CS4487 - Machine Learning

Lecture 5a - Supervised Learning - Regression

Dr. Antoni B. Chan

Dept. of Computer Science, City University of Hong Kong

Outline

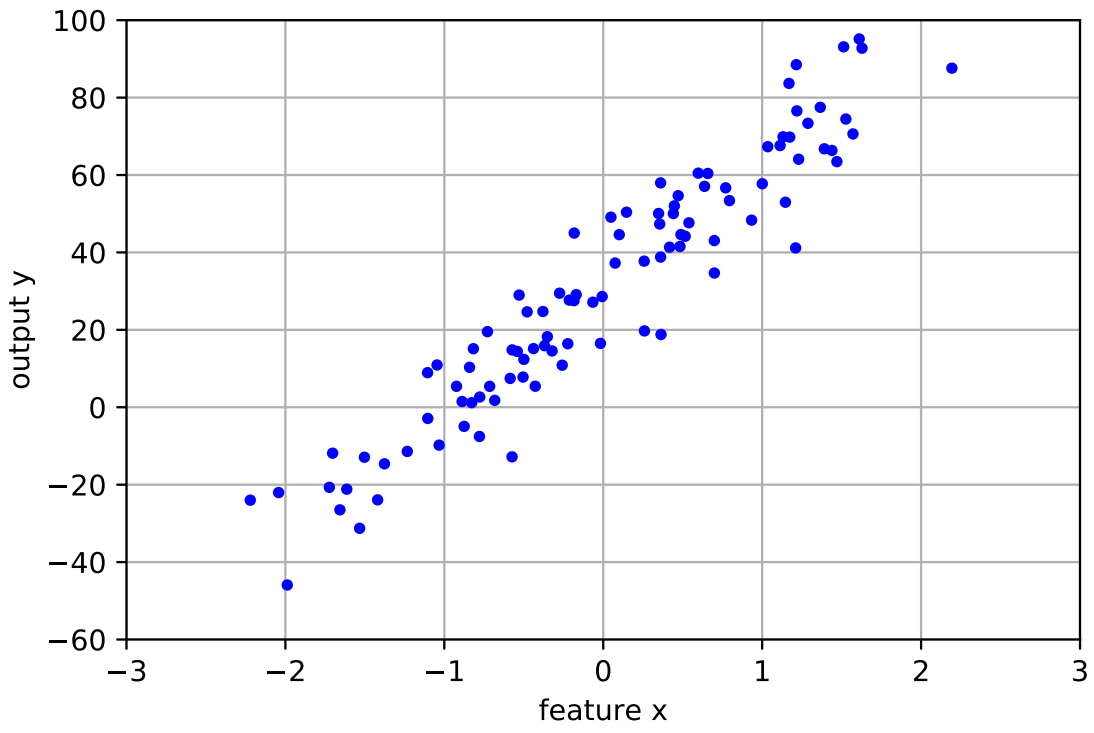
1. Linear Regression
2. Selecting Features
3. Removing Outliers
4. Non-linear regression

Regression

- Supervised learning
  - Input observation $\mathbf{x}$, typically a vector in $\mathbb{R}^d$.
  - Output $y \in \mathbb{R}$, a real number.
- **Goal:** predict output y from input $\mathbf{x}$.
  - i.e., learn the function $y = f(\mathbf{x})$.

In [3]:
```
linfig
```

Out[3]:

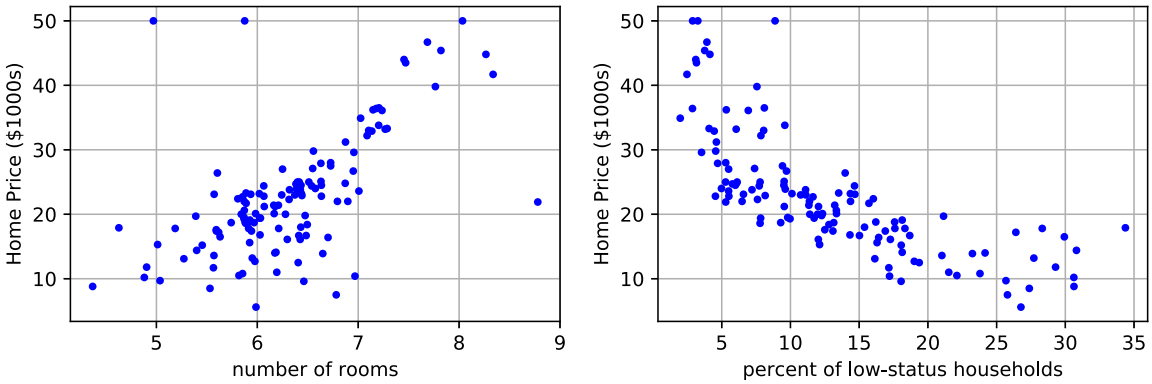


Examples:

- Predict Boston house price from number of rooms, or percentage of low-status households in neighborhood.

