CS5489 - Machine Learning

Lecture 5a - Supervised Learning - Regression

Prof. Antoni B. Chan

Dept. of Computer Science, City University of Hong Kong

Outline

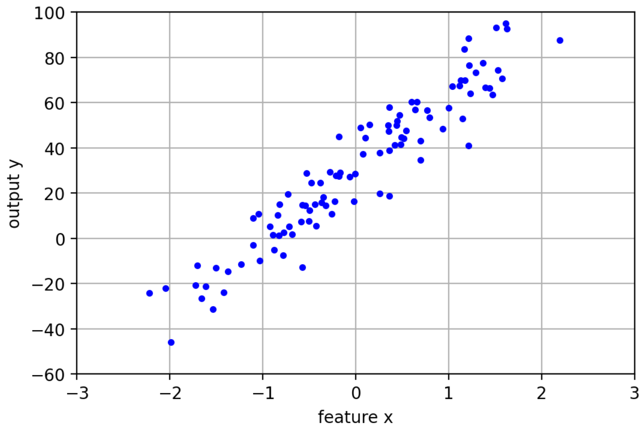
1. Linear Regression
2. Selecting Features
3. Removing Outliers
4. Non-linear regression

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]:

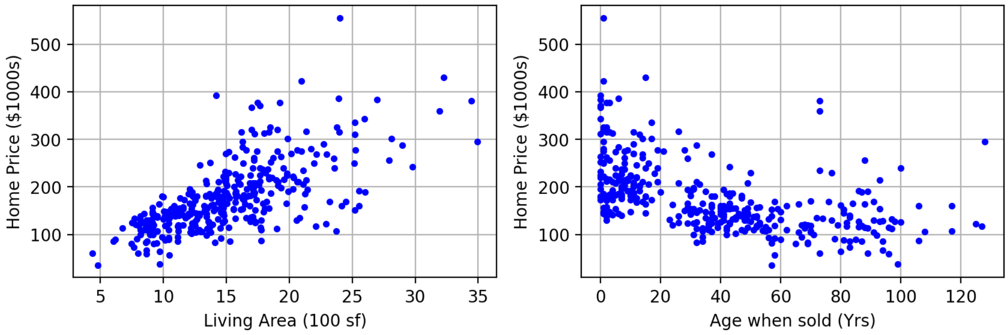


Examples:

- Predict housing price from living area and house age.

In [6]:
```
housing1dfig
```

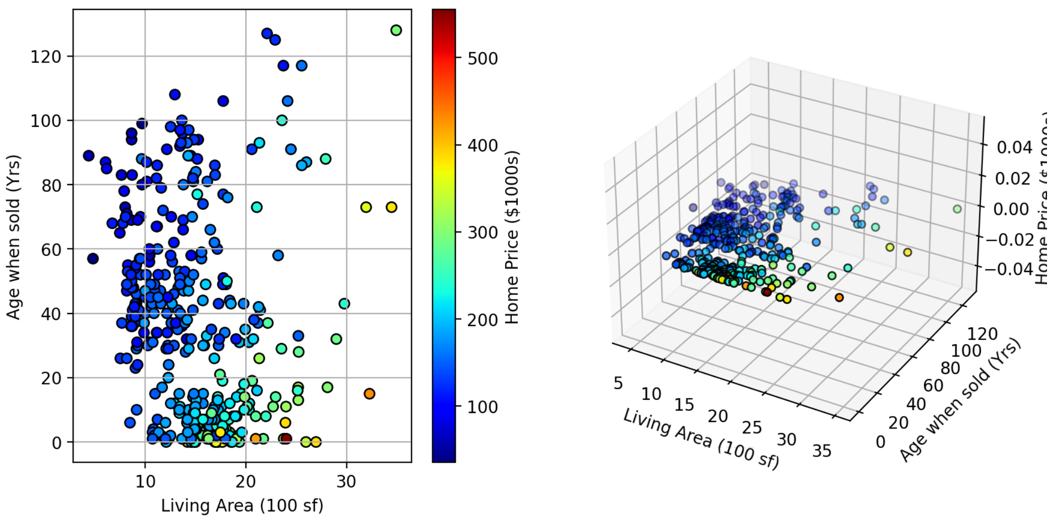
Out[6]:



- predict from both features

In [7]:
```
housing2dfig
```

Out[7]:

