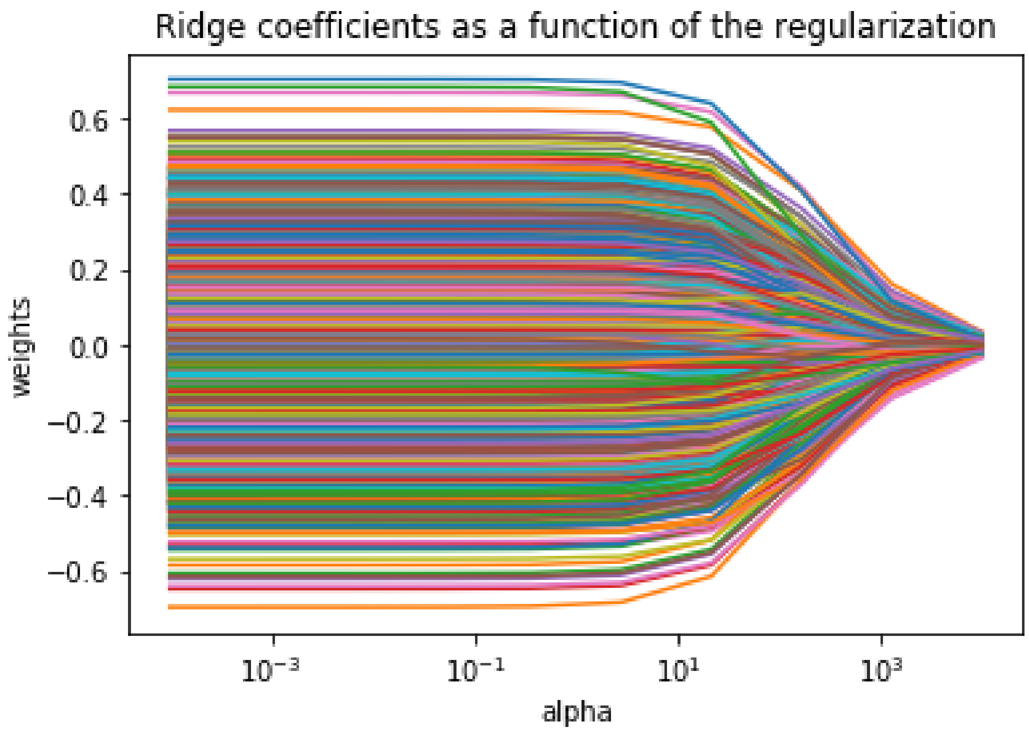


A couple really big ones, close to 1e10

Now let us see how lasso and ridge serve to decrease variance by shrinking all coefficients (ridge and lasso) and zeroing some of them out (lasso). Here is how to fit the elastic net (a mixture of ridge and lasso). We will now consider all 10346 predictors.