In [5]:
```
boston1dfig
```

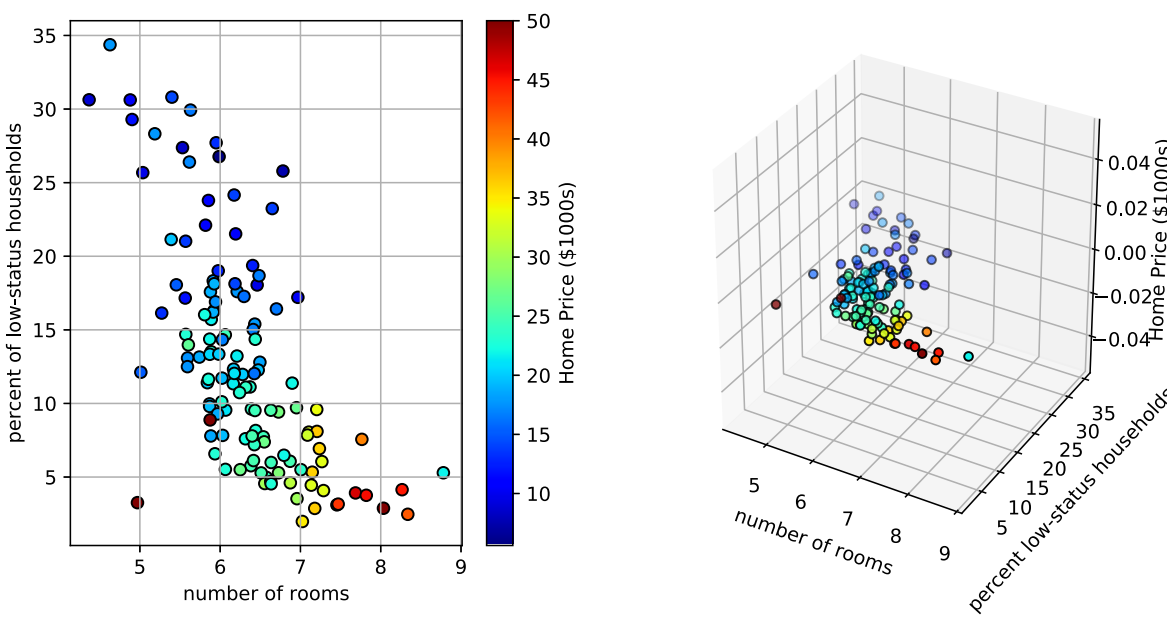
Out[5]:



- predict from both features

In [6]:
```
boston2dfig
```

Out[6]:

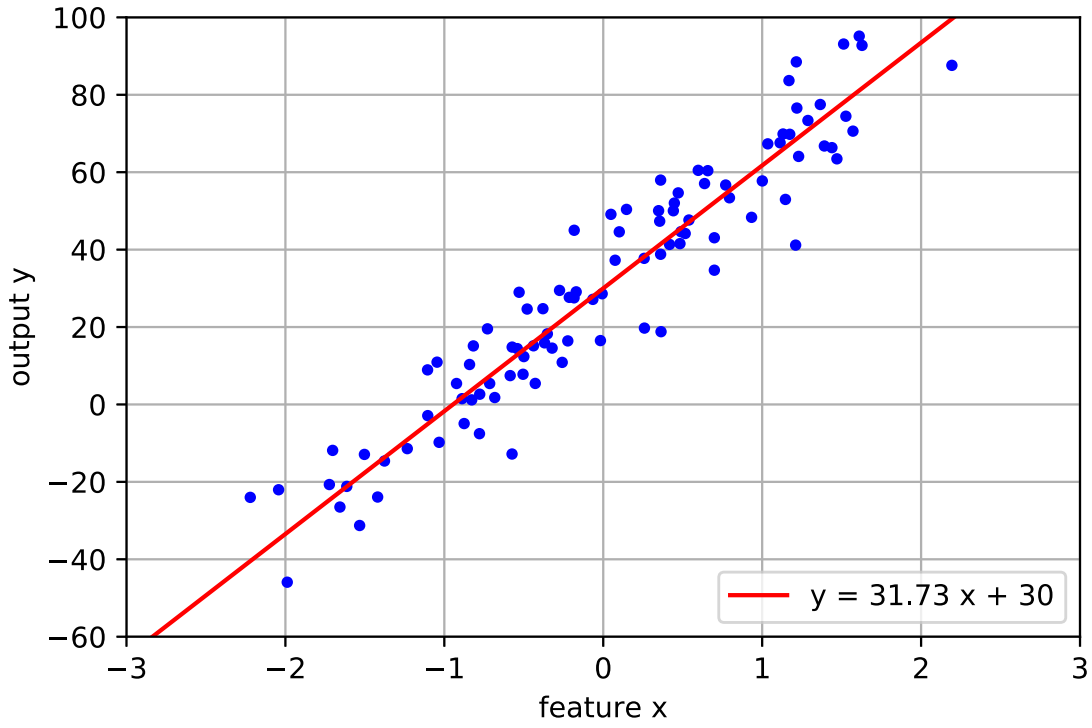


Linear Regression

- **1-d case:** the output y is a linear function of input feature x
  - $y = w * x + b$
  - w is the slope, b is the intercept.

In [8]:
```
linfig
```

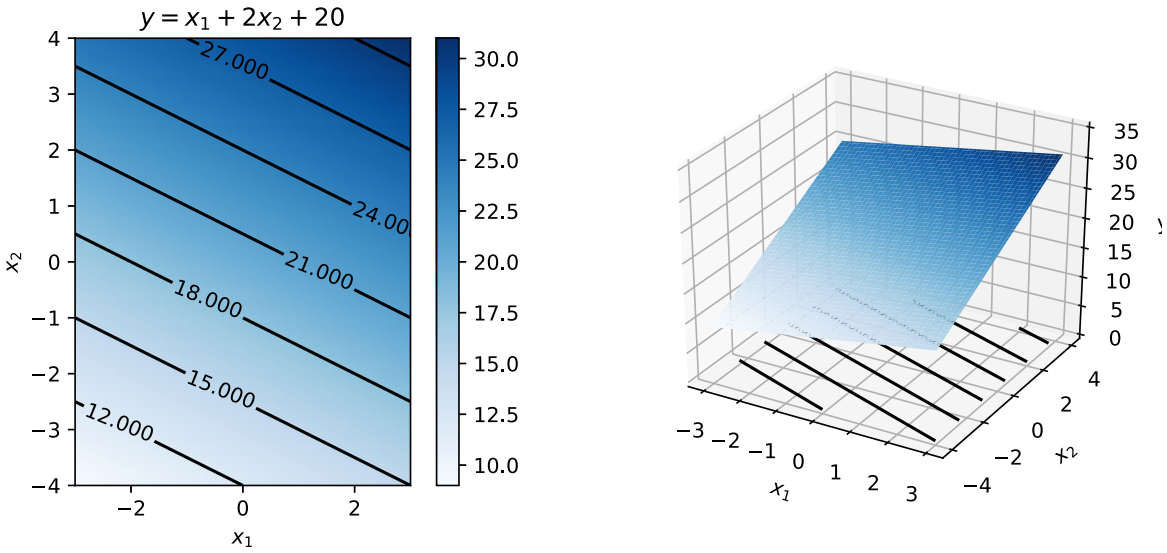
Out[8]:



- **d-dim case**: the output y is a linear combination of d input variables $x_1, \cdots, x_d$:
  - $y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$
- Equivalently,
  - $y = w_0 + \mathbf{w}^T \mathbf{x} = w_0 + \sum_{j=1}^{d} w_j x_j$
  - $\mathbf{x} \in \mathbb{R}^d$ is the vector of input values.
  - $\mathbf{w} \in \mathbb{R}^d$ are the weights of the linear function, and w_0 is the intercept (bias term).

In [10]:
```
lin2dfig
```

Out[10]:



Ordinary Least Squares (OLS)

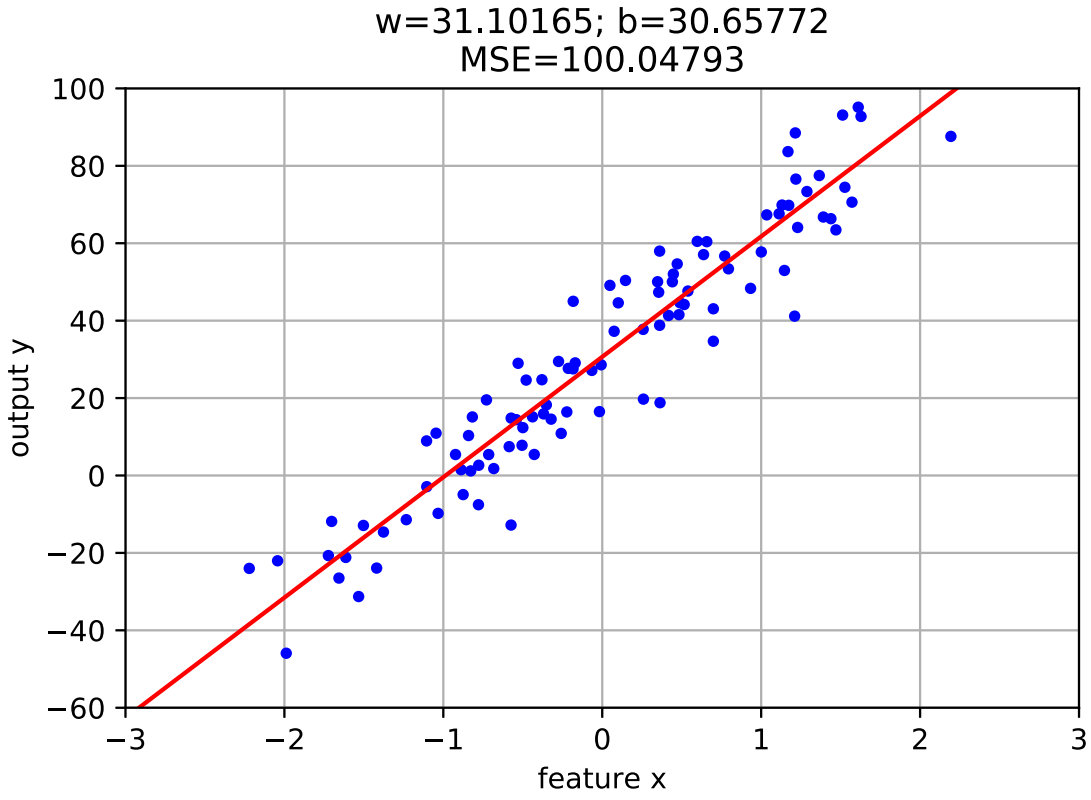
- The linear function has form $f(\mathbf{x}) = \mathbf{w}^T\mathbf{x} + b$.
- *How to estimate the parameters* $(\mathbf{w}, b)$ *from the data?*
- Fit the parameters by minimizing the squared prediction error on the training set $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$:

$$\min_{\mathbf{w},b} \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2 = \min_{\mathbf{w},b} \sum_{i=1}^{N} (y_i - (\mathbf{w}^T\mathbf{x}_i + b))^2$$

- closed-form solution: $\mathbf{w}^* = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{y}$
  - where $\mathbf{X} = [\mathbf{x}_1 \cdots \mathbf{x}_N]$ is the data matrix
  - and $\mathbf{y} = [y_1, \cdots, y_N]^T$ is vector of outputs.
  - Note: $(\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}$ is also called the *pseudo-inverse* of $\mathbf{X}$.