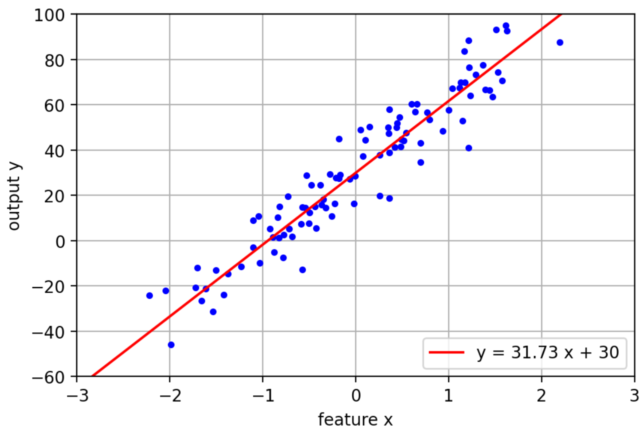


Linear Regression

- **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 [9]:
```
linfig
```

Out[9]:

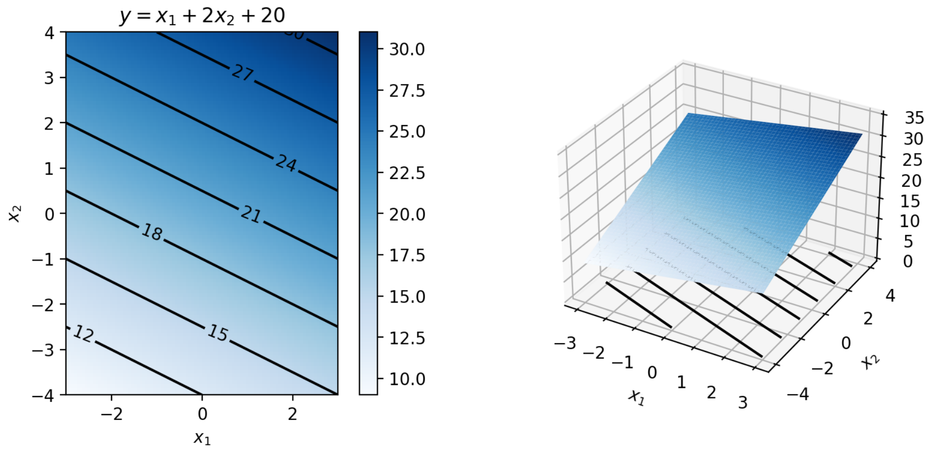


- **d-dim case**: the output y is a linear combination of d input variables $x_1, \cdots, x_d$:
  - $y = w_0 + w_1 x_1 + w_2 x_2 + \cdots + w_d x_d$
- Equivalently,
  - $y = w_0 + \mathbf{w}^T \mathbf{x} = w_0 + \sum_{j=1}^{d} w_j x_j$
    - $\mathbf{x} \in \mathbb{R}^d$ is the vector of input values.

- $\mathbf{w} \in \mathbb{R}^d$ are the weights of the linear function, and w_0 is the intercept (bias term)

In [11]:
```
lin2dfig
```

Out[11]:



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:
  - Expand the norm term:

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

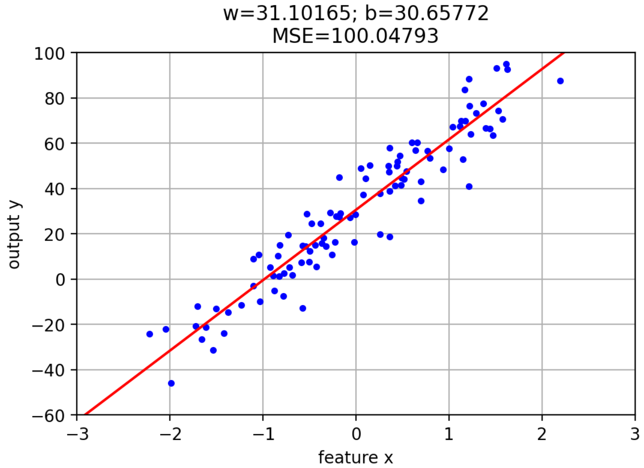
- Find the minimum by taking the derivative and setting to 0:

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

- 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 [13]:
```python
# 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');
```



