Example of Regression Analysis Using the Boston Housing Data Set.

In [1]:

```
from sklearn.linear_model import LinearRegression, Lasso, Ridge, ElasticNet,
import numpy as np
import pandas as pd
import pylab as pl
```

In [2]:

```
from sklearn.datasets import load_boston
boston = load_boston()
```

In [3]:
```
print(boston.DESCR)
```

```
Boston House Prices dataset
===========================

Notes
------
Data Set Characteristics:

    :Number of Instances: 506

    :Number of Attributes: 13 numeric/categorical predictive

    :Median Value (attribute 14) is usually the target

    :Attribute Information (in order):
        - CRIM     per capita crime rate by town
        - ZN       proportion of residential land zoned for lots over 25,000
        - INDUS    proportion of non-retail business acres per town
        - CHAS     Charles River dummy variable (= 1 if tract bounds river;
        - NOX      nitric oxides concentration (parts per 10 million)
        - RM       average number of rooms per dwelling
        - AGE      proportion of owner-occupied units built prior to 1940
        - DIS      weighted distances to five Boston employment centres
        - RAD      index of accessibility to radial highways
        - TAX      full-value property-tax rate per $10,000
        - PTRATIO  pupil-teacher ratio by town
        - B        1000(Bk - 0.63)^2 where Bk is the proportion of blacks by
        - LSTAT    % lower status of the population
        - MEDV     Median value of owner-occupied homes in $1000's

    :Missing Attribute Values: None

    :Creator: Harrison, D. and Rubinfeld, D.L.

This is a copy of UCI ML housing dataset.
http://archive.ics.uci.edu/ml/datasets/Housing


This dataset was taken from the StatLib library which is maintained at Carne

The Boston house-price data of Harrison, D. and Rubinfeld, D.L. 'Hedonic
prices and the demand for clean air', J. Environ. Economics & Management,
vol.5, 81-102, 1978.   Used in Belsley, Kuh & Welsch, 'Regression diagnostic
...', Wiley, 1980.   N.B. Various transformations are used in the table on
pages 244-261 of the latter.

The Boston house-price data has been used in many machine learning papers th
problems.

**References**

   - Belsley, Kuh & Welsch, 'Regression diagnostics: Identifying Influential
   - Quinlan,R. (1993). Combining Instance-Based and Model-Based Learning. I
   - many more! (see http://archive.ics.uci.edu/ml/datasets/Housing)
```

In [4]:
```
print(boston.feature_names)
```

```
['CRIM' 'ZN' 'INDUS' 'CHAS' 'NOX' 'RM' 'AGE' 'DIS' 'RAD' 'TAX' 'PTRATIO'
 'B' 'LSTAT']
```

In [5]:
```
print(boston.data.shape)
print(boston.target.shape)
```

```
(506, 13)
(506,)
```

In [6]:
```
bostonDF = pd. DataFrame(boston.data, columns = boston.feature_names)
bostonDF.head()
```

Out[6]:

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	L
0	0.00632	18.0	2.31	0.0	0.538	6.575	65.2	4.0900	1.0	296.0	15.3	396.90	
1	0.02731	0.0	7.07	0.0	0.469	6.421	78.9	4.9671	2.0	242.0	17.8	396.90	
2	0.02729	0.0	7.07	0.0	0.469	7.185	61.1	4.9671	2.0	242.0	17.8	392.83	
3	0.03237	0.0	2.18	0.0	0.458	6.998	45.8	6.0622	3.0	222.0	18.7	394.63	
4	0.06905	0.0	2.18	0.0	0.458	7.147	54.2	6.0622	3.0	222.0	18.7	396.90	

In [7]:
```
np.set_printoptions(precision=2, linewidth=120, suppress=True, edgeitems=7)
```