Examples: 1-d

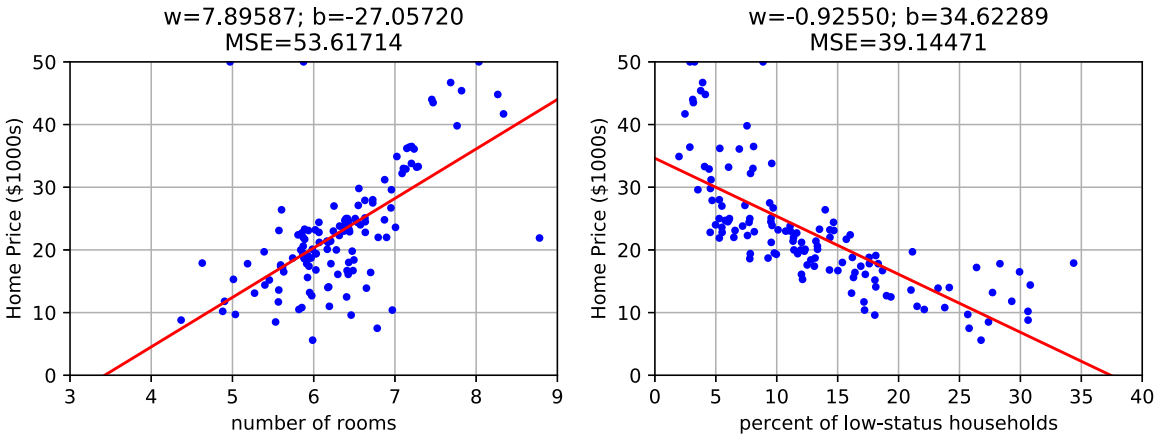
In [12]:
```
# fit using ordinary least squares
ols = linear_model.LinearRegression()
ols.fit(linX, linY)

# show plot
axbox = [-3, 3, -60, 100]
plt.figure()
plot_linear_1d(ols, axbox, linX, linY)
plt.xlabel('feature x'); plt.ylabel('output y');
```



Boston housing price (1d)

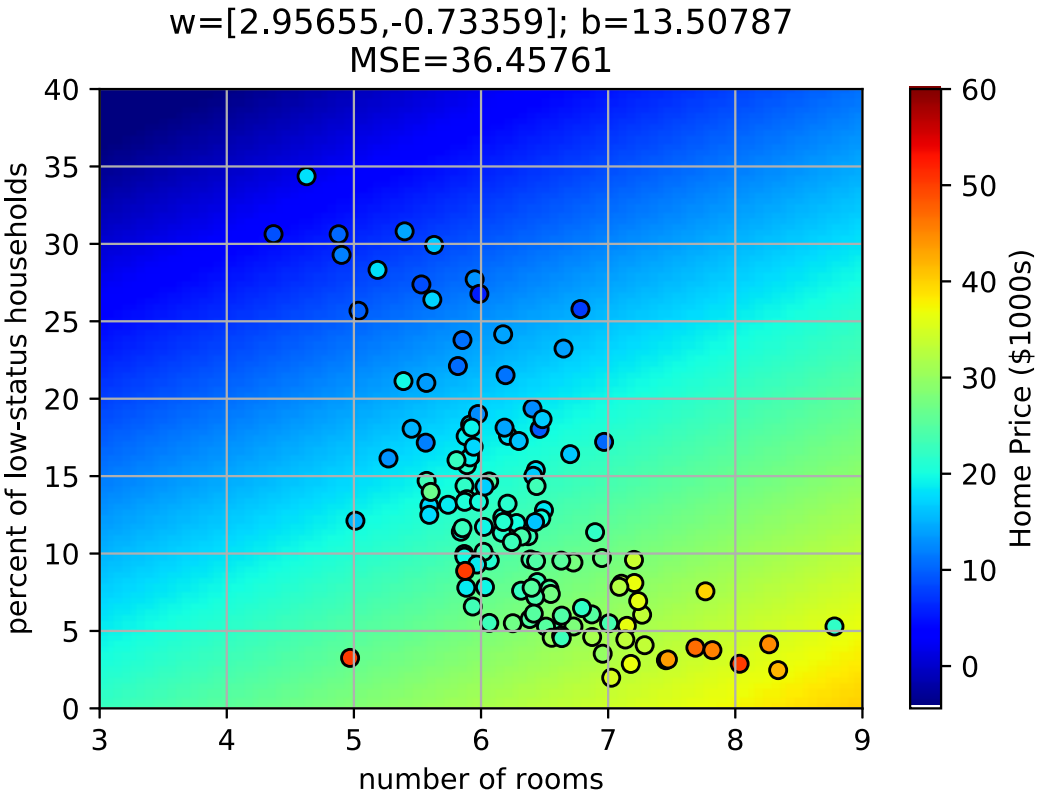
- learn regression function for each feature separately

In [13]:
```
# fit each dimension of Boston data
plt.figure(figsize=(10,3))

bostonFeats = ('number of rooms', 'percent of low-status households')
bostonaxbox = ([3,9,0,50], [0,40,0,50])

for i in range(2):
    ols = linear_model.LinearRegression()
    tmpX = bostonX[:,i][:,newaxis]
    ols.fit(tmpX, bostonY)

    plt.subplot(1,2,i+1)
    plot_linear_1d(ols, bostonaxbox[i], tmpX, bostonY)
    plt.ylabel('Home Price ($1000s)'); plt.xlabel(bostonFeats[i])
```



- for both features together

In [16]:
```
# learn with both dimensions
ols = linear_model.LinearRegression()
ols.fit(bostonX, bostonY)

# make figure
plt.figure()
plot_linear_2d(ols, bostonaxbox2, bostonX, bostonY)
cbar = plt.colorbar()
cbar.set_label('Home Price ($1000s)')
plt.xlabel('number of rooms')
plt.ylabel('percent of low-status households');
```