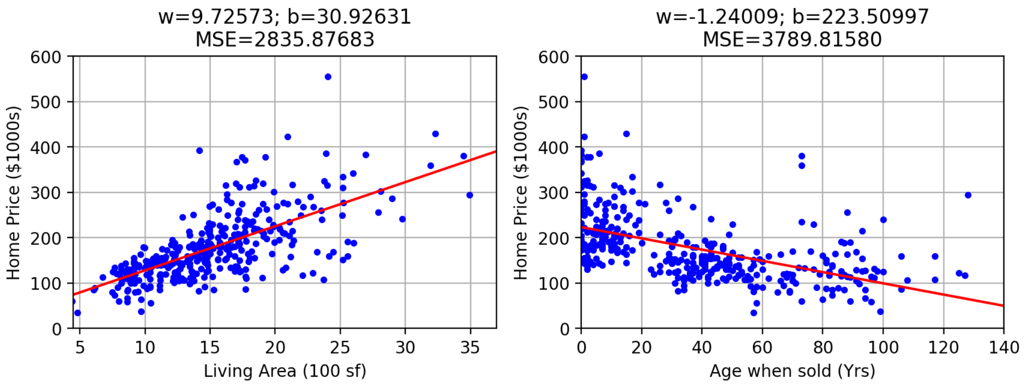
Housing price (1d)

- learn regression function for each feature separately

In [14]:
```python
ols = [None]*2
for i in range(2):
    ols[i] = linear_model.LinearRegression()
    tmpX = housingX[:,i][:,newaxis]
    ols[i].fit(tmpX, housingY)
```

In [16]:
```python
ofig
```

Out[16]:

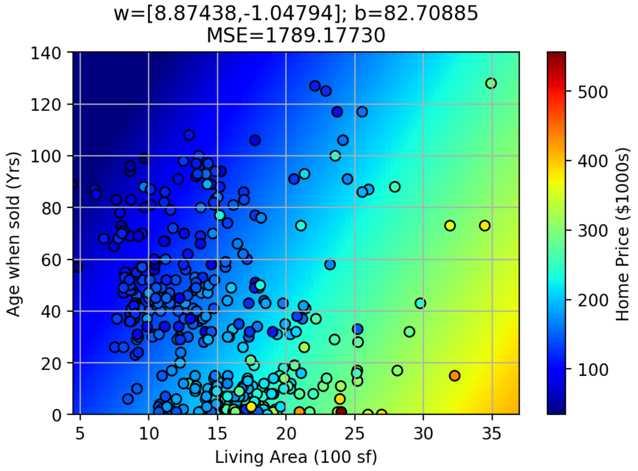


- for both features together

In [18]:
```python
# learn with both dimensions
ols = linear_model.LinearRegression()
ols.fit(housingX, housingY);
```

In [20]:
```python
ofig
```

Out[20]:



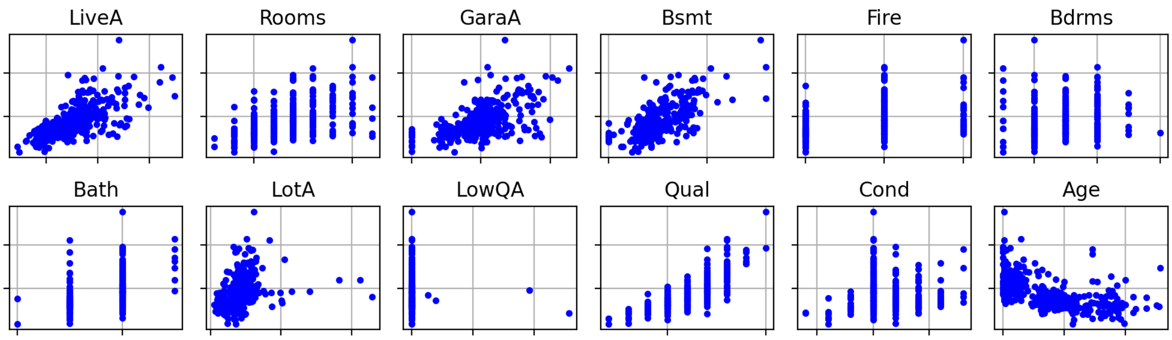
- interpretation from the linear model parameters:
  - each 100 sqf of living space increases home price by \$8874 (w_1)
  - each year of age decreases home price by \$1048 (w_2)
  - the "starting" price is \$82,709 (b).