In [8]:
```
print(boston.data)
```

```
[[  0.01  18.     2.31   0.     0.54   6.58  65.2    4.09   1.   296.    1
 [  0.03   0.     7.07   0.     0.47   6.42  78.9    4.97   2.   242.    1
 [  0.03   0.     7.07   0.     0.47   7.18  61.1    4.97   2.   242.    1
 [  0.03   0.     2.18   0.     0.46   7.    45.8    6.06   3.   222.    1
 [  0.07   0.     2.18   0.     0.46   7.15  54.2    6.06   3.   222.    1
 [  0.03   0.     2.18   0.     0.46   6.43  58.7    6.06   3.   222.    1
 [  0.09  12.5    7.87   0.     0.52   6.01  66.6    5.56   5.   311.    1
 ...
 [  0.18   0.     9.69   0.     0.58   5.57  73.5    2.4    6.   391.    1
 [  0.22   0.     9.69   0.     0.58   6.03  79.7    2.5    6.   391.    1
 [  0.06   0.    11.93   0.     0.57   6.59  69.1    2.48   1.   273.    2
 [  0.05   0.    11.93   0.     0.57   6.12  76.7    2.29   1.   273.    2
 [  0.06   0.    11.93   0.     0.57   6.98  91.     2.17   1.   273.    2
 [  0.11   0.    11.93   0.     0.57   6.79  89.3    2.39   1.   273.    2
 [  0.05   0.    11.93   0.     0.57   6.03  80.8    2.5    1.   273.    2
```

In [9]:
```
# The attribute MEDV (Median Values) is the target attribute (response varic
print(boston.target[:20])
```

```
[24.  21.6 34.7 33.4 36.2 28.7 22.9 27.1 16.5 18.9 15.  18.9 21.7 20.4 18.
```

In [10]:

```
# In order to do multiple regression we need to add a column of 1s as the cc
x = np.array([np.concatenate((v,[1])) for v in boston.data])
y = boston.target
```

In [11]:

```
# First 10 elements of the data
print(x[:10])
```

```
[[  0.01  18.     2.31   0.     0.54   6.58  65.2    4.09   1.   296.    1
 [  0.03   0.     7.07   0.     0.47   6.42  78.9    4.97   2.   242.    1
 [  0.03   0.     7.07   0.     0.47   7.18  61.1    4.97   2.   242.    1
 [  0.03   0.     2.18   0.     0.46   7.    45.8    6.06   3.   222.    1
 [  0.07   0.     2.18   0.     0.46   7.15  54.2    6.06   3.   222.    1
 [  0.03   0.     2.18   0.     0.46   6.43  58.7    6.06   3.   222.    1
 [  0.09  12.5    7.87   0.     0.52   6.01  66.6    5.56   5.   311.    1
 [  0.14  12.5    7.87   0.     0.52   6.17  96.1    5.95   5.   311.    1
 [  0.21  12.5    7.87   0.     0.52   5.63 100.     6.08   5.   311.    1
 [  0.17  12.5    7.87   0.     0.52   6.    85.9    6.59   5.   311.    1
```

In [12]:

```
# Create linear regression object
linreg = LinearRegression()

# Train the model using the training set
linreg.fit(x,y)
```

Out[12]:

```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normalize=Fals
```

In [13]:

```
# Let's see predictions for the first 10 instances and compare to actual MEL
for i in range(10):
    pred = linreg.predict(np.array([x[i]]))[0]
    print("%2d \t %2.2f \t %2.2f" % (i, pred, y[i]))
```

```
 0       30.01   24.00
 1       25.03   21.60
 2       30.57   34.70
 3       28.61   33.40
 4       27.94   36.20
 5       25.26   28.70
 6       23.00   22.90
 7       19.53   27.10
 8       11.52   16.50
 9       18.92   18.90
```