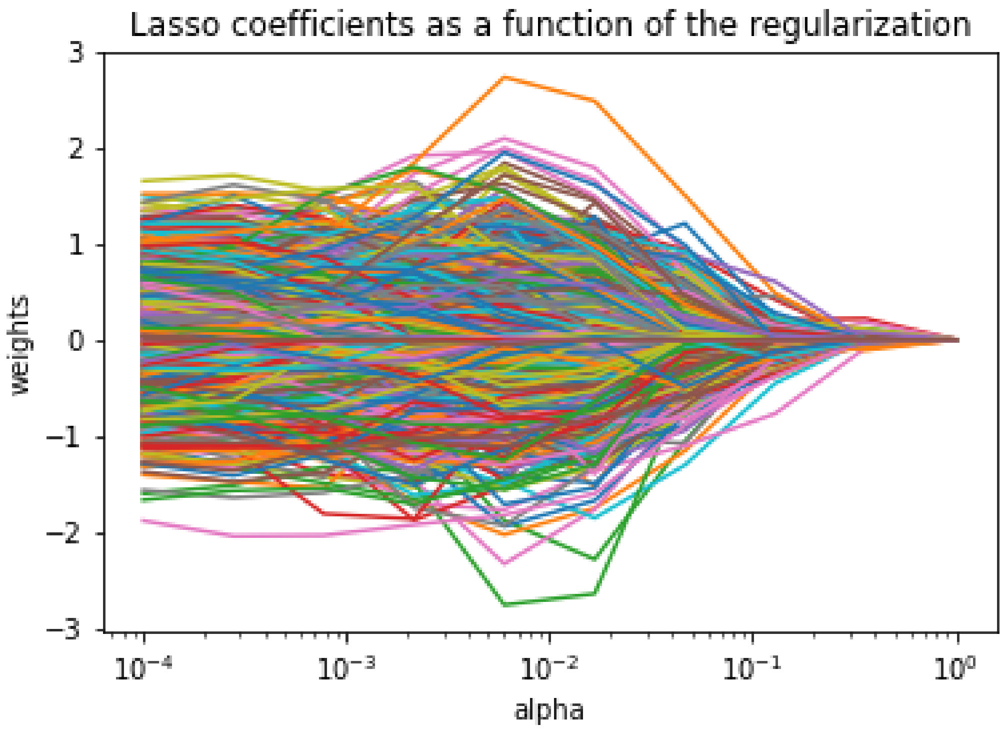
[23]:
```
from sklearn.linear_model import Ridge, Lasso, ElasticNet
import numpy as np

n_alphas = 10
alphas = np.logspace(-4, 4, n_alphas)

coefs_ridge = []
for a in alphas:
    ridge = Ridge(alpha=a, fit_intercept=False)
    ridge.fit(X_train, y_train)
    coefs_ridge.append(ridge.coef_[0])

# Display results
```

```python
ax = plt.gca()

ax.plot(alphas, coefs_ridge)
ax.set_xscale('log')
plt.xlabel('alpha')
plt.ylabel('weights')
plt.title('Ridge coefficients as a function of the regularization')
plt.axis('tight')
plt.show()

alphas = np.logspace(-4, 0, n_alphas)

coefs_lasso = []
for a in alphas:
    lasso = Lasso(alpha=a, fit_intercept=False)
    lasso.fit(X_train, y_train)
    coefs_lasso.append(lasso.coef_)

# Display results

ax = plt.gca()

ax.plot(alphas, coefs_lasso)
ax.set_xscale('log')
plt.xlabel('alpha')
plt.ylabel('weights')
plt.title('Lasso coefficients as a function of the regularization')
plt.axis('tight')
plt.show()


coefs_elastic = []
for a in alphas:
    elastic = ElasticNet(alpha = a, fit_intercept=False)
    elastic.fit(X_train, y_train)
    coefs_elastic.append(elastic.coef_)

# Display results

ax = plt.gca()

ax.plot(alphas, coefs_elastic)
ax.set_xscale('log')
plt.xlabel('alpha')
plt.ylabel('weights')
plt.title('Elastic Net coefficients as a function of the regularization')
plt.axis('tight')
plt.show()


yhat = ridge.predict(X_test)
#you will get a matrix of predictions corresponding to the predictions made at␣
↪each point in the regularization path.
```