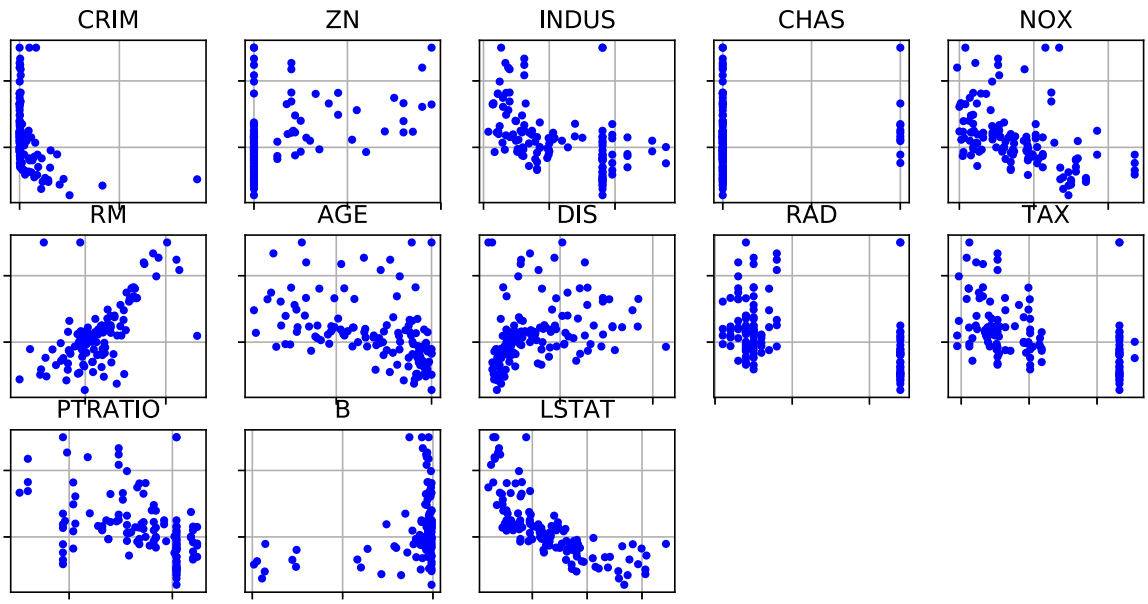
- interpretation from the linear model parameters:
  - each room increases home price by \$2956 (w_1)
  - each percentage of low-status households decreases home price by \$733 (w_2)
  - the "starting" price is \$13,508 (b).

Selecting Features

- The Boston housing data actually has 13 features.
  - plots of feature vs. housing price

In [18]:
```
bostonffig
```

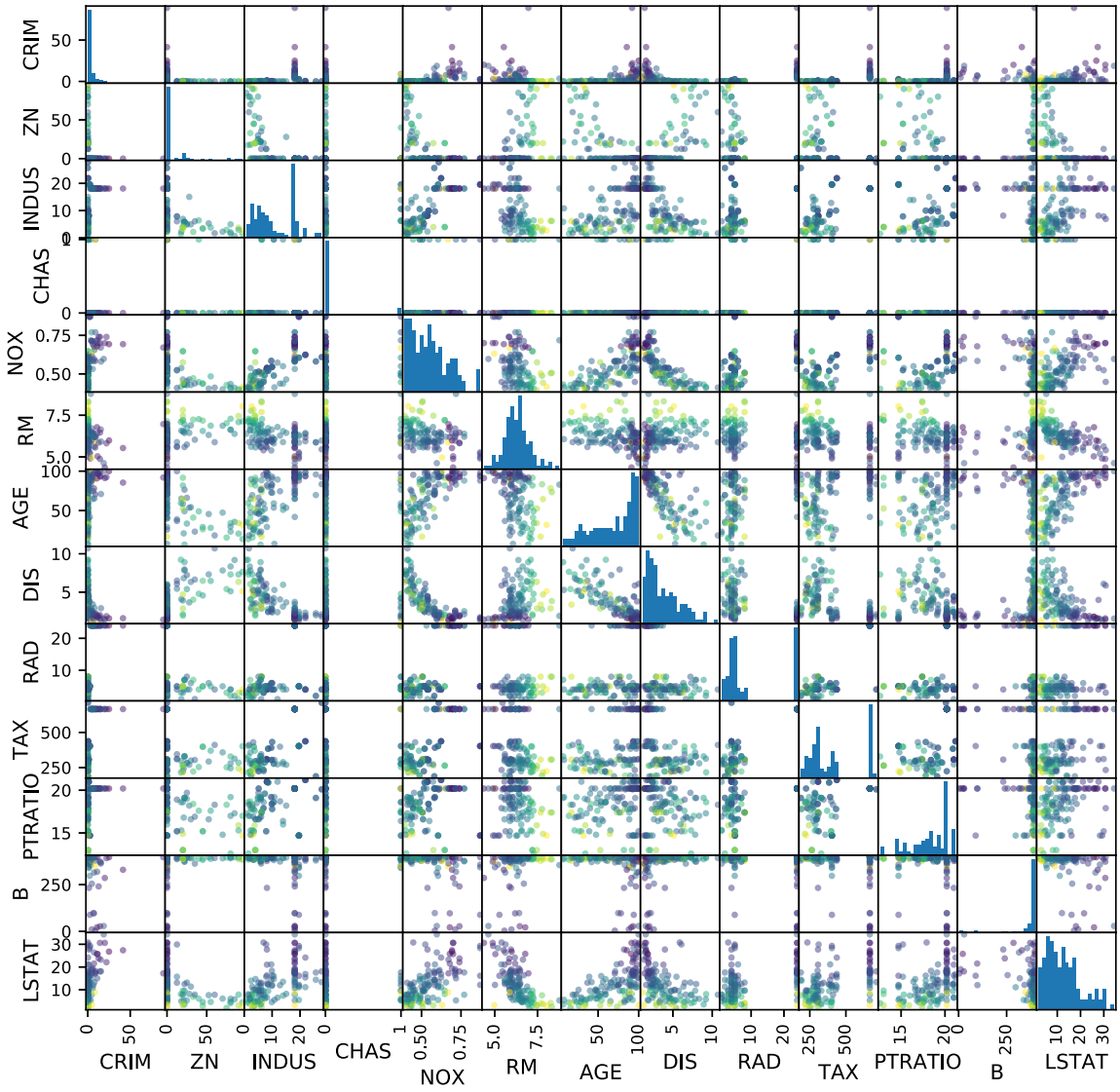
Out[18]:




```
      CRIM = per capita crime rate by town`
        ZN = proportion of residential land zoned for lots ove
r 25,000 sq.ft.`
     INDUS = proportion of non-retail business acres per town
      CHAS = Charles River dummy variable (= 1 if tract bounds
river; 0 otherwise)
       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 cent
res
       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 b
lacks by town
     LSTAT = % lower status of the population
```