Selecting Features

- There are more features from the housing data.
  - plots of feature vs. housing price

In [22]:
```
housingffig
```

Out[22]:



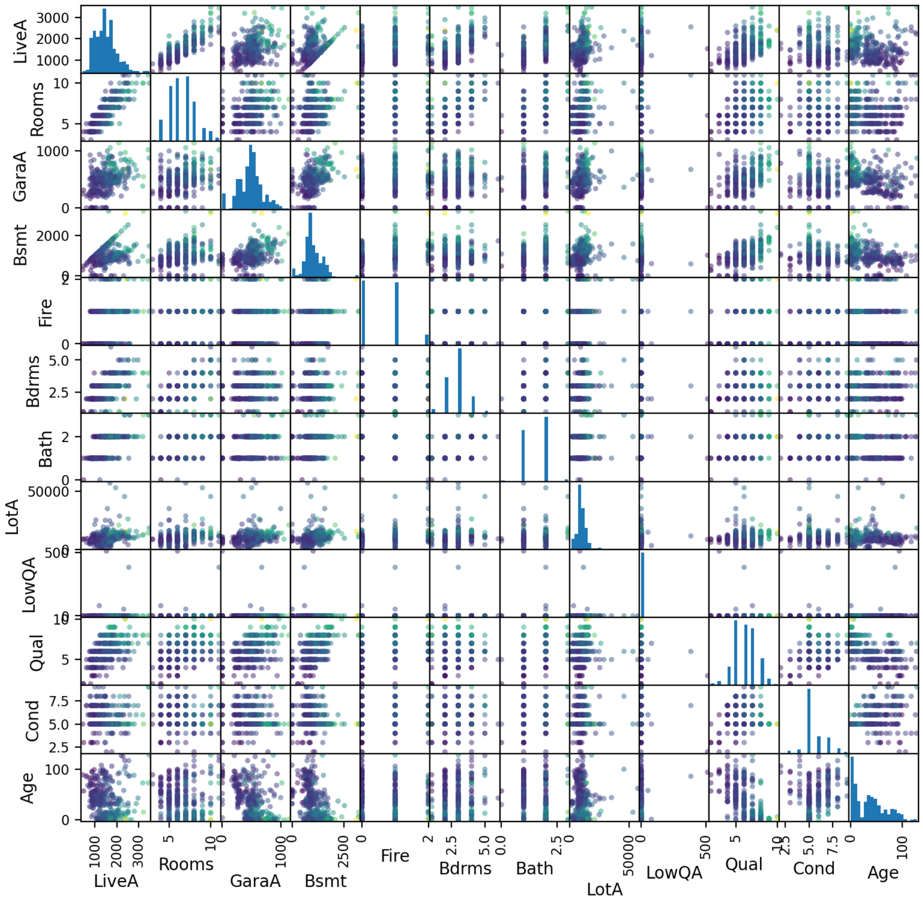
In [23]:
```
print(mydesc)
```

```
LiveA = Living Area (sqf)
Rooms = Number of Rooms
GaraA = Garage Area (sqf)
Bsmt  = Basement Area (sqf)
Fire  = Number of Fireplaces
Bdrms = Number of Bedrooms
Bath  = Number of Bathrooms
LotA  = Lot Area (sqf)
LowQA = Area of low-quality finish (sqf)
Qual  = Overall Quality of Materials
Cond  = Overall Condition
Age   = Age when Sold (yrs)
```

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

In [24]:
```
import pandas as pd
housing_df = pd.DataFrame(housingX, columns=housingXname)
```

```
tmp=pd.plotting.scatter_matrix(housing_df, c=housingY, 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.

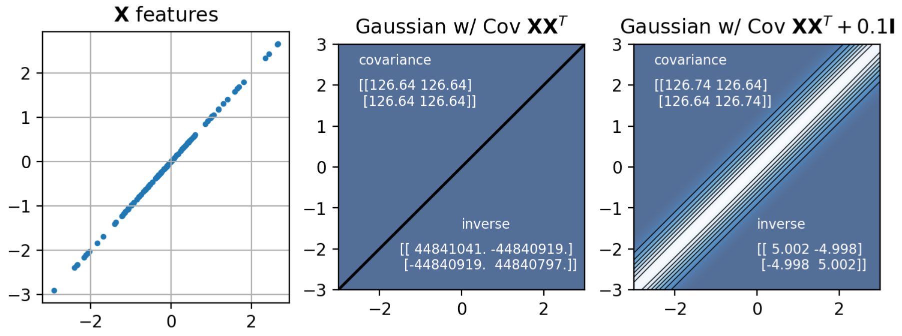
- Also has a closed-form solution (similar derivation to linear regression):

- $\mathbf{w}^* = (\mathbf{X}\mathbf{X}^T + \alpha I)^{-1}\mathbf{X}\mathbf{y}$
- Similar to the solution for linear regression: $\mathbf{w}^* = (\mathbf{X}\mathbf{X}^T)^{-1}\mathbf{X}\mathbf{y}$

- 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)

In [28]:
```
degfig
```

Out[28]:



Example on Housing data

In [29]:
```
# randomly split data into 80% train and 20% test set
trainX, testX, trainY, testY = \
  model_selection.train_test_split(housingX, housingY,
  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)
```

