Sucheta Naik

182000492

This dataset contains information collected by the U.S Census Service concerning housing in the area of Boston Mass.

In [ ]:
```
Import packages from scikitlearn
```

In [98]:
```
from sklearn.cross_validation import KFold
from sklearn.linear_model import LinearRegression, Lasso, Ridge
import numpy as np
import pylab as pl
import matplotlib.pyplot as plt
```

This code will load dataset from scikit

In [72]:
```
from sklearn.datasets import load_boston
boston = load_boston()
```

In [73]:
```
print(boston.feature_names)
```

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

In [74]:
```
print(boston.data.shape)
print(boston.target.shape)
```

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

In [75]:
```
np.set_printoptions(precision=2, linewidth=120, suppress=True, edge
items=4)
```

In [76]:
```
print(boston.data)
```

```
[[   0.01   18.       2.31    0.    ...,  296.     15.3   396.9
4.98]
 [   0.03    0.       7.07    0.    ...,  242.     17.8   396.9
9.14]
 [   0.03    0.       7.07    0.    ...,  242.     17.8   392.83
4.03]
 [   0.03    0.       2.18    0.    ...,  222.     18.7   394.63
2.94]
 ...,
 [   0.05    0.      11.93    0.    ...,  273.     21.    396.9
9.08]
 [   0.06    0.      11.93    0.    ...,  273.     21.    396.9
5.64]
 [   0.11    0.      11.93    0.    ...,  273.     21.    393.45
6.48]
 [   0.05    0.      11.93    0.    ...,  273.     21.    396.9
7.88]]
```

This code will add a column of 1s for x0 in order to do multiple regression

In [77]:
```
x = np.array([np.concatenate((v,[1])) for v in boston.data])
y = boston.target
```

This code will print first 10 elements of the data

In [78]:
```
print(x[:10])
```

```
[[   0.01   18.       2.31    0.       0.54    6.58   65.2     4.09
1.    296.     15.3   396.9     4.98    1.  ]
 [   0.03    0.       7.07    0.       0.47    6.42   78.9     4.97
2.    242.     17.8   396.9     9.14    1.  ]
 [   0.03    0.       7.07    0.       0.47    7.18   61.1     4.97
2.    242.     17.8   392.83    4.03    1.  ]
 [   0.03    0.       2.18    0.       0.46    7.     45.8     6.06
3.    222.     18.7   394.63    2.94    1.  ]
 [   0.07    0.       2.18    0.       0.46    7.15   54.2     6.06
3.    222.     18.7   396.9     5.33    1.  ]
 [   0.03    0.       2.18    0.       0.46    6.43   58.7     6.06
3.    222.     18.7   394.12    5.21    1.  ]
 [   0.09   12.5      7.87    0.       0.52    6.01   66.6     5.56
5.    311.     15.2   395.6    12.43    1.  ]
 [   0.14   12.5      7.87    0.       0.52    6.17   96.1     5.95
5.    311.     15.2   396.9    19.15    1.  ]
 [   0.21   12.5      7.87    0.       0.52    5.63  100.      6.08
5.    311.     15.2   386.63   29.93    1.  ]
 [   0.17   12.5      7.87    0.       0.52    6.     85.9     6.59
5.    311.     15.2   386.71   17.1     1.  ]]
```

This code will print first 10 elements of the response variable

In [79]:
```
print(y[:10])
```

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

Implement Regular Linear Regression:

This code will create linear regression object

In [80]:
```
linreg = LinearRegression()
```

This code will train the model using the training sets

In [81]:
```
linreg.fit(x,y)
```

Out[81]:
```
LinearRegression(copy_X=True, fit_intercept=True, n_jobs=1, normal
ize=False)
```

This code will give predictions for the first 10 instances

In [82]:
```
print(linreg.predict(x[:10]))
```

```
[ 30.01   25.03   30.57   28.61   27.94   25.26   23.     19.53   11.52   1
8.92]
```

This code will compute RMSE on training data

In [83]:
```
#p = np.array([linreg.predict(xi) for xi in x])
p = linreg.predict(x)
```

This code will constuct a vector of errors

In [41]:
```
err = abs(p-y)
```

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

This code will show the error on the first 10 predictions

In [67]:
```
print(err[:10])
```

```
[ 6.82  3.63 -4.31 -3.9  -7.85 -2.06  0.88 -7.58 -5.62  0.81]
```

This code will give dot product of error vector with itself that gives us the sum of squared errors

In [63]:
```
total_error = np.dot(err,err)
```

This code will compute RMSE

In [64]:
```
rmse_train = np.sqrt(total_error/len(p))
print(rmse_train)
```

```
15.6326388252
```