Compute RMSE on training data

In [14]:

```
# First, let's compute errors on all training instances

p = linreg.predict(x) # p is the array of predicted values

# Now we can constuct an array of errors
err = abs(p-y)

# Let's see the error on the first 10 predictions
print(err[:10])
```

```
[6.01 3.43 4.13 4.79 8.26 3.44 0.1  7.57 4.98 0.02]
```

In [15]:

```
# Dot product of error vector with itself gives us the sum of squared errors
total_error = np.dot(err,err)

# Finally compute RMSE
rmse_train = np.sqrt(total_error/len(p))
print("RMSE on Training Data: ", rmse_train)
```

```
RMSE on Training Data:  4.679506300635516
```

In [16]:

```
# We can view the regression coefficients
print('Regression Coefficients: \n', linreg.coef_)
```

```
Regression Coefficients: 
 [ -0.11   0.05   0.02   2.69 -17.8    3.8    0.    -1.48   0.31  -0.01  -
```

In [17]:

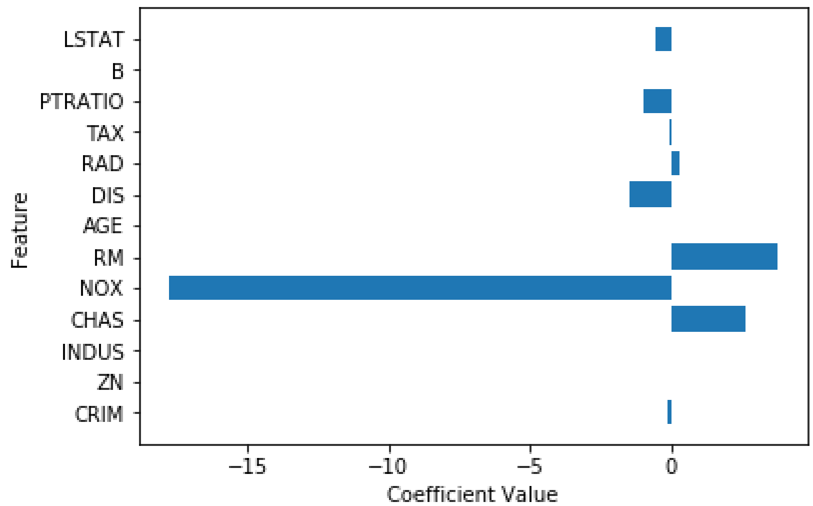
```
# Let's put some names to the faces
for i in range(len(boston.feature_names)):
    print("%7s   %2.2f" % (boston.feature_names[i], linreg.coef_[i]))
```

```
   CRIM   -0.11
     ZN   0.05
  INDUS   0.02
   CHAS   2.69
    NOX   -17.80
     RM   3.80
    AGE   0.00
    DIS   -1.48
    RAD   0.31
    TAX   -0.01
PTRATIO   -0.95
      B   0.01
  LSTAT   -0.53
```

In [18]:

```
# The following function can be used to plot the model coefficients
%matplotlib inline
def plot_coefficients(model, n_features, feature_names):
    pl.barh(range(n_features), model.coef_[:-1], align='center')
    pl.yticks(np.arange(n_features), feature_names)
    pl.xlabel("Coefficient Value")
    pl.ylabel("Feature")
    pl.ylim(-1, n_features)

plot_coefficients(linreg, len(boston.feature_names), boston.feature_names)
```