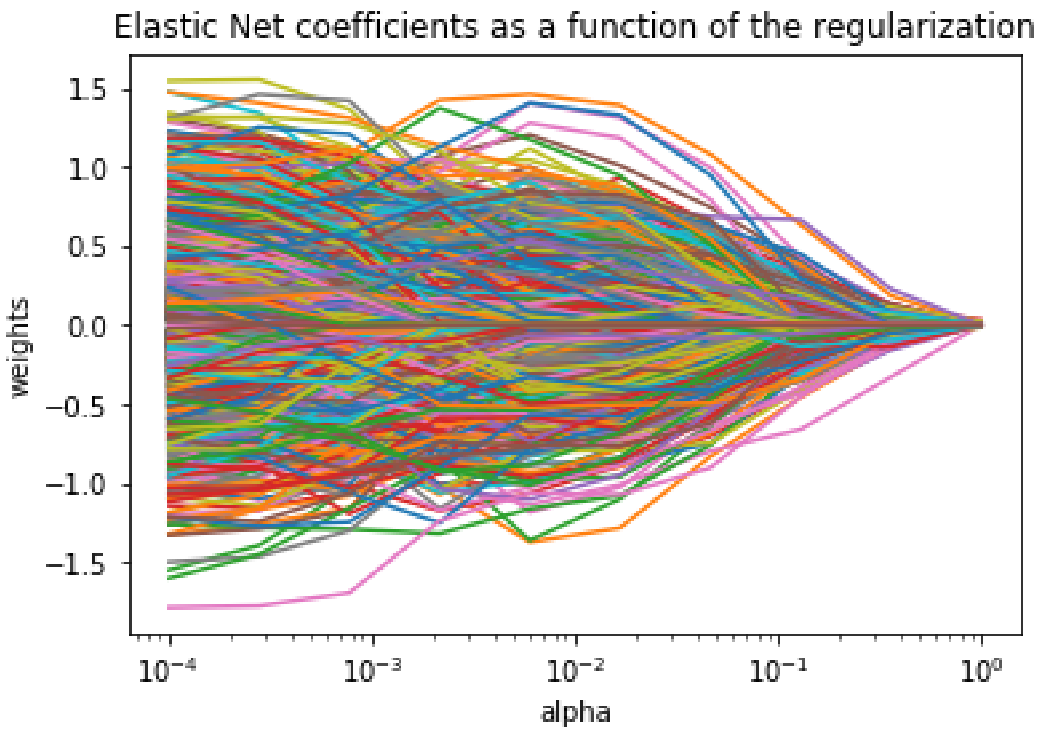
```
/Users/luischavesrodriguez/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 150.75399259833844, tolerance: 7.282365700000001
  positive)
/Users/luischavesrodriguez/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 303.33152123916585, tolerance: 7.282365700000001
  positive)
/Users/luischavesrodriguez/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 532.4195813988762, tolerance: 7.282365700000001
  positive)
```

```
/Users/luischavesrodriguez/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 909.0249714865685, tolerance: 7.282365700000001
  positive)
/Users/luischavesrodriguez/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 845.3355295587378, tolerance: 7.282365700000001
  positive)
/Users/luischavesrodriguez/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 157.74207399987063, tolerance: 7.282365700000001
  positive)
/Users/luischavesrodriguez/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 13.064564286920358, tolerance: 7.282365700000001
  positive)
```



```
/Users/luischavesrodriguez/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
```

```
Duality gap: 106.92901368379337, tolerance: 7.282365700000001
  positive)
/Users/luischavesrodriguez/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 228.09820994528442, tolerance: 7.282365700000001
  positive)
/Users/luischavesrodriguez/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 426.61425914213527, tolerance: 7.282365700000001
  positive)
/Users/luischavesrodriguez/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 800.4671438418975, tolerance: 7.282365700000001
  positive)
/Users/luischavesrodriguez/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 1105.416937754393, tolerance: 7.282365700000001
  positive)
/Users/luischavesrodriguez/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 795.379804516404, tolerance: 7.282365700000001
  positive)
/Users/luischavesrodriguez/opt/anaconda3/lib/python3.7/site-
packages/sklearn/linear_model/_coordinate_descent.py:476: ConvergenceWarning:
Objective did not converge. You might want to increase the number of iterations.
Duality gap: 168.47968701932405, tolerance: 7.282365700000001
  positive)
```