```
(292, 12)
(73, 12)
```

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

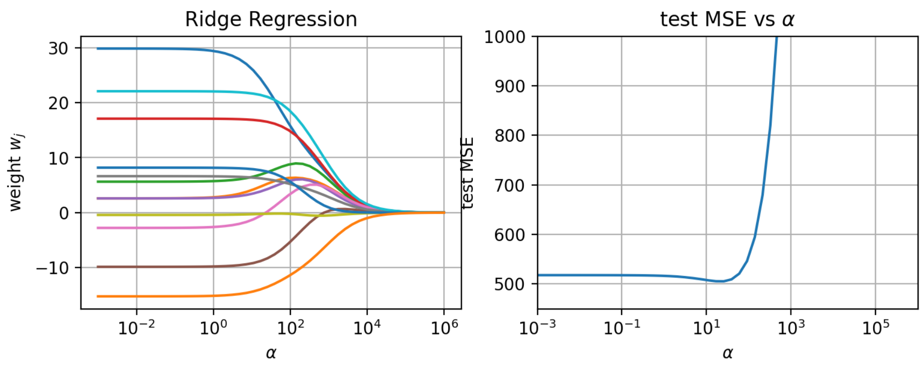
In [30]:
```
# alpha values to try
alphas = logspace(-3,6,50)

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.

In [32]:
```
rfig
```

Out[32]:



Selecting α using cross-validation

- built-in cross-validation (RidgeCV)

In [33]:
```
# 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 = 509.7348023994934
alpha = 7.196856730011521
w = [ 27.0687271    3.7674307    6.26844302  16.95682861   3.20616604
  -9.42397403  -1.71271811   6.44770492  -0.33614936  21.81101743
   7.99606554 -14.63428106]
```

Interpretation

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

In [34]:
```
# print out sorted coefficients with descriptions
def print_coefs(coefs, name, desc):
    # 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} : {:5s} = {}".format(coefs[i], name[i], desc[i]))
```

- Which weights are most important?
    - positive weights indicate factors that increase the house price
        - *Examples:* LiveA, Qual, Bsmt, Cond
    - negative weights indicate factors that decrease the house price
        - *Examples:* Age, Bdrms, Bath

In [35]:
```
print_coefs(rr.coef_, housingXname, housingXdesc)
```

```
weight : feature description
 27.069 : LiveA = Living Area (sqf)
 21.811 : Qual  = Overall Quality of Materials
 16.957 : Bsmt  = Basement Area (sqf)
-14.634 : Age   = Age when Sold (yrs)
-9.424 : Bdrms = Number of Bedrooms
```

```
 7.996 : Cond  = Overall Condition
 6.448 : LotA  = Lot Area (sqf)
 6.268 : GaraA = Garage Area (sqf)
 3.767 : Rooms = Number of Rooms
 3.206 : Fire  = Number of Fireplaces
-1.713 : Bath  = Number of Bathrooms
-0.336 : LowQA = Area of low-quality finish (sqf)
```

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.

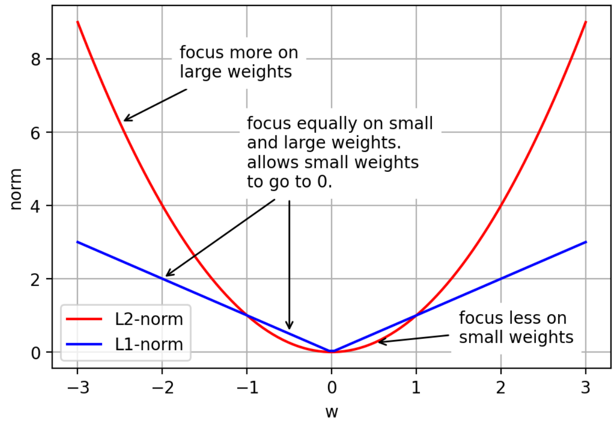
$$\min_{\mathbf{w},b} \alpha \sum_{j=1}^{d} |w_j| + \sum_{i=1}^{N} (y_i - f(\mathbf{x}_i))^2$$

Comparison of L2 and L1 norms.