- Use `pandas` to view pairwise relationships
  - diagonal shows the histogram
  - off-diagonal shows plots for two features at a time

In [19]:
```
import pandas as pd
boston_feature_names = [x[0] for x in bostonAttr]
boston_df = pd.DataFrame(bostonX, columns=boston_feature_names)

tmp=pd.plotting.scatter_matrix(boston_df, c=bostonY, figsize=(9, 9),
                               marker='o', hist_kwds={'bins': 20}, s=10,
                               alpha=.5)
```



- *Can we select a few features that are good for predicting the price?*
  - This will provide some insight about our data and what is important.

Shrinkage

- Add a *regularization* term to "shrink" some linear weights to zero.
  - features associated with zero weight are not important since they aren't used to calculate the function output.
  - $y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$

Ridge Regression

- Add regularization term to OLS:

$$\min_{\mathbf{w},b} \alpha||\mathbf{w}||^2 + \sum_{i=1}^{N}(y_i - f(\mathbf{x}_i))^2$$

- the first term is the *regularization term*
  - $||\mathbf{w}||^2 = \sum_{j=1}^{d} w_j^2$ penalizes large weights.
  - α is the hyperparameter that controls the amount of shrinkage
    - larger α means more shrinkage.
    - $\alpha = 0$ is the same as OLS.

- the second term is the *data-fit term*
  - sum-squared error of the prediction.
- Also has a closed-form solution:
  - $\mathbf{w}^* = (\mathbf{X}\mathbf{X}^T + \alpha I)^{-1}\mathbf{X}\mathbf{y}$
  - (The term "ridge regression" comes from the closed-form solution, where a "ridge" is added to the diagonal of the covariance matrix)

Example on Boston data

In [20]:
```
# randomly split data into 80% train and 20% test set
trainX, testX, trainY, testY = \
  model_selection.train_test_split(boston['data'], boston['target'],
  train_size=0.8, test_size=0.2, random_state=4487)

# normalize feature values to zero mean and unit variance
# this makes comparing weights more meaningful
#   feature value 0 means the average value for that features
#   feature value of +1 means one standard deviation above average
#   feature value of -1 means one standard deviation below average
scaler = preprocessing.StandardScaler()
trainXn = scaler.fit_transform(trainX)
testXn  = scaler.transform(testX)

print(trainXn.shape)
print(testXn.shape)
```

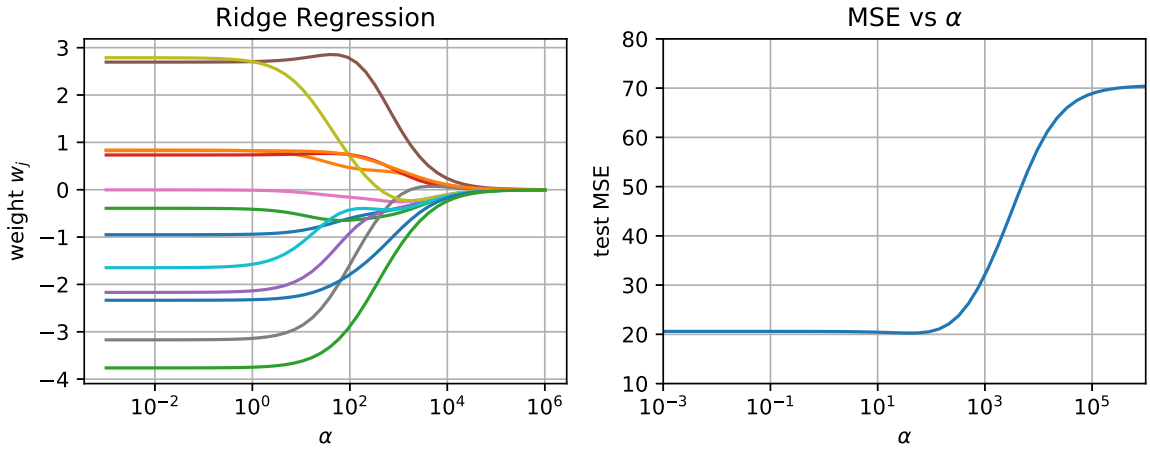
```
(404, 13)
(102, 13)
```

- vary α from 10^{-3} (little shrinkage) to 10^6 (lots of shrinkage)
  - for small α, all weights are non-zero.
  - for large α, all weights shrink to 0.
  - somewhere in between is the best model...