[24]:
```
coefs_lasso
```

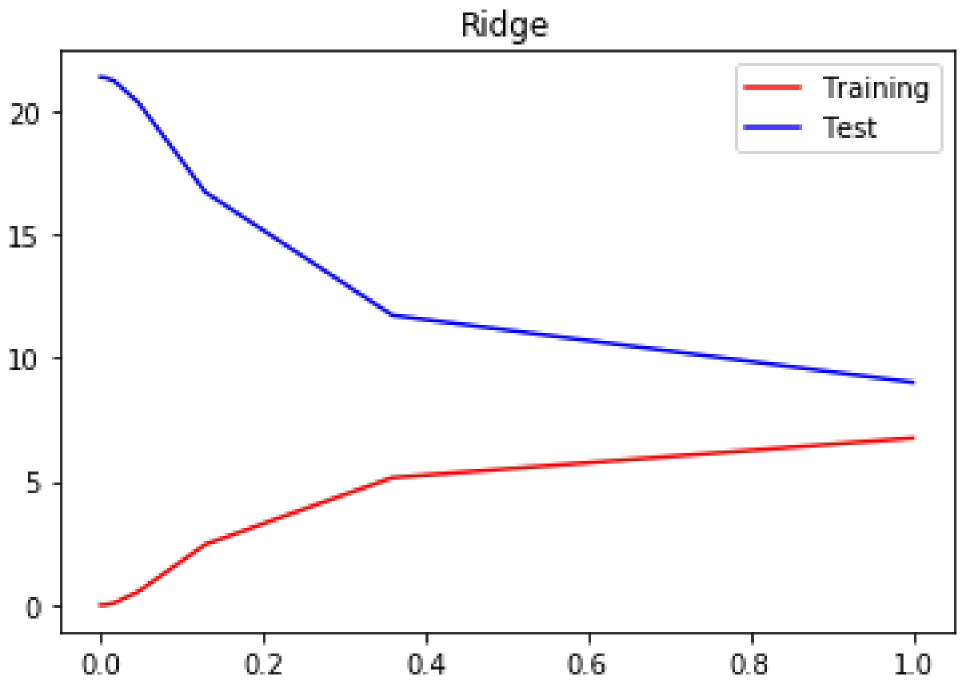
[24]:
```
[array([-0.        , -0.0910898 , -0.67916669, …, -0.26295957,
        -0.22526761,  0.04642263]),
 array([-0.        , -0.        , -0.5251626 , …, -0.09307056,
        -0.05903153,  0.        ]),
 array([ 0.        , -0.        , -0.2516506 , …, -0.0088479 ,
        -0.26243536, -0.        ]),
 array([ 0.        , -0.        , -0.        , …,  0.        ,
        -0.21592919, -0.        ]),
 array([ 0.        , -0.        , -0.        , …,  0.        ,
        -0.01753853, -0.04520285]),
 array([ 0., -0., -0., …,  0., -0., -0.]),
 array([-0., -0., -0., …, -0., -0.,  0.]),
 array([-0.,  0., -0., …, -0., -0.,  0.]),
 array([-0.,  0., -0., …, -0., -0.,  0.]),
 array([-0.,  0., -0., …, -0., -0., -0.])]
```

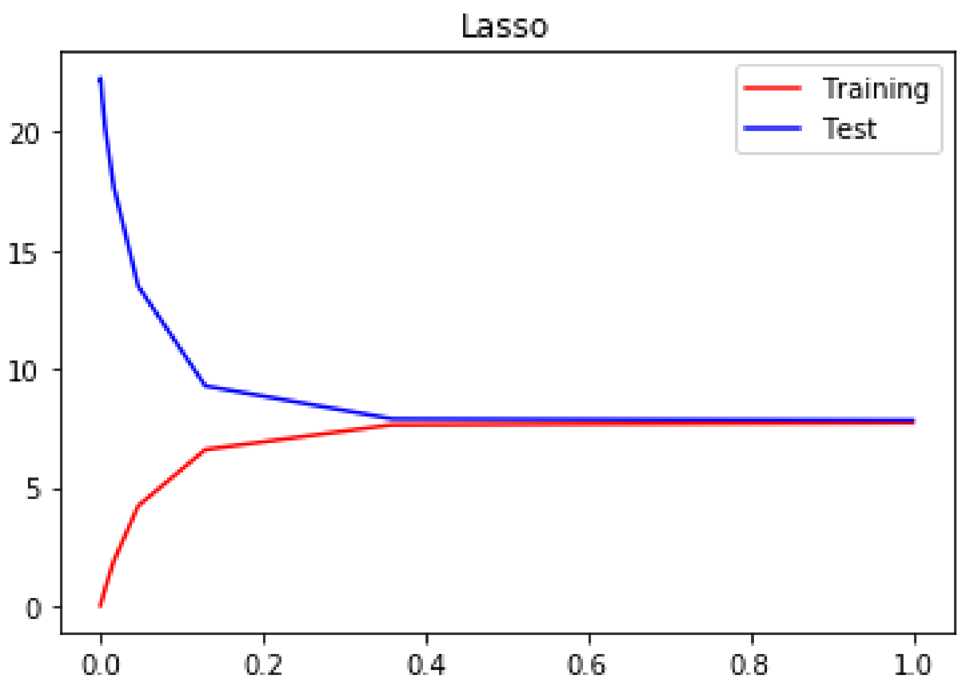
[25]:
```
results = {'Ridge': {alpha: coefs  for alpha, coefs in zip(alphas,coefs_ridge)},
           'Lasso': {alpha: coefs  for alpha, coefs in zip(alphas,coefs_lasso)},
           'Elastic': {alpha: coefs  for alpha, coefs in␣
 ↪zip(alphas,coefs_elastic)}}
import pandas as pd
```