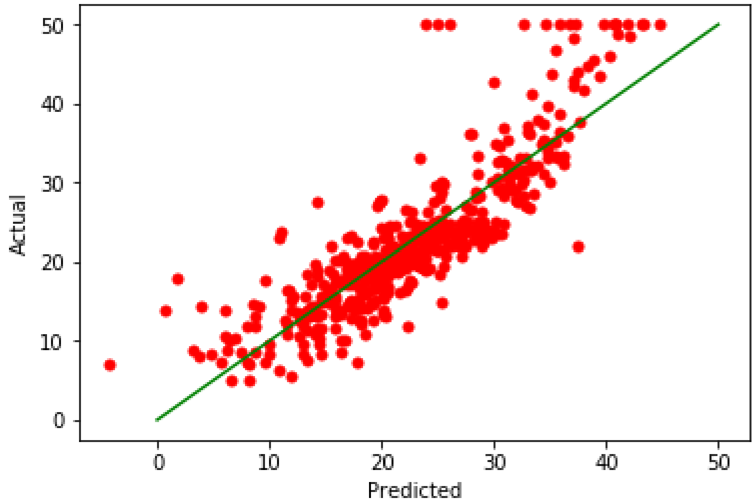
In [19]:

```
print("Linear Regression Intercept: ", linreg.intercept_)
```

```
Linear Regression Intercept:  36.491103280362715
```

In [20]:

```
# Plot predicted against actual (in the training data)
%matplotlib inline
pl.plot(p, y,'ro', markersize=5)
pl.plot([0,50],[0,50], 'g-')
pl.xlabel('Predicted')
pl.ylabel('Actual')
pl.show()
```



Let's now compute RMSE using 10-fold cross-validation

In [21]:

```
def cross_validate(model, X, y, n, verbose=False):
    # model: regression model to be trained
    # X: the data matrix
    # y: the target variable array
    # n: the number of fold for x-validation
    # Returns mean RMSE across all folds

    from sklearn.model_selection import KFold
    kf = KFold(n_splits=n, random_state=22)
    xval_err = 0
    f = 1
    for train,test in kf.split(x):
        model.fit(X[train],y[train])
        p = model.predict(x[test])
        e = p-y[test]
        rmse = np.sqrt(np.dot(e,e)/len(x[test]))
        if verbose:
            print("Fold %2d RMSE: %.4f" % (f, rmse))
        xval_err += rmse
        f += 1
    return xval_err/n
```

In [22]:

```
rmse_10cv = cross_validate(linreg, x, y, 10, verbose=True)
```

```
Fold  1 RMSE: 3.0498
Fold  2 RMSE: 3.7646
Fold  3 RMSE: 3.7558
Fold  4 RMSE: 5.9325
Fold  5 RMSE: 5.6502
Fold  6 RMSE: 4.4563
Fold  7 RMSE: 3.1556
Fold  8 RMSE: 12.9819
Fold  9 RMSE: 5.7981
Fold 10 RMSE: 3.3116
```

In [23]:

```
method_name = 'Simple Linear Regression'
print('Method: %s' %method_name)
print('RMSE on training: %.4f' %rmse_train)
print('RMSE on 10-fold CV: %.4f' %rmse_10cv)
```

```
Method: Simple Linear Regression
RMSE on training: 4.6795
RMSE on 10-fold CV: 5.1856
```

Let's try Ridge Regression:

In [24]:

```
# Create linear regression object with a ridge coefficient 0.5
ridge = Ridge(alpha=0.8)

# Train the model using the training set
ridge.fit(x,y)
```

Out[24]:

```
Ridge(alpha=0.8, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
```

In [25]:

```
# Compute RMSE on training data
p = ridge.predict(x)
err = p-y
total_error = np.dot(err,err)
rmse_train = np.sqrt(total_error/len(p))

# Compute RMSE using 10-fold x-validation

rmse_10cv = cross_validate(ridge, x, y, 10, verbose=True)

method_name = 'Ridge Regression'
print("\n")
print('Method: %s' %method_name)
print('RMSE on training: %.4f' %rmse_train)
print('RMSE on 10-fold CV: %.4f' %rmse_10cv)
```

```
Fold  1 RMSE: 3.0523
Fold  2 RMSE: 3.5777
Fold  3 RMSE: 3.3268
Fold  4 RMSE: 6.0322
Fold  5 RMSE: 5.4693
Fold  6 RMSE: 4.3329
Fold  7 RMSE: 3.0825
Fold  8 RMSE: 12.9915
Fold  9 RMSE: 5.8317
Fold 10 RMSE: 3.3695


Method: Ridge Regression
RMSE on training: 4.6916
RMSE on 10-fold CV: 5.1066
```