In [21]:
```
# alpha values to try
alphas = logspace(-3,6,50)

MSEs = empty(len(alphas))
ws   = empty((len(alphas), trainXn.shape[1]))
for i,alpha in enumerate(alphas):
    # learn the RR model
    rr = linear_model.Ridge(alpha=alpha)
    rr.fit(trainXn, trainY)
    ws[i,:] = rr.coef_   # save weights

    MSEs[i] = metrics.mean_squared_error(testY, rr.predict(testXn))
```

In [23]:
```
rfig
```

Out[23]:



Selecting α using cross-validation

- built-in cross-validation (RidgeCV)

In [24]:
```
# train RR with cross-validation
rr = linear_model.RidgeCV(alphas=alphas, cv=5)
rr.fit(trainXn, trainY)

MSE = metrics.mean_squared_error(testY, rr.predict(testXn))
print("MSE =", MSE)
print("alpha =", rr.alpha_)
print("w =", rr.coef_)
```

```
MSE = 20.427112422027655
alpha = 10.985411419875595
w = [-0.85419239  0.71385132 -0.53970282  0.75571123 -1.85577534  2.78267491
 -0.06063119 -2.8538377   2.10475562 -1.0968367  -2.22304827  0.81576653
 -3.61306393]
```

Interpretation

- Which weights are most important?
  - negative weights indicate factors that decrease the house price
    - *Examples:* LSTAT (having higher percentage of lower status population), DIS (distance to business areas), PTRATIO (higher student-teacher ratio)
  - positive weights indicate factors that increase the house price
    - *Examples:* RM (having more rooms), RAD (proximity to highways)

In [25]:
```
# print out sorted coefficients with descriptions
def print_coefs(coefs, bostonAttr):
    # sort coefficients from smallest to largest, then reverse it
    inds = argsort(abs(coefs))[::-1]
    # print out
    print("weight : feature description")
    for i in inds:
        print("{: .3f} : {:7s} {}".format(coefs[i], bostonAttr[i][0], bostonAttr
[i][1]))

print_coefs(rr.coef_, bostonAttr)
```

```
weight : feature description
-3.613 : LSTAT   % lower status of the population
-2.854 : DIS     weighted distances to five Boston employment centres
 2.783 : RM      average number of rooms per dwelling
-2.223 : PTRATIO pupil-teacher ratio by town
 2.105 : RAD     index of accessibility to radial highways
-1.856 : NOX     nitric oxides concentration (parts per 10 million)
-1.097 : TAX     full-value property-tax rate per $10,000
-0.854 : CRIM    per capita crime rate by town
 0.816 : B       1000(Bk - 0.63)^2 where Bk is the proportion of blacks by t
own
 0.756 : CHAS    Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)
 0.714 : ZN      proportion of residential land zoned for lots over 25,000 s
q.ft.
-0.540 : INDUS   proportion of non-retail business acres per town
-0.061 : AGE     proportion of owner-occupied units built prior to 1940
```

Better shrinkage

- With ridge regression, some weights are small but still non-zero.
  - these are less important, but somehow still necessary.
- To get better shrinkage to zero, we can change the regularization term to encourage more weights to be 0.

LASSO

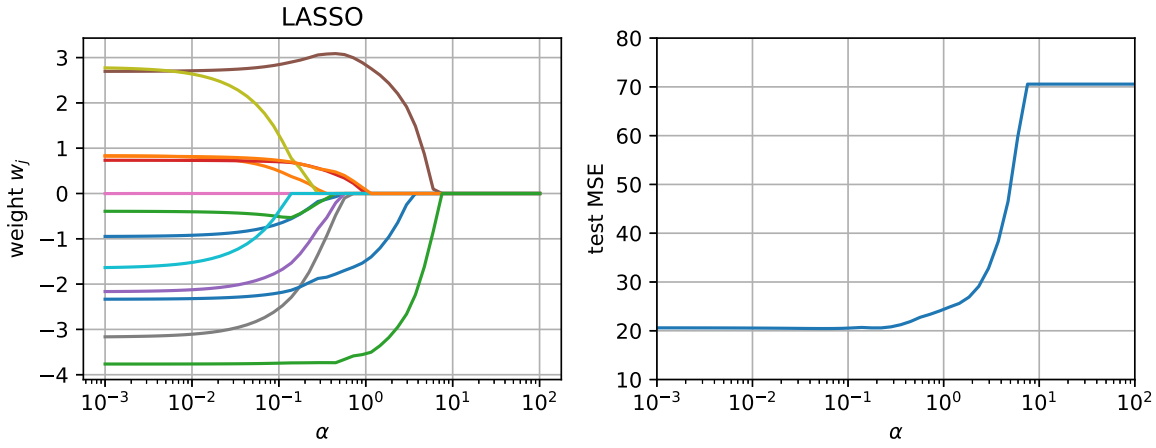
- LASSO = "Least absolute shrinkage and selection operator"
- keep the same data fit term, but change the regularization term:
  - sum of absolute weight values: $\sum_{j=1}^{d} |w_j|$
  - when a weight is close to 0, the regularization term will force it to be equal to 0.

$$\min_{\mathbf{w},b} \alpha \sum_{j=1}^{d} |w_j| + \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2$$

In [26]:
```
lasalphas = logspace(-3,2,50)

lassoMSEs  = empty(len(alphas))
lassows    = empty((len(alphas), trainXn.shape[1]))
for i,alpha in enumerate(lasalphas):
    # learn the LASSO model
    las = linear_model.Lasso(alpha=alpha)
    las.fit(trainXn, trainY)
    lassows[i,:] = las.coef_    # save weights

    lassoMSEs[i] = metrics.mean_squared_error(testY, las.predict(testXn))
```