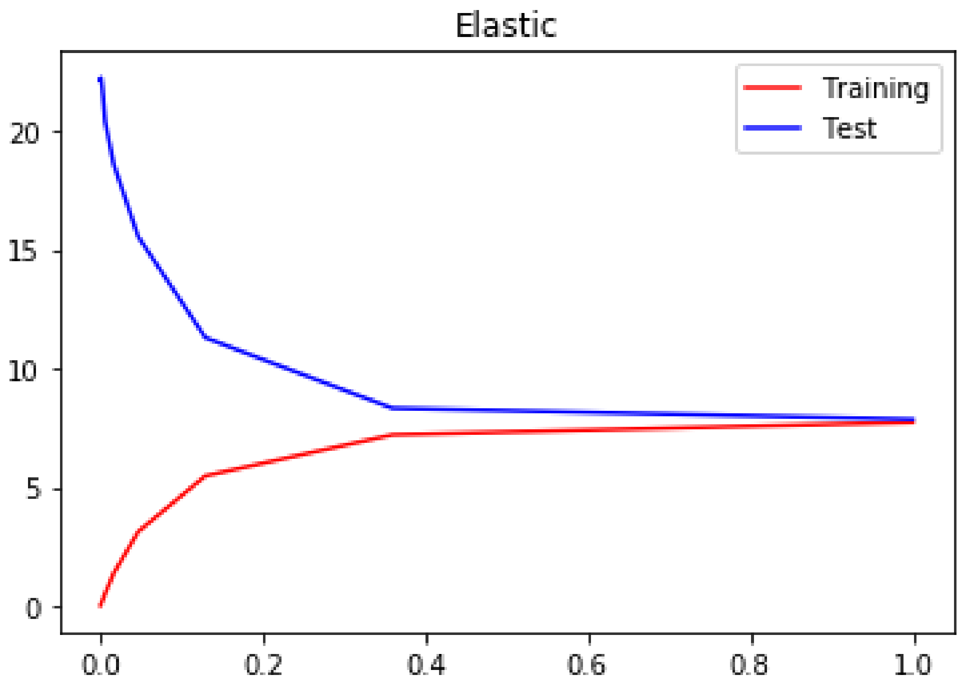
```
pd.DataFrame(results)
results.keys()
```

[25]:
```
dict_keys(['Ridge', 'Lasso', 'Elastic'])
```

[26]:
```
for model in results.keys():
    fig = plt.figure()
    error_train = np.zeros(alphas.shape)
    error_test = np.zeros(alphas.shape)
    i = 0
    for alpha in alphas:
        coef = results[model][alpha]
        coef = np.reshape(coef,(len(coef), 1))
        X_train = np.array(X_train)
        X_test = np.array(X_test)
        y_train = np.array(y_train)
        y_test = np.array(y_test)
        error_train[i] = np.sqrt(np.mean((y_train - np.matmul(X_train,␣
 ↪coef))**2))
        error_test[i] = np.sqrt(np.mean((y_test - np.matmul(X_test,coef))**2))
        i += 1
    plt.plot(alphas, error_train, 'r')
    ax = fig.gca()
    plt.plot(alphas, error_test, 'b')
    plt.title(str(model))
    ax.legend(['Training', 'Test'])
    plt.show()
```




Lasso
Training
Test
20
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0


Elastic
Training
Test
20
15
10
5
0
0.0
0.2
0.4
0.6
0.8
1.0

[35]:
```

```

[35]:
```
array([22.12317186, 22.08324142, 22.23769033, 21.8936044 , 20.37951688,
       18.551532  , 15.54006014, 11.28498287,  8.31664145,  7.85152396])
```

[ ]:
```
print(coefs_lasso[2].shape)
print(X_train.shape)
y_train.shape

coef = np.reshape(coefs_lasso[2], (len(coefs_lasso[2]),1))
print(coef.shape)
(np.matmul(X_train,coef)).shape
```

- Calculate a loss function (which one is appropriate for a linear model?) at each value of λ for lasso, ridge, and the elastic net, using the training data. Repeat for the testing data. Plot both of these curves—these are the training and testing learning curves.
- What values of λ give solutions that approach that of ordinary linear regression (without regularization)? For lasso, what value of λ corresponds to a model with no predictors? What are the predictions for this model?

[30]:
```
results = {Model: {alpha: coefs  for alpha,
                   coefs in zip(alphas,
                                [coefs_ridge, coefs_lasso, coefs_elastic])}
           for Model in ['Ridge','Lasso','Elastic']}
```