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

In [37]:
```
normfig
```

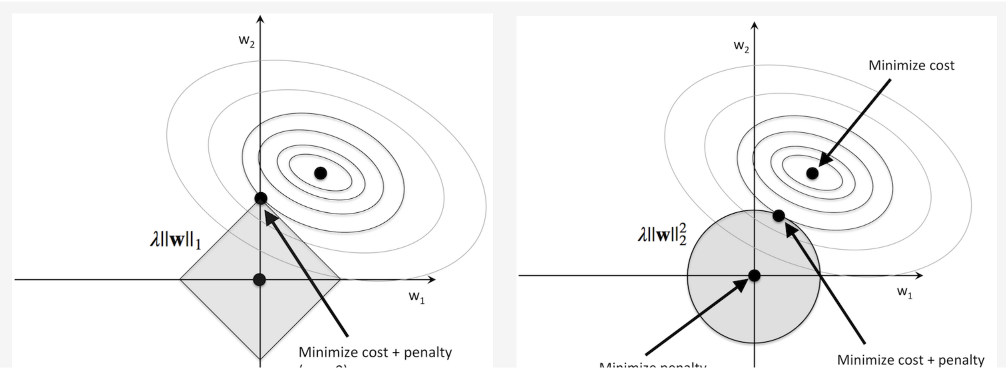
Out[37]:



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.

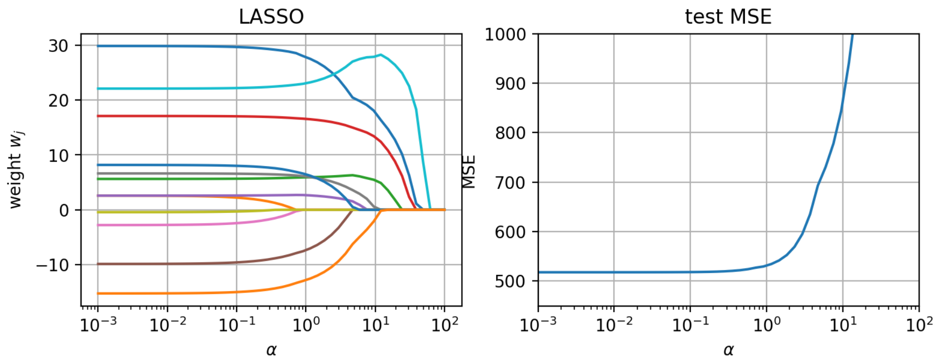
L1 L2



In [38]:
```
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 [40]:
```
lfig
```

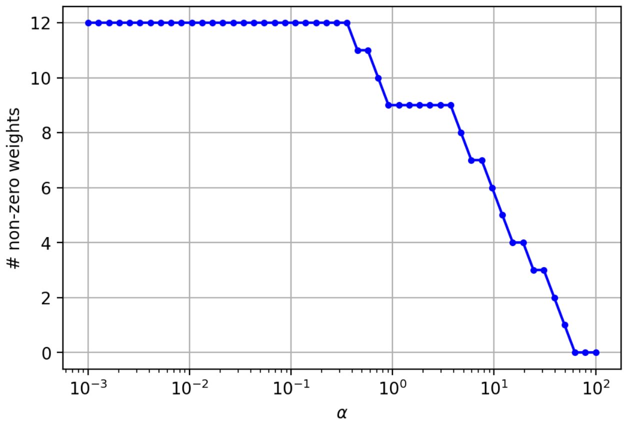
Out[40]:



Feature selection

- Select α to obtain a given number of features

In [41]:
```
# 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 [42]:
```
# 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= 12.067926406393289
MSE = 941.847017977837
w = [16.35781583  0.          4.89090713 12.39985828  0.         -0.
  0.          0.         -0.         28.27633526  0.         -0.17302193]
```

Interpretation

- important features have non-zero weights
  - Qual, LiveA, Bsmt, GaraA, Age
- weights for unimportant features are set to 0
  - Cond, LowQA, LotA, Bath, Bdrms, Fire, Rooms