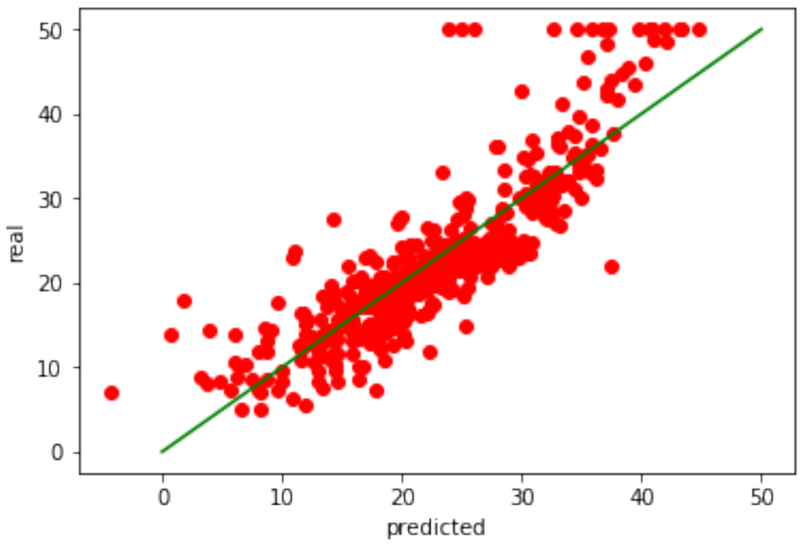
This code will view the regression coefficients

In [43]:
```
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  -0.95   0.01  -0.53   0.  ]
```

This code will plot outputs

In [44]:
```
%matplotlib inline
pl.plot(p, y,'ro')
pl.plot([0,50],[0,50], 'g-')
pl.xlabel('predicted')
pl.ylabel('real')
pl.show()
```



This code will compute RMSE using 10-fold x-validation

In [114]:
```
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]])
    p = linreg.predict(x[test])
    e = p-y[test]
    xval_err += np.dot(e,e)

rmse_10cv = np.sqrt(xval_err/len(x))
```

This code will give results of rmse on training and rmse on 10 fold cv

In [46]:
```
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: 5.8819
```

Implement Ridge Regression:

This code will create linear regression object with a ridge coefficient 0.5

In [84]:
```
ridge = Ridge(fit_intercept=True, alpha=0.5)
```

This code will train the model using the training set

In [85]:
```
ridge.fit(x,y)
```

Out[85]:
```
Ridge(alpha=0.5, copy_X=True, fit_intercept=True, max_iter=None,
   normalize=False, random_state=None, solver='auto', tol=0.001)
```

This code will compute RMSE on training data

In [86]:
```
# p = np.array([ridge.predict(xi) for xi in x])
p = ridge.predict(x)
err = p-y
total_error = np.dot(err,err)
rmse_train = np.sqrt(total_error/len(p))
print(rmse_train)
```

```
4.68570791668
```

This code will compute RMSE using 10-fold x-validation and print rmse on training and rmse on 10 fold cv

In [87]:
```
kf = KFold(len(x), n_folds=10)
xval_err = 0
for train,test in kf:
    ridge.fit(x[train],y[train])
    p = ridge.predict(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
```

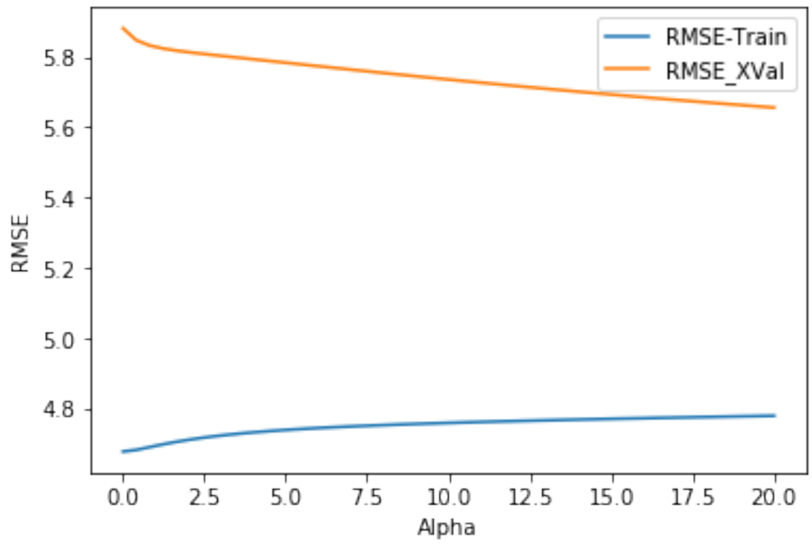
This code will try different values of alpha and observe the impact on x-validation RMSE

