Example of Regression Based on the Boston Housing Data Set.

In [65]:
```
from sklearn.cross_validation import KFold
from sklearn.linear_model import LinearRegression, Lasso, Ridge,
ElasticNet
from sklearn.linear_model import LassoCV, RidgeCV
import numpy as np
import pylab as pl
from sklearn.datasets import load_boston
boston = load_boston()
x = np.array([np.concatenate((v,[1])) for v in boston.data])
```

In [2]:
```
print y[:10]
```

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

In [4]:
```
print x[:2]
```

```
[[  6.32000000e-03   1.80000000e+01   2.31000000e+00   0.00000000e+00
    5.38000000e-01   6.57500000e+00   6.52000000e+01   4.09000000e+00
    1.00000000e+00   2.96000000e+02   1.53000000e+01   3.96900000e+02
    4.98000000e+00   1.00000000e+00]
 [  2.73100000e-02   0.00000000e+00   7.07000000e+00   0.00000000e+00
    4.69000000e-01   6.42100000e+00   7.89000000e+01   4.96710000e+00
    2.00000000e+00   2.42000000e+02   1.78000000e+01   3.96900000e+02
    9.14000000e+00   1.00000000e+00]]
```

In [17]:
```
# Create linear regression object
linreg = LinearRegression()

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

Out[17]:
```
LinearRegression(copy_X=True, fit_intercept=True, normalize=False)
```

In [69]:
```
# Compute RMSE on training data
p = np.array([linreg.predict(xi) for xi in x])
err = p-y
# Dot product of error vector with itself gives us the sum of squared
errors
total_error = np.dot(err,err)
# Compute RMSE
rmse_train = np.sqrt(total_error/len(p))
```

```
4.68866198421
```

In [75]:

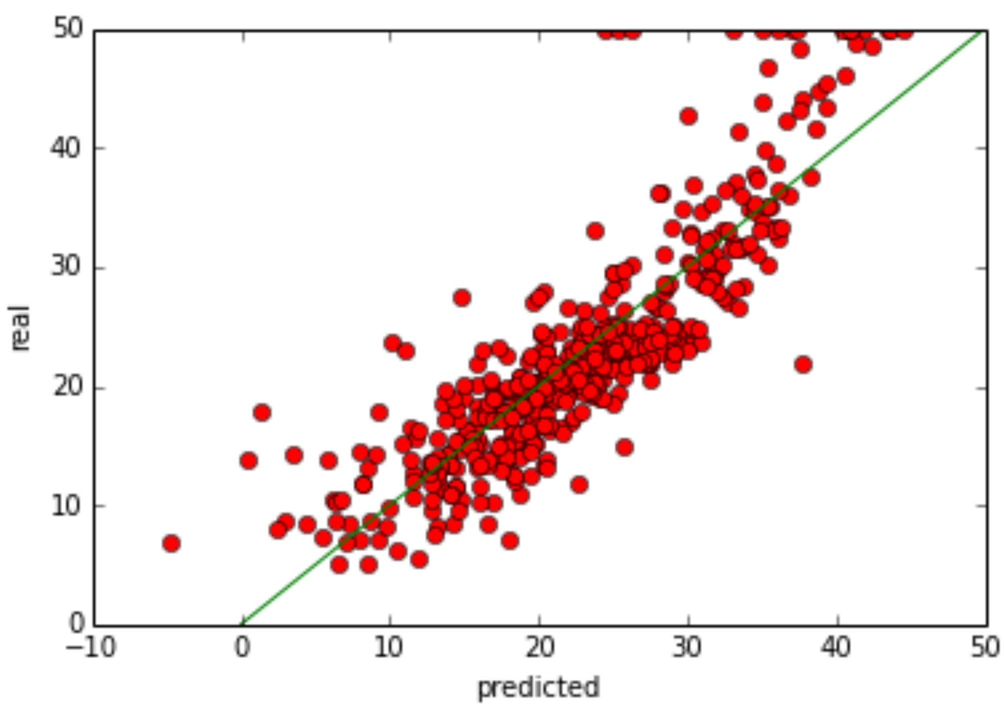
```
# We can view the coefficients
print('Regression Coefficients: ', linreg.coef_)
```

```
('Regression Coefficients: ', array([ -1.04755725e-01,   4.91233643e-02,
 3.24299720e-02,
         2.51517135e+00,  -1.76585750e+01,   3.81259444e+00,
         1.06438518e-02,  -1.43651798e+00,   3.60959247e-01,
        -1.54635990e-02,  -9.13025678e-01,   9.94705988e-03,
        -5.55769911e-01,   0.00000000e+00]))
```

In [77]:

```
# Plot outputs
pl.plot(p, y,'ro')
pl.plot([0,50],[0,50], 'g-')
pl.xlabel('predicted')
pl.ylabel('real')
pl.show()