In [28]:
```
lfig
```

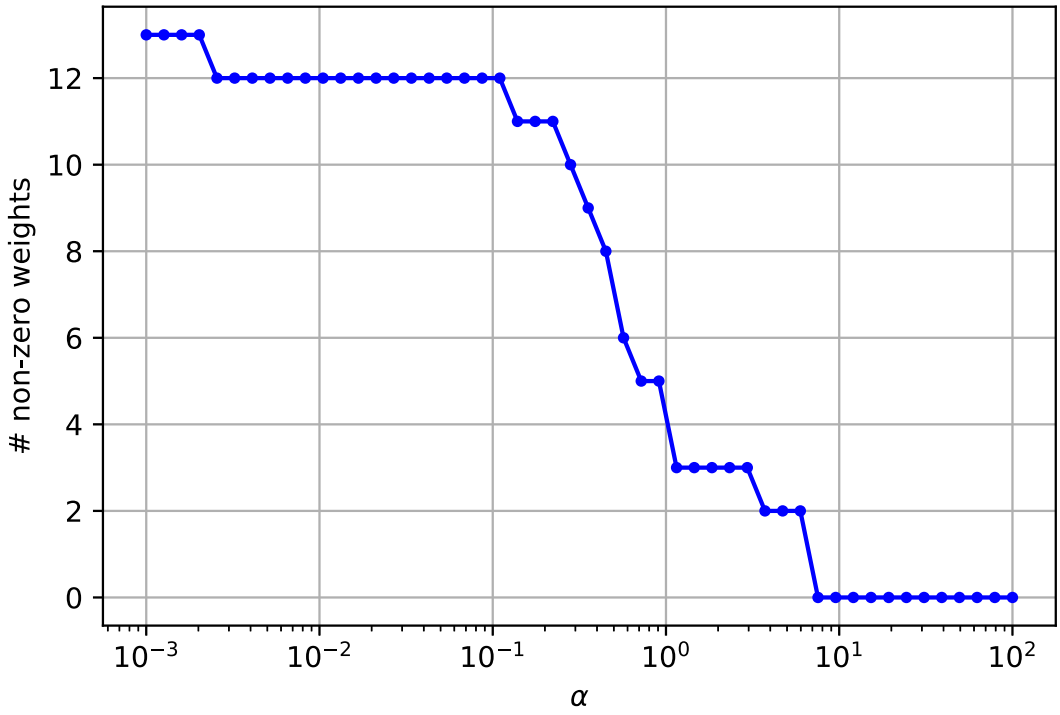
Out[28]:



Feature selection

- Select α to obtain a given number of features

In [29]:
```
# count the number of non-zero weights
nzweights = sum(abs(lassows)>1e-6, axis=1)

plt.semilogx(lasalphas, nzweights, 'b.-')
plt.grid(True)
plt.xlabel('$\\alpha$'); plt.ylabel('# non-zero weights');
```



In [30]:
```
# get alpha where non-zero weights = 5
myi = where(nzweights==5)[0][0]
print("alpha=", lasalphas[myi])
print("MSE =", lassoMSEs[myi])
print("w =", lassows[myi,:])
```

```
alpha= 0.7196856730011522
MSE = 23.373225554067442
w = [-0.          0.         -0.          0.22535088 -0.          2.99639992
 -0.         -0.         -0.         -0.         -1.62498304  0.27522116
 -3.58518563]
```

Interpretation

- weights for unimportant features are set to 0
  - TAX, RAD, DIS, AGE, ...
- important features have non-zero weights
  - LSTAT, RM, PTRATIO, B, CHAS

In [31]:
```
print_coefs(lassows[myi,:], bostonAttr)
```

```
weight : feature description
-3.585 : LSTAT   % lower status of the population
 2.996 : RM      average number of rooms per dwelling
-1.625 : PTRATIO pupil-teacher ratio by town
 0.275 : B       1000(Bk - 0.63)^2 where Bk is the proportion of blacks by t
own
 0.225 : CHAS    Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)
-0.000 : TAX     full-value property-tax rate per $10,000
-0.000 : RAD     index of accessibility to radial highways
-0.000 : DIS     weighted distances to five Boston employment centres
-0.000 : AGE     proportion of owner-occupied units built prior to 1940
-0.000 : NOX     nitric oxides concentration (parts per 10 million)
-0.000 : INDUS   proportion of non-retail business acres per town
 0.000 : ZN      proportion of residential land zoned for lots over 25,000 s
q.ft.
-0.000 : CRIM    per capita crime rate by town
```

Cross-validation to select α

- Use built-in CV function
  - selects α with lowest error.

In [32]:
```
# fit with cross-validation (alpha range is determined automatically)
las = linear_model.LassoCV()
las.fit(trainXn, trainY)

MSE = metrics.mean_squared_error(testY, las.predict(testXn))
print("MSE =", MSE)
print("alpha =", las.alpha_)
print("w =", las.coef_)
```

```
MSE = 20.573076053150526
alpha = 0.006866525794375745
w = [-0.93072851  0.8144078  -0.39980611  0.73120286 -2.1383923   2.70456431
 -0.         -3.12587305  2.68508642 -1.5614193  -2.32482345  0.81823908
 -3.76347777]
```

Interpretation

- only AGE is an unimportant feature.

In [33]:
```
print_coefs(las.coef_, bostonAttr)
```

```
weight : feature description
-3.763 : LSTAT   % lower status of the population
-3.126 : DIS     weighted distances to five Boston employment centres
 2.705 : RM      average number of rooms per dwelling
 2.685 : RAD     index of accessibility to radial highways
-2.325 : PTRATIO pupil-teacher ratio by town
-2.138 : NOX     nitric oxides concentration (parts per 10 million)
-1.561 : TAX     full-value property-tax rate per $10,000
-0.931 : CRIM    per capita crime rate by town
 0.818 : B       1000(Bk - 0.63)^2 where Bk is the proportion of blacks by t
own
 0.814 : ZN      proportion of residential land zoned for lots over 25,000 s
q.ft.
 0.731 : CHAS    Charles River dummy variable (= 1 if tract bounds river; 0
otherwise)
-0.400 : INDUS   proportion of non-retail business acres per town
-0.000 : AGE     proportion of owner-occupied units built prior to 1940
```