In [90]:
```
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(fit_intercept=True, 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))

    # computing RMSE using 10-fold cross validation
    kf = KFold(len(x), n_folds=10)
    xval_err = 0
    for train, test in kf:
        ridge.fit(x[train], y[train])
        p = ridge.predict(x[test])
        err = p - y[test]
        xval_err += np.dot(err,err)
    rmse_10cv = np.sqrt(xval_err/len(x))

    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_10c
v))
```

```
Ridge Regression
alpha    RMSE_train      RMSE_10cv

0.010    4.6795          5.8806
0.418    4.6842          5.8467
0.826    4.6921          5.8319
1.234    4.7000          5.8234
1.642    4.7070          5.8175
2.050    4.7133          5.8126
2.458    4.7187          5.8082
2.866    4.7234          5.8041
3.274    4.7276          5.8000
3.682    4.7313          5.7960
4.090    4.7346          5.7920
4.498    4.7375          5.7880
4.906    4.7402          5.7840
5.313    4.7426          5.7800
5.721    4.7448          5.7760
6.129    4.7469          5.7720
6.537    4.7488          5.7680
6.945    4.7505          5.7641
7.353    4.7522          5.7602
7.761    4.7537          5.7563
8.169    4.7552          5.7524
8.577    4.7565          5.7485
8.985    4.7578          5.7447
9.393    4.7591          5.7410
9.801    4.7603          5.7372
10.209   4.7614          5.7335
10.617   4.7625          5.7298
11.025   4.7635          5.7262
11.433   4.7646          5.7226
11.841   4.7655          5.7190
12.249   4.7665          5.7155
12.657   4.7674          5.7120
13.065   4.7683          5.7086
13.473   4.7692          5.7052
13.881   4.7700          5.7018
14.289   4.7708          5.6985
14.697   4.7717          5.6952
15.104   4.7724          5.6919
15.512   4.7732          5.6887
15.920   4.7740          5.6856
16.328   4.7747          5.6824
16.736   4.7755          5.6793
17.144   4.7762          5.6762
17.552   4.7769          5.6732
17.960   4.7776          5.6702
18.368   4.7783          5.6672
18.776   4.7790          5.6643
19.184   4.7797          5.6614
19.592   4.7804          5.6585
20.000   4.7811          5.6557
```

This code will plot output

In [50]:
```
pl.plot(alpha, t_rmse, label='RMSE-Train')
pl.plot(alpha, cv_rmse, label='RMSE_XVal')
pl.legend( ('RMSE-Train', 'RMSE_XVal') )
pl.ylabel('RMSE')
pl.xlabel('Alpha')
pl.show()
```



Implement Lasso Regression:

This code will create linear regression object with a lasso coefficient 0.5

In [91]:
```
lasso = Lasso(fit_intercept=True, alpha=0.5)
```

This code will train the model using the training set

In [92]:
```
lasso.fit(x,y)
```

Out[92]:
```
Lasso(alpha=0.5, copy_X=True, fit_intercept=True, max_iter=1000,
   normalize=False, positive=False, precompute=False, random_state
=None,
   selection='cyclic', tol=0.0001, warm_start=False)
```

This code will compute RMSE on training data

In [94]:
```
# p = np.array([ridge.predict(xi) for xi in x])
p = lasso.predict(x)
err = p-y
total_error = np.dot(err,err)
rmse_train = np.sqrt(total_error/len(p))
print(rmse_train)
```

```
4.91407791181
```

This code will compute RMSE using 10-fold x-validation and will give results of rmse on training and rmse on 10 fold cv

In [95]:
```
kf = KFold(len(x), n_folds=10)
xval_err = 0
for train,test in kf:
    lasso.fit(x[train],y[train])
    p = lasso.predict(x[test])
    e = p-y[test]
    xval_err += np.dot(e,e)
rmse_10cv = np.sqrt(xval_err/len(x))

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

```
Method: Lasso Regression
RMSE on training: 4.9141
RMSE on 10-fold CV: 5.7368
```

This code will compare across methods easier, let's parametrize the regression methods:

In [113]:
```
a = 0.5
for name,met in [
        ('Regular linear regression', LinearRegression()),
        ('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])
    p = met.predict(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 = met.predict(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)
    print("\n")
```

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


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
```

The Regular linear regression can yield either a more accurate or a more interpretable model than lasso and ridge regression as the root mean square error is less than lasso and ridge regression. The lasso regression has lower root mean square error(rmse)on 10-fold cross validation(cv) than regular linear regression and ridge regression. The RMSE difference between Regular linear regression and ridge regression is quite small and hence proves that both are closely related. But RMSE for 10cv between lasso and ridge regression is small but significant indicating their uniqueness.