In [43]:
```
print_coefs(lassows[myi,:], housingXname, housingXdesc)
```

```
weight : feature description
 28.276 : Qual  = Overall Quality of Materials
 16.358 : LiveA = Living Area (sqf)
 12.400 : Bsmt  = Basement Area (sqf)
 4.891 : GaraA = Garage Area (sqf)
-0.173 : Age   = Age when Sold (yrs)
 0.000 : Cond  = Overall Condition
-0.000 : LowQA = Area of low-quality finish (sqf)
 0.000 : LotA  = Lot Area (sqf)
 0.000 : Bath  = Number of Bathrooms
-0.000 : Bdrms = Number of Bedrooms
 0.000 : Fire  = Number of Fireplaces
 0.000 : Rooms = Number of Rooms
```

Cross-validation to select α

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

In [44]:
```
# 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 = 527.9102135105119
alpha = 0.8213828052493439
w = [ 28.20284029   0.          5.90009657  16.64554095   2.71954575
  -7.72793088  -0.          6.19124241  -0.          22.87482329
   6.74864601 -13.12006246]
```

Interpretation

- LowQA, Bath, Rooms are unimportant features

In [45]:
```
print_coefs(las.coef_, housingXname, housingXdesc)
```

```
weight : feature description
 28.203 : LiveA = Living Area (sqf)
 22.875 : Qual  = Overall Quality of Materials
 16.646 : Bsmt  = Basement Area (sqf)
-13.120 : Age   = Age when Sold (yrs)
-7.728 : Bdrms = Number of Bedrooms
 6.749 : Cond  = Overall Condition
 6.191 : LotA  = Lot Area (sqf)
```

```
 5.900 : GaraA = Garage Area (sqf)
 2.720 : Fire  = Number of Fireplaces
-0.000 : LowQA = Area of low-quality finish (sqf)
-0.000 : Bath  = Number of Bathrooms
 0.000 : Rooms = Number of Rooms
```

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:

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

- 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: $j = \mathrm{argmax}_j\, |\mathbf{r}^T \mathbf{x}_j|$, where $\mathbf{x}_j$ is the j-th row of $\mathbf{X}$ (the j-th features).
    - Compute the weight: $w_j = \mathrm{argmin}_{w_j}\, ||\mathbf{r} - \mathbf{x}_j w_j||^2$
    - Update the residual: $\mathbf{r} - = \mathbf{x}_j w_j$

In [46]:
```
# Example
omp = linear_model.OrthogonalMatchingPursuit(n_nonzero_coefs=2, normalize=False)
omp.fit(trainXn, trainY)

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

```
MSE = 1161.3764110385423
[26.78089631  0.          0.          0.          0.          0.
  0.          0.          0.         43.93862629  0.          0.        ]
180.67125342465755
```

In [47]:
```
print_coefs(omp.coef_, housingXname, housingXdesc)
```

```
weight : feature description
 43.939 : Qual  = Overall Quality of Materials
 26.781 : LiveA = Living Area (sqf)
 0.000 : Age   = Age when Sold (yrs)
 0.000 : Cond  = Overall Condition
 0.000 : LowQA = Area of low-quality finish (sqf)
 0.000 : LotA  = Lot Area (sqf)
 0.000 : Bath  = Number of Bathrooms
```

```
 0.000 : Bdrms = Number of Bedrooms
 0.000 : Fire  = Number of Fireplaces
 0.000 : Bsmt  = Basement Area (sqf)
 0.000 : GaraA = Garage Area (sqf)
 0.000 : Rooms = Number of Rooms
```

- Note that LASSO selects the same features, but has worse MSE (due to regularization on the weights).

In [48]:

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

```
MSE = 2812.8482393181357
weight : feature description
 18.287 : Qual  = Overall Quality of Materials
 1.129 : LiveA = Living Area (sqf)
-0.000 : Age   = Age when Sold (yrs)
 0.000 : Cond  = Overall Condition
-0.000 : LowQA = Area of low-quality finish (sqf)
 0.000 : LotA  = Lot Area (sqf)
 0.000 : Bath  = Number of Bathrooms
 0.000 : Bdrms = Number of Bedrooms
 0.000 : Fire  = Number of Fireplaces
 0.000 : Bsmt  = Basement Area (sqf)
 0.000 : GaraA = Garage Area (sqf)
 0.000 : Rooms = Number of Rooms
```