We can try different values of alpha and observe the impact on x-validation RMSE

In [26]:

```
print('Ridge Regression')
print('alpha\t RMSE_train\t RMSE_10cv\n')
alpha = np.linspace(.01,20,50)
t_rmse = np.array([])
cv_rmse = np.array([])

for a in alpha:
    ridge = Ridge(alpha=a)

    # computing the RMSE on training data
    ridge.fit(x,y)
    p = ridge.predict(x)
    err = p-y
    total_error = np.dot(err,err)
    rmse_train = np.sqrt(total_error/len(p))

    rmse_10cv = cross_validate(ridge, x, y, 10)

    t_rmse = np.append(t_rmse, [rmse_train])
    cv_rmse = np.append(cv_rmse, [rmse_10cv])
    print('{:.3f}\t {:.4f}\t\t {:.4f}'.format(a,rmse_train,rmse_10cv))
```

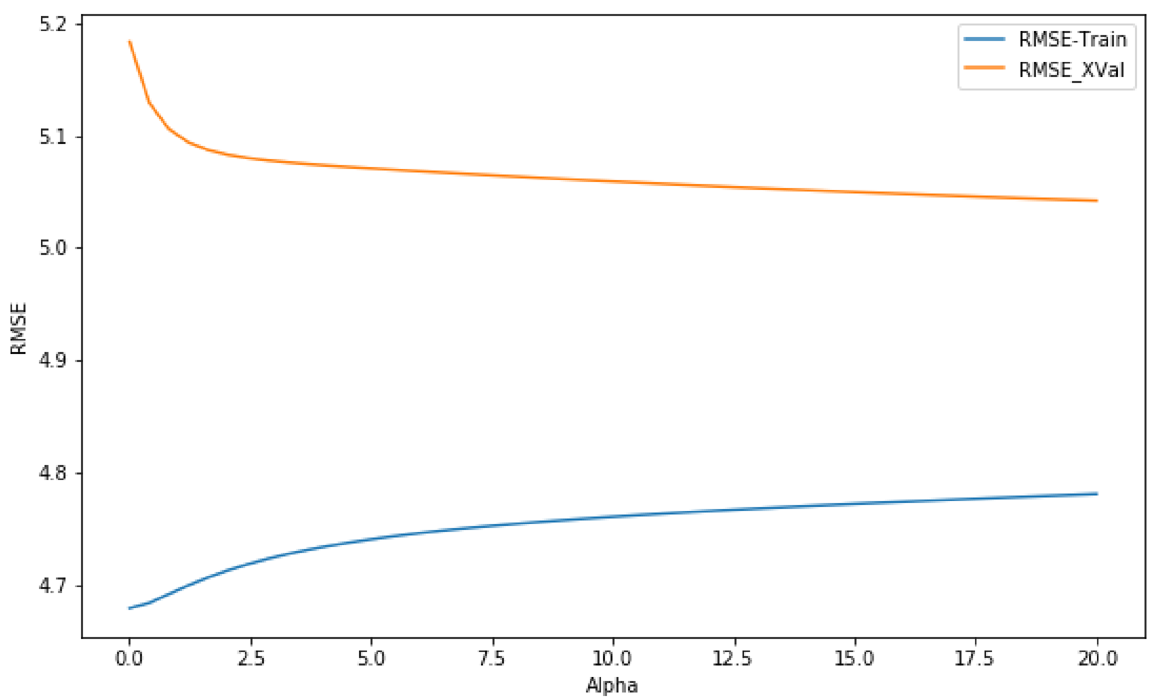
```
Ridge Regression
alpha    RMSE_train      RMSE_10cv

0.010    4.6795          5.1835
0.418    4.6842          5.1291
0.826    4.6921          5.1056
1.234    4.7000          5.0938
1.642    4.7070          5.0871
2.050    4.7133          5.0829
2.458    4.7187          5.0800
2.866    4.7234          5.0778
3.274    4.7276          5.0761
3.682    4.7313          5.0747
4.090    4.7346          5.0734
4.498    4.7375          5.0722
4.906    4.7402          5.0710
5.313    4.7426          5.0700
5.721    4.7448          5.0689
6.129    4.7469          5.0679
6.537    4.7488          5.0669
6.945    4.7505          5.0659
7.353    4.7522          5.0650
7.761    4.7537          5.0640
8.169    4.7552          5.0631
8.577    4.7565          5.0622
8.985    4.7578          5.0613
9.393    4.7591          5.0604
9.801    4.7603          5.0595
10.209   4.7614          5.0587
10.617   4.7625          5.0578
11.025   4.7635          5.0570
11.433   4.7646          5.0562
11.841   4.7655          5.0554
12.249   4.7665          5.0546
```

```
12.657   4.7674          5.0538
13.065   4.7683          5.0531
13.473   4.7692          5.0523
13.881   4.7700          5.0516
14.289   4.7708          5.0509
14.697   4.7717          5.0502
15.104   4.7724          5.0495
15.512   4.7732          5.0488
15.920   4.7740          5.0482
16.328   4.7747          5.0475
16.736   4.7755          5.0469
17.144   4.7762          5.0462
17.552   4.7769          5.0456
17.960   4.7776          5.0450
18.368   4.7783          5.0444
18.776   4.7790          5.0439
19.184   4.7797          5.0433
19.592   4.7804          5.0427
20.000   4.7811          5.0422
```