# pl.scatter(p, y,  color='black')
```



In [78]:

```
# Compute RMSE using 10-fold x-validation
kf = KFold(len(x), n_folds=10)
xval_err = 0
for train,test in kf:
    linreg.fit(x[train],y[train])
    p = np.array([linreg.predict(xi) for xi in x[test]])
    e = p-y[test]
    xval_err += np.dot(e,e)
rmse_10cv = np.sqrt(xval_err/len(x))
```

In [79]:

```
method_name = 'Linear Regression'
print('Method: %s' %method_name)
print('RMSE on training: %.4f' %rmse_train)
```

```
print('RMSE on 10-fold CV: %.4f' %rmse_10cv)
```

```
Method: Linear Regression
RMSE on training: 4.6887
RMSE on 10-fold CV: 5.8819
```

Let's try Ridge Regression:

In [54]:

```
# Create linear regression object
ridge = Ridge(fit_intercept=True, alpha=0.5)

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

Out[54]:

```
Ridge(alpha=0.5, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, solver='auto', tol=0.001)
```

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

In [55]:

```
# Compute RMSE on training data
p = np.array([ridge.predict(xi) for xi in 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
kf = KFold(len(x), n_folds=10)
xval_err = 0
for train,test in kf:
    ridge.fit(x[train],y[train])
    p = np.array([ridge.predict(xi) for xi in x[test]])
    e = p-y[test]
    xval_err += np.dot(e,e)
rmse_10cv = np.sqrt(xval_err/len(x))

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

```
Method: Ridge Regression
RMSE on training: 4.6857
RMSE on 10-fold CV: 5.8428
```

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

In [62]:

```
a = 0.5
for name,met in [
        ('linear regression', LinearRegression()),
```

```
        ('elastic-net', ElasticNet(fit_intercept=True, alpha=a)),
        ('lasso', Lasso(fit_intercept=True, alpha=a)),
        ('ridge', Ridge(fit_intercept=True, alpha=a)),
        ]:
    met.fit(x,y)
    p = np.array([met.predict(xi) for xi in x])
    e = p-y
    total_error = np.dot(e,e)
    rmse_train = np.sqrt(total_error/len(p))

    kf = KFold(len(x), n_folds=10)
    err = 0
    for train,test in kf:
        met.fit(x[train],y[train])
        p = np.array([met.predict(xi) for xi in x[test]])
        e = p-y[test]
        err += np.dot(e,e)

    rmse_10cv = np.sqrt(err/len(x))
    print('Method: %s' %name)
    print('RMSE on training: %.4f' %rmse_train)
    print('RMSE on 10-fold CV: %.4f' %rmse_10cv)
```

```
Method: linear regression
RMSE on training: 4.6795
RMSE on 10-fold CV: 5.8819


Method: elastic-net
RMSE on training: 4.9855
RMSE on 10-fold CV: 5.4779


Method: lasso
RMSE on training: 4.9141
RMSE on 10-fold CV: 5.7368


Method: ridge
RMSE on training: 4.6857
RMSE on 10-fold CV: 5.8428
```

In [86]:

```
cd C:\WinPython27\Data
```

```
C:\WinPython27\Data
```

Using the regression implementation from Machine Learning in Action, Chapter 8:

In [94]:

```
def standRegres(xArr,yArr):
```

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

In [95]:

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

In [96]:

```
print w
```

```
[[ -1.07170557e-01]
 [  4.63952195e-02]
 [  2.08602395e-02]
 [  2.68856140e+00]
 [ -1.77957587e+01]
 [  3.80475246e+00]
 [  7.51061703e-04]
 [ -1.47575880e+00]
 [  3.05655038e-01]
 [ -1.23293463e-02]
 [ -9.53463555e-01]
 [  9.39251272e-03]
 [ -5.25466633e-01]
 [  3.64911033e+01]]
```

In [99]:

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

In [100]:

```
w_ridge = ridgeRegres(x,y,0.5)
print w_ridge
```

```
[[ -1.00258044e-01]
 [  4.76559911e-02]
 [ -6.63573226e-04]
 [  2.68040479e+00]
 [ -9.55123875e+00]
 [  4.55214996e+00]
 [ -4.67446118e-03]
 [ -1.25507957e+00]
```

```
[  2.52066137e-01]
[ -1.15766049e-02]
[ -7.26125030e-01]
[  1.14804636e-02]
[ -4.92130481e-01]
[  2.17772079e+01]]
```

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

In [104]:
```
xMat=mat(x)
yMat=mat(y)
yHat = xMat*w_ridge
```

In [105]:
```
print yHat[0:10]
```

```
[[ 29.80808276]
 [ 24.75277329]
 [ 30.78188454]
 [ 29.12268607]
 [ 28.60788228]
 [ 25.35402577]
 [ 22.47871664]
 [ 19.28185025]
 [ 11.2059811 ]
 [ 18.64883549]]
```

In [107]:
```
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.