CS5489 - Machine Learning

Lecture 5a - Supervised Learning - Regression

Dr. Antoni B. Chan

Dept. of Computer Science, City University of Hong Kong

Outline

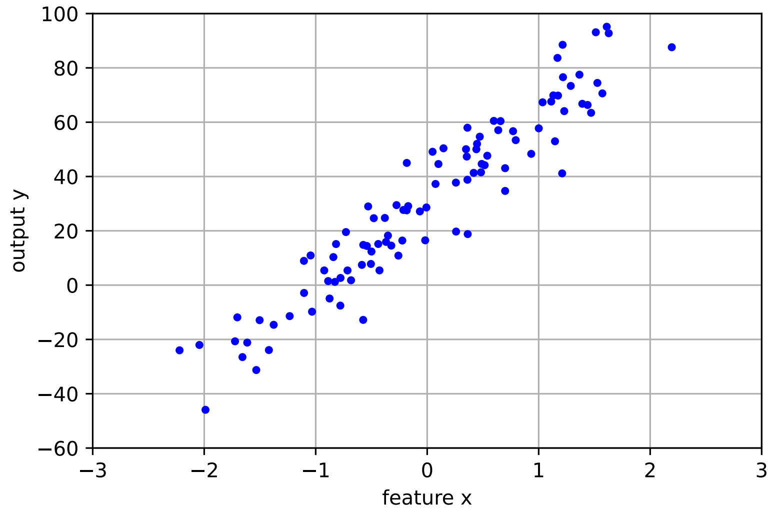
1. Linear Regression
2. Selecting Features
3. Removing Outliers
4. Non-linear regression

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]:

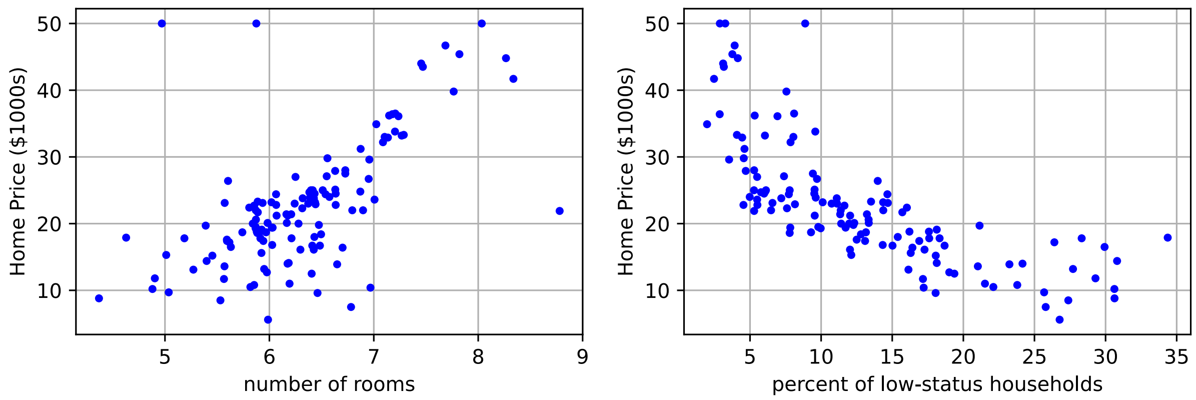


Examples:

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

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

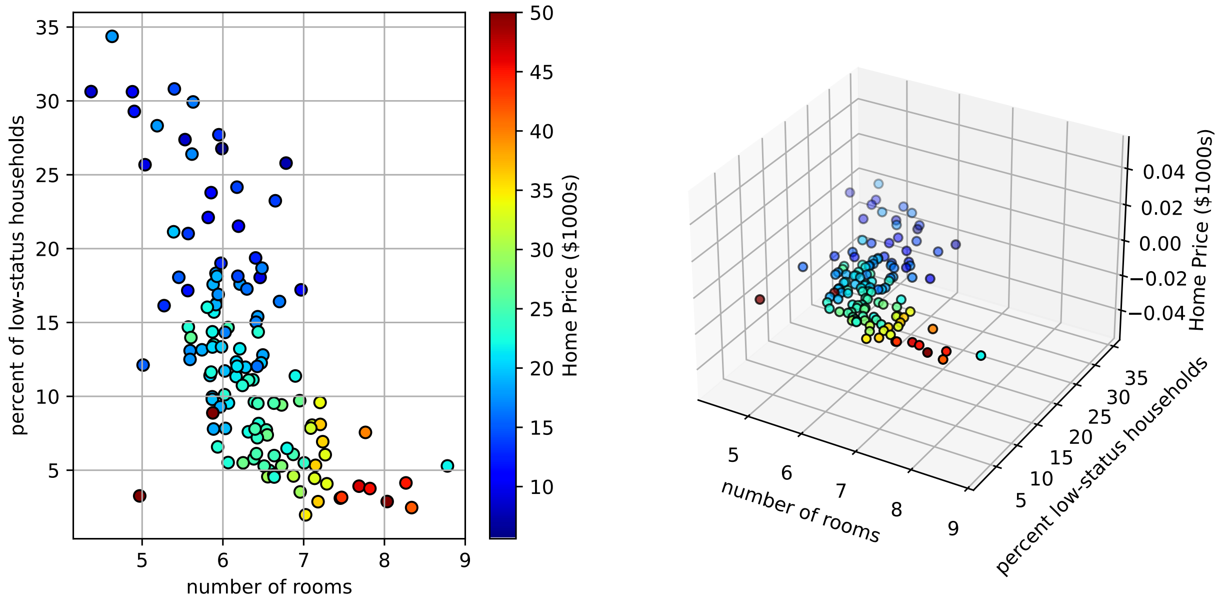
Out[5]:



- predict from both features

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

Out[6]:

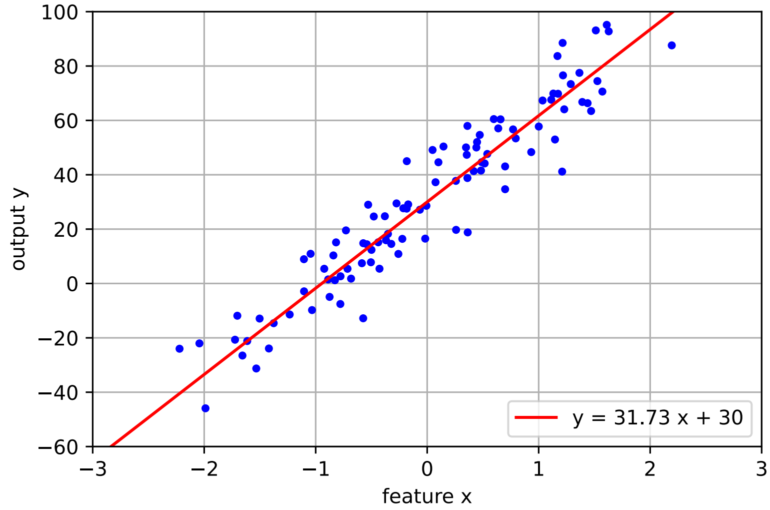


Linear Regression

- **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
```

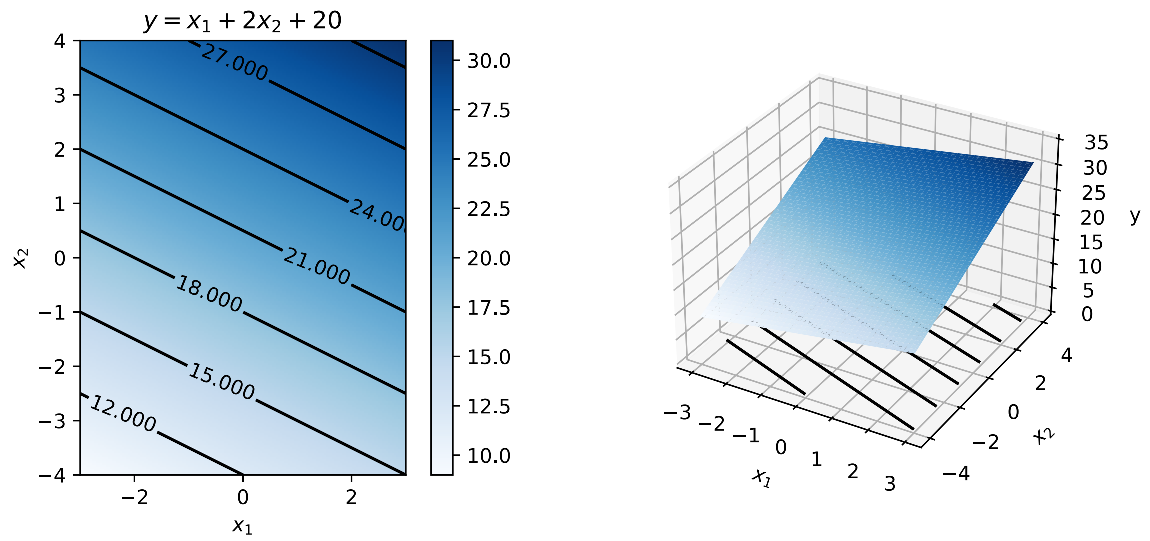
Out[8]:



- **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 $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$:

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

- The bias term b can be absorbed into $\mathbf{w}$ by redefining as follows:
  - $\mathbf{w} \leftarrow \begin{bmatrix}\mathbf{w} \\ b\end{bmatrix}, \mathbf{x} \leftarrow \begin{bmatrix}\mathbf{x} \\ 1\end{bmatrix}$
- We can write the minimization problem as:

$$\min_{\mathbf{w}} ||\mathbf{y} - \mathbf{X}^T\mathbf{w}||^2$$

  - 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.

- To obtain the solution:
  - 1) Expand the norm term:

$$\begin{aligned} ||\mathbf{y} - \mathbf{X}^T\mathbf{w}||^2 &= (\mathbf{y} - \mathbf{X}^T\mathbf{w})^T(\mathbf{y} - \mathbf{X}^T\mathbf{w}) \\ &= \mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}^T\mathbf{w} + \mathbf{w}^T\mathbf{X}\mathbf{X}^T\mathbf{w} \end{aligned}$$

  - Find the minimum by taking the derivative and setting to 0:

$$\begin{aligned} \frac{d}{d\mathbf{w}}(\mathbf{y}^T\mathbf{y} - 2\mathbf{y}^T\mathbf{X}^T\mathbf{w} + \mathbf{w}^T\mathbf{X}\mathbf{X}^T\mathbf{w}) &= -2\mathbf{X}\mathbf{y} + 2\mathbf{X}\mathbf{X}^T\mathbf{w} = 0 \\ &\Rightarrow \ \mathbf{X}\mathbf{X}^T\mathbf{w} = \mathbf{X}\mathbf{y} \\ &\Rightarrow \ \mathbf{w}^* = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{y} \end{aligned}$$

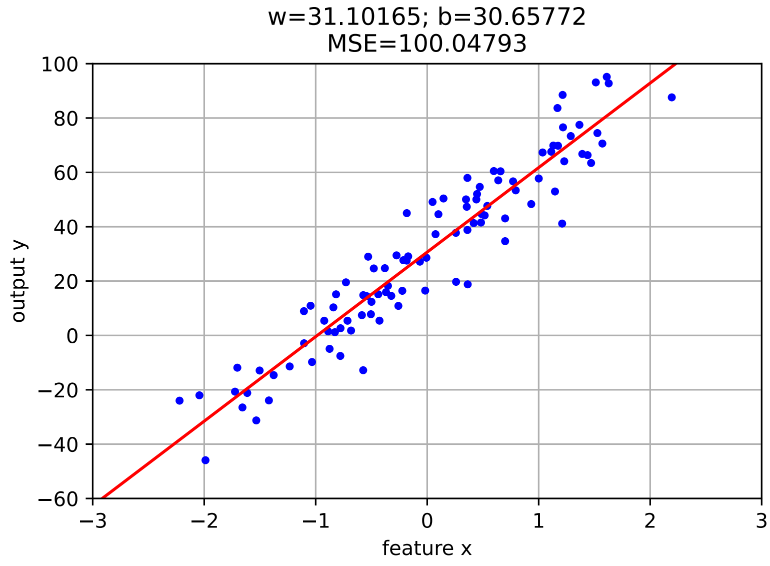
  - closed-form solution!
    - Note: $(\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}$ is also called the *pseudo-inverse* of $\mathbf{X}$.

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');
```