In [27]:

```
fig = pl.figure(figsize=(10,6))
ax = fig.add_subplot(111)
ax.plot(alpha, t_rmse, label='RMSE-Train')
ax.plot(alpha, cv_rmse, label='RMSE_XVal')
pl.legend( ('RMSE-Train', 'RMSE_XVal') )
pl.ylabel('RMSE')
pl.xlabel('Alpha')
pl.show()
```



To make comparisons across methods easier, let's parametrize the regression methods:

In [28]:

```
a = 0.01
for name,met in [
        ('linear regression', LinearRegression()),
        ('lasso', Lasso(alpha=a)),
        ('ridge', Ridge(alpha=a)),
        ('elastic-net', ElasticNet(alpha=a))
        ]:

    # computing the RMSE on training data
    met.fit(x,y)
    p = met.predict(x)
    e = p-y
    total_error = np.dot(e,e)
    rmse_train = np.sqrt(total_error/len(p))

    # computing the RMSE for x-validation
    rmse_10cv = cross_validate(met, x, y, 10)


    print('Method: %s' %name)
    print('RMSE on training: %.4f' %rmse_train)
    print('RMSE on 10-fold CV: %.4f' %rmse_10cv)
    print("\n")
```

```
Method: linear regression
RMSE on training: 4.6795
RMSE on 10-fold CV: 5.1856


Method: lasso
RMSE on training: 4.6834
RMSE on 10-fold CV: 5.1461


Method: ridge
RMSE on training: 4.6795
RMSE on 10-fold CV: 5.1835


Method: elastic-net
RMSE on training: 4.7245
RMSE on 10-fold CV: 5.0813
```

Now let's try to do regression via Stochastic Gradient Descent.

In [29]:

```
# SGD is very senstitive to varying-sized feature values. So, first we need

from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()
scaler.fit(x)
x_s = scaler.transform(x)

sgdreg = SGDRegressor(penalty='l2', alpha=0.01, max_iter=300)

# Compute RMSE on training data
sgdreg.fit(x_s,y)
p = sgdreg.predict(x_s)
err = p-y
total_error = np.dot(err,err)
rmse_train = np.sqrt(total_error/len(p))

# Compute RMSE using 10-fold x-validation
from sklearn.model_selection import KFold

n = 10
kf = KFold(n_splits=n, random_state=22)
xval_err = 0
f = 1
for train,test in kf.split(x):
    scaler = StandardScaler()
    scaler.fit(x[train])  # Don't cheat - fit only on training data
    xtrain_s = scaler.transform(x[train])
    xtest_s = scaler.transform(x[test])  # apply same transformation to test
    sgdreg.fit(xtrain_s,y[train])
    p = sgdreg.predict(xtest_s)
    e = p-y[test]

    rmse = np.sqrt(np.dot(e,e)/len(x[test]))
    print("Fold %2d RMSE: %.4f" % (f, rmse))
    xval_err += rmse
    f += 1

rmse_10cv = xval_err/n

method_name = 'Stochastic Gradient Descent Regression'
print('Method: %s' %method_name)
print('RMSE on training: %.4f' %rmse_train)
print('RMSE on 10-fold CV: %.4f' %rmse_10cv)
```

```
Fold  1 RMSE: 2.9811
Fold  2 RMSE: 3.7054
Fold  3 RMSE: 3.6829
Fold  4 RMSE: 6.0139
Fold  5 RMSE: 5.5694
Fold  6 RMSE: 4.4626
Fold  7 RMSE: 3.0826
Fold  8 RMSE: 12.8734
Fold  9 RMSE: 5.7801
Fold 10 RMSE: 3.2109
```

```
Method: Stochastic Gradient Descent Regression
RMSE on training: 4.6818
RMSE on 10-fold CV: 5.1362
```

Instead of Scikit-learn, let's implement the closed form solution for linear regression

In [30]:

```
def standRegres(xArr,yArr):
    xMat = np.mat(xArr); yMat = np.mat(yArr).T
    xTx = xMat.T*xMat
    if np.linalg.det(xTx) == 0.0:
        print("This matrix is singular, cannot do inverse")
        return
    ws = xTx.I * (xMat.T*yMat)
    return ws
```

In [31]:

```
w = standRegres(x,y)
```

In [32]:

```
print(w)
```

```
[[ -0.11]
 [  0.05]
 [  0.02]
 [  2.69]
 [-17.8 ]
 [  3.8 ]
 [  0.  ]
 [ -1.48]
 [  0.31]
 [ -0.01]
 [ -0.95]
 [  0.01]
 [ -0.53]
 [ 36.49]]
```

In [33]:

```
def ridgeRegres(xArr,yArr,lam=0.2):
    xMat = np.mat(xArr); yMat = np.mat(yArr).T
    xTx = xMat.T*xMat
    denom = xTx + np.eye(np.shape(xMat)[1])*lam
    if np.linalg.det(denom) == 0.0:
        print("This matrix is singular, cannot do inverse")
        return
    ws = denom.I * (xMat.T*yMat)
    return ws
```

In [34]:
```
w_ridge = ridgeRegres(x,y,0.5)
print(w_ridge)
```

```
[[-0.1 ]
 [ 0.05]
 [-0.  ]
 [ 2.68]
 [-9.55]
 [ 4.55]
 [-0.  ]
 [-1.26]
 [ 0.25]
 [-0.01]
 [-0.73]
 [ 0.01]
 [-0.49]
 [21.78]]
```

Now that we have the regression coefficients, we can compute the predictions:

In [35]:
```
xMat=np.mat(x)
yMat=np.mat(y)
yHat = xMat*w_ridge
```

In [36]:
```
yHat.shape
```

Out[36]:
```
(506, 1)
```

In [37]:
```
print(yHat[0:10])
```

```
[[29.81]
 [24.75]
 [30.78]
 [29.12]
 [28.61]
 [25.35]
 [22.48]
 [19.28]
 [11.21]
 [18.65]]
```

In [38]:
```
print(yMat.T[0:10])
```

```
[[24. ]
 [21.6]
 [34.7]
 [33.4]
 [36.2]
 [28.7]
 [22.9]
 [27.1]
 [16.5]
 [18.9]]
```

Model evaluation and cross validation can be performed as before.