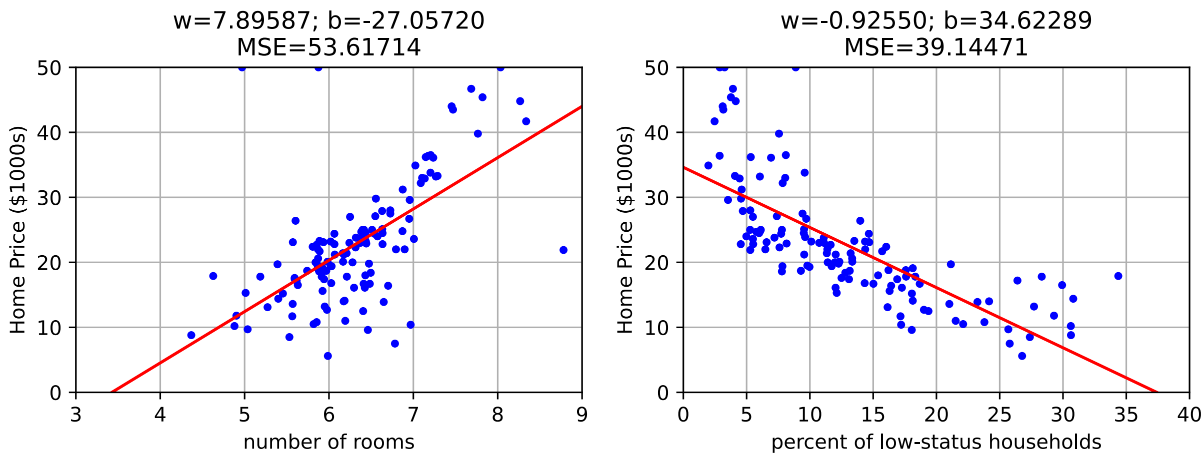
Boston housing price (1d)

- learn regression function for each feature separately

In [13]:
```
ols = [None]*2
for i in range(2):
    ols[i] = linear_model.LinearRegression()
    tmpX = bostonX[:,i][:,newaxis]
    ols[i].fit(tmpX, bostonY)
```

In [15]:
```
ofig
```

Out[15]:

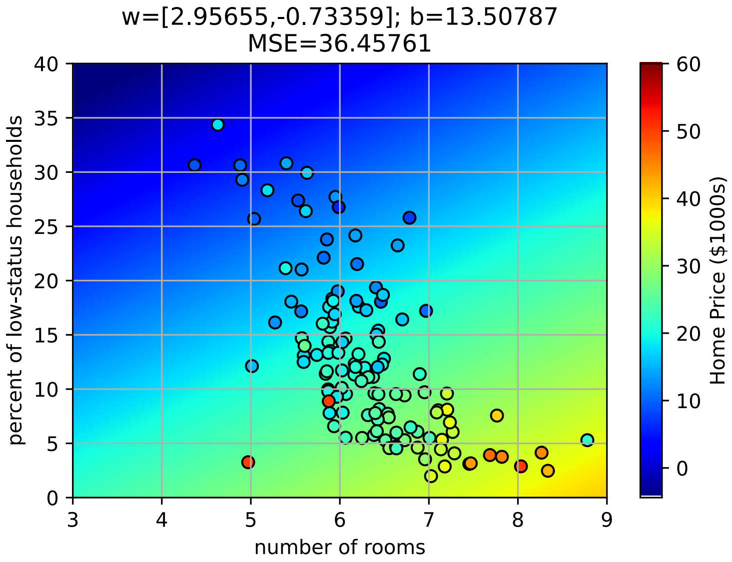


- for both features together

In [17]:
```
# learn with both dimensions
ols = linear_model.LinearRegression()
ols.fit(bostonX, bostonY);
```

In [19]:
```
ofig
```

Out[19]:



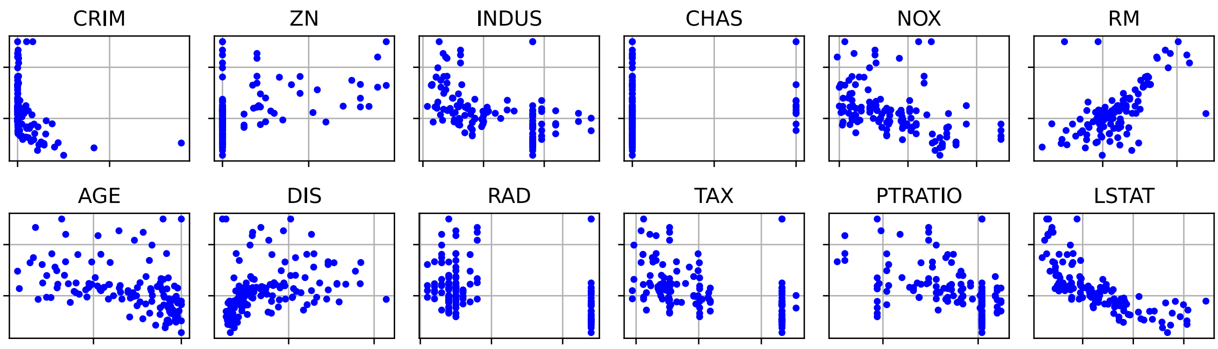
- interpretation from the linear model parameters:
  - each room increases home price by \$2956 (w_1)
  - each percentage of low-status households decreases home price by \$733 (w_2)
  - the "starting" price is \$13,508 (b).

Selecting Features

- The Boston housing data actually has 12 features.
  - plots of feature vs. housing price

In [21]:
```
bostonffig
```

Out[21]:



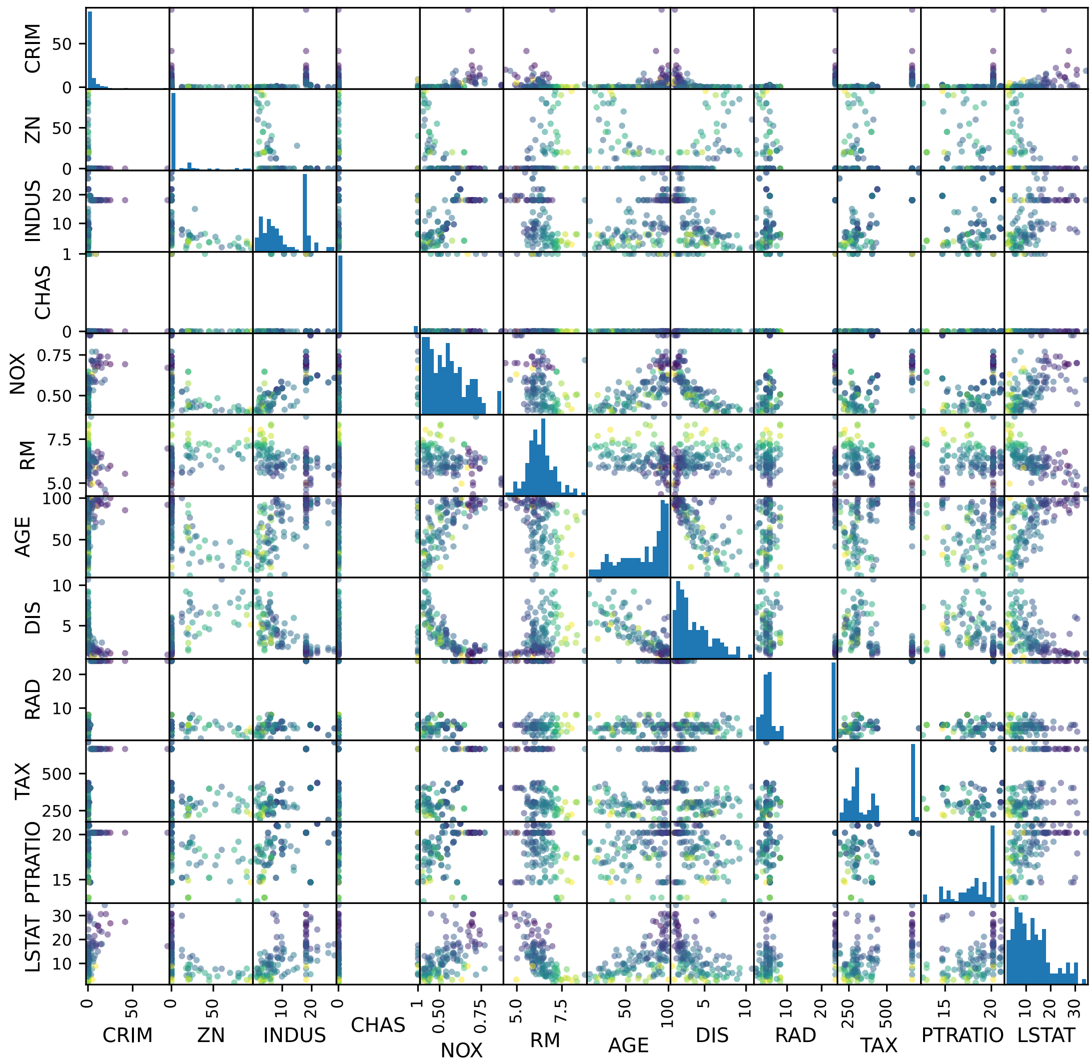

```
   CRIM = per capita crime rate by town`
     ZN = proportion of residential land zoned for lots over 25,000 sq.ft.`
  INDUS = proportion of non-retail business acres per town
   CHAS = Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
    NOX = nitric oxides concentration (parts 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 centres
    RAD = index of accessibility to radial highways
    TAX = full-value property-tax rate per $10,000
PTRATIO = pupil-teacher ratio 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 [22]:
```
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 (aka L2-norm)
  - 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, same as linear regression.

- Also has a closed-form solution (similar derivation to linear regression):
  - $\mathbf{w}^* = (\mathbf{X}\mathbf{X}^T + \alpha I)^{-1}\mathbf{X}\mathbf{y}$
  - Similar to the solution for linear regression: $\mathbf{w}^* = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{y}$
- What is the effect of the scaled identity matrix αI?

- with linear regression, if $\mathbf{X}$ does not span the input space $\mathbb{R}^d$, then $\mathbf{X}\mathbf{X}^T$ could be ill-conditioned or non-invertible.
  - i.e., we don't know how the data varies in the space orthogonal to $\mathbf{X}$.
- with ridge regression, the scaled identity conditions the matrix $\mathbf{X}\mathbf{X}^T$ so that the inverse can be computed.
  - (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 [23]:

```
# randomly split data into 80% train and 20% test set
trainX, testX, trainY, testY = \
  model_selection.train_test_split(bostonX, bostonY,
  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)
```

```
(101, 12)
(26, 12)
```

- vary α from 10^{-3} (little shrinkage) to 10^6 (lots of shrinkage)

In [24]:

```
# alpha values to try
alphas = logspace(-3,6,50)

MSEs = empty(len(alphas))
```

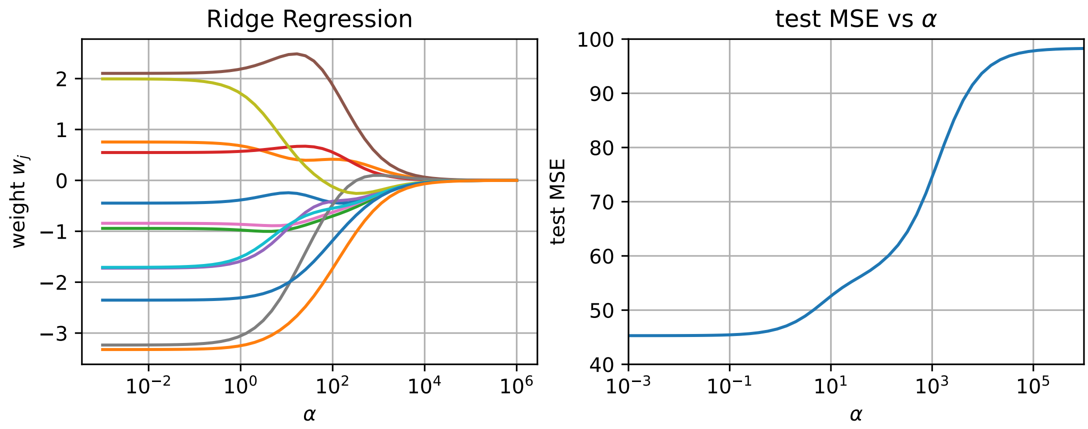
```
ws    = empty((len(alphas), trainXn.shape[1]))
for i,alpha in enumerate(alphas):
    # learn the RR model
    rr = linear_model.Ridge(alpha=alpha)
    rr.fit(trainXn, trainY)
    ws[i,:] = rr.coef_    # save weights

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

- Effect...
  - for small α, all weights are non-zero.
  - for large α, all weights shrink to 0.
  - somewhere in between is the best model...

In [26]:
```
rfig
```

Out[26]:



Selecting α using cross-validation

- built-in cross-validation (RidgeCV)

In [27]:
```
# 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 = 55.24072984514399
alpha = 25.595479226995383
w = [-0.29070744  0.39475303 -0.87840328  0.67118967 -0.62982053  2.448381
06
 -0.80391822 -1.41878524  0.27755795 -0.69521816 -1.75392601 -2.48098356]
```

Interpretation

- Which weights are most important?
  - look at weights with large magnitude.

In [28]:
```
# 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]))
```

- 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 [29]:
```
print_coefs(rr.coef_, bostonAttr)
```

```
weight : feature description
-2.481 : LSTAT   % lower status of the population
 2.448 : RM      average number of rooms per dwelling
-1.754 : PTRATIO pupil-teacher ratio by town
-1.419 : DIS     weighted distances to five Boston employment centres
-0.878 : INDUS   proportion of non-retail business acres per town
-0.804 : AGE     proportion of owner-occupied units built prior to 1940
-0.695 : TAX     full-value property-tax rate per $10,000
 0.671 : CHAS    Charles River dummy variable (= 1 if tract bounds river;
0 otherwise)
-0.630 : NOX     nitric oxides concentration (parts per 10 million)
 0.395 : ZN      proportion of residential land zoned for lots over 25,000
sq.ft.
-0.291 : CRIM    per capita crime rate by town
 0.278 : RAD     index of accessibility to radial highways
```

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.
  - also called "sparse" weights, or encouraging "sparsity".

LASSO

- LASSO = "Least absolute shrinkage and selection operator"
- keep the same data fit term, but change the regularization term:
  - sum of absolute weight values: $\sum_{j=1}^{d} |w_j|$
    - also called L1-norm: $||\mathbf{w}||_1$
  - when a weight is close to 0, the regularization term can move the weight to be equal to 0.

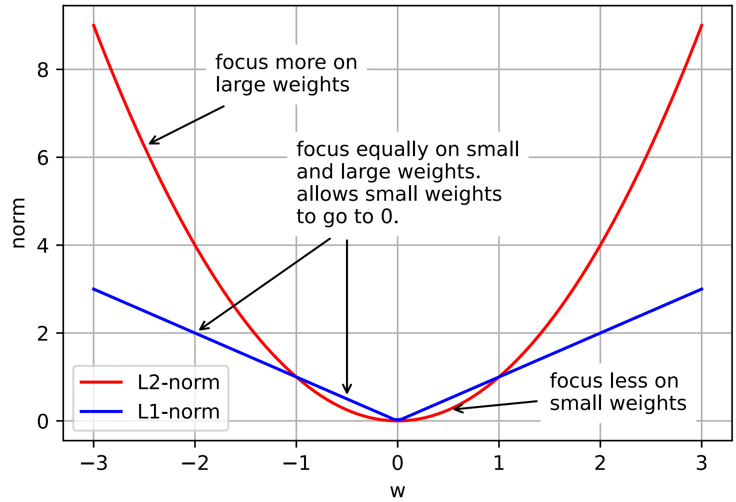
$$\min_{\mathbf{w},b} \alpha \sum_{j=1}^{d} |w_j| + \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2$$

Comparison of L2 and L1 norms.

- L2 focuses more on large weights.
- L1 treats all weights equally.

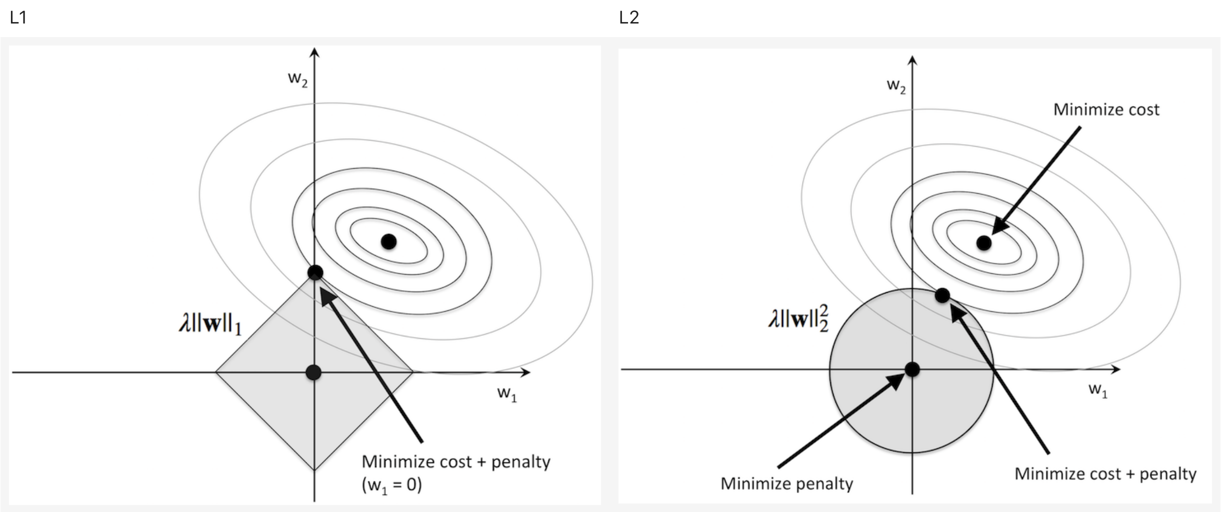
In [31]:
```
normfig
```

Out[31]:



Comparison of L2 and L1 norms

- During optimization with L1 norm:
  - for a given value of L1 norm, the minimal objective is usually in a "corner" of the L1 norm contour.
  - The "corner" corresponds to some weights equal to 0.



In [32]:
```
lasalphas = logspace(-3,2,50)

lassoMSEs  = empty(len(alphas))
lassows   = empty((len(alphas), trainXn.shape[1]))
```

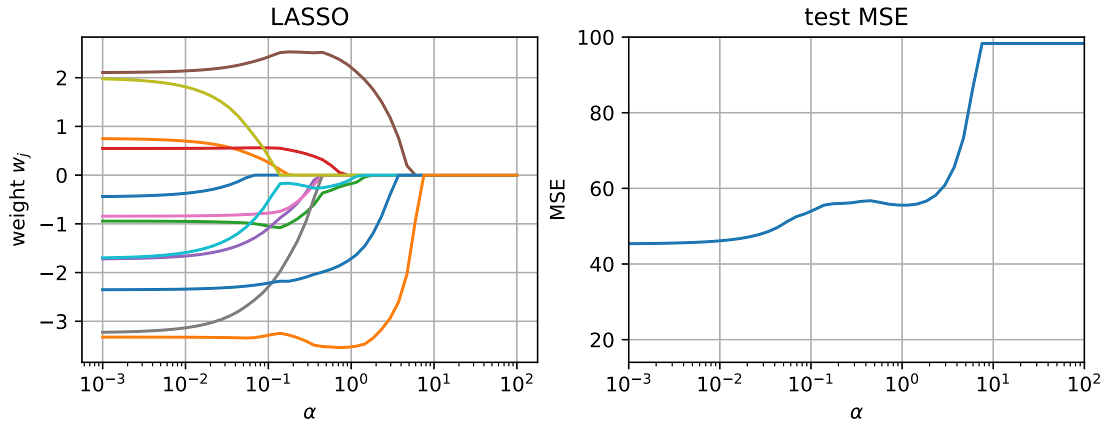
```
for i,alpha in enumerate(lasalphas):
    # learn the LASSO model
    las = linear_model.Lasso(alpha=alpha)
    las.fit(trainXn, trainY)
    lassows[i,:] = las.coef_    # save weights

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

In [34]:

```
lfig
```

Out[34]:



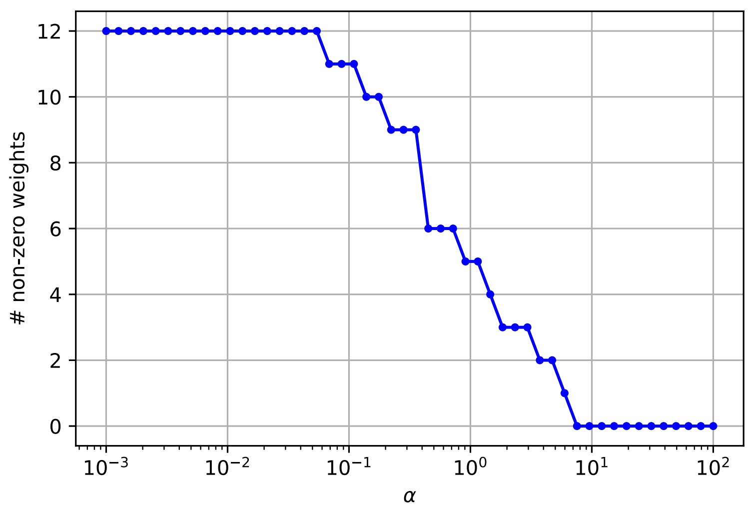
Feature selection

- Select α to obtain a given number of features

In [35]:

```
# 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 [36]:

```
# 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= 0.9102981779915218
MSE = 55.579375077684396
w = [-0.          0.          -0.19049149  0.          -0.          2.264558
```

```
08
 -0.          -0.          -0.          -0.11944939 -1.76235357 -3.53224554]
```

Interpretation

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

In [37]:
```
print_coefs(lassows[myi,:], bostonAttr)
```

```
weight : feature description
-3.532 : LSTAT   % lower status of the population
 2.265 : RM      average number of rooms per dwelling
-1.762 : PTRATIO pupil-teacher ratio by town
-0.190 : INDUS   proportion of non-retail business acres per town
-0.119 : 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 : CHAS    Charles River dummy variable (= 1 if tract bounds river;
0 otherwise)
 0.000 : ZN      proportion of residential land zoned for lots over 25,000
sq.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 [38]:
```
# 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 = 56.05514450375403
alpha = 0.6426533625838364
w = [-0.          0.         -0.27751219  0.13119465 -0.          2.416023
96
 -0.          -0.          -0.          -0.20770521 -1.89713883 -3.53302188]
```

Interpretation

- RAD, DIS, AGE, NOX, ZN, CRIM are unimportant features.

In [39]:
```
print_coefs(las.coef_, bostonAttr)
```

```
weight : feature description
-3.533 : LSTAT   % lower status of the population
```

```
 2.416 : RM      average number of rooms per dwelling
-1.897 : PTRATIO pupil-teacher ratio by town
-0.278 : INDUS   proportion of non-retail business acres per town
-0.208 : TAX     full-value property-tax rate per $10,000
 0.131 : CHAS    Charles River dummy variable (= 1 if tract bounds river;
0 otherwise)
-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 : ZN      proportion of residential land zoned for lots over 25,000
sq.ft.
-0.000 : CRIM    per capita crime rate by town
```

Sparsity Constraints

- In previous formulations, LASSO and Ridge Regression only encourage sparisty using a regularizer.
- We can also formulate the regression problem with *explicit* sparsity constraints:

$$\min_{\mathbf{w},b} \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2, \text{s. t. } ||\mathbf{w}||_0 \leq K$$

  - L0-norm: $||\mathbf{w}||_0$ = the number of non-zero entries in $\mathbf{w}$.
    - (not really a norm)
  - K is a hyperparameter - how many non-zero coefficients are desired.

Serious problem...

- LASSO and Ridge Regression are convex problems
  - Ridge Regression - closed-form solution
  - LASSO - efficient optimization algorithms to get exact solution
- Optimization problems with L0-norm constraints are NP-hard.
  - Combinatorial problem - all combinations of features need to be tried.

Orthogonal Matching Pursuit (OMP)

- **Idea:** greedy algorithm that iteratively selects the feature that is most correlated with the current residual error.
- Algorithm
  - Initialize the residual: $\mathbf{r} = \mathbf{y}$
  - For t in 1 to K
    - Find the most correlated feature: $j = \text{argmax}_j \, |\mathbf{r}^T \mathbf{x}_j|$, where $\mathbf{x}_j$ is the j-th row of $\mathbf{X}$ (the j-th features).
    - Compute the weight: $w_j = \text{argmin}_{w_j} \, ||\mathbf{r} - \mathbf{x}_j w_j||^2$
    - Update the residual: $\mathbf{r} -= \mathbf{x}_j w_j$

In [40]:
```
# Example
omp = linear_model.OrthogonalMatchingPursuit(n_nonzero_coefs=2)
omp.fit(trainXn, trainY)

MSE = metrics.mean_squared_error(testY, omp.predict(testXn))
print("MSE =", MSE)
print(omp.coef_)
print(omp.intercept_)
```

```
MSE = 53.86974967354984
[ 0.          0.          0.          0.          0.          0.
  0.          0.          0.          0.         -2.62819004 -5.96007042]
22.85940594059405
```

In [41]:
```
print_coefs(omp.coef_, bostonAttr)
```

```
weight : feature description
-5.960 : LSTAT    % lower status of the population
-2.628 : PTRATIO pupil-teacher ratio by town
 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 : RM       average number of rooms per dwelling
 0.000 : NOX      nitric oxides concentration (parts per 10 million)
 0.000 : CHAS     Charles River dummy variable (= 1 if tract bounds river;
0 otherwise)
 0.000 : INDUS    proportion of non-retail business acres per town
 0.000 : ZN       proportion of residential land zoned for lots over 25,000
sq.ft.
 0.000 : CRIM     per capita crime rate by town
```

- Note that LASSO selects different features, and also has worse MSE.

In [42]:
```
# get alpha where non-zero weights = 2
myi = where(nzweights==2)[0][0]
print("MSE =", lassoMSEs[myi])
print_coefs(lassows[myi,:], bostonAttr)
```

```
MSE = 65.57831140414783
weight : feature description
-2.605 : LSTAT    % lower status of the population
 0.765 : RM       average number of rooms per dwelling
-0.000 : PTRATIO pupil-teacher ratio by town
-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 : CHAS     Charles River dummy variable (= 1 if tract bounds river;
0 otherwise)
-0.000 : INDUS    proportion of non-retail business acres per town
 0.000 : ZN       proportion of residential land zoned for lots over 25,000
sq.ft.
-0.000 : CRIM     per capita crime rate by town
```