In [39]:

```
# You can "ravel" the 2d matrix above to get a 1d Numpy array more suitable
print(yHat.A.ravel())
```

```
[29.81 24.75 30.78 29.12 28.61 25.35 22.48 19.28 11.21 18.65 19.03 21.15 2
 18.68 12.82 18.15 16.6  14.34 16.27 13.61 15.97 15.28 20.45 21.86 11.99 1
 22.8  31.37 34.5  28.21 24.95 24.47 22.57 21.38 19.91 17.7   8.69 16.69 2
 32.9  22.18 21.13 17.87 18.44 24.3  23.48 24.42 29.71 24.62 20.81 16.99 2
 22.21 22.65 20.85 21.65 28.49 26.86 25.86 24.72 24.63 27.58 21.78 25.22 3
 28.09 24.01 36.23 35.56 32.35 25.09 26.15 19.35 20.46 21.48 18.39 17.15 2
 24.68 19.8  23.06 23.15 19.72 20.24 21.62 22.16 20.26 16.16 20.16 22.12 1
 16.35 13.56 18.22 16.49 20.26 14.45 17.22 14.54  4.39 15.54 13.16  9.4  1
 20.29 17.88 23.4  20.74 13.58 32.49 27.61 26.72 31.4  36.4  40.43 42.23 2
 23.06 21.69 28.26 25.4  30.25 24.94 28.36 30.98 32.73 35.04 27.03 33.61 3
 30.59 29.92 32.84 31.8  31.   40.83 35.66 32.11 34.28 30.6  31.24 28.41 3
 16.22 21.91 16.36 22.25 25.11 10.92 24.23 25.84 27.95 24.1  29.05 32.76 2
 35.83 30.75 23.64 33.25 38.71 37.76 31.29 24.53 29.77 32.81 27.97 27.98 2
 19.94 21.61 24.73 24.6  25.3  25.93 31.94 23.9  21.61 37.41 44.16 36.35 3
 31.22 41.11 39.03 25.15 21.9  27.03 28.25 36.16 35.68 33.42 35.7  34.68 2
 26.89 20.52 26.45 26.64 26.65 33.49 34.68 31.78 24.86 23.32 27.87 26.6  1
 33.48 30.41 35.59 32.37 28.63 23.24 17.54 26.55 22.9  25.35 25.61 20.22 1
 19.16 25.37 25.18 24.07 19.7  20.66 24.03 21.34 19.63 23.03 23.   22.44 2
 22.49 28.01 29.2  16.92 15.07 25.87 27.99 23.62 21.56 21.56 17.59 26.34 1
 19.89 18.39 20.8  39.57 12.47 14.91  8.22 22.11 32.26 34.45 24.36 25.44
 15.71 19.04 13.52 13.16  2.6   8.23  5.81  5.74  6.28 14.13 17.3  17.77
 11.35 12.48 18.38 18.81 12.92  7.72  8.73  6.72 19.24 12.9  19.5  13.63 1
  6.01 15.21 20.12 18.36 17.62 12.28 12.99  9.44 14.98 13.77 14.13 12.99 1
  9.07  5.37 13.62 13.48 18.33 19.51 19.1  11.96 13.46 18.45 18.99 18.18 1
 12.48 12.67 17.79 19.14 19.82 20.93 20.57 23.21 20.68 17.89 13.71 16.42 1
 15.66 20.63 10.87 19.01 21.58 22.86 26.68 28.37 20.35 19.18 22.24 19.44 2
 20.86 17.23 14.03 19.23 21.47 18.32 20.64 24.45 22.85 28.52 26.97 22.74]
```

In [ ]: