

Introduction to Data Science

WFAiS UJ, Informatyka Stosowana
I stopień studiów

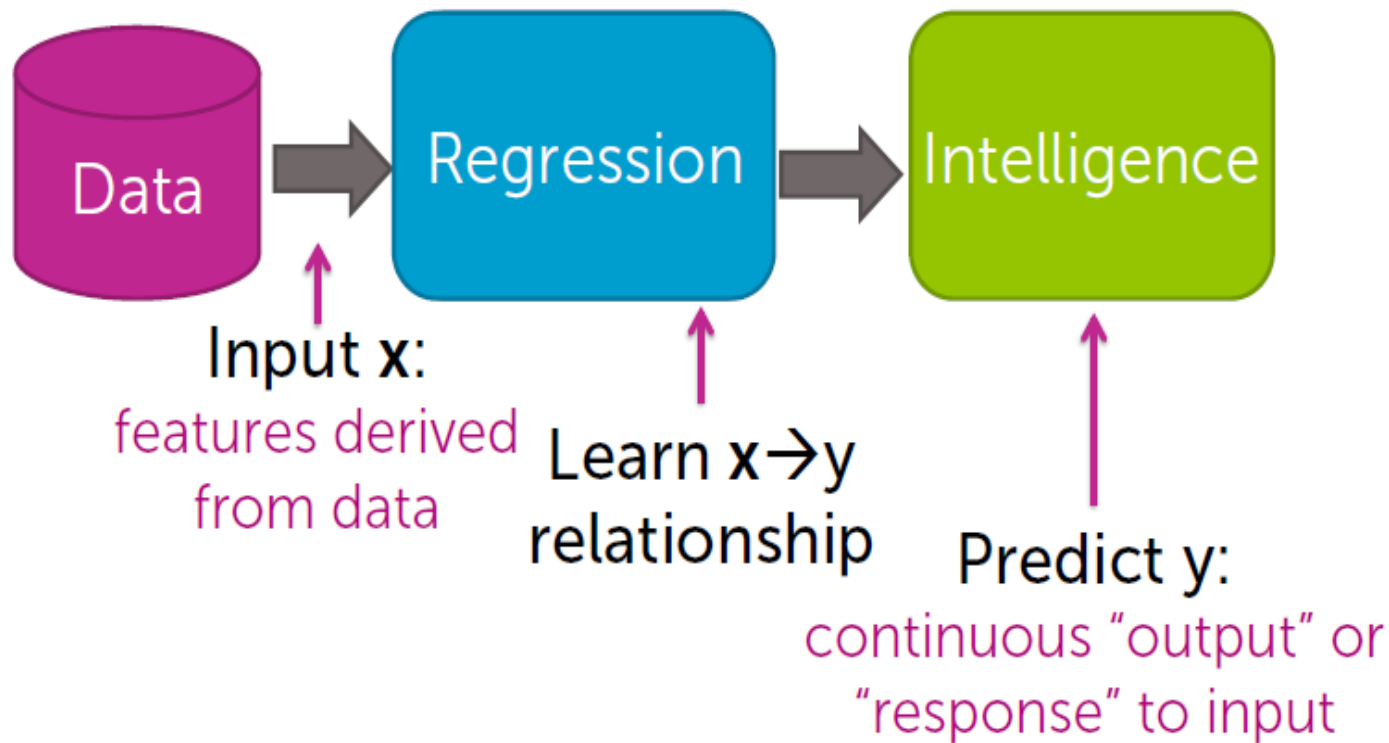
This lecture is
based on course by E. Fox and C. Guestrin, Univ of Washington

Regression for predictions

- ❑ **Simple regression**
- ❑ **Multiple regression**
- ❑ **Assesing performance**
- ❑ **Ridge regression**
- ❑ **Feature selection and lasso regression**
- ❑ **Nearest neighbor and kernel regression**

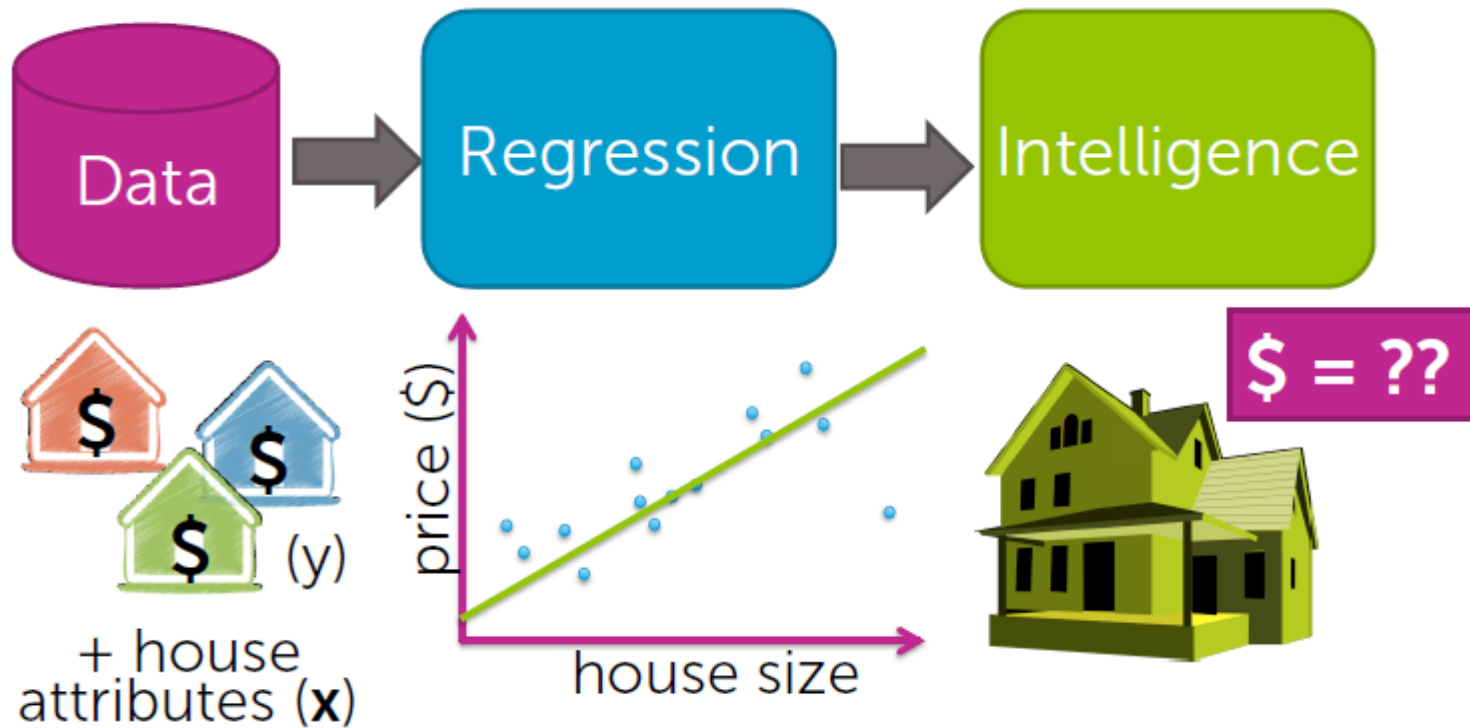
What is regression?

From features to predictions

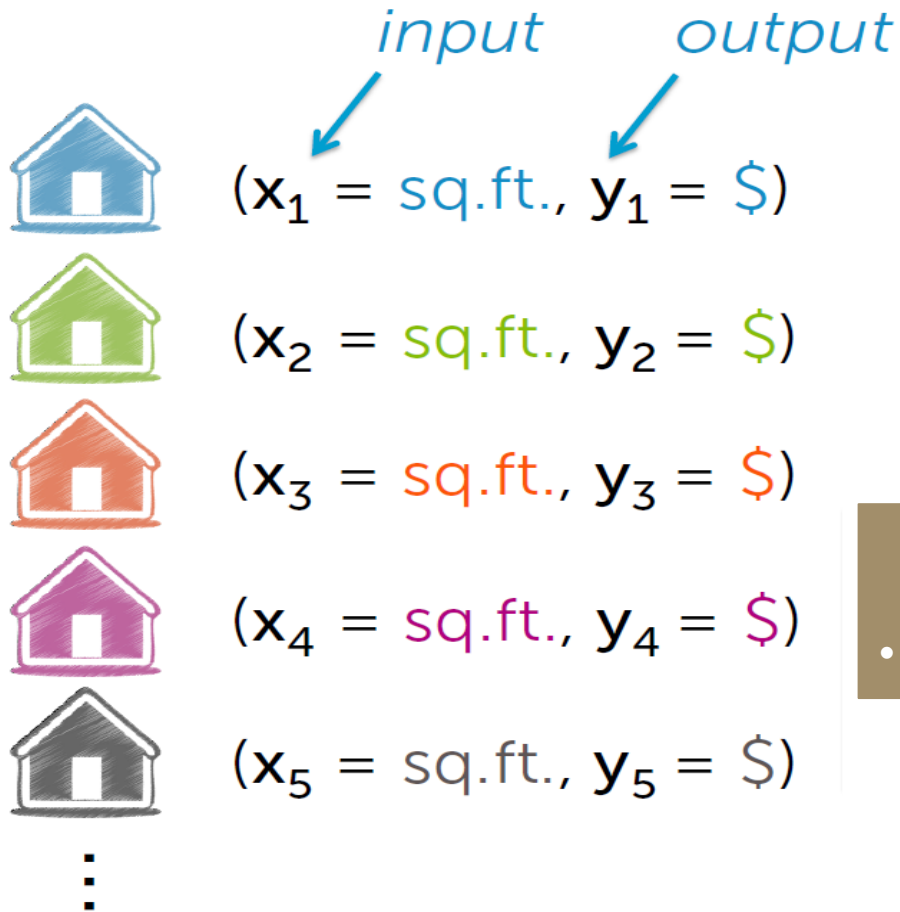


Case study

Predicting house prices



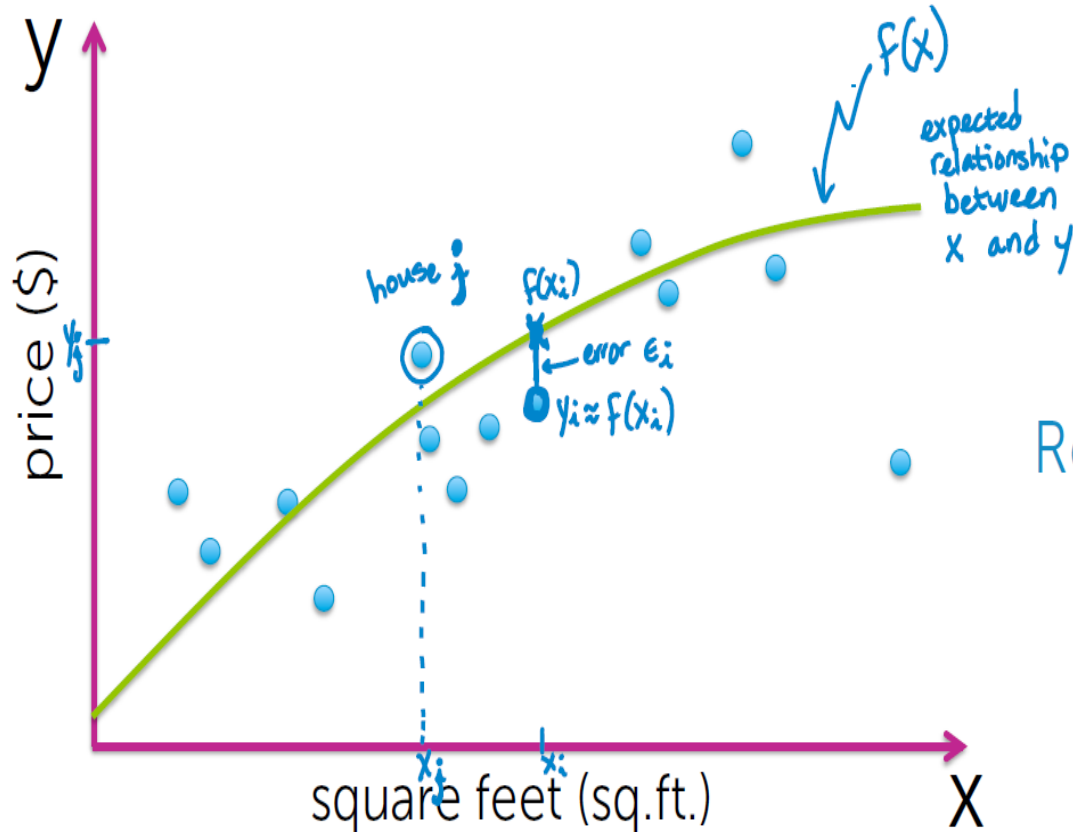
Data



Input vs output

- y is quantity of interest
- assume y can be predicted from x

Model: assume functional relationship



„Essentially, all models are wrong but some are usefull.”
George Box, 1987.

Regression model:

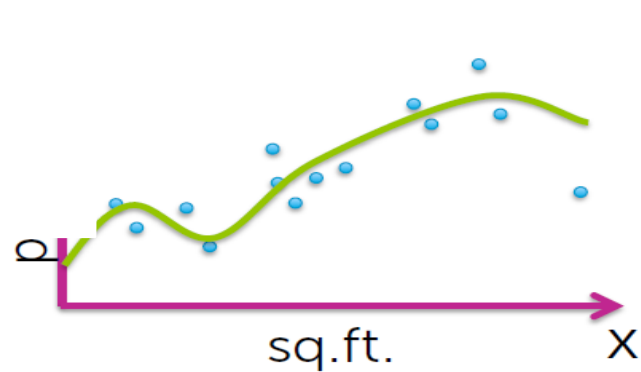
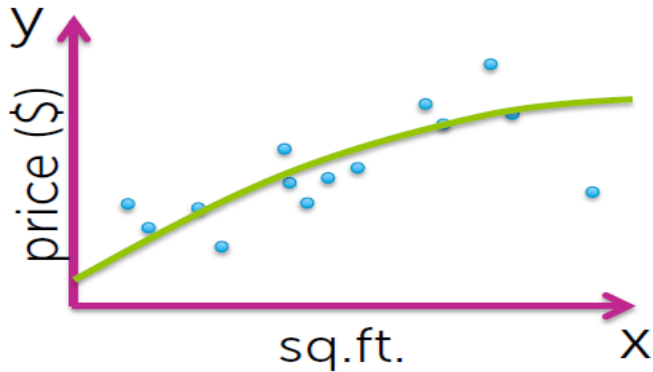
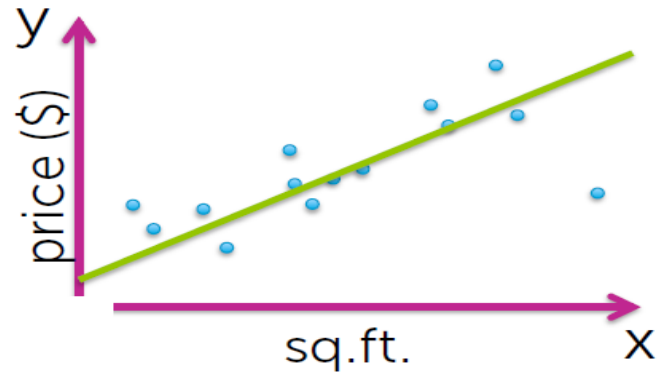
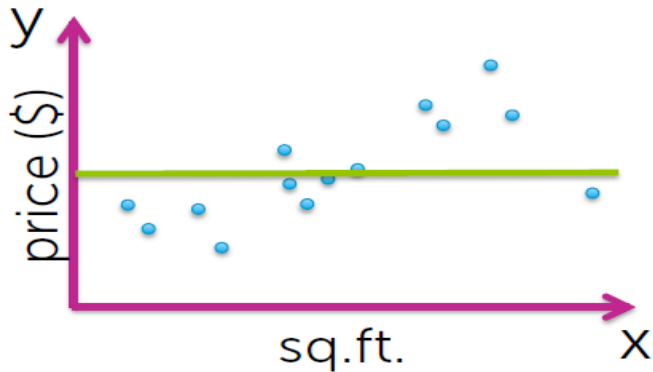
$$y_i = f(x_i) + \epsilon_i$$

$E[\epsilon_i] = 0$ ← equally likely that error is + or -
↑ expected value

⇓
 y_i is equally likely to be above or below $f(x_i)$

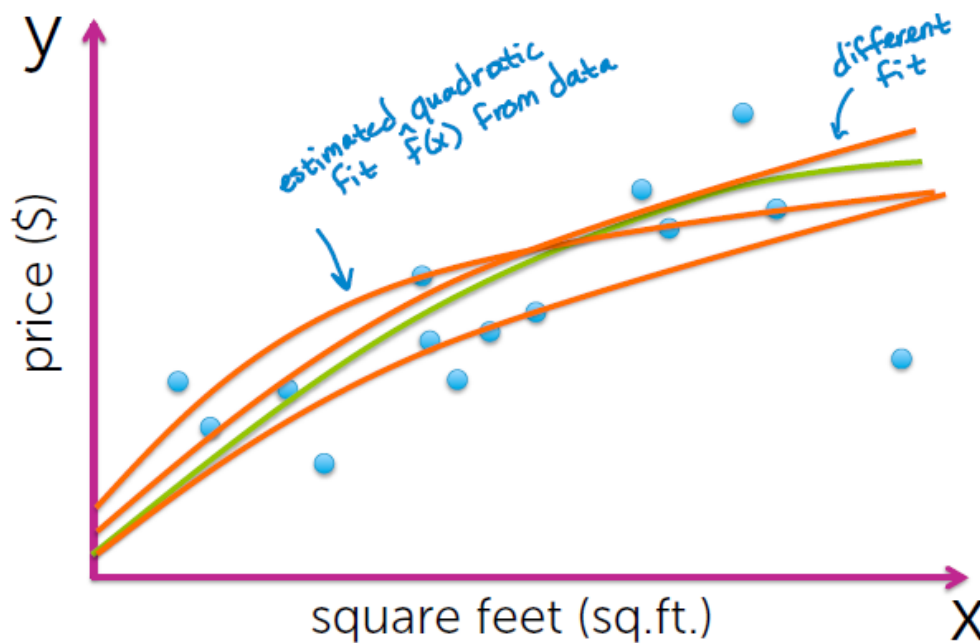
Task 1:

Which model to fit?



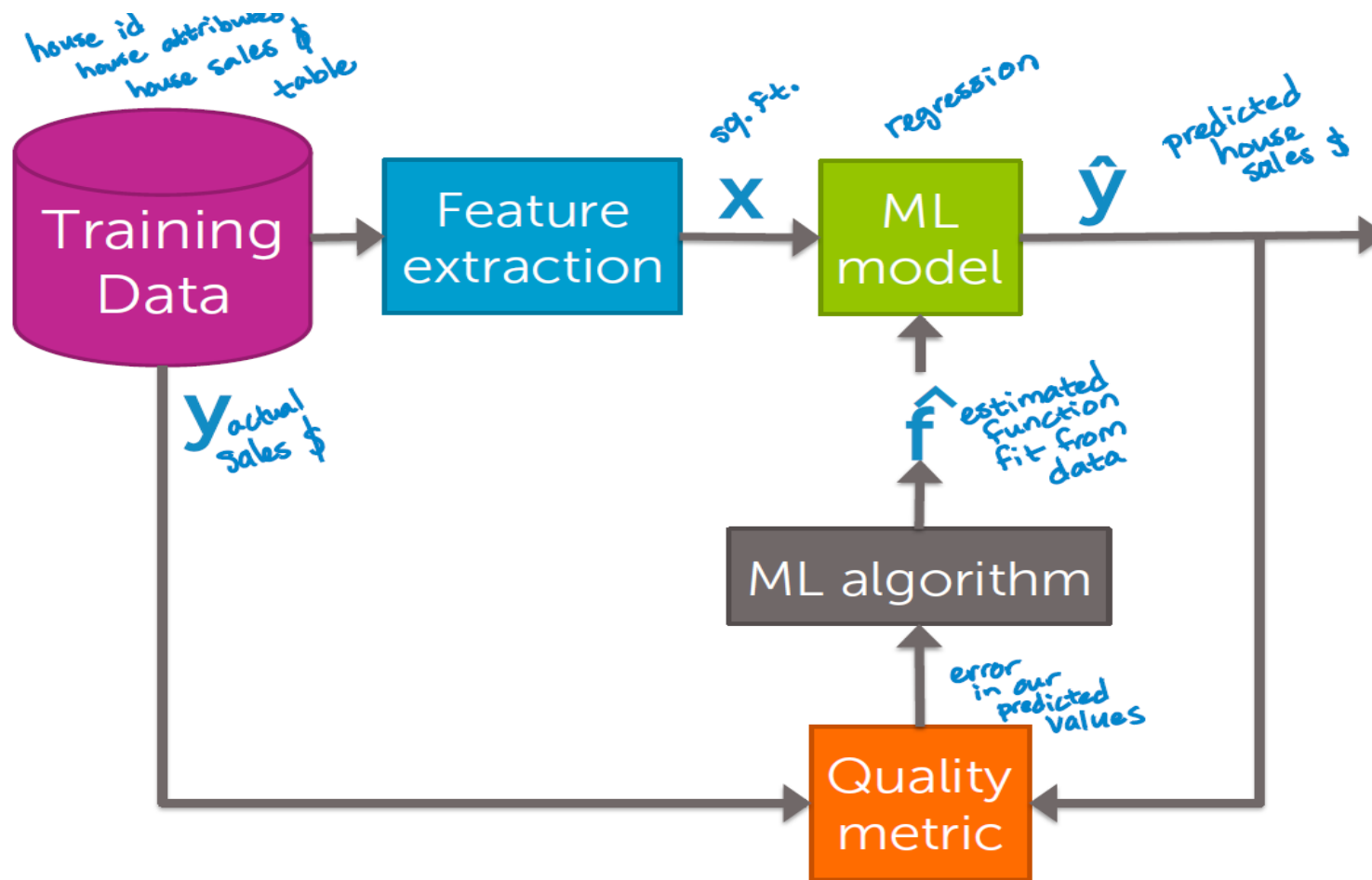
Task 2:

For a given model $f(x)$ estimate function $\hat{f}(x)$ from data



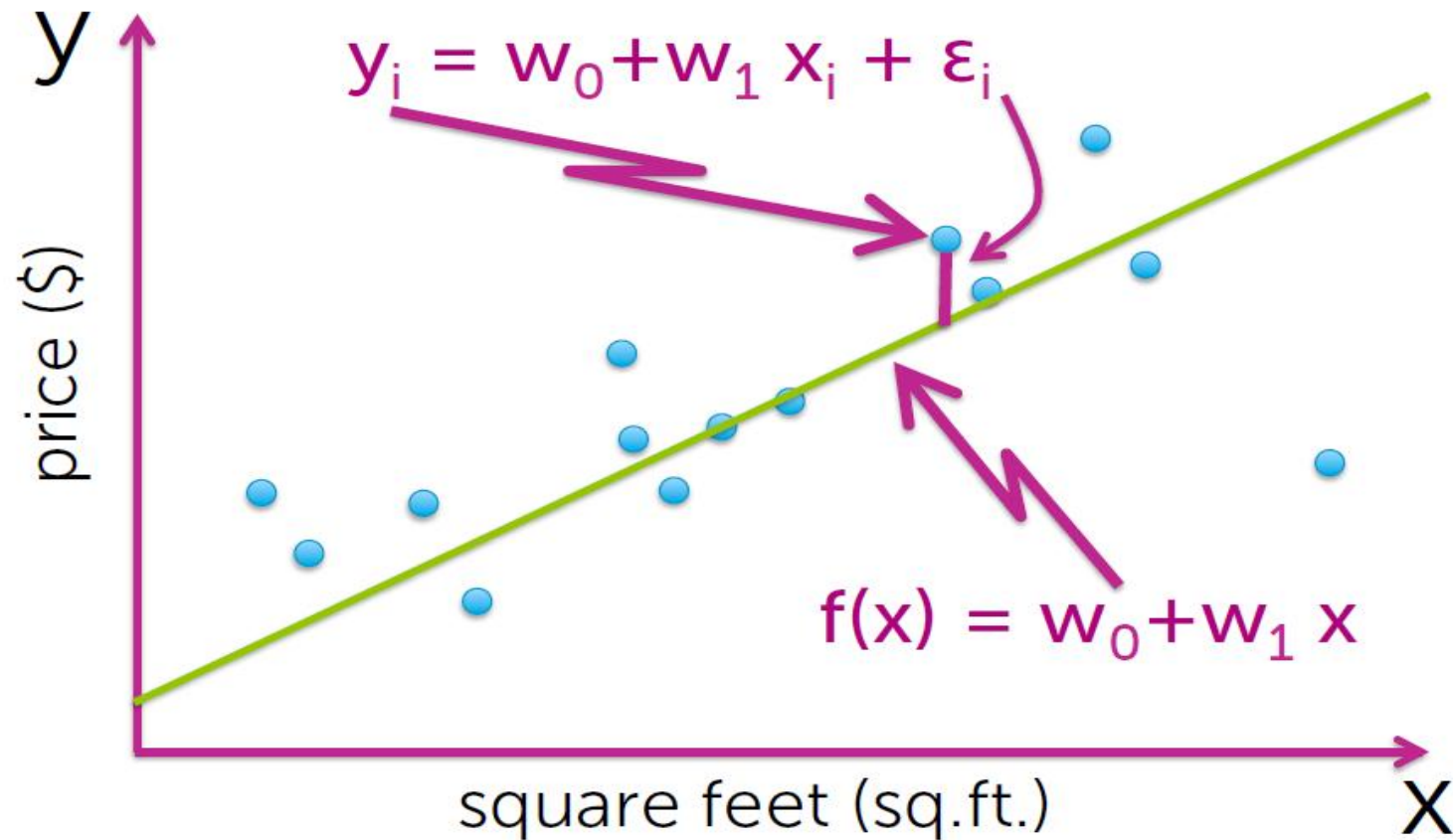
Assume model $f(x)$ is a quadratic function

How it works: baseline flow chart

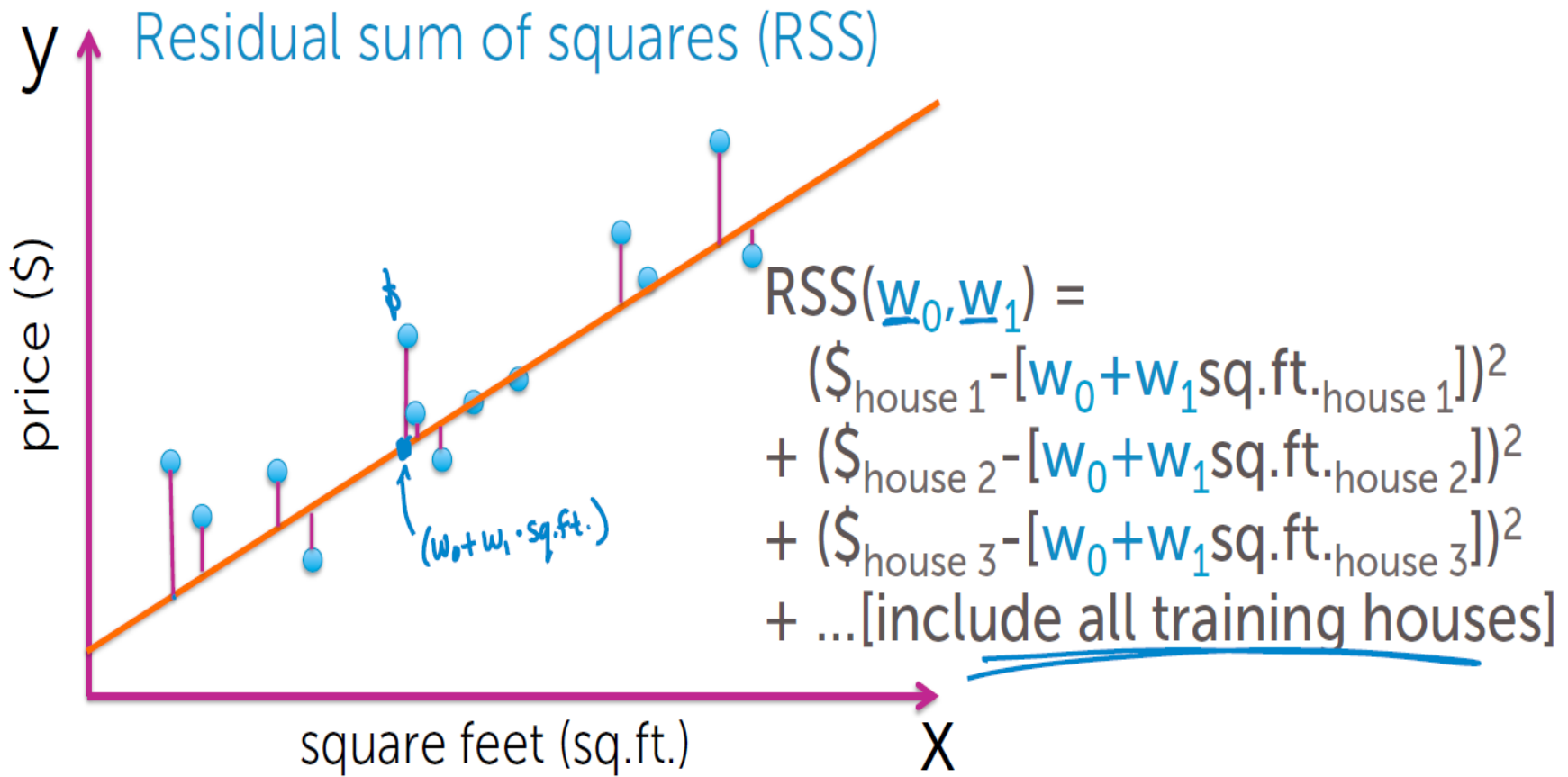


SIMPLE LINEAR REGRESSION

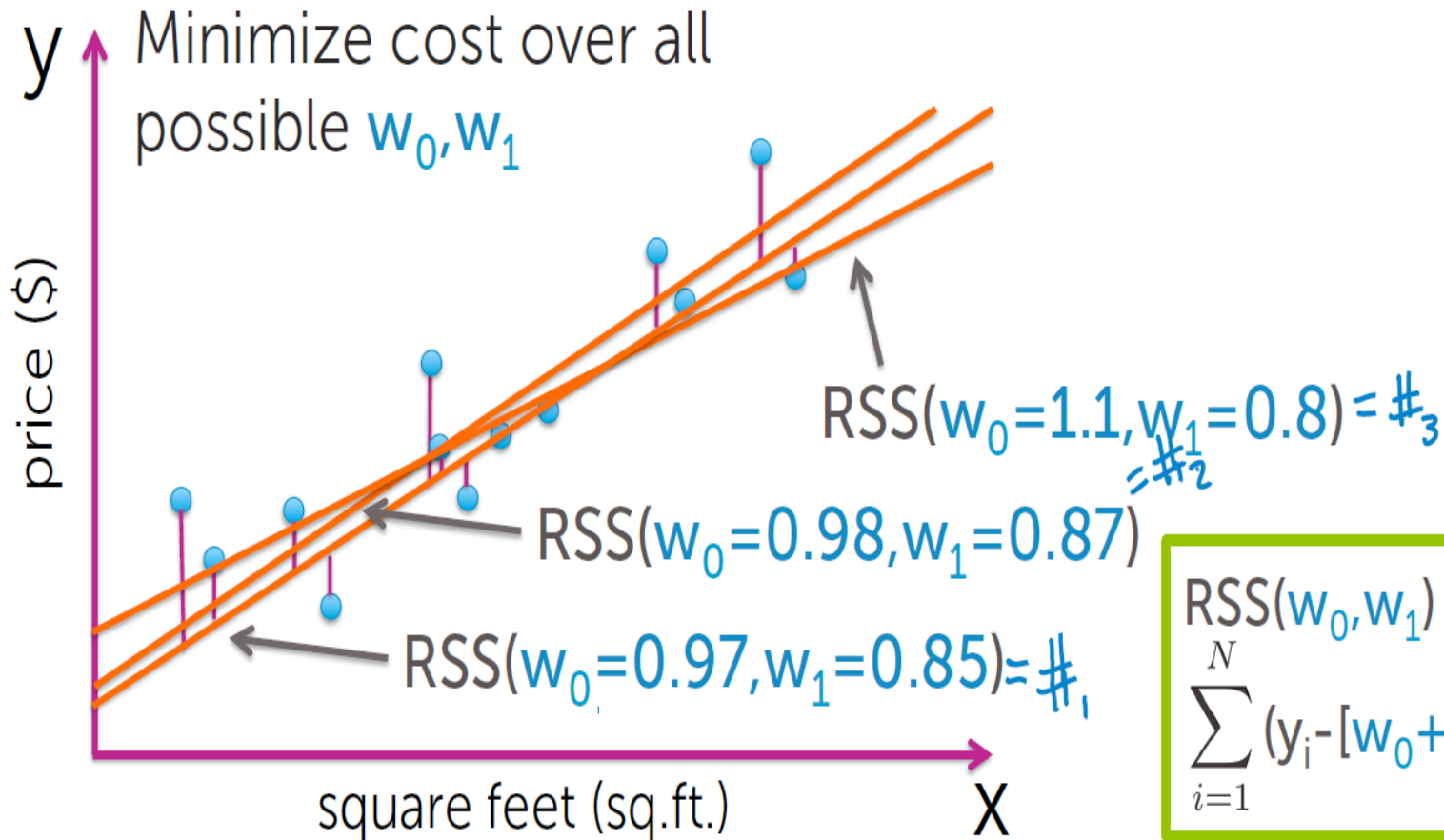
Simple linear regression model



The cost of using a given line

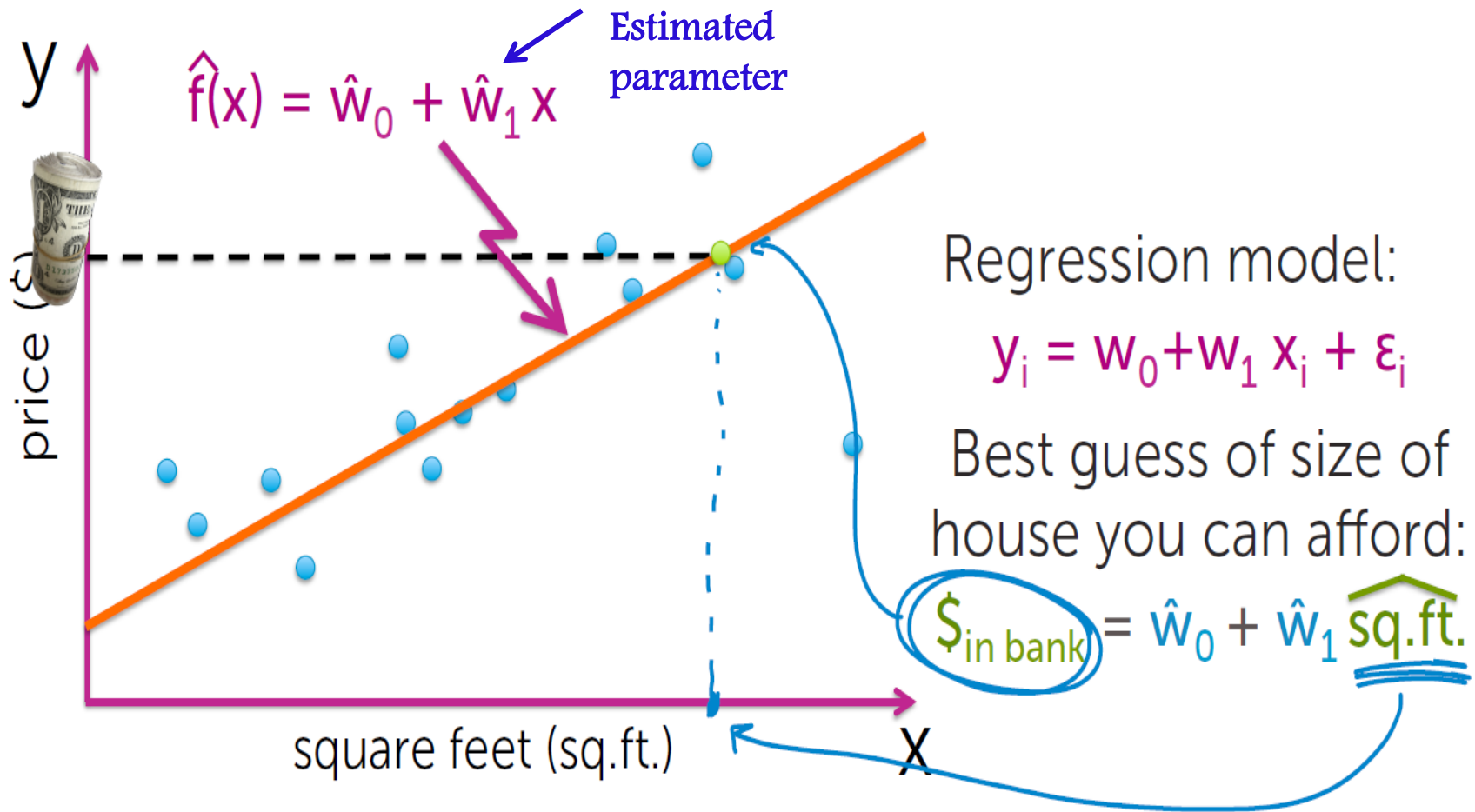


Find „best” line

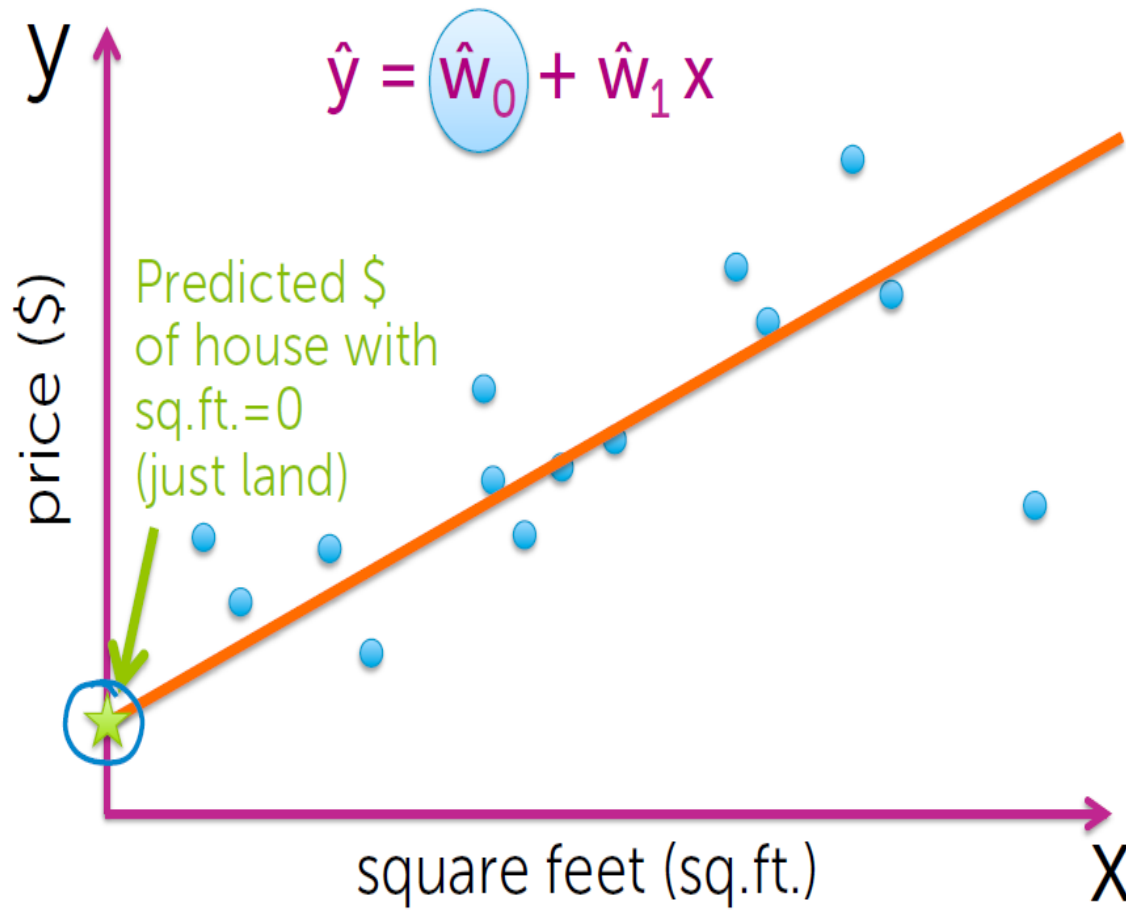


$$RSS(w_0, w_1) = \sum_{i=1}^N (y_i - [w_0 + w_1 x_i])^2$$

Predicting size of house you can afford



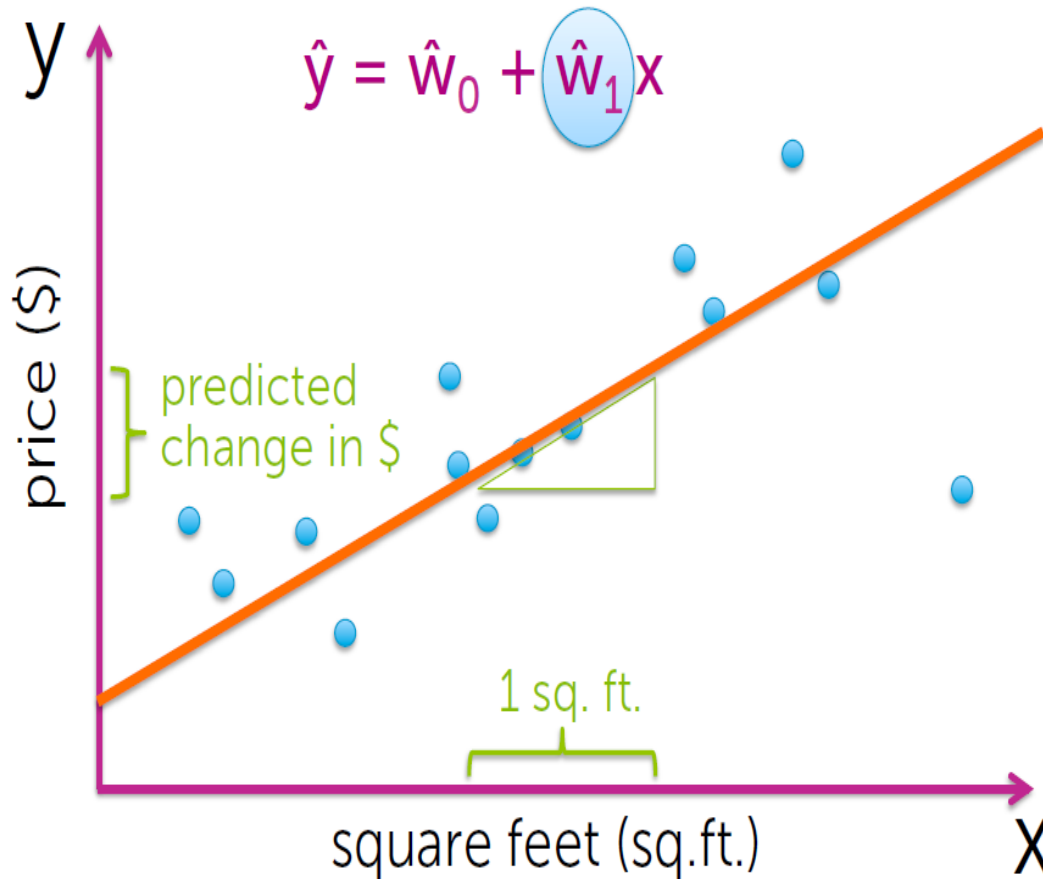
Interpreting the coefficients



$$\hat{y} = \hat{w}_0 \text{ when } x=0$$

not very meaningful

Interpreting the coefficients

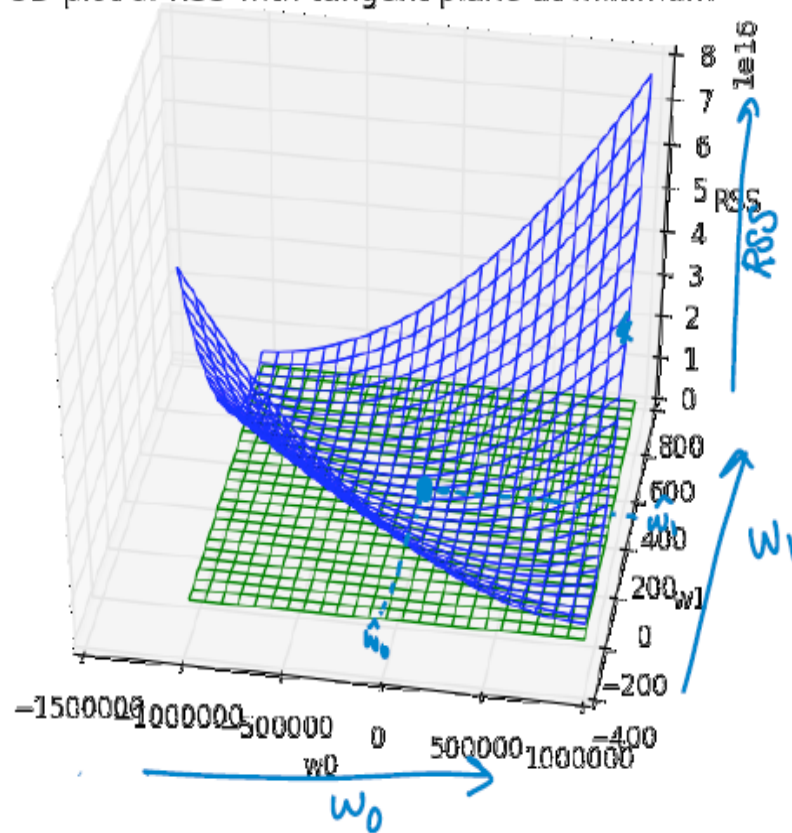


Magnitude of fit parameters depend on the units of both features and observations

$$\begin{aligned} & \hat{\$}_{1001 \text{ sq.ft.}} - \hat{\$}_{1000 \text{ sq.ft.}} \\ &= \hat{w}_0 + \hat{w}_1 \cdot 1001 \text{ sq.ft.} \\ &\quad - (\hat{w}_0 + \hat{w}_1 \cdot 1000 \text{ sq.ft.}) \\ &= \hat{w}_1 \\ &\text{predicted change in the output} \\ &\quad \text{per unit change in input} \end{aligned}$$

ML algorithm: minimising the cost

3D plot of RSS with tangent plane at minimum

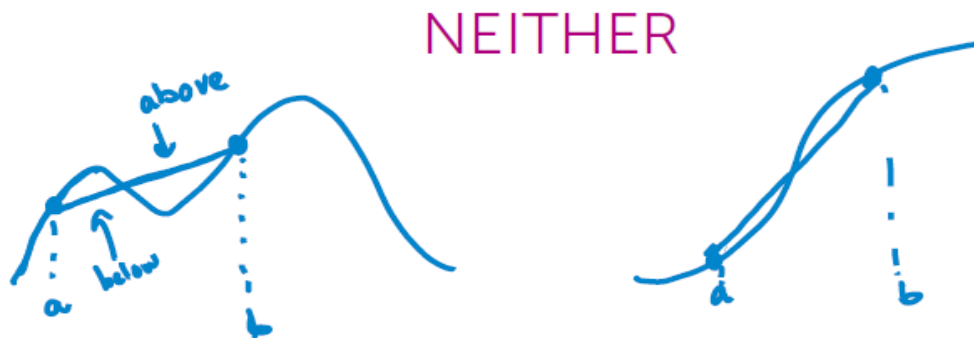
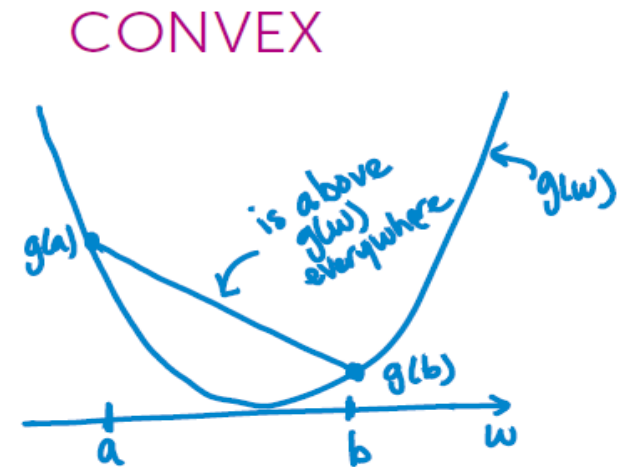
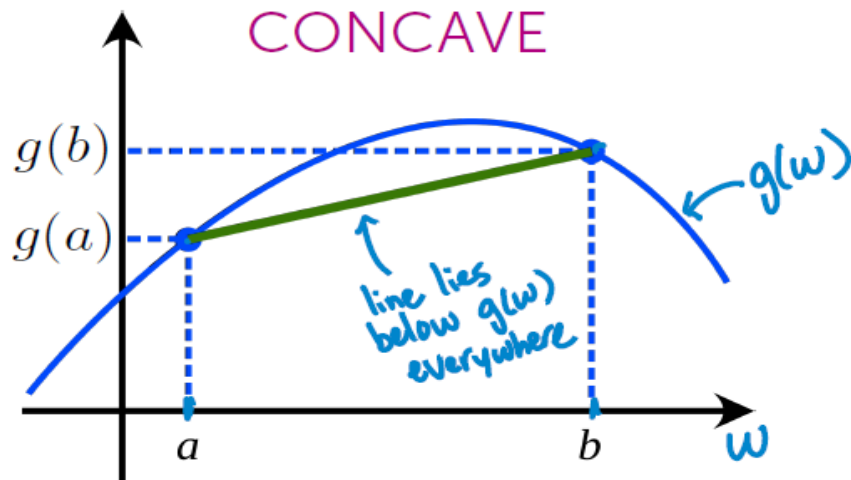


Minimize function
over all possible w_0, w_1

$$\min_{w_0, w_1} \sum_{i=1}^N (y_i - [w_0 + w_1 x_i])^2$$

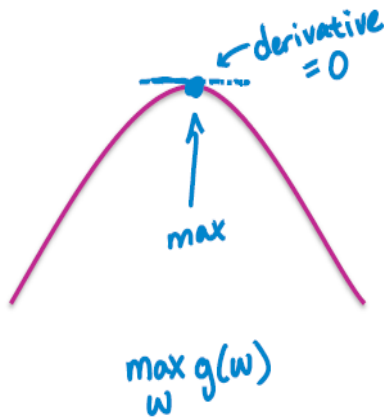
RSS(w_0, w_1) is a function
of 2 variables = $q(w_0, w_1)$

Convex/concave function

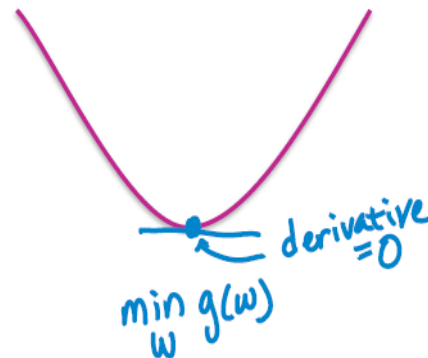


Finding max/min analytically

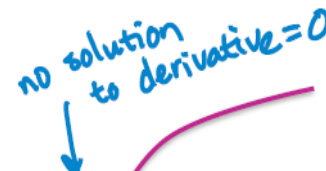
CONCAVE



CONVEX



NEITHER



Example:

$$g(w) = 5 - \underline{(w-10)^2}$$

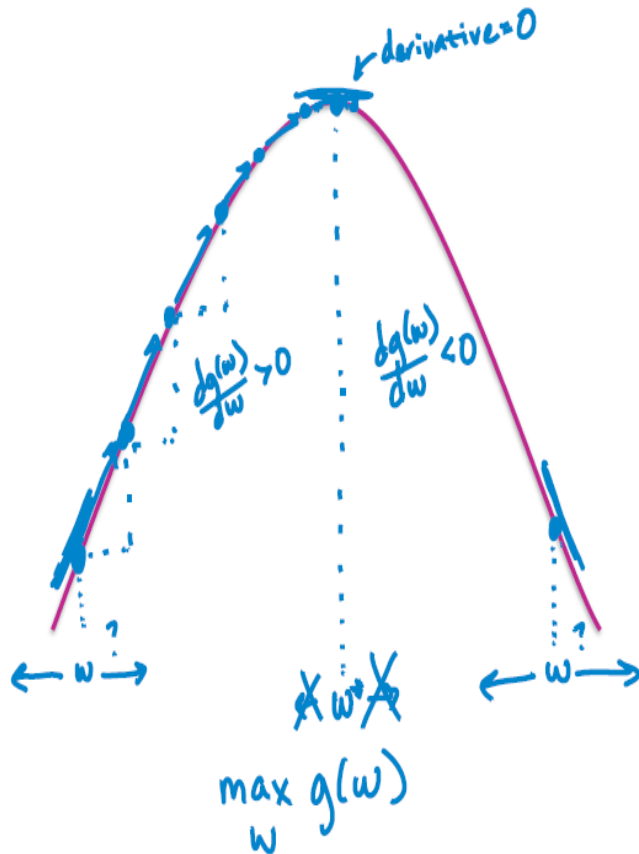
$$\begin{aligned}\frac{dg(w)}{dw} &= 0 - 2(w-10)^1 \cdot 1 \\ &= -2w + 20\end{aligned}$$

Set derivate = 0:

$$\begin{aligned}-2w + 20 &= 0 \\ w &= 10\end{aligned}$$



Finding the max via hill climbing



Sign of the derivative is saying me
what I want to do :move left or right
or stay where I am

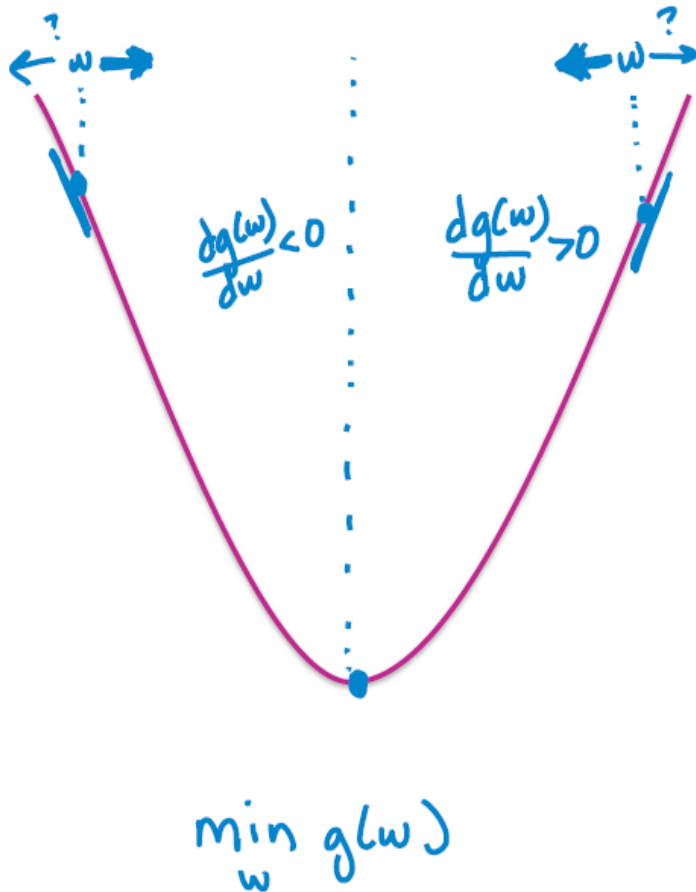
How do we know whether to move
 w to right or left?
(Inc. or dec. the value of w ?)

While not converged

$$w^{(t+1)} \leftarrow w^{(t)} + \eta \frac{dg(w)}{dw}$$

iteration t stepsize

Finding the min via hill descent



When derivative is positive, we want to decrease w
and when derivative is negative, we want to increase w

Algorithm:

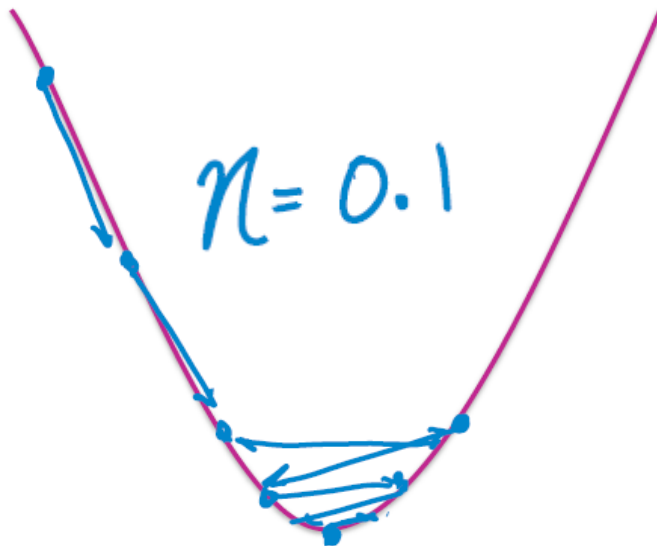
while not converged

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \left. \frac{dg}{dw} \right|_{w^{(t)}}$$

Choosing the step size (stepsize schedule)

Fixed

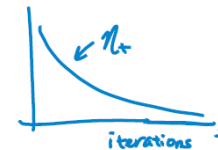
Works well for strongly convex functions



Varying

Common choices:

$$\eta_t = \frac{\alpha}{t}$$
$$\eta_t = \frac{\alpha}{\sqrt{t}}$$



Try not to decrease η too fast

Convergence criteria

For convex functions,
optimum occurs when

$$\frac{dg(w)}{dw} = 0$$

In practice, stop when

$$\left| \frac{dg(w)}{dw} \right| < \epsilon$$

threshold to be set

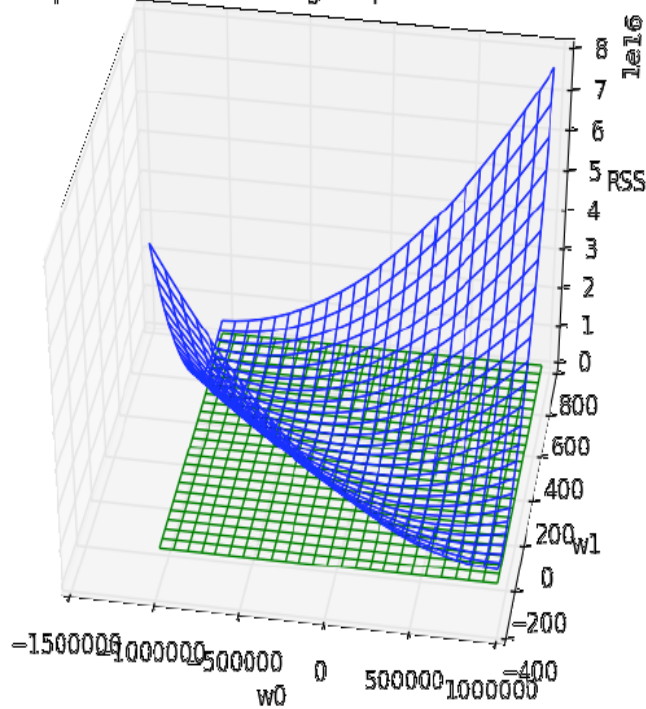
That will be „good enough”
value of ϵ depends on the data we are looking at

Algorithm:

while not converged
 $w^{(t+1)} \leftarrow w^{(t)} - \eta \left. \frac{dg}{dw} \right|_{w^{(t)}}$

Moving to multiple dimensions

3D plot of RSS with tangent plane at minimum



$$\nabla g(\mathbf{w}) = \begin{bmatrix} \frac{\partial g}{\partial w_0} \\ \frac{\partial g}{\partial w_1} \\ \vdots \\ \frac{\partial g}{\partial w_p} \end{bmatrix}$$

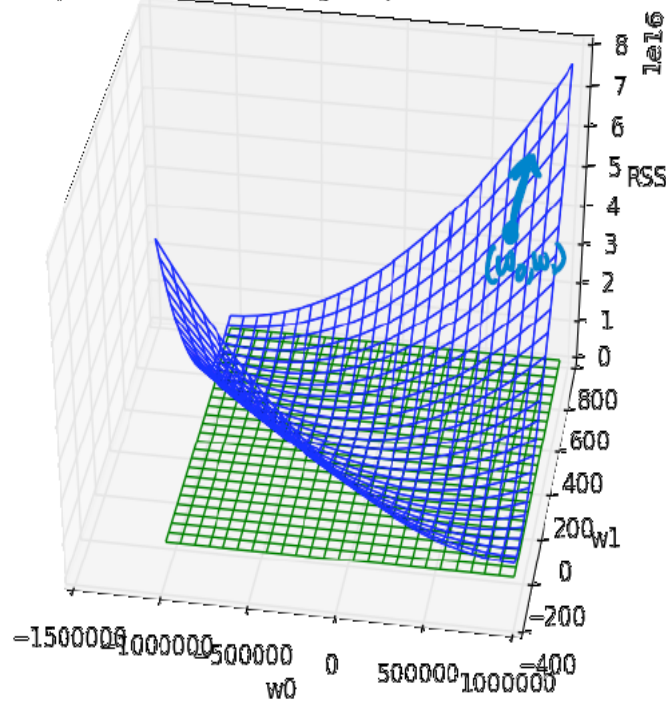
gradient $[w_0, w_1, \dots, w_p]$

$(p+1)$ -dimensional vector

partial derivative is like a derivative with respect to w_i treating all other variables as constants

Gradient example

3D plot of RSS with tangent plane at minimum



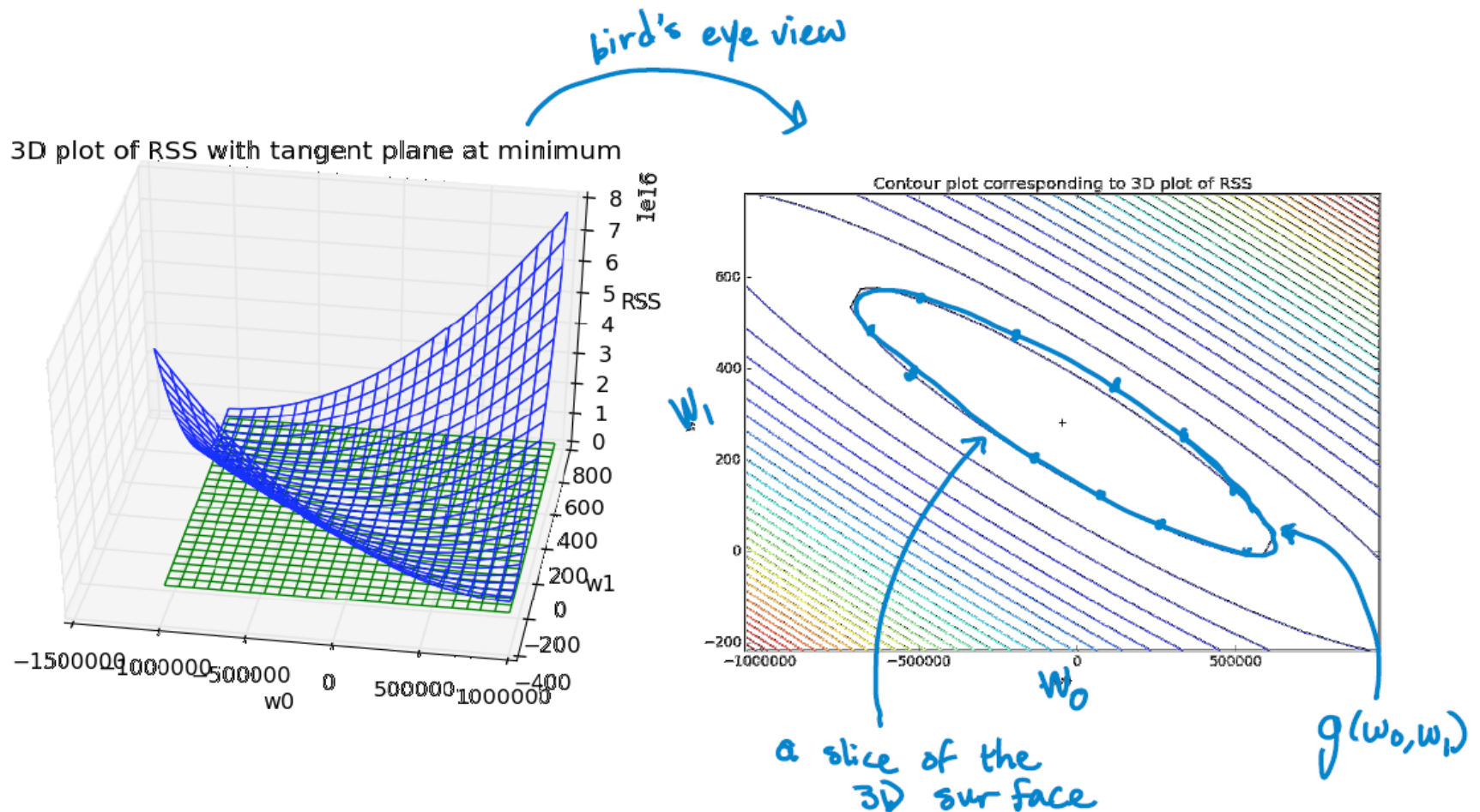
$$g(w) = 5w_0 + 10w_0w_1 + 2w_1^2$$

$$\frac{\partial g}{\partial w_0} = 5 + 10w_1$$

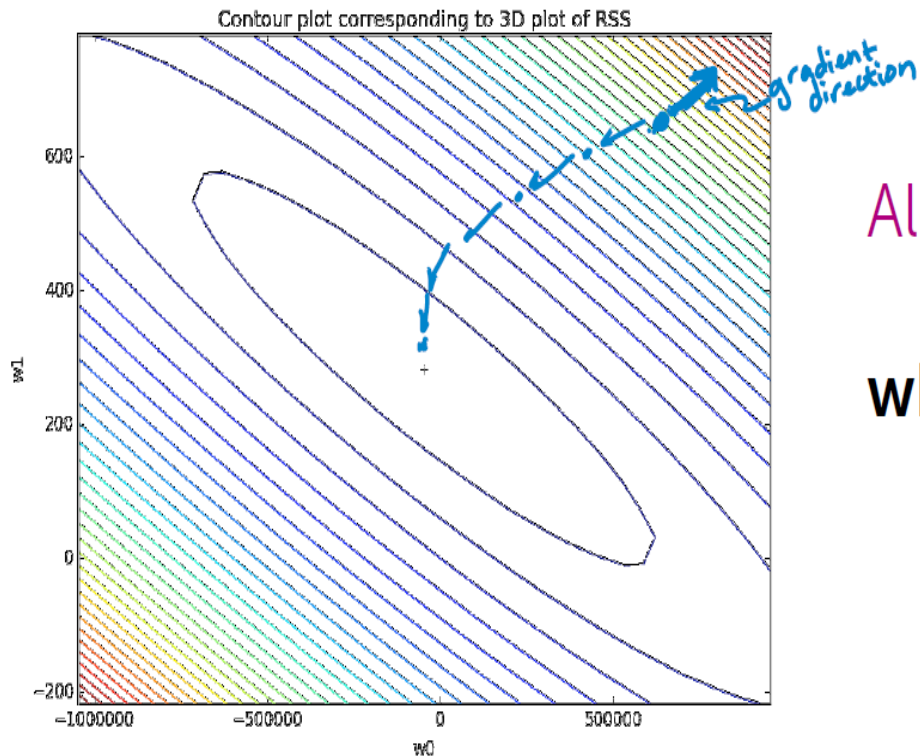
$$\frac{\partial g}{\partial w_1} = 10w_0 + 4w_1$$

$$\nabla g(w) = \begin{bmatrix} 5 + 10w_1 \\ 10w_0 + 4w_1 \end{bmatrix}$$

Contour plots



Gradient descent



Algorithm:

while not converged

$$w^{(t+1)} \leftarrow w^{(t)} - \eta \nabla g(w^{(t)})$$

$$\begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} \leftarrow \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix} - \eta \begin{bmatrix} \vdots \\ \vdots \\ \vdots \end{bmatrix}$$

Convergence:
 $\|\nabla g(w)\| < \epsilon$

Compute the gradient

$$\text{RSS}(w_0, w_1) = \sum_{i=1}^N (y_i - [w_0 + w_1 x_i])^2$$

Taking the derivative w.r.t. w_0

$$\begin{aligned} & \sum_{i=1}^N 2 (y_i - [w_0 + w_1 x_i])' \cdot (-1) \\ &= -2 \sum_{i=1}^N (y_i - [w_0 + w_1 x_i]) \end{aligned}$$

Putting it together:

$$\nabla \text{RSS}(w_0, w_1) = \begin{bmatrix} -2 \sum_{i=1}^N [y_i - (w_0 + w_1 x_i)] \\ -2 \sum_{i=1}^N [y_i - (w_0 + w_1 x_i)] x_i \end{bmatrix}$$

Taking the derivative w.r.t. w_1

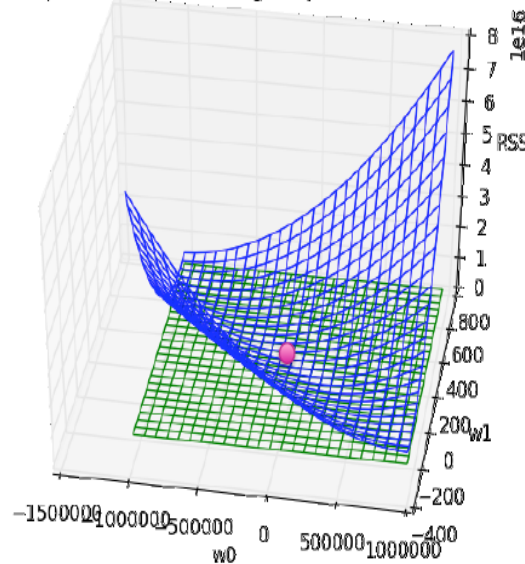
$$\begin{aligned} & \sum_{i=1}^N 2 \underline{(y_i - [w_0 + w_1 x_i])'} \cdot (-x_i) \\ &= -2 \sum_{i=1}^N (y_i - [w_0 + w_1 x_i]) \underline{x_i} \end{aligned}$$

Approach 1: set gradient to 0

$$\nabla \text{RSS}(w_0, w_1) = \begin{bmatrix} -2 \sum_{i=1}^N [y_i - (w_0 + w_1 x_i)] \\ -2 \sum_{i=1}^N [y_i - (w_0 + w_1 x_i)] x_i \end{bmatrix}$$

This method is called „Closed form solution”

3D plot of RSS with tangent plane at minimum



top term:

$$\hat{w}_0 = \frac{\sum_{i=1}^N y_i}{N} - \hat{w}_1 \frac{\sum_{i=1}^N x_i}{N}$$

average house sales price ← $\frac{\sum y_i}{N}$
estimate of the slope ← $\hat{w}_1$
average sq.-ft. ← $\frac{\sum x_i}{N}$

bottom term:

$$\sum y_i x_i - \hat{w}_0 \sum x_i - \hat{w}_1 \sum x_i^2 = 0$$

plug in →

$$\hat{w}_1 = \frac{\sum y_i x_i - \frac{\sum y_i \sum x_i}{N}}{\sum x_i^2 - \frac{\sum x_i \sum x_i}{N}}$$

Note:

$$\begin{aligned} &\sum_{i=1}^N y_i \\ &\sum_{i=1}^N x_i \\ &\sum_{i=1}^N y_i x_i \\ &\sum_{i=1}^N x_i^2 \end{aligned}$$

Approach 2: gradient descent

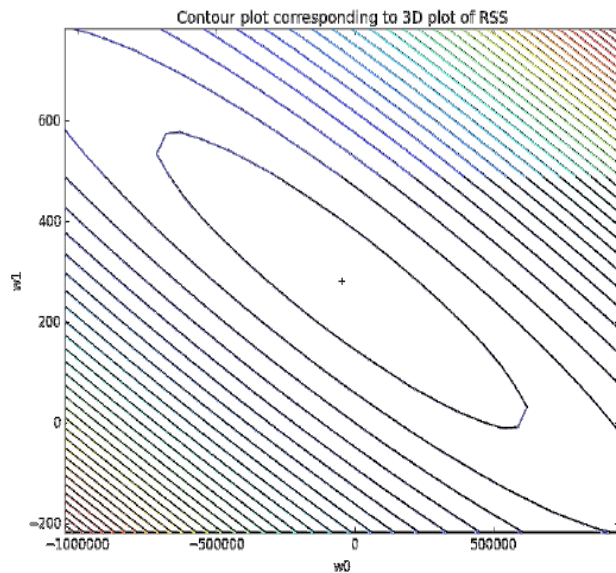
Interpreting the gradient:

$$\nabla \text{RSS}(w_0, w_1) = \begin{bmatrix} -2 \sum_{i=1}^N [y_i - (w_0 + w_1 x_i)] \\ -2 \sum_{i=1}^N [y_i - (w_0 + w_1 x_i)] x_i \end{bmatrix} = \begin{bmatrix} -2 \sum_{i=1}^N [y_i - \hat{y}_i(w_0, w_1)] \\ -2 \sum_{i=1}^N [y_i - \hat{y}_i(w_0, w_1)] x_i \end{bmatrix}$$

Handwritten annotations:
- y_i : actual house sales observation
- $\hat{y}_i(w_0, w_1)$: predicted value

Approach 2: gradient descent

$$\nabla \text{RSS}(w_0, w_1) = \begin{bmatrix} -2 \sum_{i=1}^N [y_i - \hat{y}_i(w_0, w_1)] \\ -2 \sum_{i=1}^N [y_i - \hat{y}_i(w_0, w_1)] x_i \end{bmatrix}$$



while not converged $(-2) \cdot (-\eta)$

$$\begin{bmatrix} w_0^{(t+1)} \\ w_1^{(t+1)} \end{bmatrix} \leftarrow \begin{bmatrix} w_0^{(t)} \\ w_1^{(t)} \end{bmatrix} + 2\eta \begin{bmatrix} \sum_{i=1}^N [y_i - \hat{y}_i(w_0^{(t)}, w_1^{(t)})] \\ \sum_{i=1}^N [y_i - \hat{y}_i(w_0^{(t)}, w_1^{(t)})] x_i \end{bmatrix}$$

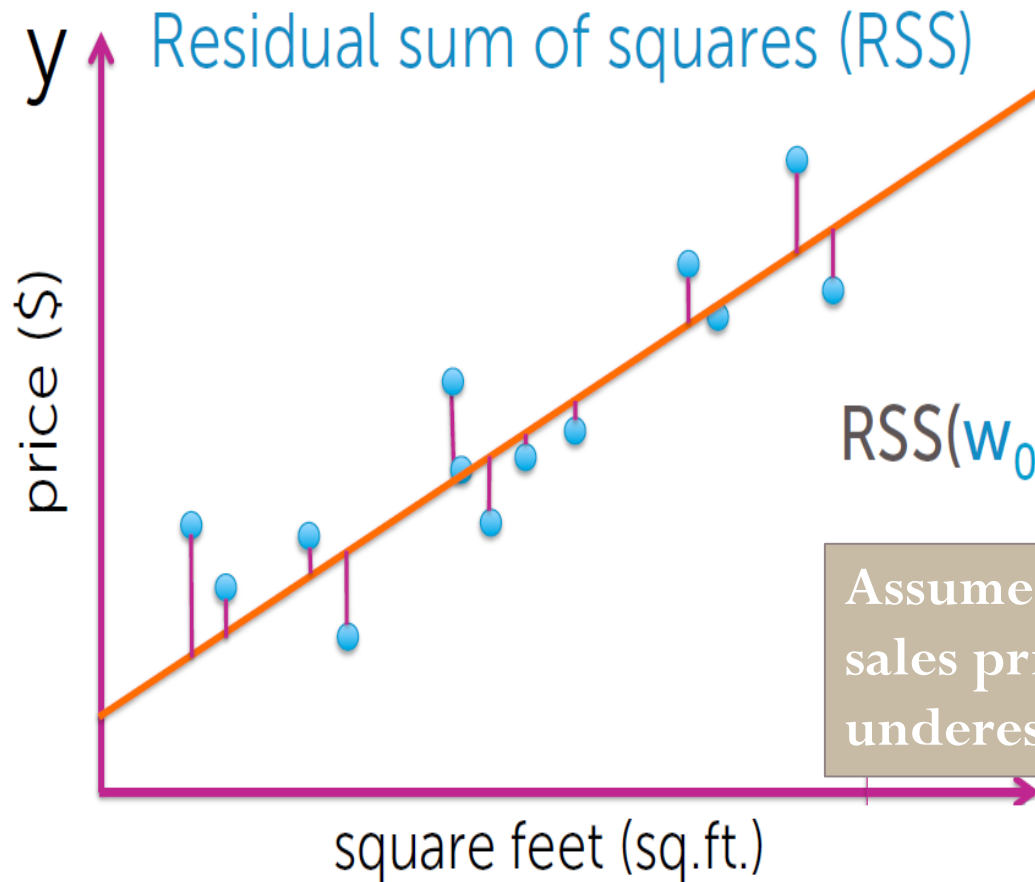
If overall, under predicting $\hat{y}_i$, then $\sum [y_i - \hat{y}_i]$ is positive
 $\rightarrow w_0$ is going to increase

similar intuition for w_1 , but multiply by x_i

Comparing the approaches

- For most ML problems, cannot solve $\text{gradient} = 0$
- Even if solving $\text{gradient} = 0$ is feasible, gradient descent can be more efficient
- Gradient descent relies on choosing stepsize and convergence criteria

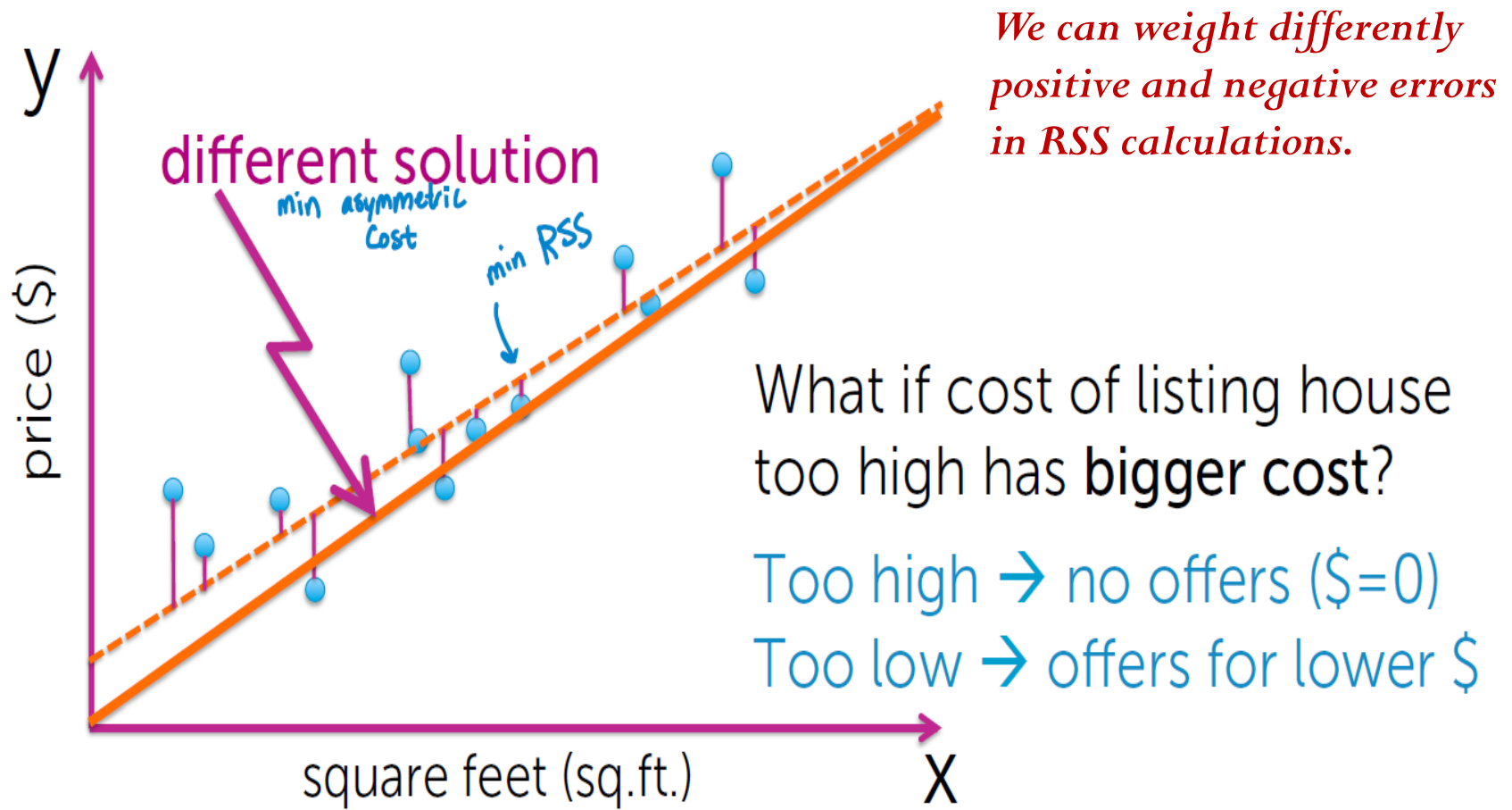
Symmetric cost function



$$RSS(w_0, w_1) = \sum_{i=1}^N (y_i - [w_0 + w_1 x_i])^2$$

Assumes error of overestimating sales price is the same as underestimating it

Asymmetric cost functions

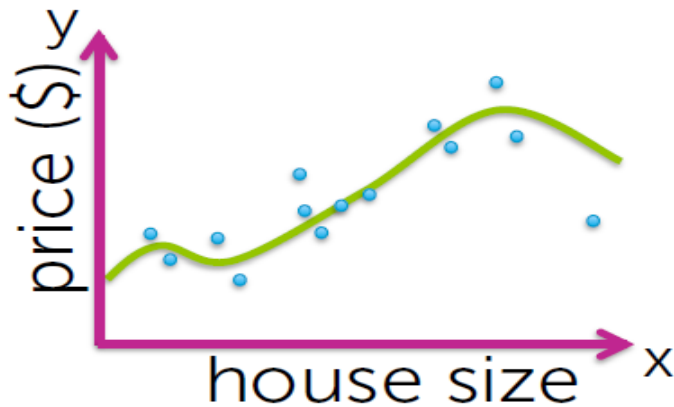


What you can do now

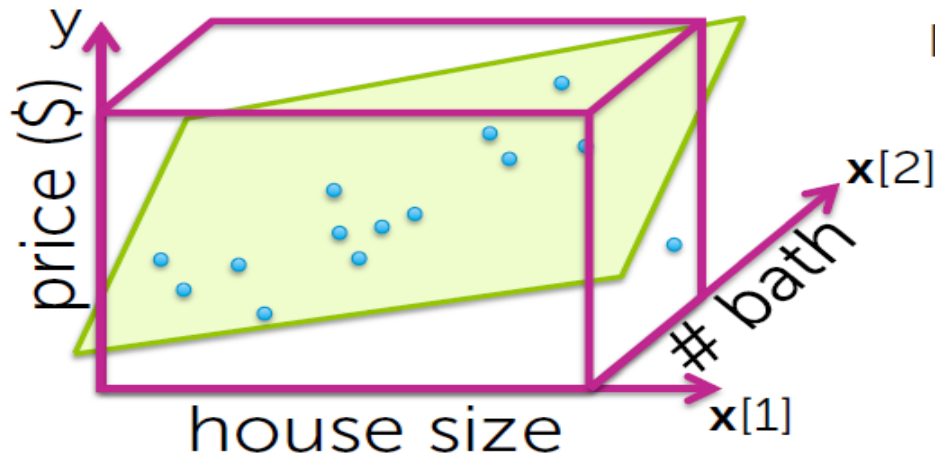
- Describe the input (features) and output (real-valued predictions) of a regression model
- Calculate a goodness-of-fit metric (e.g., RSS)
- Estimate model parameters to minimize RSS using gradient descent
- Interpret estimated model parameters
- Exploit the estimated model to form predictions
- Discuss the possible influence of high leverage points
- Describe intuitively how fitted line might change when assuming different goodness-of-fit metrics

MULTIPLE REGRESSION

Multiple regression



Fit **more complex relationships** than just a line



Incorporate more inputs

- Square feet
- # bathrooms
- # bedrooms
- Lot size
- Year built
- ...

Polynomial regression

Model:

$$y_i = w_0 + w_1 x_i + w_2 x_i^2 + \dots + w_p x_i^p + \varepsilon_i$$

treat as different **features**

feature 1 = 1 (constant) *parameter 1 = w_0*

feature 2 = x *parameter 2 = w_1*

feature 3 = x^2 *parameter 3 = w_2*

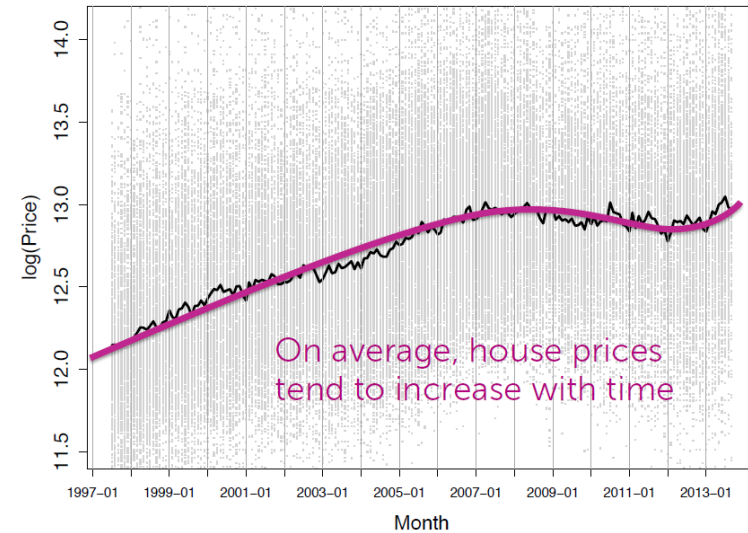
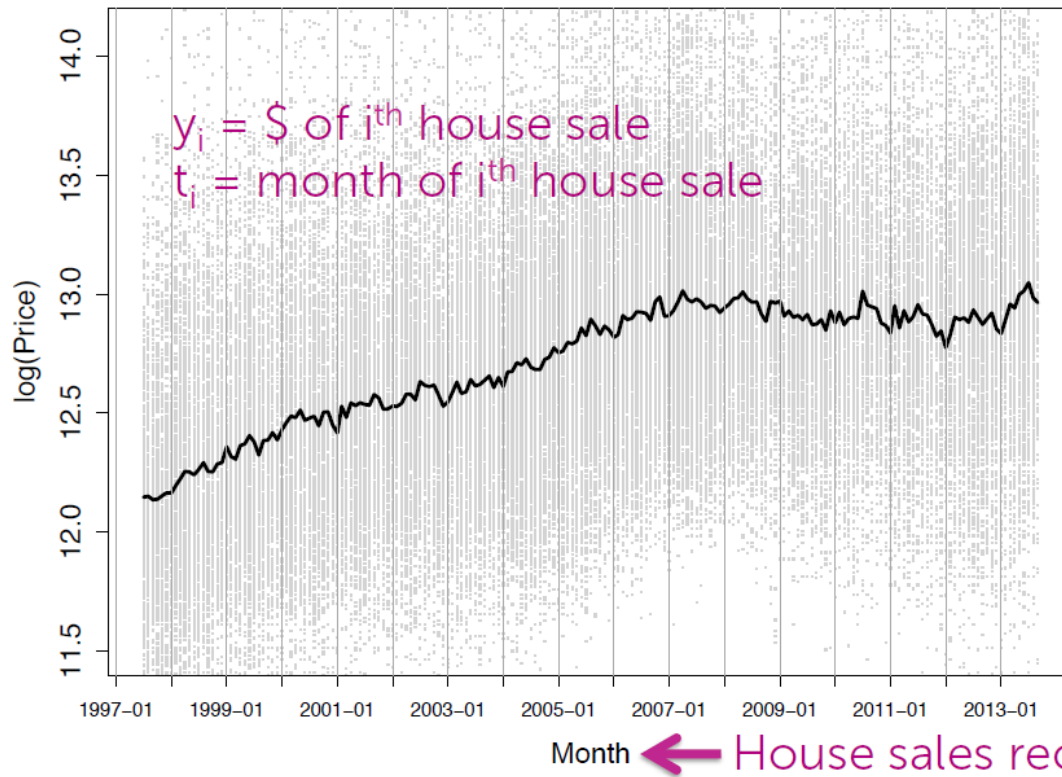
...

...

feature $p+1 = x^p$ *parameter $p+1 = w_p$*

Other functional forms of one input

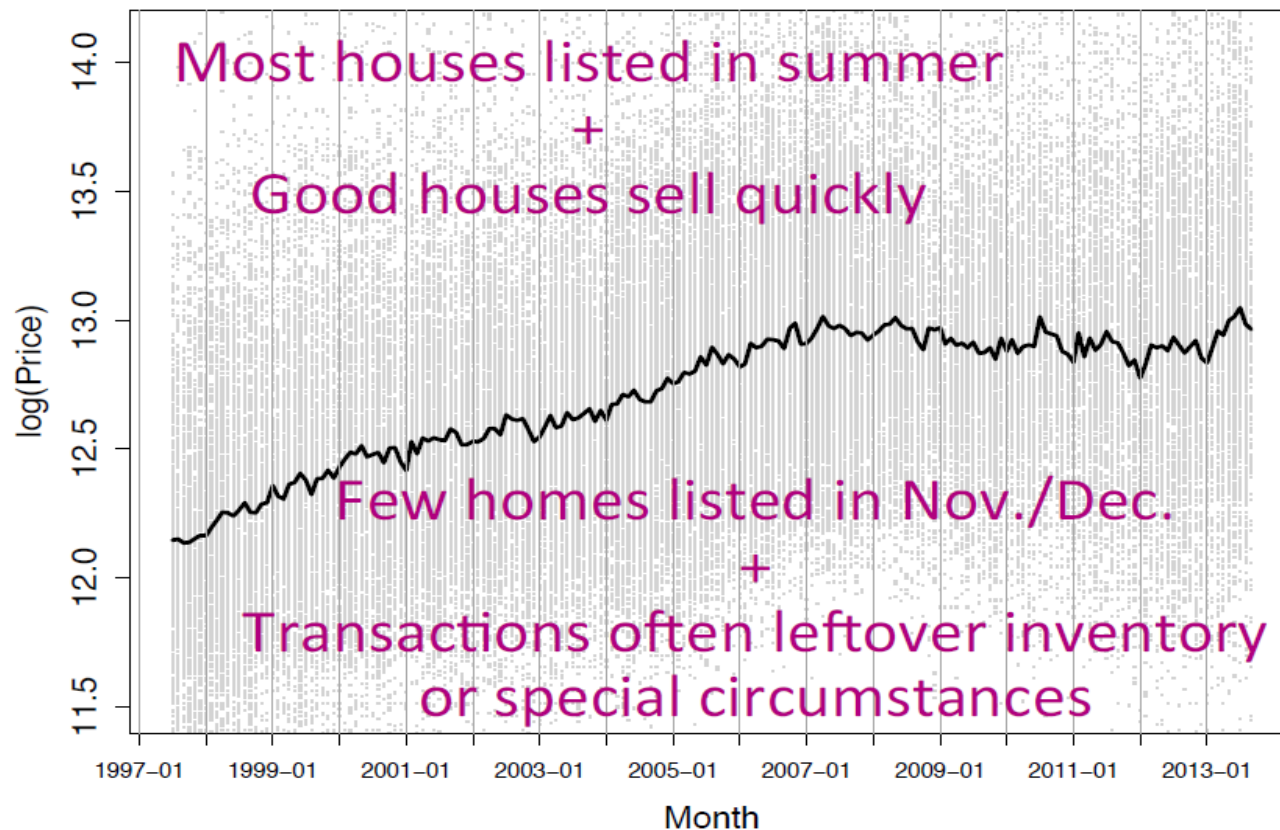
- Trends in time series



This trend can be modeled with polynomial function.

Other functional forms of one input

- **Seasonality**



Example of detrending

Model:

$$y_i = w_0 + w_1 t_i + w_2 \sin(2\pi t_i / 12 - \Phi) + \varepsilon_i$$

Linear trend

Unknown phase/shift

Seasonal component =
Sinusoid with period 12
(resets annually)



Trigonometric identity: $\sin(a - b) = \sin(a)\cos(b) - \cos(a)\sin(b)$

$$\rightarrow \sin(2\pi t_i / 12 - \Phi) = \sin(2\pi t_i / 12)\cos(\Phi) - \cos(2\pi t_i / 12)\sin(\Phi)$$

Example of detrending

Equivalently,

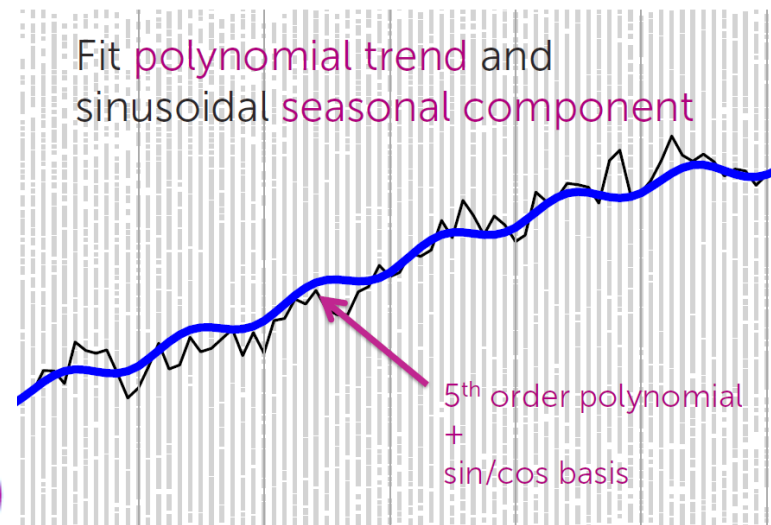
$$y_i = w_0 + w_1 t_i + w_2 \sin(2\pi t_i / 12) + w_3 \cos(2\pi t_i / 12) + \varepsilon_i$$

feature 1 = 1 (constant)

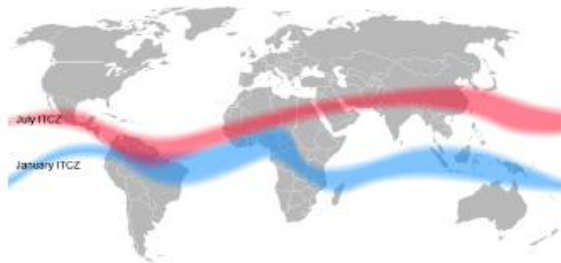
feature 2 = t

feature 3 = $\sin(2\pi t/12)$

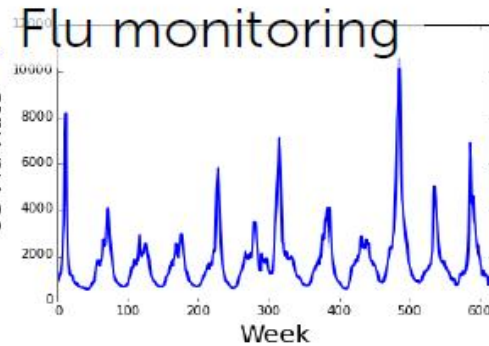
feature 4 = $\cos(2\pi t/12)$



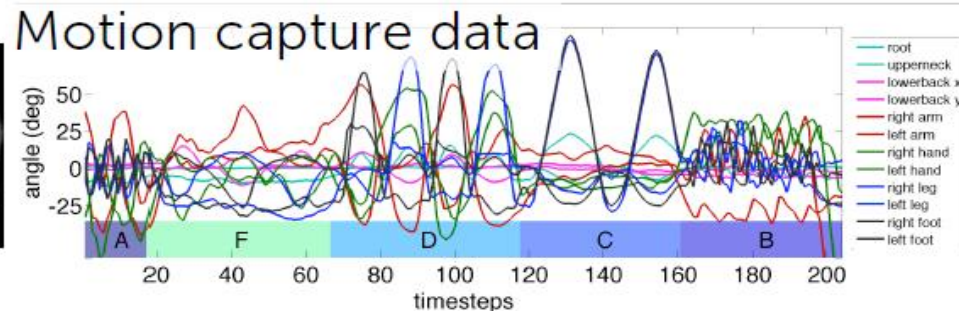
Other examples of seasonality



Weather modeling
(e.g., temperature, rainfall)



Demand forecasting
(e.g., jacket purchases)



Generic basic expansion

Model:

$$\begin{aligned} y_i &= w_0 h_0(x_i) + w_1 h_1(x_i) + \dots + w_D h_D(x_i) + \varepsilon_i \\ &= \sum_{j=0}^D w_j h_j(x_i) + \varepsilon_i \end{aligned}$$

feature 1 = $h_0(x)$...often 1 (constant)

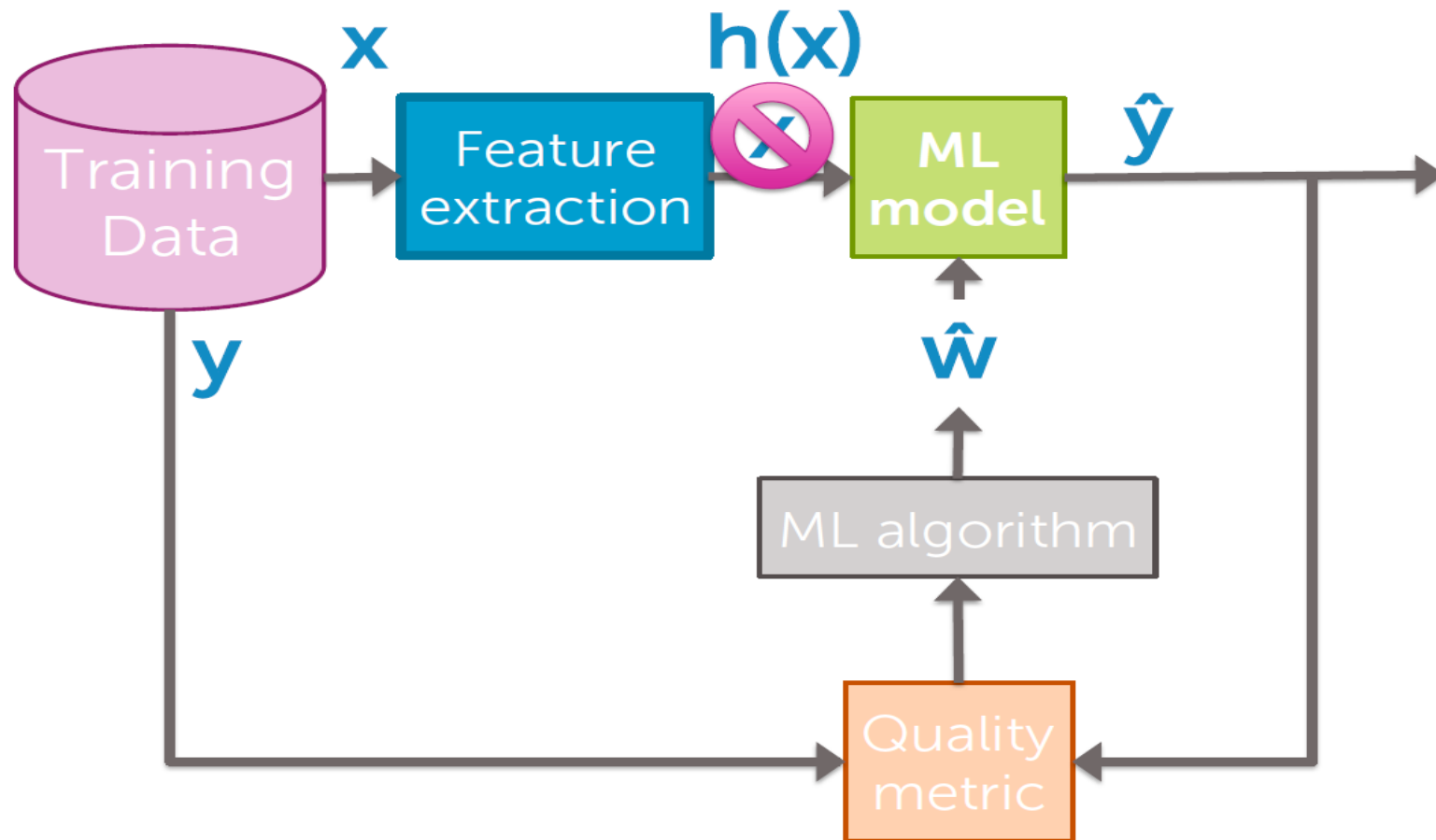
feature 2 = $h_1(x)$... e.g., x

feature 3 = $h_2(x)$... e.g., x^2 or $\sin(2\pi x/12)$

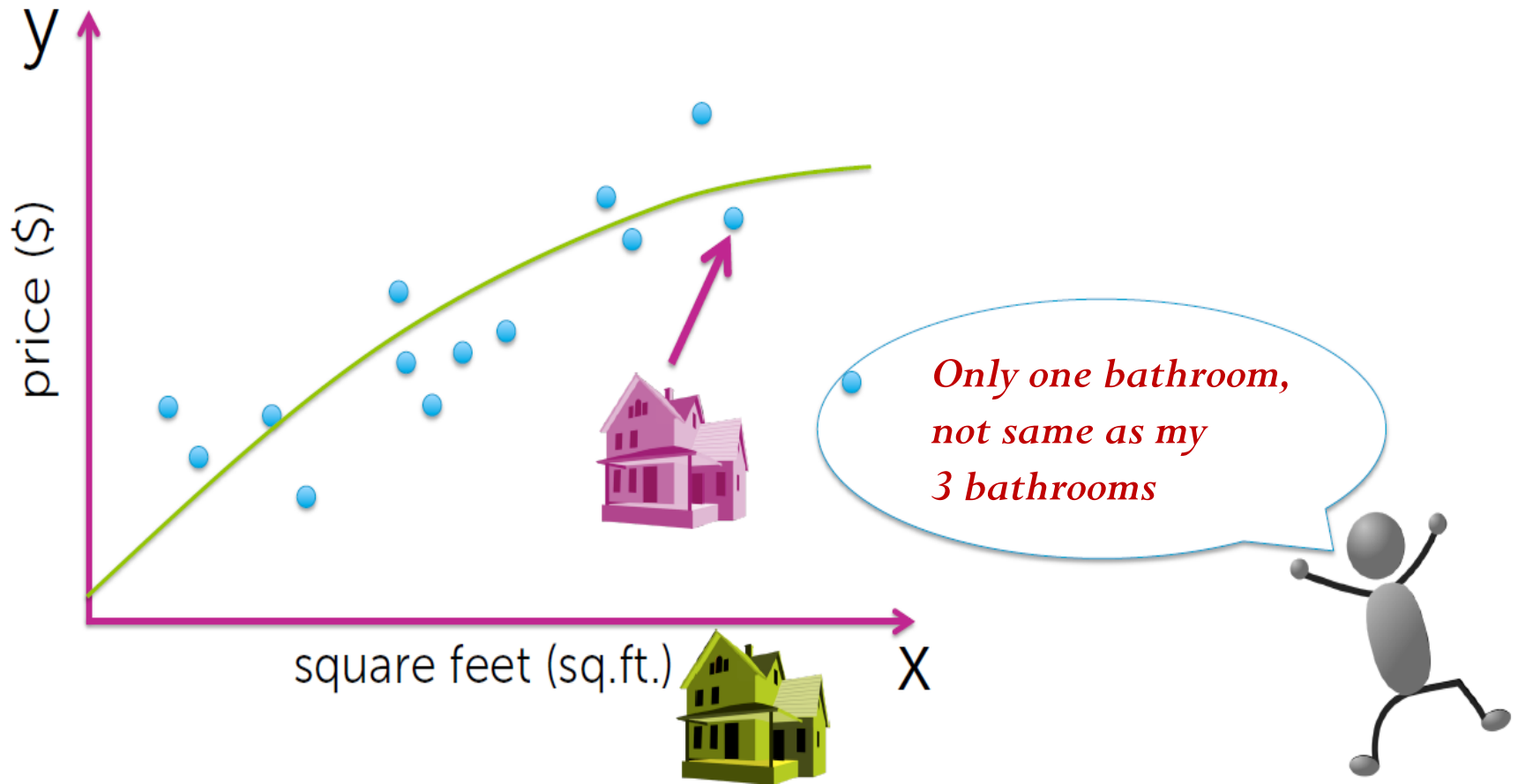
...

feature $D+1$ = $h_D(x)$... e.g., x^p

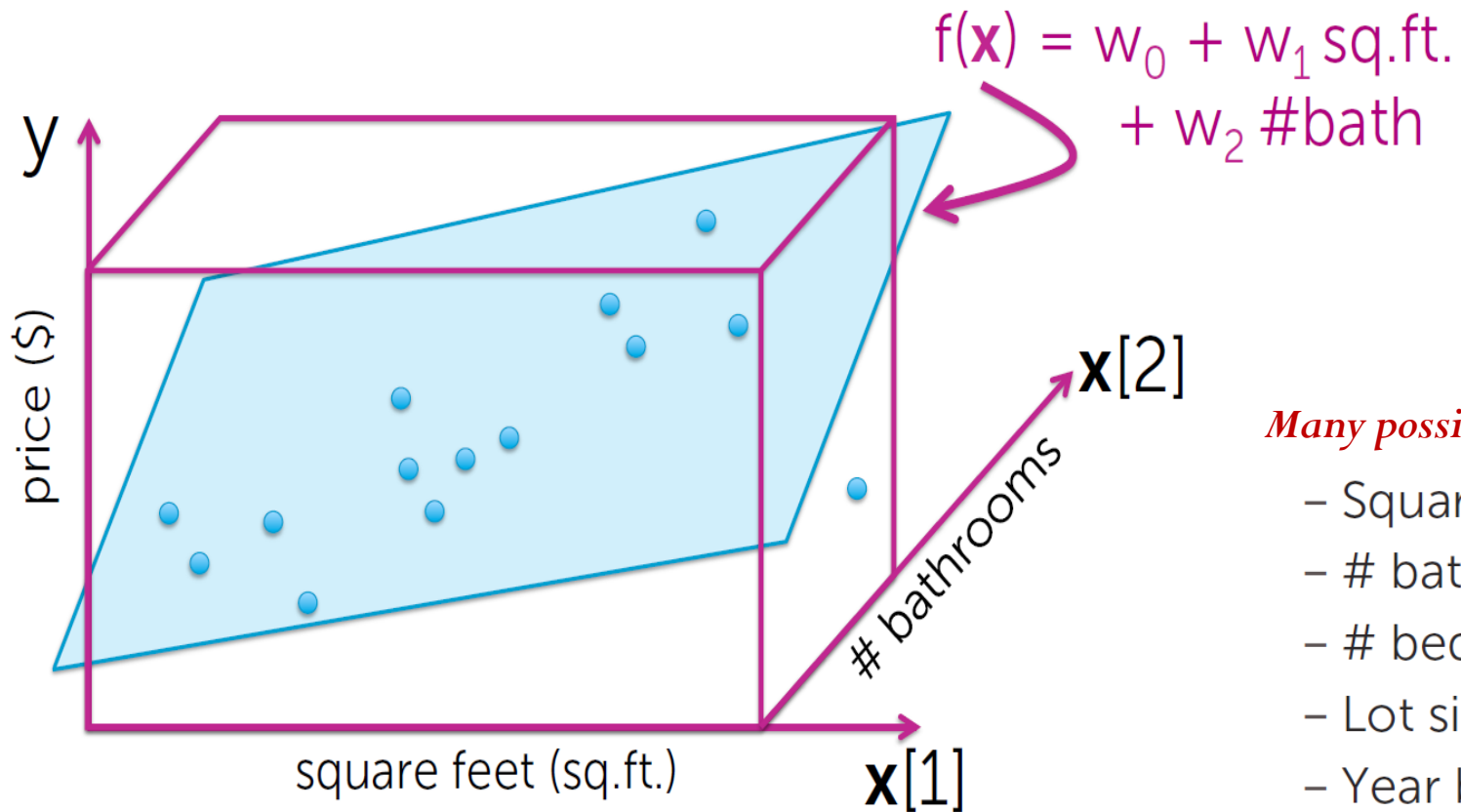
More realistic flow chart



Incorporating multiple inputs



Incorporating multiple inputs



Many possible inputs

- Square feet
- # bathrooms
- # bedrooms
- Lot size
- Year built
- ...

General notation

Output: y  scalar

Inputs: $\mathbf{x} = (\mathbf{x}[1], \mathbf{x}[2], \dots, \mathbf{x}[d])$

 d-dim vector

Notational conventions:

$\mathbf{x}[j]$ = j^{th} input (*scalar*)

$h_j(\mathbf{x})$ = j^{th} feature (*scalar*)

$\mathbf{x}_i$ = input of i^{th} data point (*vector*)

$\mathbf{x}_i[j]$ = j^{th} input of i^{th} data point (*scalar*)

Simple hyperplane

Model:

Noise term



$$y_i = w_0 + w_1 \mathbf{x}_i[1] + \dots + w_d \mathbf{x}_i[d] + \epsilon_i$$

feature 1 = 1

feature 2 = $\mathbf{x}[1]$... e.g., sq. ft.

feature 3 = $\mathbf{x}[2]$... e.g., #bath

...

feature $d+1$ = $\mathbf{x}[d]$... e.g., lot size

More generally: D-dimensional curve

Model:

$$y_i = w_0 h_0(\mathbf{x}_i) + w_1 h_1(\mathbf{x}_i) + \dots + w_D h_D(\mathbf{x}_i) + \varepsilon_i$$
$$= \sum_{j=0}^D w_j h_j(\mathbf{x}_i) + \varepsilon_i$$

More on notation

observations $(\mathbf{x}_i, y_i) : N$

inputs $\mathbf{x}[j] : d$

features $h_j(\mathbf{x}) : D$

feature 1 = $h_0(\mathbf{x})$... e.g., 1

feature 2 = $h_1(\mathbf{x})$... e.g., $\mathbf{x}[1] = \text{sq. ft.}$

feature 3 = $h_2(\mathbf{x})$... e.g., $\mathbf{x}[2] = \text{\#bath}$

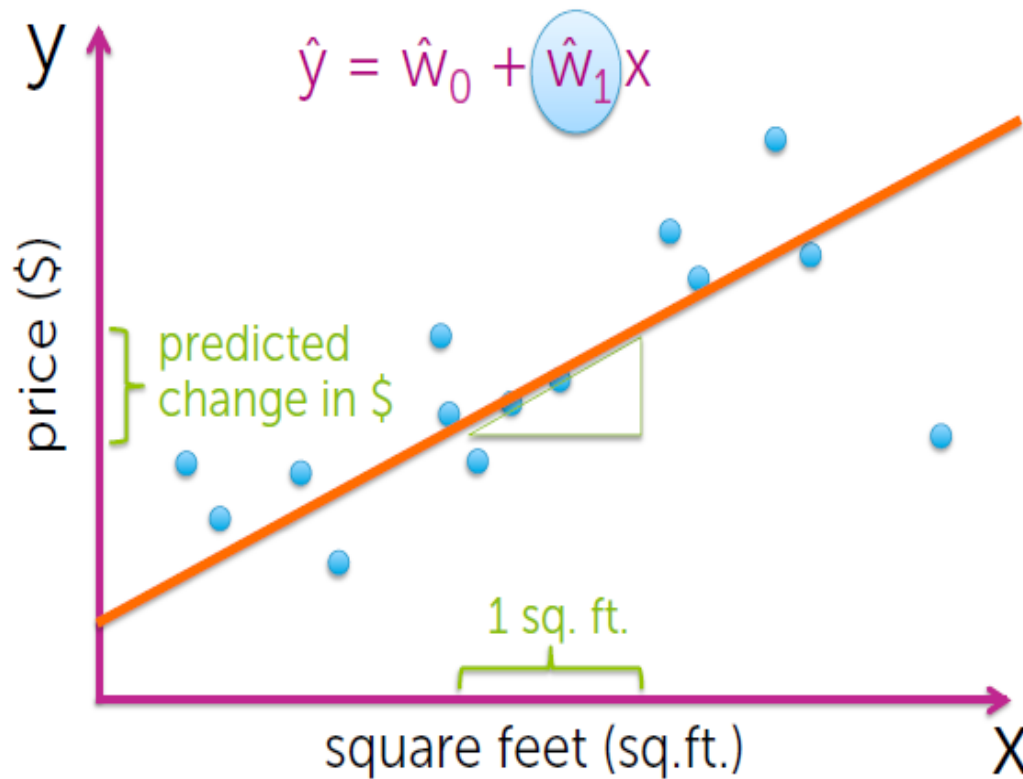
or, $\log(\mathbf{x}[7]) \mathbf{x}[2] = \log(\text{\#bed}) \times \text{\#bath}$

...

feature $D+1 = h_D(\mathbf{x})$... some other function of $\mathbf{x}[1], \dots, \mathbf{x}[d]$

Interpreting coefficients

Simple linear regression

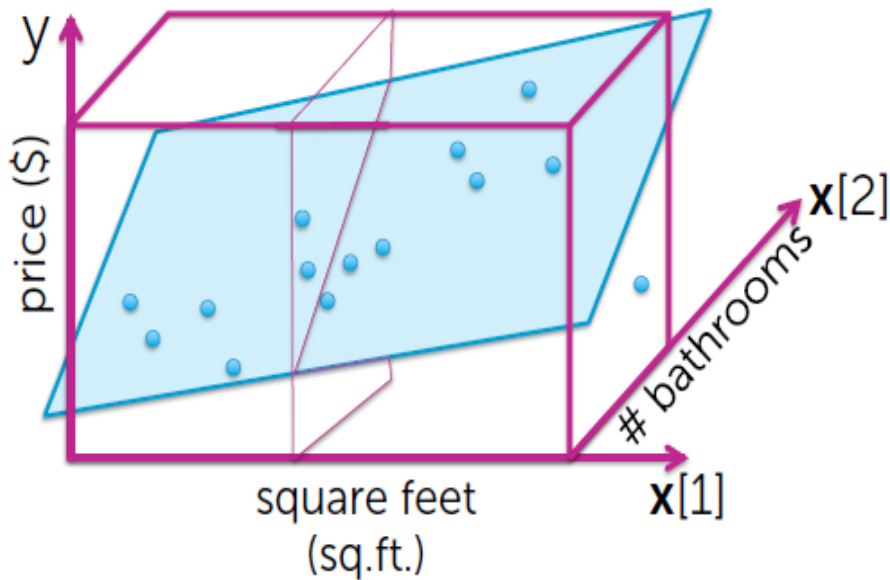


Interpreting coefficients

Two linear features

$$\hat{y} = \hat{w}_0 + \hat{w}_1 \mathbf{x}[1] + \hat{w}_2 \mathbf{x}[2]$$

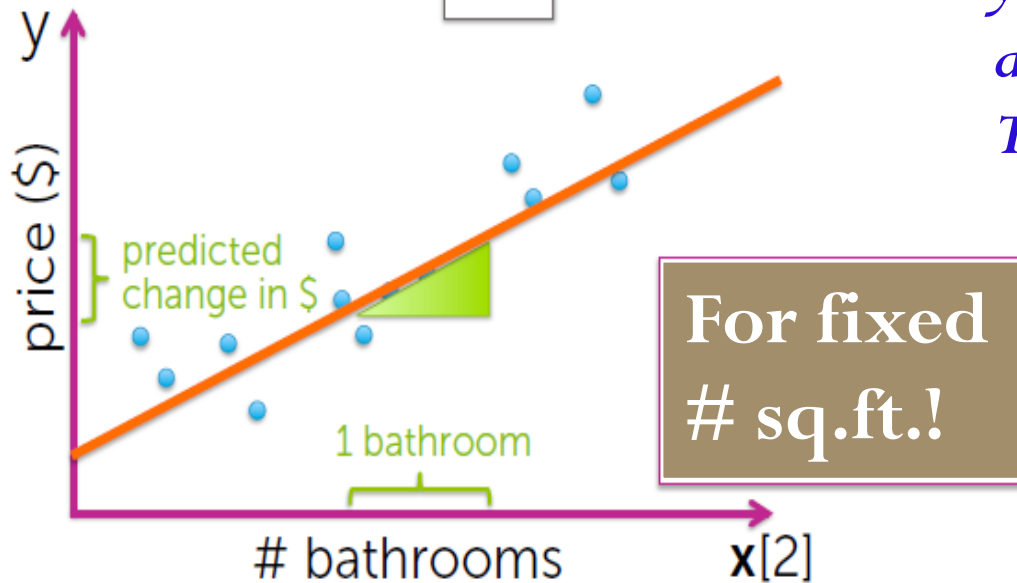
fix



Interpreting coefficients

Two linear features

$$\hat{y} = \hat{w}_0 + \hat{w}_1 \underset{\text{fix}}{\mathbf{x}[1]} + \hat{w}_2 \mathbf{x}[2]$$



But...

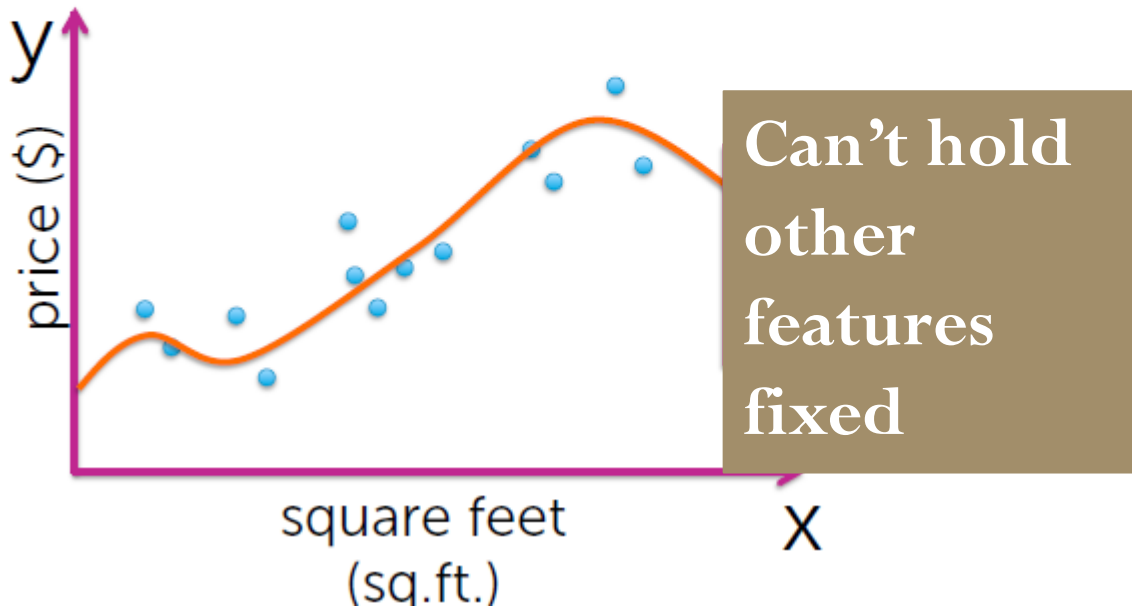
*increasing #bathrooms
for fixed #sq.ft will make
your bedrooms smaller
and smaller.*

Think about interpretation.

Interpreting coefficients

Polynomial regression

$$\hat{y} = \hat{w}_0 + \hat{w}_1 X + \dots + \hat{w}_j X^j + \dots + \hat{w}_p X^p$$

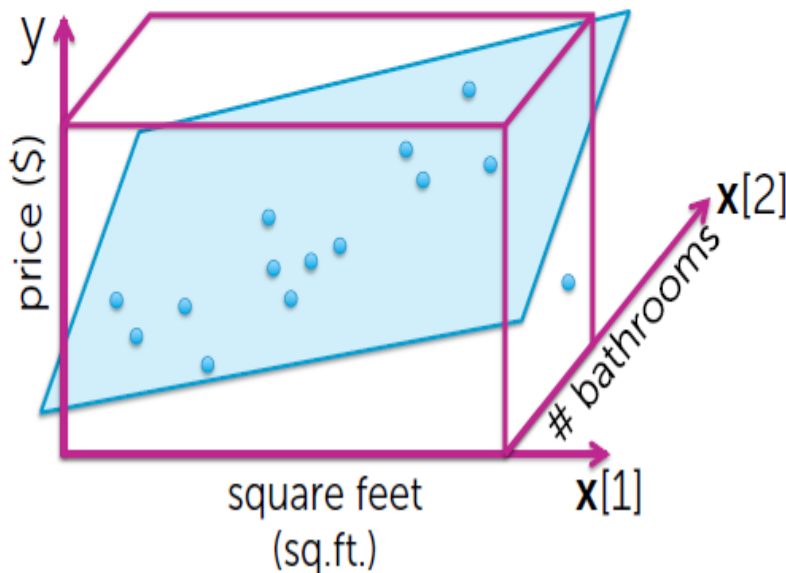


*Then ...
can't interpret
coefficients*

Interpreting coefficients

Multiple linear features

$$\hat{y} = \hat{w}_0 + \hat{w}_1 \underset{\text{fix}}{\mathbf{x}[1]} + \dots + \hat{w}_j \underset{\text{fix}}{\mathbf{x}[j]} + \dots + \hat{w}_d \underset{\text{fix}}{\mathbf{x}[d]}$$



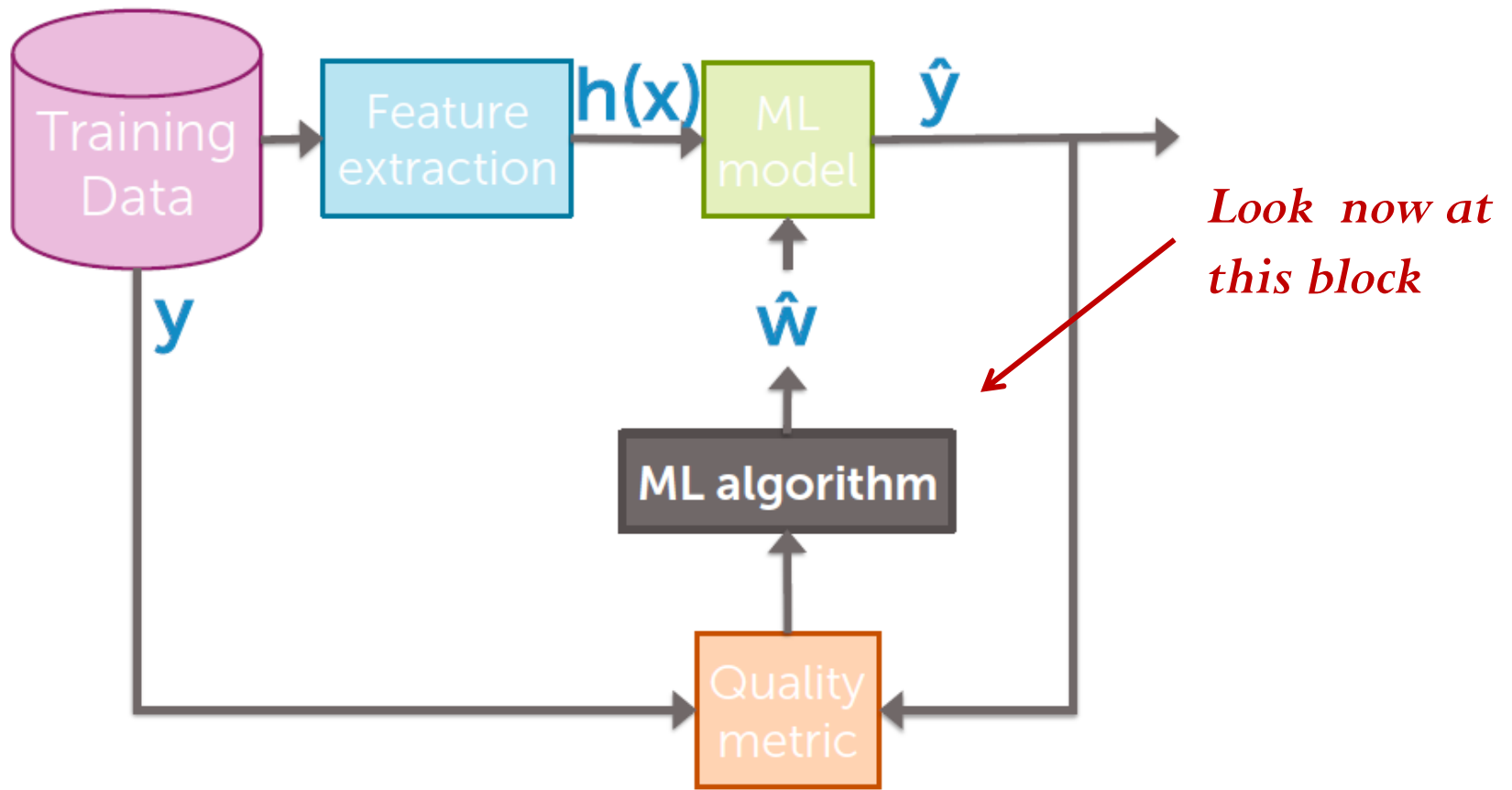
But...

increasing #bedrooms for fixed #sq.ft will make your bedrooms smaller and smaller.

You can end with negative coefficient. Might not be so if you removed #sq.ft from the model.

Think about interpretation in context of the model you put in.

Fitting in D-dimensions



Rewriting in vector notation

For observation i

$$y_i = \sum_{j=0}^D w_j h_j(\mathbf{x}_i) + \varepsilon_i$$

The diagram illustrates the conversion of the scalar equation into vector notation using visual representations:

- Left side:** A pink box labeled y_i is shown. Below it, a row of blue boxes represents the weight vector w^T , with labels $w_0, w_1, w_2, \dots, w_D$ underneath. A blue bracket above this row is labeled w^T . Below the row, the expression $= w_0 h_0(x_i) + w_1 h_1(x_i) + \dots + w_D h_D(x_i) + \varepsilon_i$ is written. An arrow points from the first term $w_0 h_0(x_i)$ to the word "scalar". At the bottom, the vector notation $= w^T h(x_i) + \varepsilon_i$ is shown.
- Middle:** A vertical column of green boxes represents the feature vector $h(x_i)$. A blue arrow points to this column with the label $h(x_i)$. The boxes are labeled $h_0(x_i), h_1(x_i), h_2(x_i), \dots, h_D(x_i)$ to their right.
- Right side:** An equals sign is followed by a row of green boxes representing the feature vector $h^T(x_i)$, with labels $h_0(x_i), h_1(x_i), \dots, h_D(x_i)$ underneath. A blue bracket above this row is labeled $h^T(x_i)$. To the right of this row is a vertical column of blue boxes representing the weight vector w , with labels $w_0, w_1, \dots, w_D$ to its right. A blue bracket to the right of this column is labeled w . The expression $= h_0(x_i)w_0 + h_1(x_i)w_1 + \dots + h_D(x_i)w_D + \varepsilon_i$ is written below the green boxes. At the bottom right, the vector notation $= h^T(x_i)w + \varepsilon_i$ is shown.

Rewriting in matrix notation

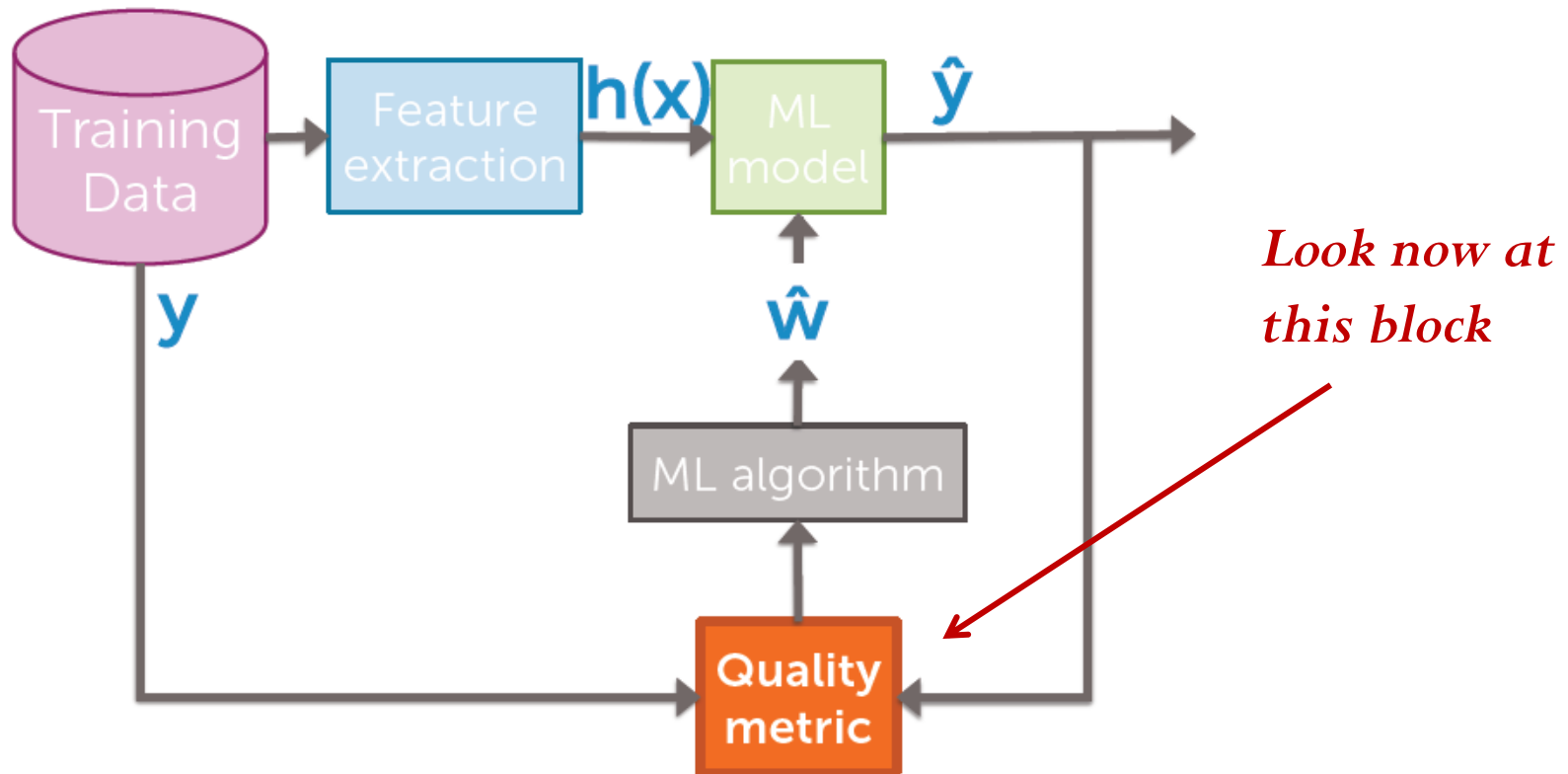
For all observations together

$$\begin{array}{c} \mathbf{y} \\ y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_n \end{array} = \begin{array}{c} \mathbf{H} \\ h_1(x_1) \ h_1(x_2) \ \dots \ h_1(x_n) \\ h_2(x_1) \ h_2(x_2) \ \dots \ h_2(x_n) \\ \vdots \\ h_D(x_1) \ h_D(x_2) \ \dots \ h_D(x_n) \end{array} \begin{array}{c} \mathbf{w} \\ w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_D \end{array} + \begin{array}{c} \mathbf{\epsilon} \\ \epsilon_1 \\ \epsilon_2 \\ \epsilon_3 \\ \vdots \\ \epsilon_n \end{array}$$

*Here is our
ML algorithm*

$$\Rightarrow \boxed{\mathbf{y} = \mathbf{H}\mathbf{w} + \mathbf{\epsilon}}$$

Fitting in D-dimensions



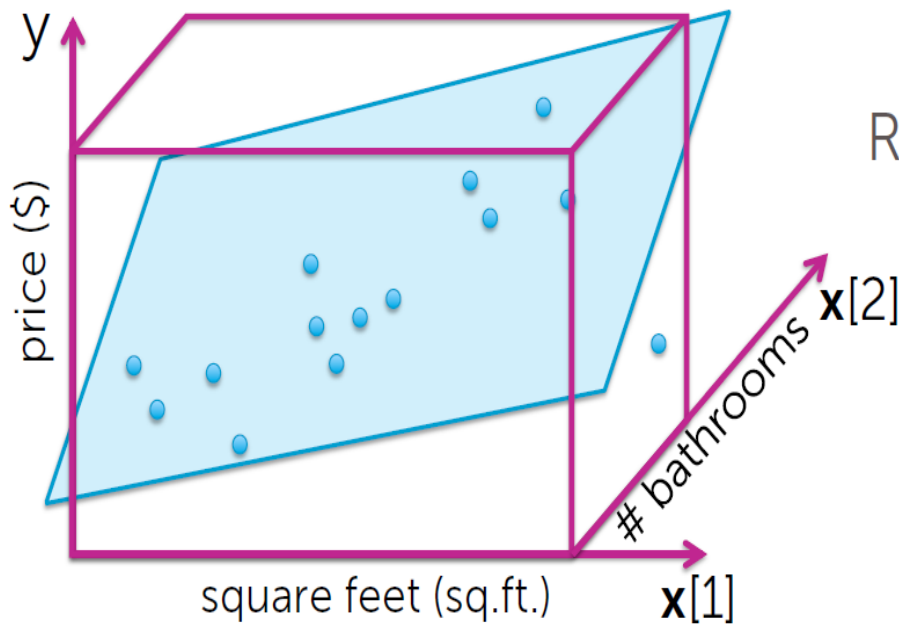
46

©2015 Emily Fox & Carlos Guestrin

Machine Learning Specialization

Cost function in D-dimension

RSS in vector notation



$$\text{RSS}(\underline{w}) = \sum_{i=1}^N (y_i - \underbrace{h^T(x_i) w}_{\hat{y}_i(w)})^2$$

Diagram illustrating the vector notation for the hypothesis function $\hat{y}_i$:

$\hat{y}_i =$ $h^T(x_i)$ w

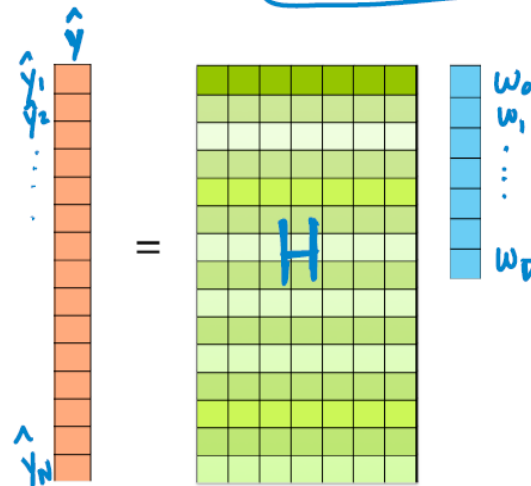
The hypothesis function $h^T(x_i)$ is represented as a row vector of green boxes, with labels $h_0(x_i), h_1(x_i), \dots, h_p(x_i)$ below them. The weight vector w is represented as a column vector of blue boxes, with labels $w_0, w_1, w_2, \dots, w_p$ to its right.

Cost function in D-dimension

RSS in matrix notation

$$\text{RSS}(\mathbf{w}) = \sum_{i=1}^N (y_i - h(\mathbf{x}_i)^T \mathbf{w})^2$$
$$= (\mathbf{y} - \mathbf{H}\mathbf{w})^T (\mathbf{y} - \mathbf{H}\mathbf{w})$$

Why? (part 1)



$$\hat{\mathbf{y}} = \mathbf{H}\mathbf{w}$$

$$(\mathbf{y} - \mathbf{H}\mathbf{w}) = (\mathbf{y} - \hat{\mathbf{y}}) = \begin{bmatrix} \text{residual}_1 \\ \text{residual}_2 \\ \vdots \\ \text{residual}_N \end{bmatrix}$$

residual i
 $= y_i - \hat{y}_i$

Regression model for D-dimension

RSS in matrix notation

$$\begin{aligned} \text{RSS}(\mathbf{w}) &= \sum_{i=1}^N (y_i - h(\mathbf{x}_i)^T \mathbf{w})^2 \\ &= (\mathbf{y} - \mathbf{H}\mathbf{w})^T (\mathbf{y} - \mathbf{H}\mathbf{w}) \end{aligned}$$

Why? (part 2)

residual ₁	residual ₂	residual ₃	...	residual _N
residual ₁	residual ₂	residual ₃	...	residual _N

$$\begin{aligned} &= \begin{pmatrix} \text{residual}_1^2 + \text{residual}_2^2 + \dots + \text{residual}_N^2 \end{pmatrix} \\ &= \sum_{i=1}^N \text{residual}_i^2 \\ &\triangleq \text{RSS}(\mathbf{w}) \end{aligned}$$

Regression model for D-dimension

Gradient of RSS

$$\begin{aligned}\nabla_{\text{RSS}}(\mathbf{w}) &= \nabla [(\mathbf{y} - \mathbf{H}\mathbf{w})^T (\mathbf{y} - \mathbf{H}\mathbf{w})] \\ &= -2\mathbf{H}^T (\mathbf{y} - \mathbf{H}\mathbf{w})\end{aligned}$$

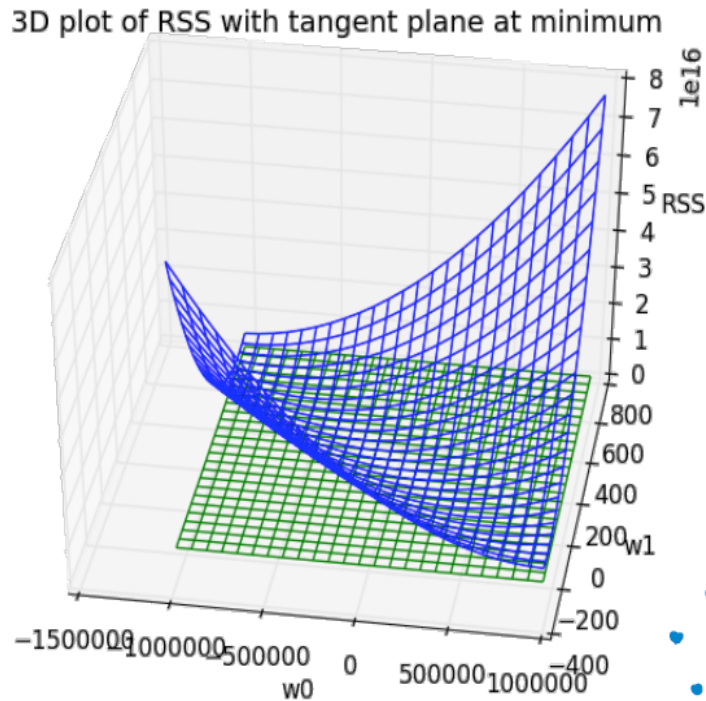
Why? By analogy to 1D case:

$$\begin{aligned}\frac{d}{dw} (y - hw)(y - hw) &= \frac{d}{dw} (y - hw)^2 = 2 \cdot (y - hw)' (-h) \\ &= -2h(y - hw)\end{aligned}$$

↑
scalars

Regression model for D-dimension

Approach 1: set gradient to zero



Closed form solution

$$\nabla \text{RSS}(\mathbf{w}) = -2\mathbf{H}^T(\mathbf{y} - \mathbf{H}\mathbf{w}) = 0$$

Solve for $\mathbf{w}$:

$$-\cancel{2}\mathbf{H}^T\mathbf{y} + \cancel{2}\mathbf{H}^T\mathbf{H}\hat{\mathbf{w}} = 0$$

$$\mathbf{H}^T\mathbf{H}\hat{\mathbf{w}} = \mathbf{H}^T\mathbf{y}$$

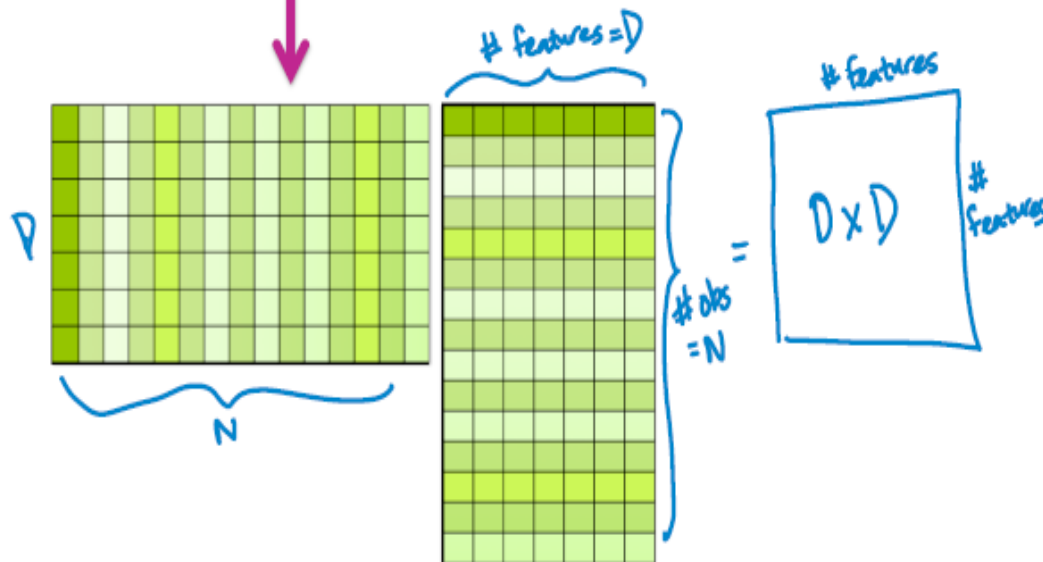
$$\underbrace{(\mathbf{H}^T\mathbf{H})^{-1}}_{\mathbf{I}} \mathbf{H}^T\mathbf{H}\hat{\mathbf{w}} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}$$

$$\hat{\mathbf{w}} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}$$

$$\begin{aligned} \bullet \mathbf{A}^{-1}\mathbf{A} &= \mathbf{I} \\ \bullet \mathbf{I}\mathbf{v} &= \mathbf{v} \\ \bullet \mathbf{I}\mathbf{v} &= \mathbf{v} \end{aligned}$$

Closed-form solution

$$\hat{\mathbf{W}} = (\mathbf{H}^T \mathbf{H})^{-1} \mathbf{H}^T \mathbf{y}$$



This matrix might not be invertible.

Invertible if:
In most cases is $N > D$
really, # of linearly ind. observations

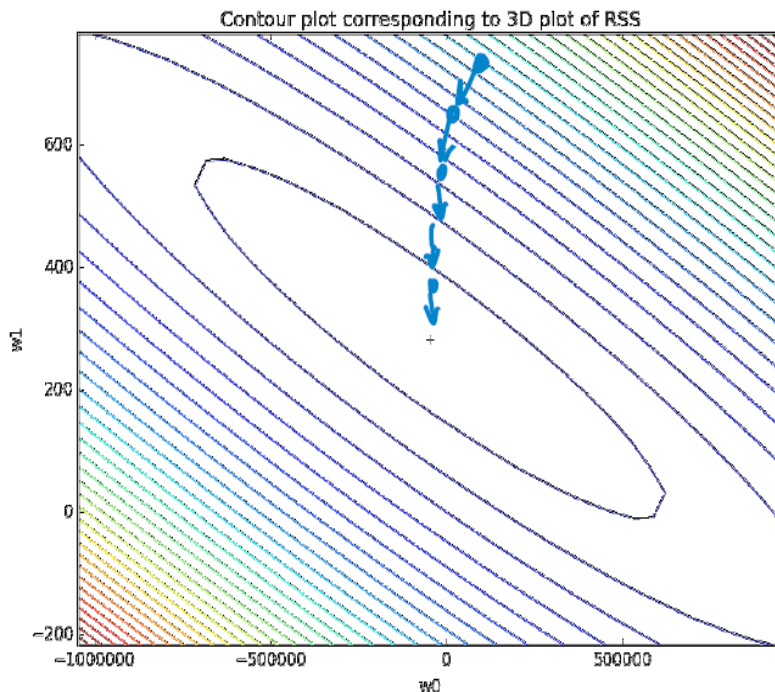
Complexity of inverse:

$$O(D^3)$$

This might not be CPU feasible.

Regression model for D-dimension

Approach 2: gradient descent



We initialise our solution somewhere and then ...

while not converged

$$\mathbf{w}^{(t+1)} \leftarrow \mathbf{w}^{(t)} - \underbrace{\eta \nabla \text{RSS}(\mathbf{w}^{(t)})}_{-2\mathbf{H}^T(\mathbf{y} - \mathbf{H}\mathbf{w})}$$

$$\leftarrow \mathbf{w}^{(t)} + 2\eta \mathbf{H}^T(\mathbf{y} - \underbrace{\mathbf{H}\mathbf{w}^{(t)}}_{\hat{\mathbf{y}}(\mathbf{w}^{(t)})})$$

Gradient descent

$$\text{RSS}(\mathbf{w}) = \sum_{i=1}^N (y_i - \mathbf{h}(\mathbf{x}_i)^T \mathbf{w})^2$$

$$= \sum_{i=1}^N (y_i - w_0 h_0(x_i) - w_1 h_1(x_i) - \dots - w_D h_D(x_i))^2$$

Partial with respect to w_j

$$\sum_{i=1}^N 2(y_i - w_0 h_0(x_i) - w_1 h_1(x_i) - \dots - w_D h_D(x_i)) \cdot (-h_j(x_i))$$

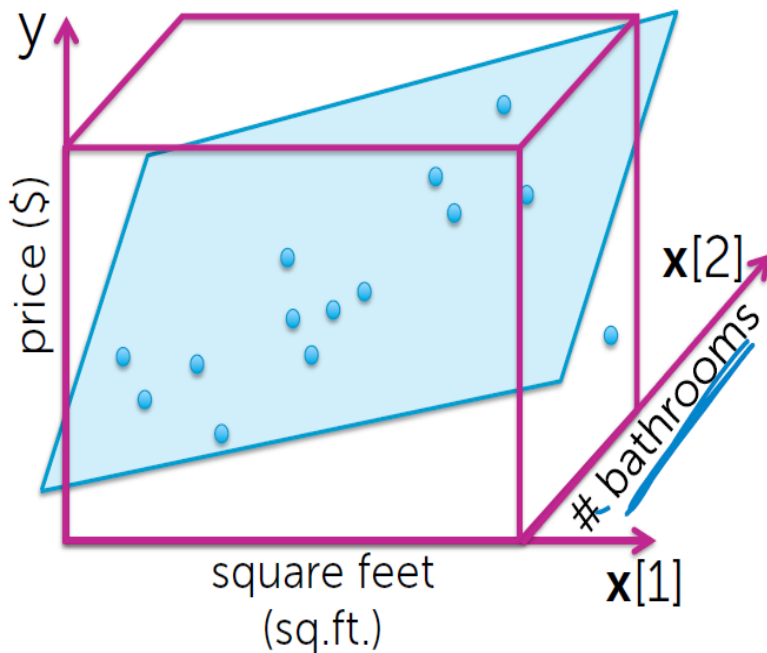
$$= -2 \sum_{i=1}^N h_j(x_i) (y_i - \mathbf{h}(\mathbf{x}_i)^T \mathbf{w})$$

Update to j^{th} feature weight:

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta \left(-2 \sum_{i=1}^N h_j(x_i) (y_i - \underbrace{\mathbf{h}^T(\mathbf{x}_i) \mathbf{w}^{(t)}}_{\hat{y}_i(\mathbf{w}^{(t)})}) \right)$$

Regression model for D-dimension

Interpreting elementwise



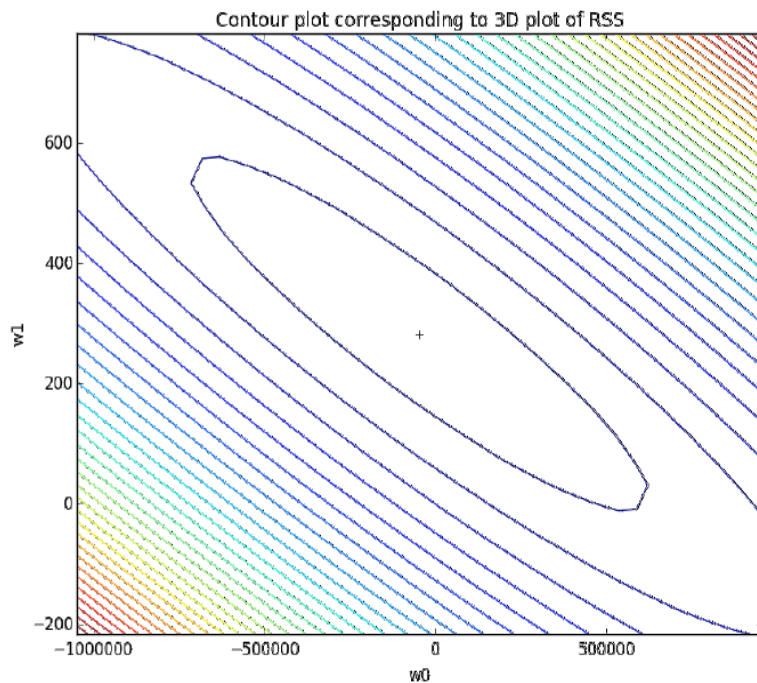
Update to j^{th} feature weight:

$$w_j^{(t+1)} \leftarrow w_j^{(t)} + 2\eta \sum_{i=1}^N h_j(\mathbf{x}_i) (y_i - \hat{y}_i(\mathbf{w}^{(t)}))$$

If underestimating impact of # bath ($\hat{w}_j^{(t)}$ is too small)
 then $(y_i - \hat{y}_i(\mathbf{w}^{(t)}))$ on average
 weighted by # bath will be positive
 $\Rightarrow w_j^{(t+1)} > w_j^{(t)}$ (increase)

Summary of gradient descent

Extremely useful algorithm in several applications



```
init  $\mathbf{w}^{(1)} = \mathbf{0}$  (or randomly, or smartly),  $t = 1$   
while  $\|\nabla \text{RSS}(\mathbf{w}^{(t)})\| > \epsilon$  ← tolerance  
    for  $j = 0, \dots, D$   
         $\text{partial}[j] = -2 \sum_{i=1}^N h_j(\mathbf{x}_i)(y_i - \hat{y}_i(\mathbf{w}^{(t)}))$   
         $\mathbf{w}_j^{(t+1)} \leftarrow \mathbf{w}_j^{(t)} - \eta \text{partial}[j]$   
     $t \leftarrow t + 1$ 
```

Handwritten notes:
- The norm in the while loop is defined as $\sqrt{\text{partial}[0]^2 + \dots + \text{partial}[D]^2}$.
- The gradient vector ∇ is highlighted in pink.

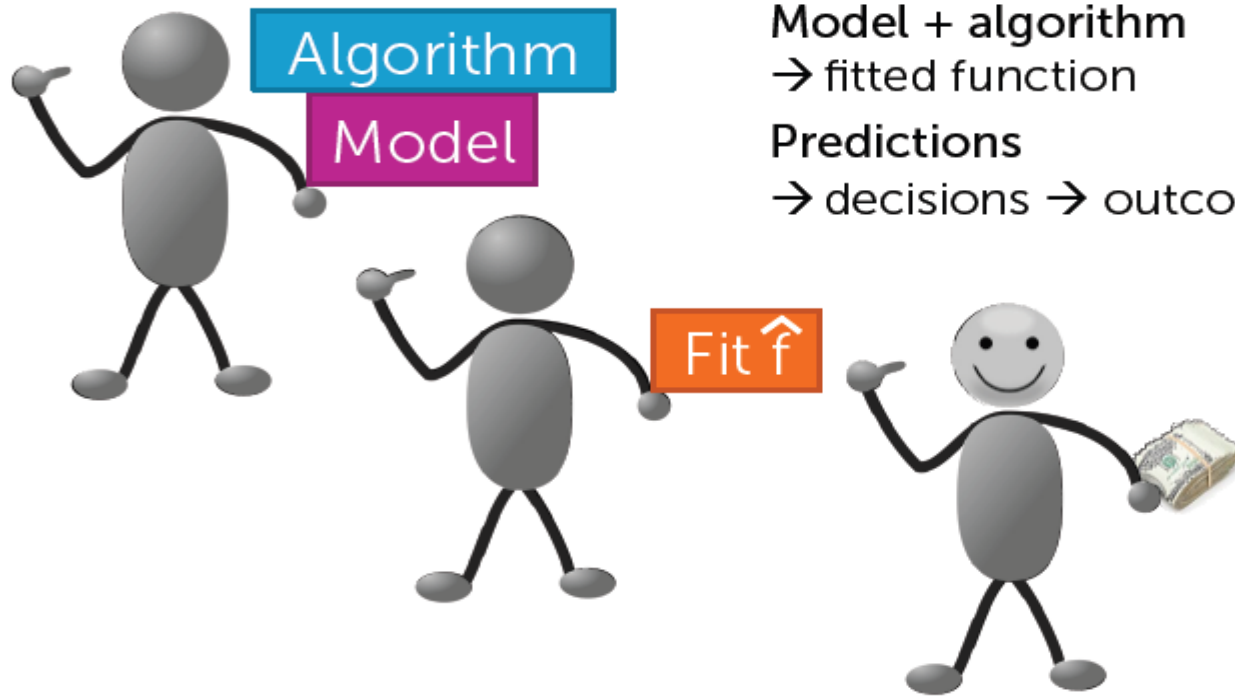
What you should know how to do now

- Describe polynomial regression
- Detrend a time series using trend and seasonal components
- Write a regression model using multiple inputs or features thereof
- Cast both polynomial regression and regression with multiple inputs as regression with multiple features
- Calculate a goodness-of-fit metric (e.g., RSS)
- Estimate model parameters of a general multiple regression model to minimize RSS:
 - In closed form
 - Using an iterative gradient descent algorithm
- Interpret the coefficients of a non-featurized multiple regression fit
- Exploit the estimated model to form predictions
- Explain applications of multiple regression beyond house price modeling

ACCESSING PERFORMANCE

Assessing performance

Make predictions, get \$, right??



Assessing performance

Or, how much am I losing?

Example: Lost \$ due to inaccurate listing price

- Too low → low offers
- Too high → few lookers + no/low offers

How much am I **losing** compared to perfection?

Perfect predictions: Loss = 0

My predictions: Loss = ???

Measuring loss

"Remember that all models are wrong; the practical question is how wrong do they have to be to not be useful." George Box, 1987.

Loss function:

Cost of using $\hat{w}$ at x
when y is true

$L(y, f_{\hat{w}}(\mathbf{x}))$

actual value $\hat{f}(\mathbf{x}) = \text{predicted value } \hat{y}$

Examples:

(assuming loss for underpredicting = overpredicting)

Absolute error: $L(y, f_{\hat{w}}(\mathbf{x})) = |y - f_{\hat{w}}(\mathbf{x})|$

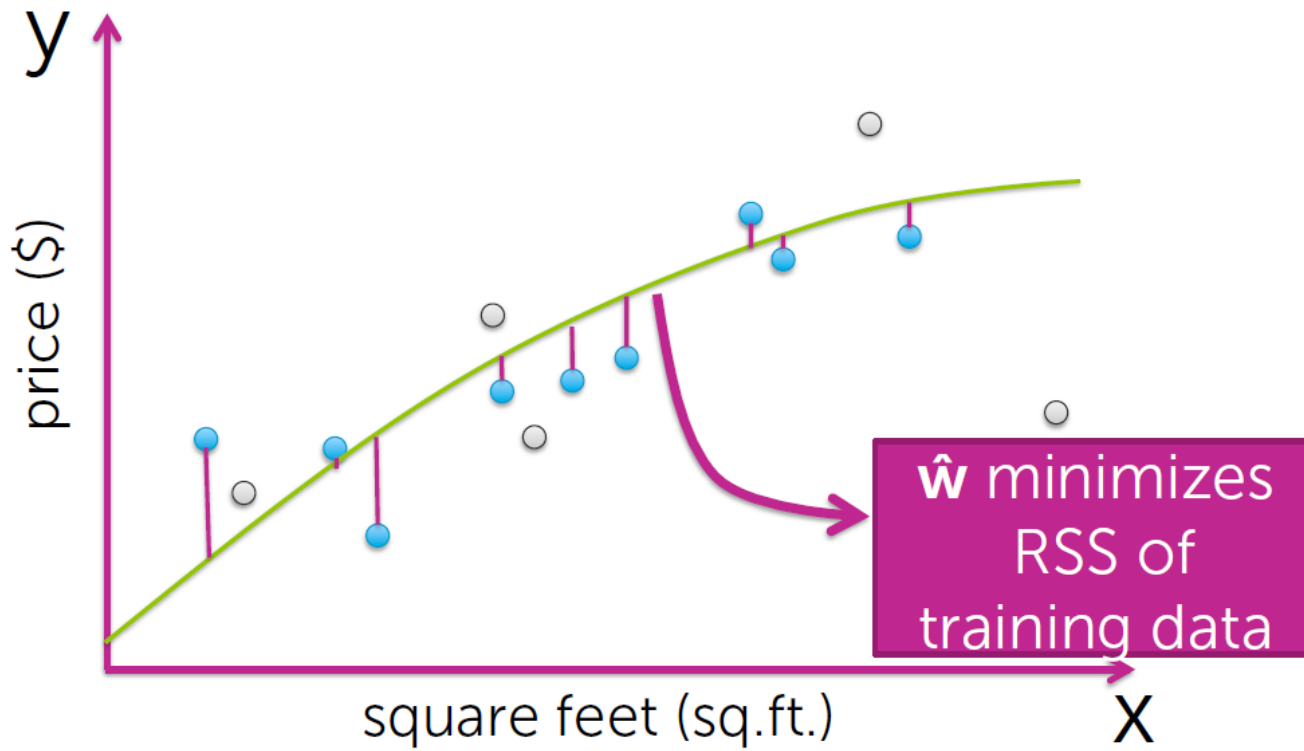
Squared error: $L(y, f_{\hat{w}}(\mathbf{x})) = (y - f_{\hat{w}}(\mathbf{x}))^2$

Symmetric loss functions



Accessing the loss

Use training data



Compute training error

1. Define a loss function $L(y, f_{\hat{\mathbf{w}}}(\mathbf{x}))$
 - E.g., squared error, absolute error,...

2. Training error

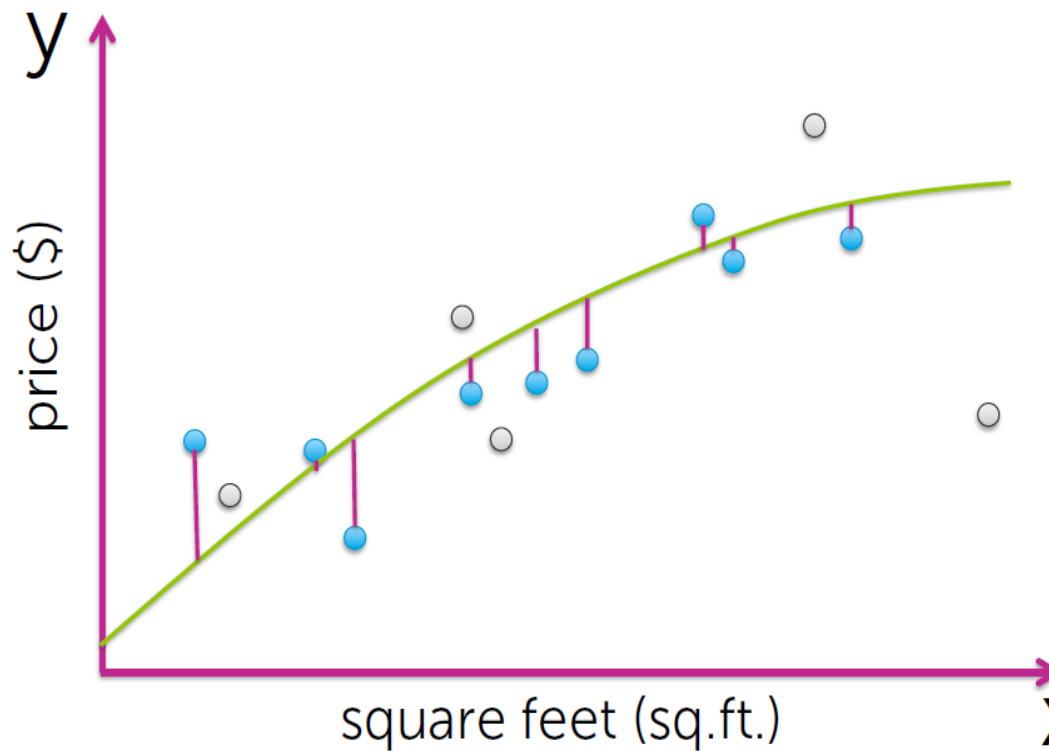
= avg. loss on houses in training set

$$= \frac{1}{N} \sum_{i=1}^N L(y_i, f_{\hat{\mathbf{w}}}(\mathbf{x}_i))$$

 fit using training data

Training error

Use **squared error loss** $(y - f_{\hat{\mathbf{w}}}(\mathbf{x}))^2$

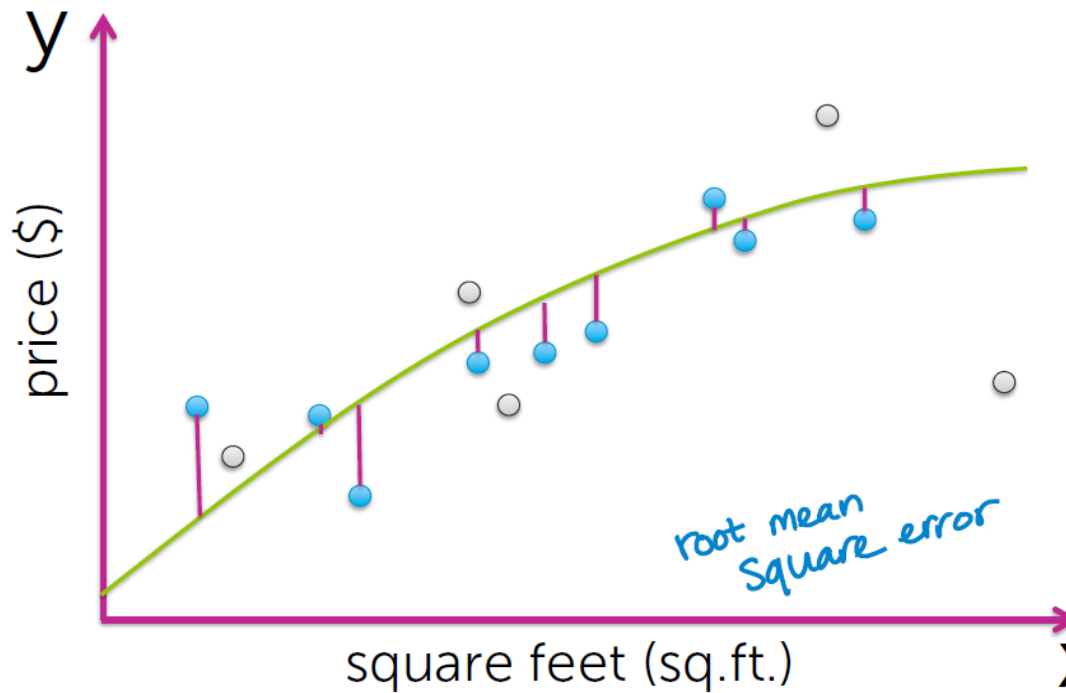


Convention is to take average here

Training error ($\hat{\mathbf{w}}$) = $1/N * [(\$_{\text{train } 1} - f_{\hat{\mathbf{w}}}(\text{sq.ft.}_{\text{train } 1}))^2 + (\$_{\text{train } 2} - f_{\hat{\mathbf{w}}}(\text{sq.ft.}_{\text{train } 2}))^2 + (\$_{\text{train } 3} - f_{\hat{\mathbf{w}}}(\text{sq.ft.}_{\text{train } 3}))^2 + \dots \text{include all training houses}]$

Training error

More intuitive is to take RMSE, same units as y



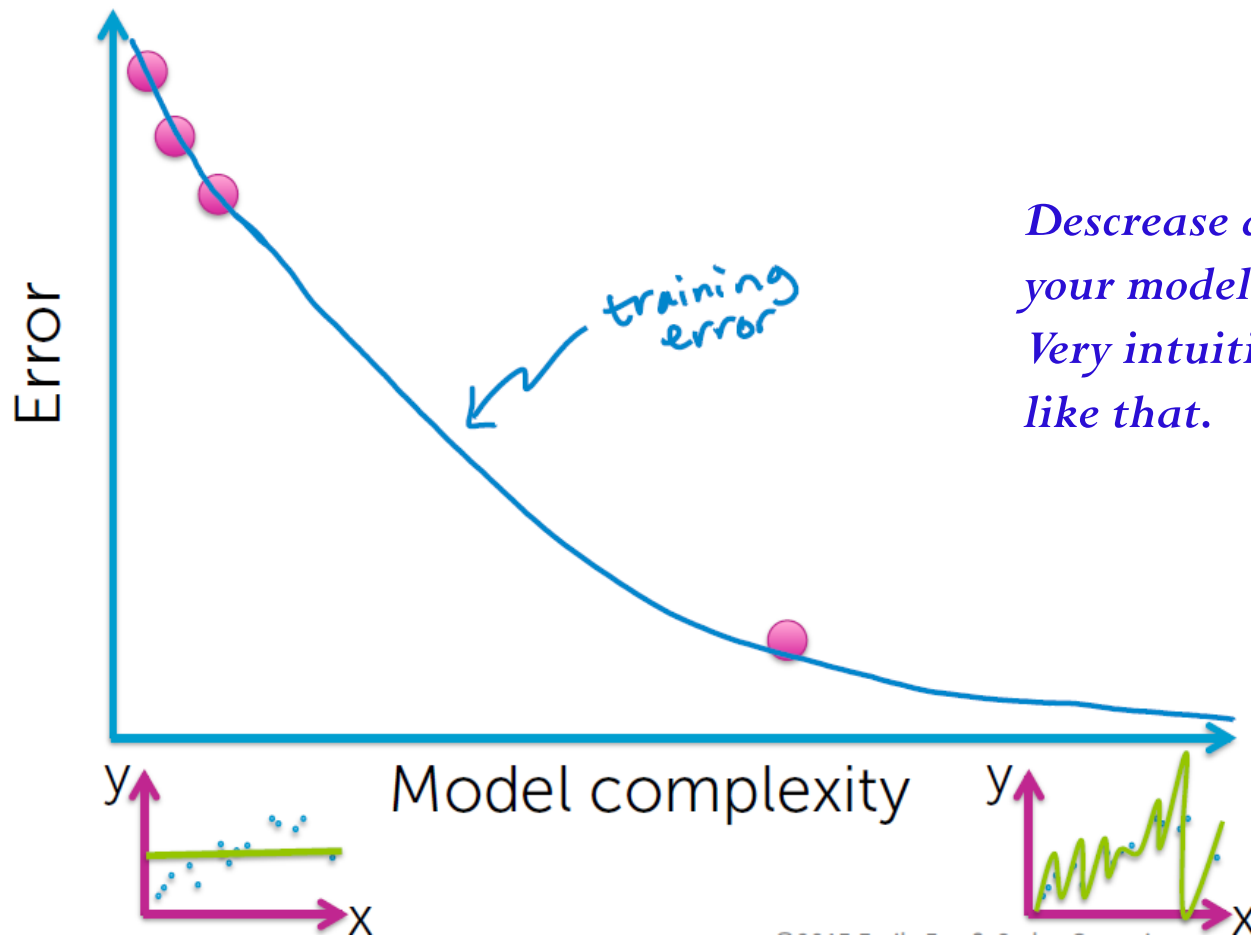
Training error ($\hat{\mathbf{w}}$) =

$$\frac{1}{N} \sum_{i=1}^N (y_i - f_{\hat{\mathbf{w}}}(\mathbf{x}_i))^2$$

RMSE =

$$\sqrt{\frac{1}{N} \sum_{i=1}^N (y_i - f_{\hat{\mathbf{w}}}(\mathbf{x}_i))^2}$$

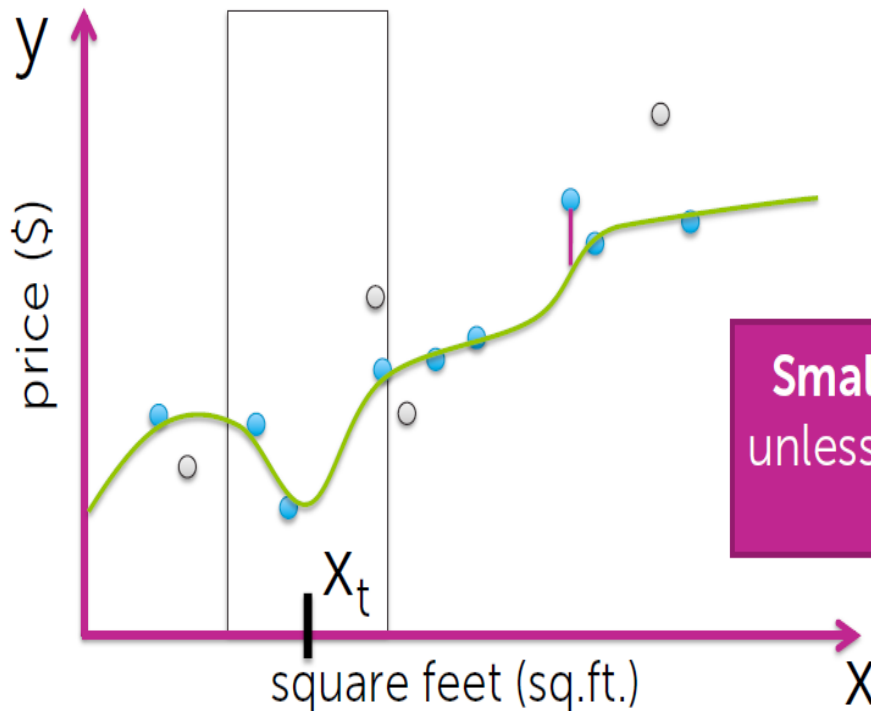
Training error vs. model complexity



*Decrease as you increase
your model complexity.
Very intuitive why it is
like that.*

Is training error a good measure?

Issue: Training error is overly optimistic
because $\hat{w}$ was fit to training data



Is there something particularly wrong about having x_t square feet ???

Small training error $\nRightarrow$ good predictions
unless training data includes everything you might ever see

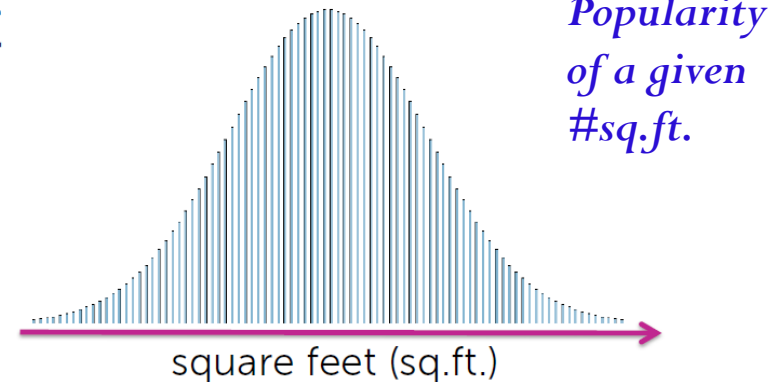
Generalisation (true) error

Really want estimate of loss
over all possible (, ) pairs



Distribution over house

In our neighborhood, houses of what # sq.ft. (🏠) are we likely to see?



For houses with a given # sq.ft. (🏠), what house prices \$ are we likely to see?



Generalisation error definition

Really want estimate of loss
over all possible (🏠, \$) pairs

Formally:

average over all possible
($\mathbf{x}, y$) pairs weighted by
how likely each is

$$\text{generalization error} = E_{\mathbf{x}, y} [L(y, f_{\hat{\mathbf{w}}}(\mathbf{x}))]$$

fit using training data

Generalisation error definition+

Really want estimate of loss over all possible (🏠, \$) pairs

Formally:

average over all possible
($\mathbf{x}, y$) pairs weighted by
how likely each is

$$\text{generalization error} = E_{\mathbf{x}, y} [L(y, f_{\hat{\mathbf{w}}}(\mathbf{x}))]$$

$$= \frac{\int L(y, f_{\hat{\mathbf{w}}}(\mathbf{x})) p(\mathbf{x}, y) d\mathbf{x} dy}{p(\mathbf{x}) p(y|\mathbf{x})}$$

fit using training data

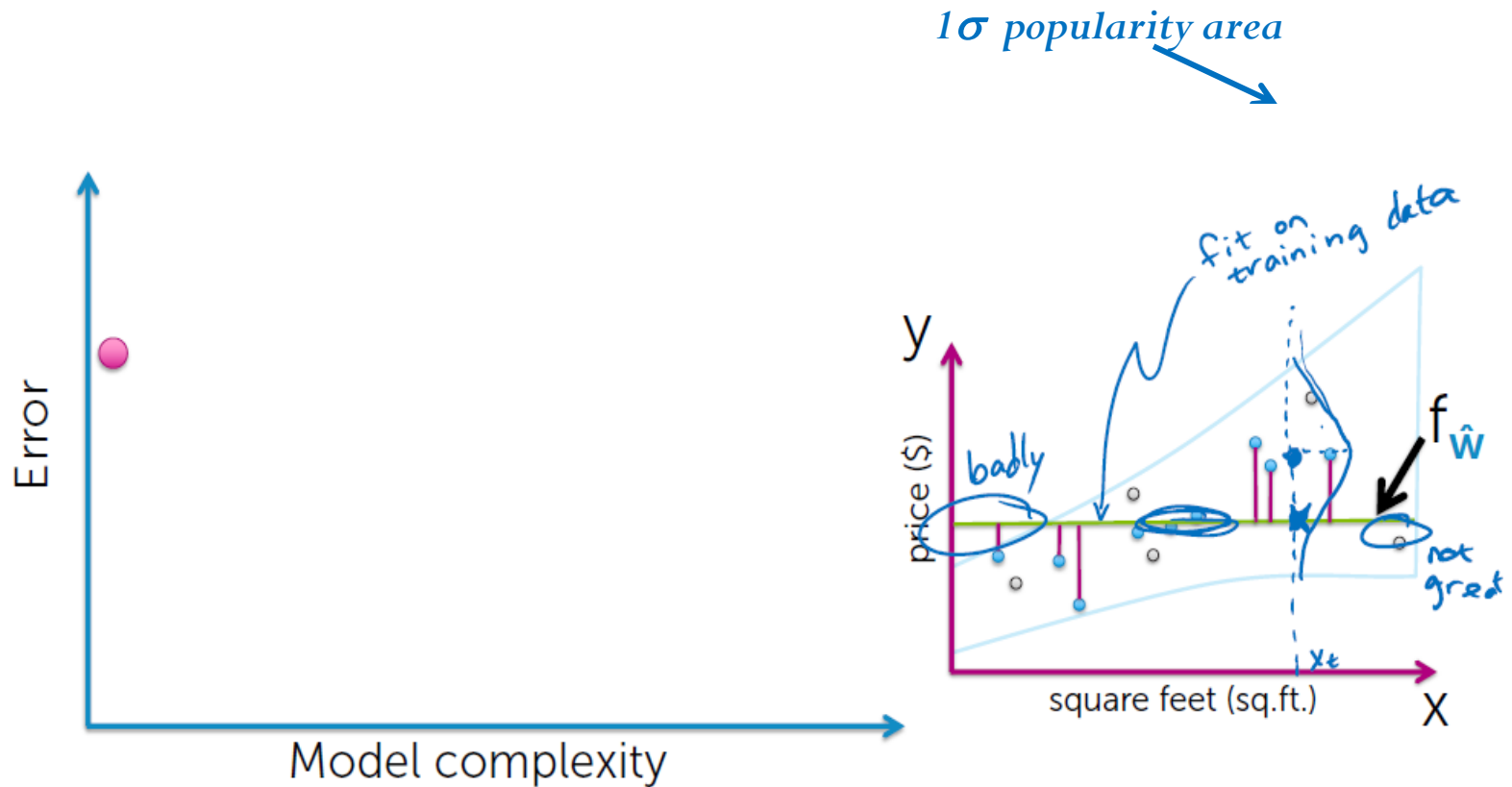
discrete r.v.

$$E[x] = \sum_i x_i p(x_i)$$

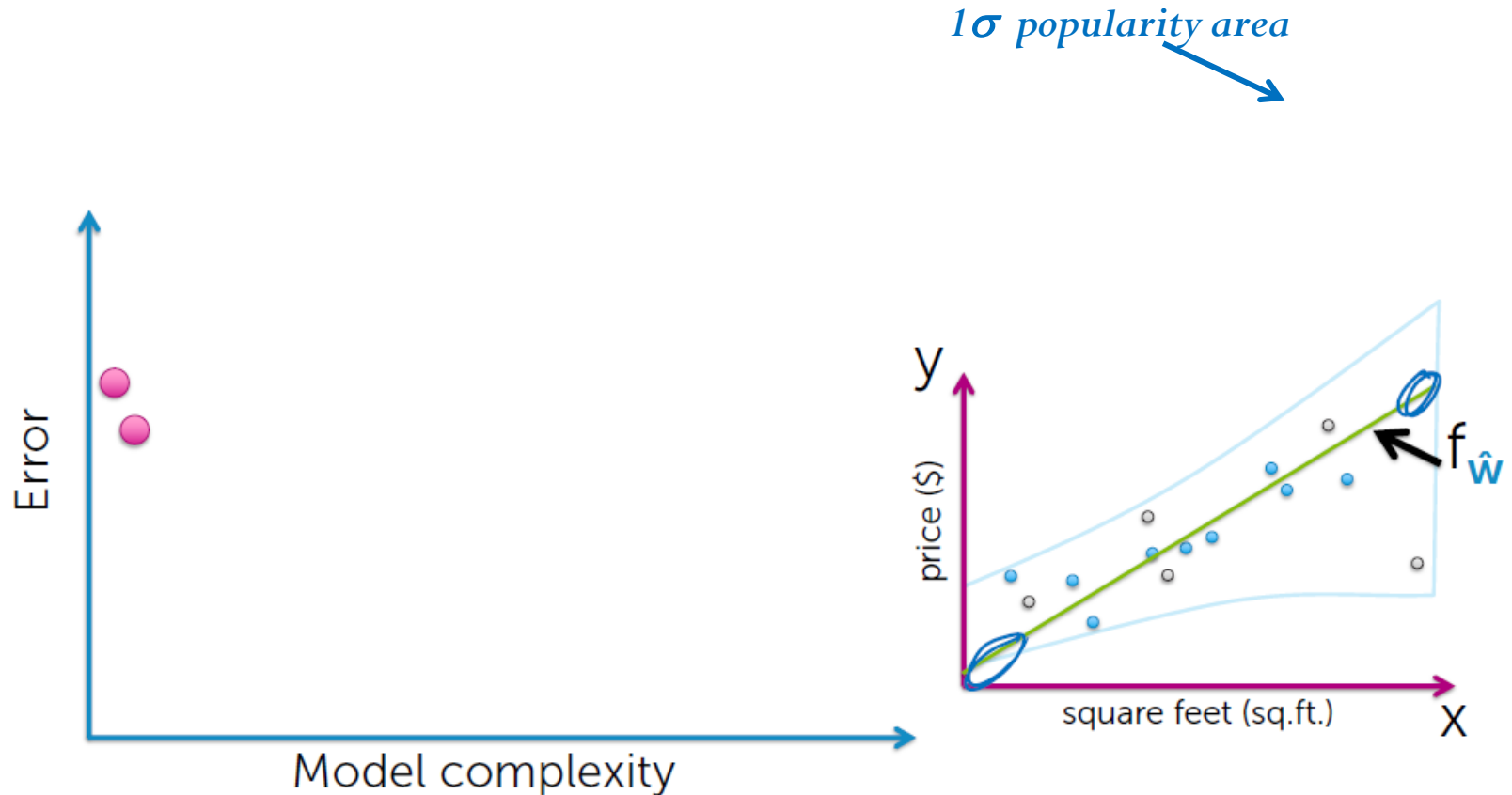
cont. r.v.

$$E[x] = \int x p(x) dx$$
$$E[f(x)] = \int f(x) p(x) dx$$

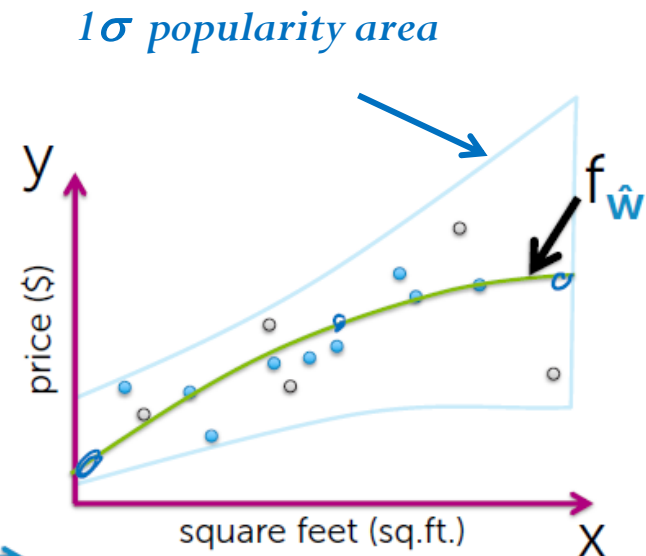
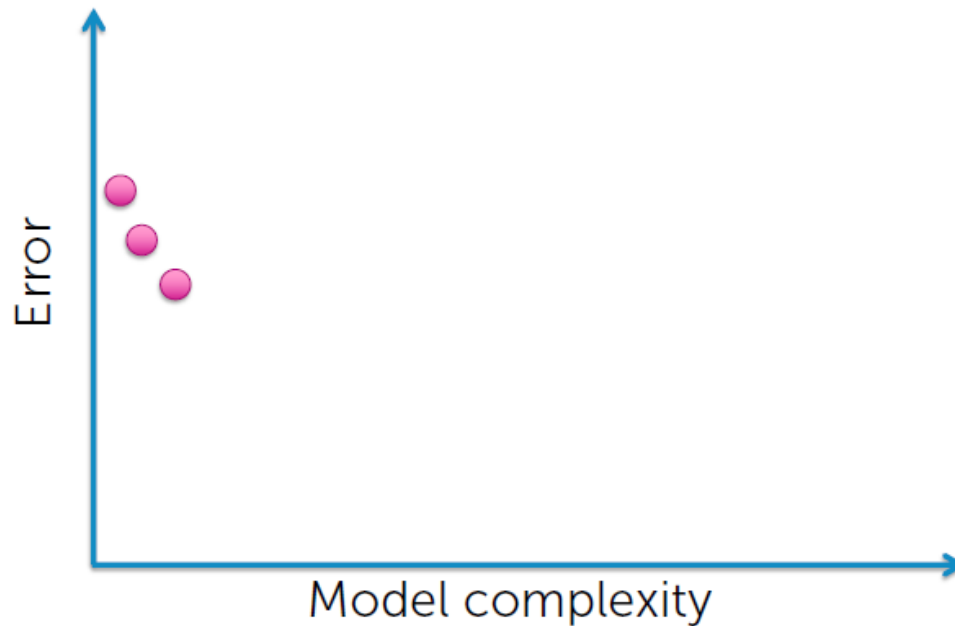
Generalisation error vs model complexity



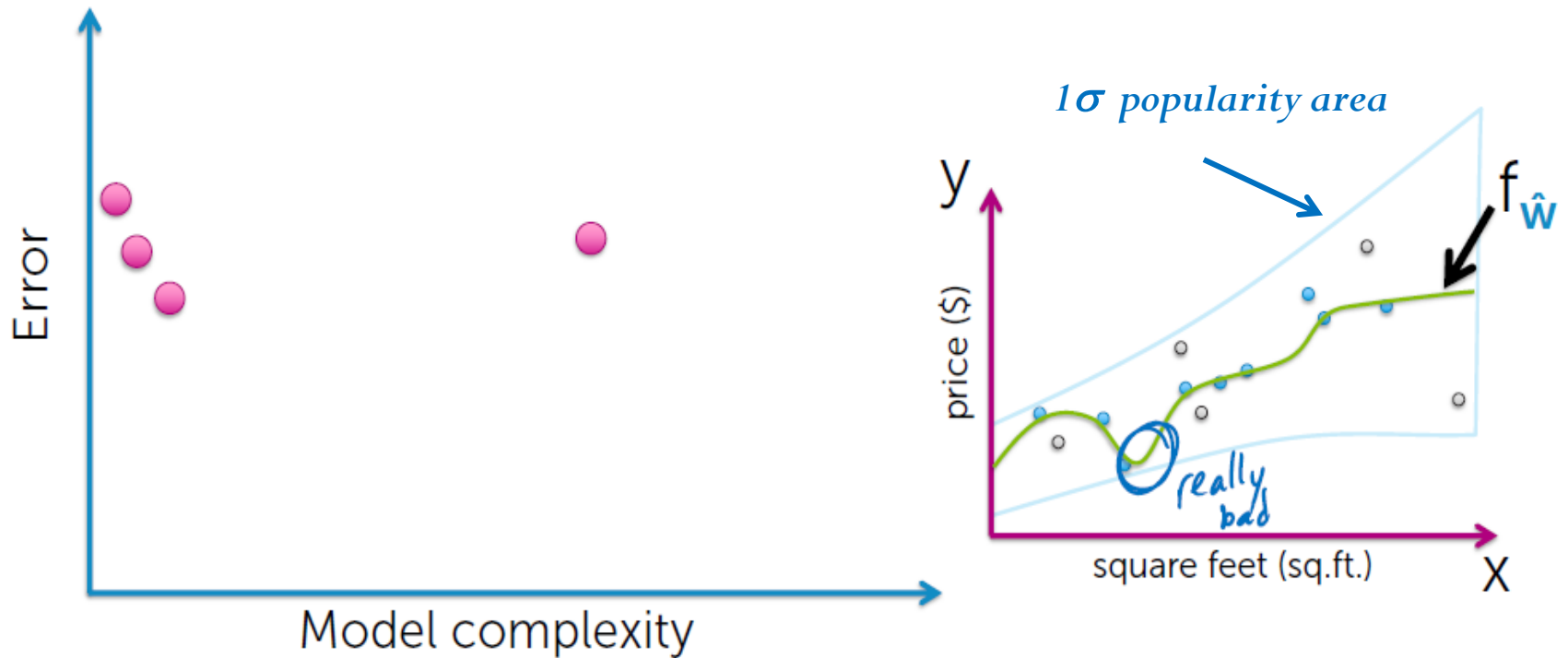
Generalisation error vs model complexity



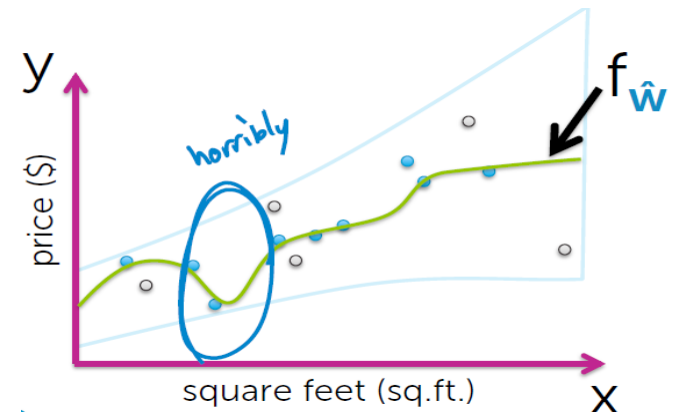
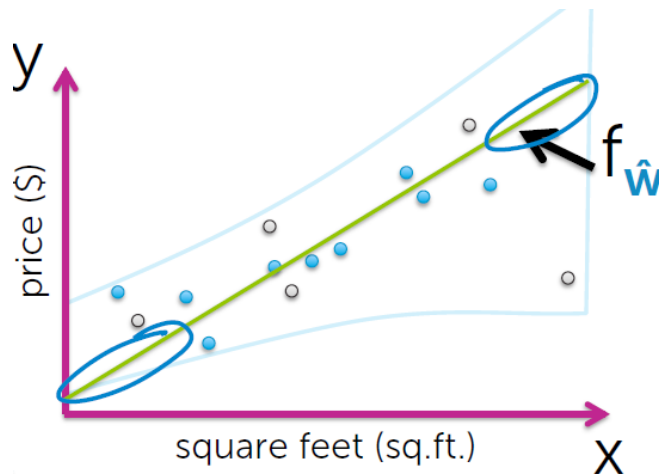
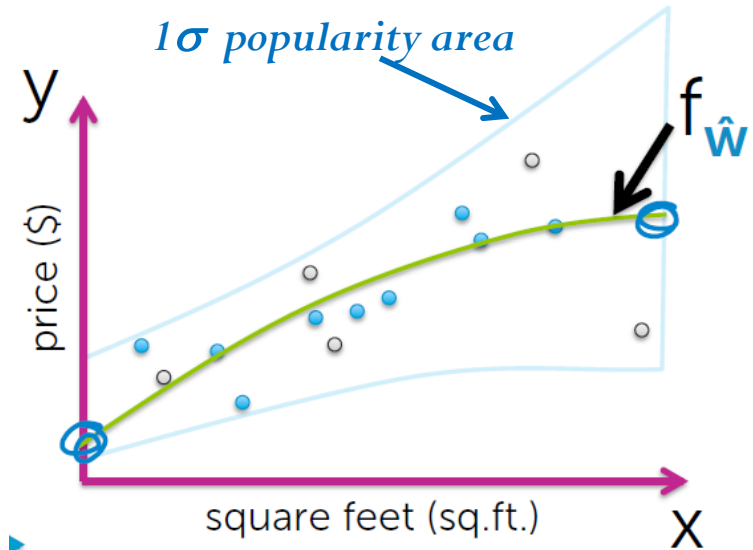
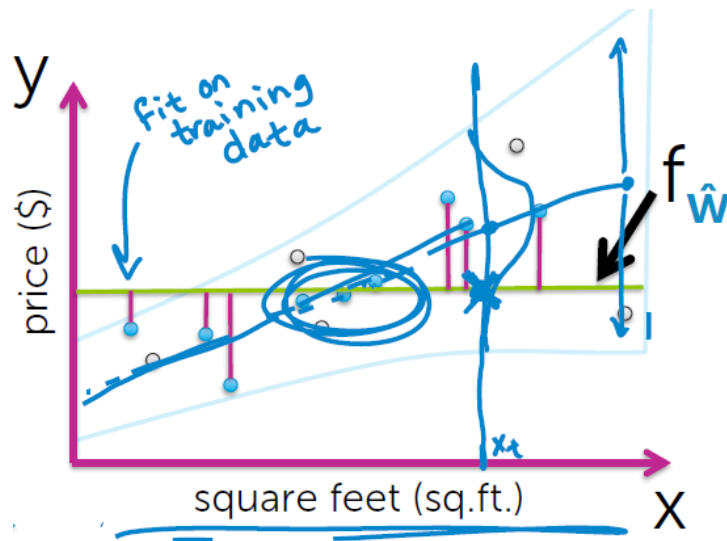
Generalisation error vs model complexity



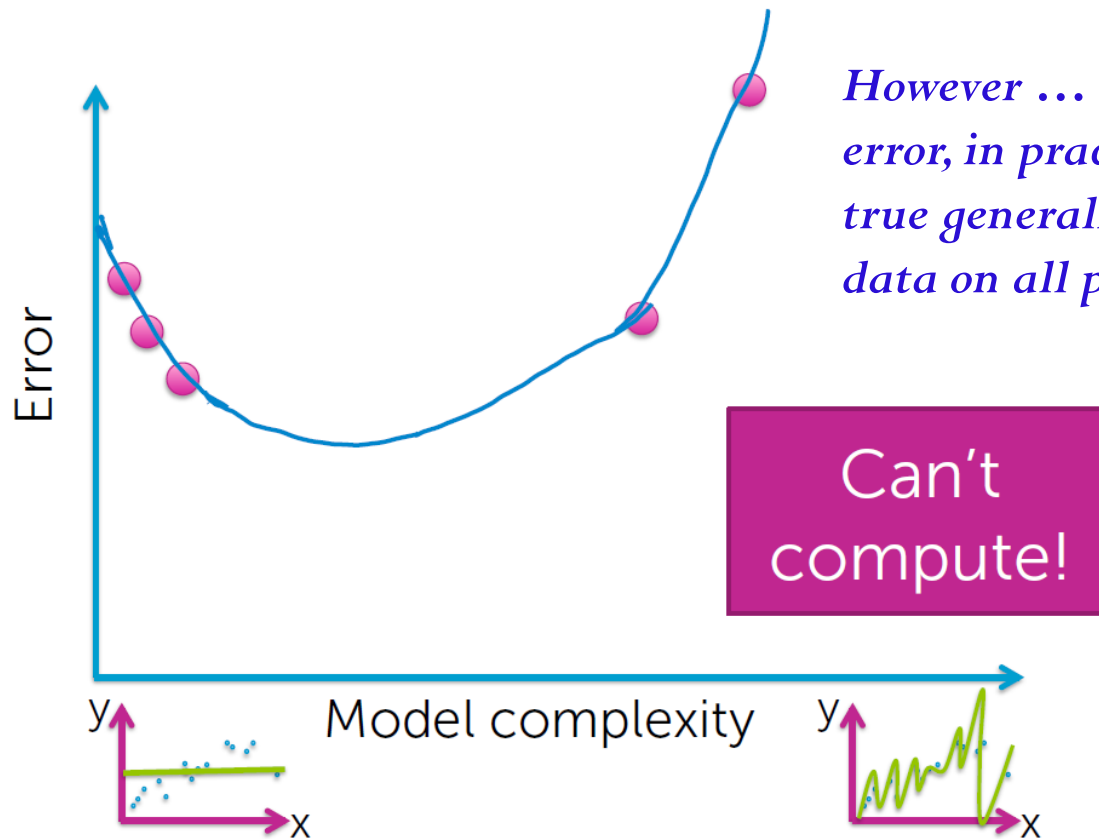
Generalisation error vs model complexity



Generalisation error (weighted with popularity) vs model complexity



Generalisation error vs model complexity



However ... in contrast to the training error, in practice we cannot really compute true generalisation error. We don't have data on all possible houses in the area.

Can't
compute!

Forming a test set

Hold out some (🏠, \$) that are *not* used for fitting the model



We want to approximate generalisation error.

Test set: proxy for „everything you might see”

Training set



Test set



Compute test error

Test error

= avg. loss on houses in test set

$$= \frac{1}{N_{test}} \sum_{i \text{ in test set}} L(y_i, f_{\hat{\mathbf{w}}}(\mathbf{x}_i))$$



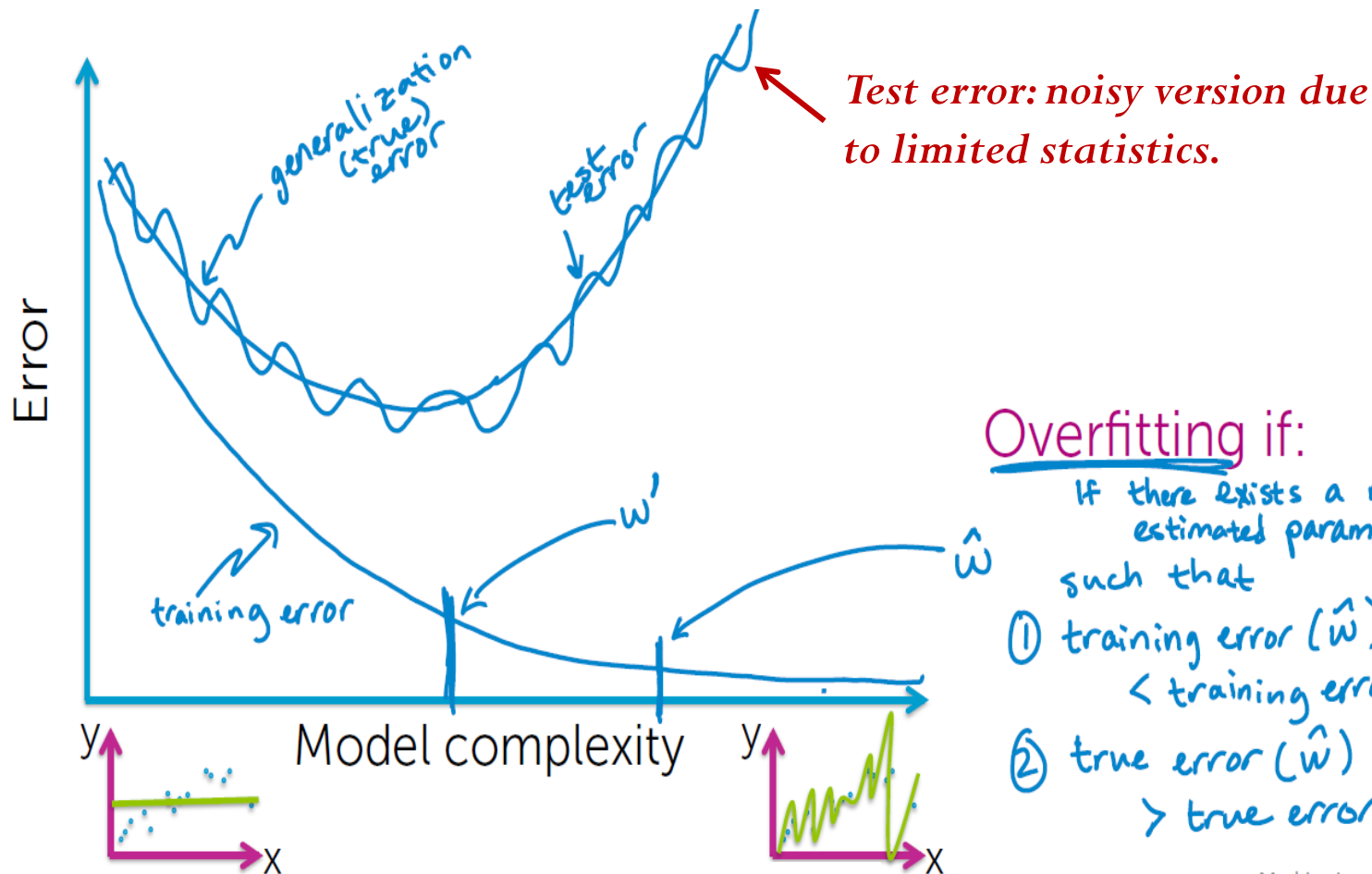
test points



fit using training data

**has never seen
test data!**

Training, true and test error vs. model complexity. Notion of overfitting.



Overfitting if:

If there exists a model with estimated params w' such that

- ① training error ($\hat{w}$) $<$ training error (w')
- ② true error ($\hat{w}$) $>$ true error (w')

Training/test splits



↑
Too few → $\hat{w}$ poorly estimated



↑
Too few → test error bad approximation of generalization error



Typically, just enough test points to form a reasonable estimate of generalization error

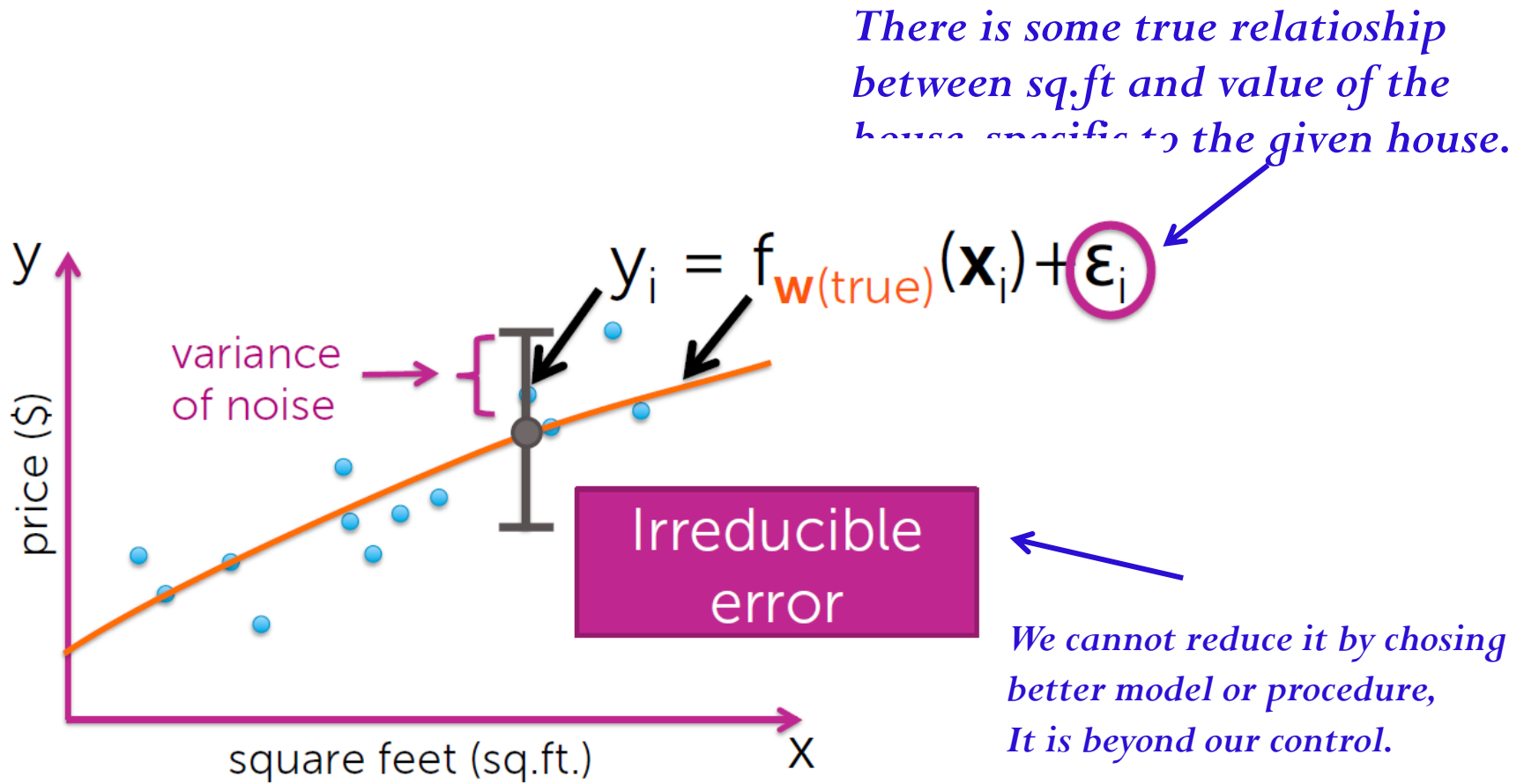
If this leaves too few for training, other methods like **cross validation** (will see later...)

Three sources of errors

In forming predictions, there are 3 sources of error:

1. Noise
2. Bias
3. Variance

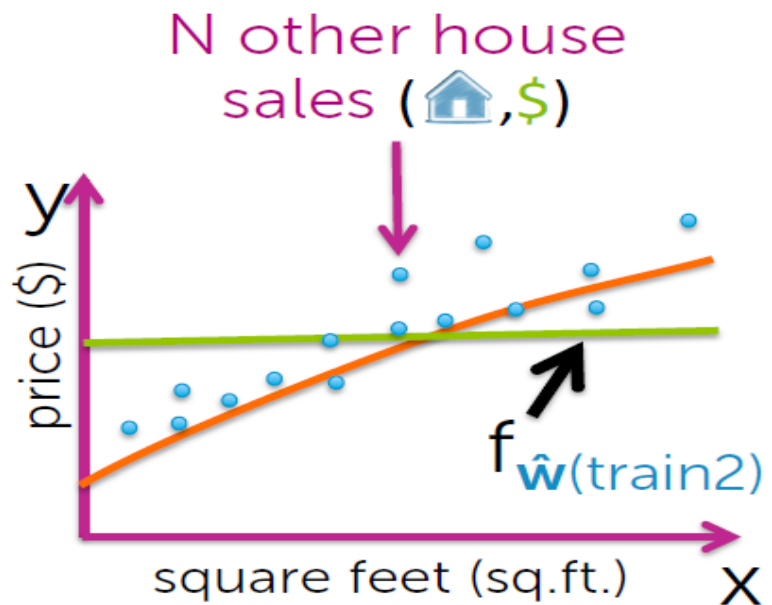
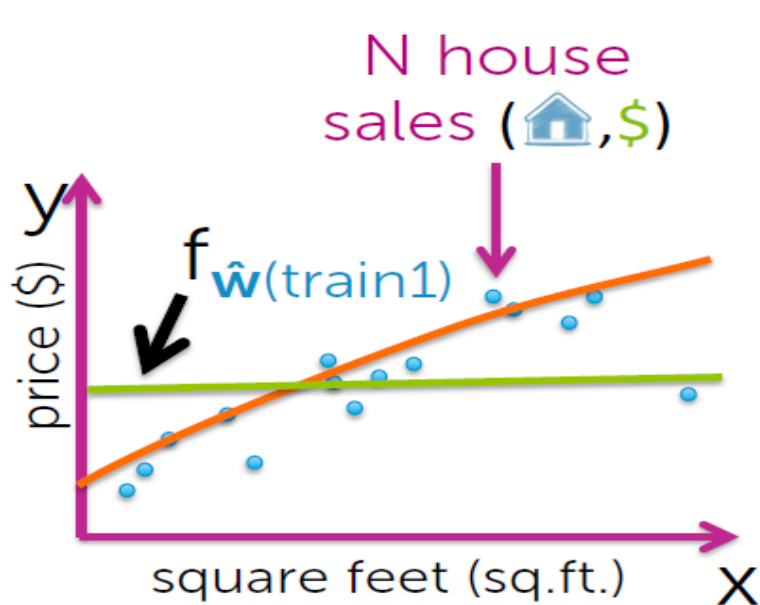
Data are inherently noisy



Bias contribution

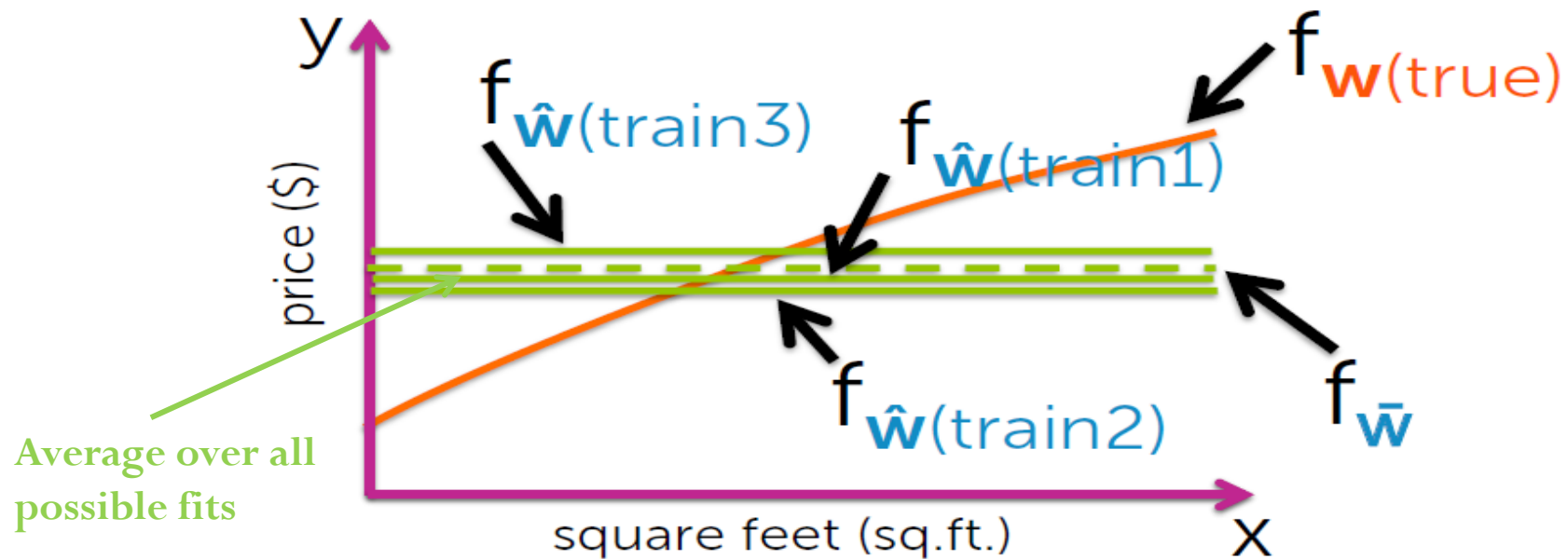
This contribution we can control.

Assume we fit a constant function



Bias contribution

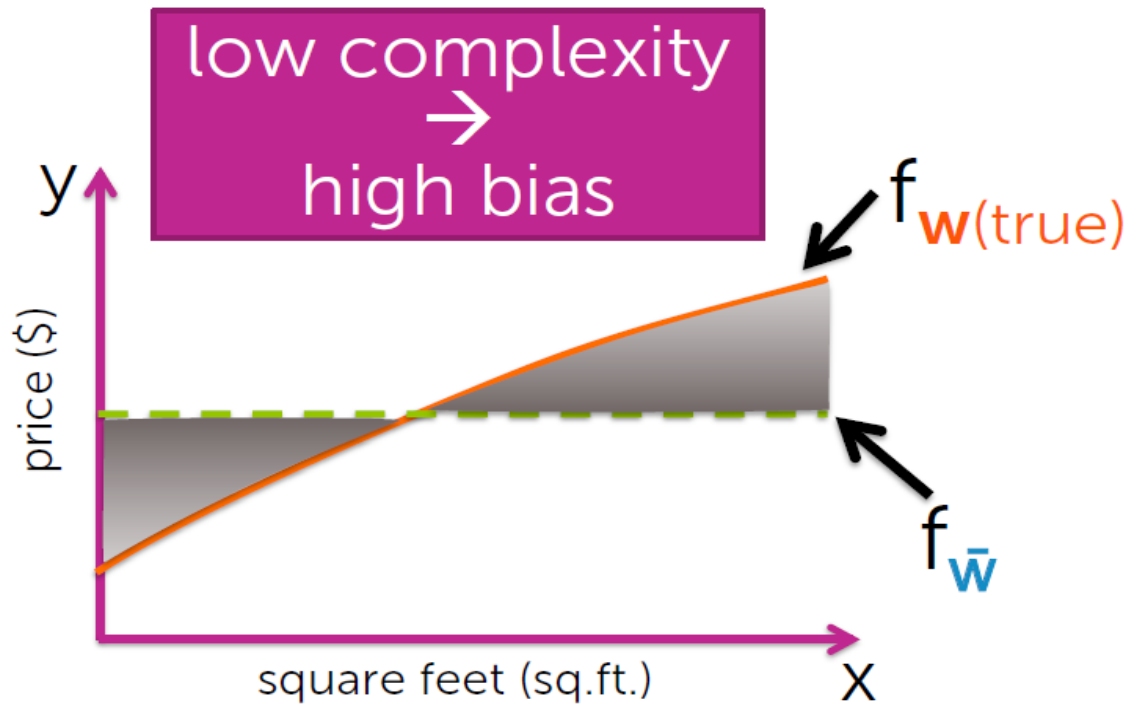
Over all possible size N training sets, what do I expect my fit to be?



Bias contribution

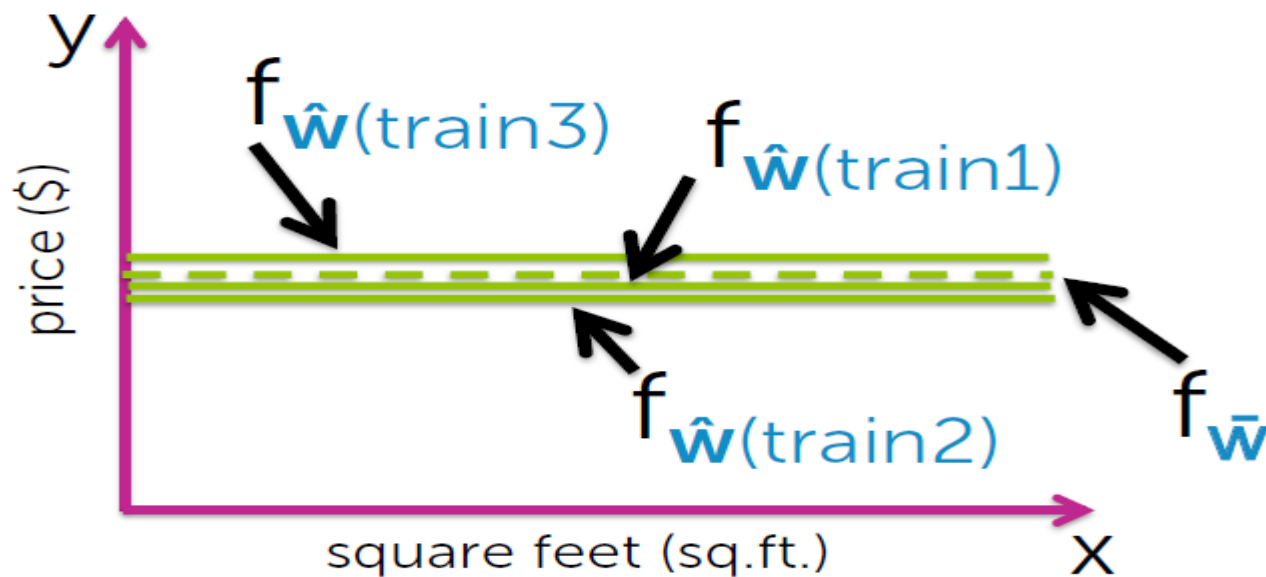
$$\text{Bias}(\mathbf{x}) = f_{\mathbf{w}(\text{true})}(\mathbf{x}) - f_{\bar{\mathbf{w}}}(\mathbf{x})$$

Is our approach flexible enough to capture $f_{\mathbf{w}(\text{true})}$?
If not, error in predictions.



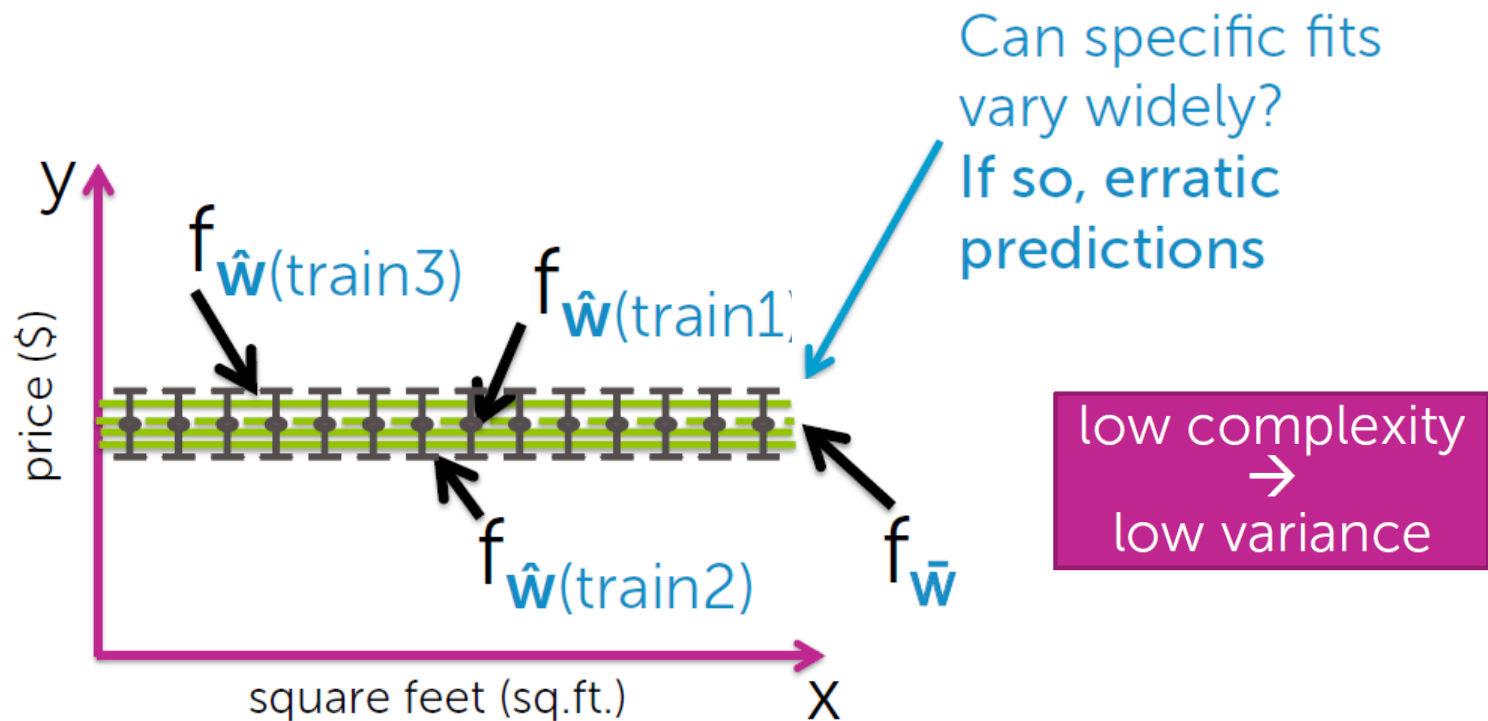
Variance contribution

How much do specific fits vary from the expected fit?



Variance contribution

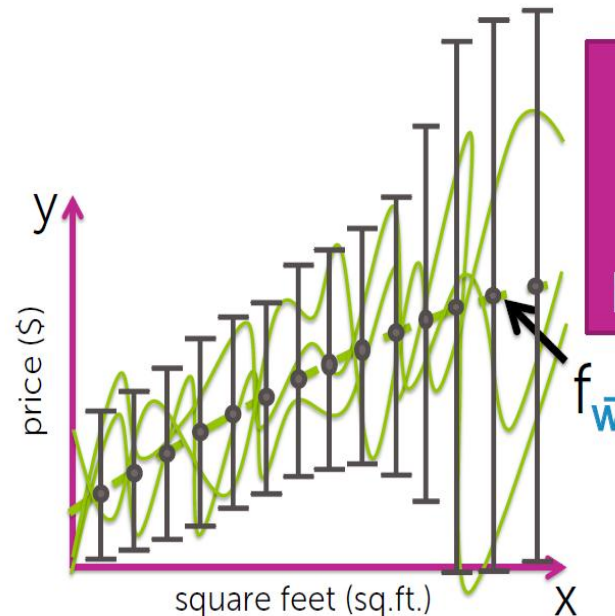
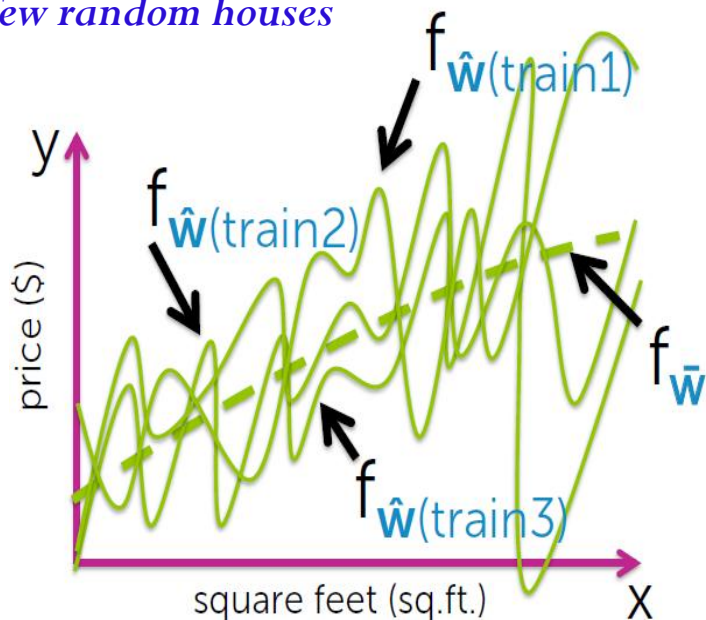
How much do specific fits vary from the expected fit?



Variance of high complexity models

Assume we fit a high-order polynomial

*For each train remove
few random houses*

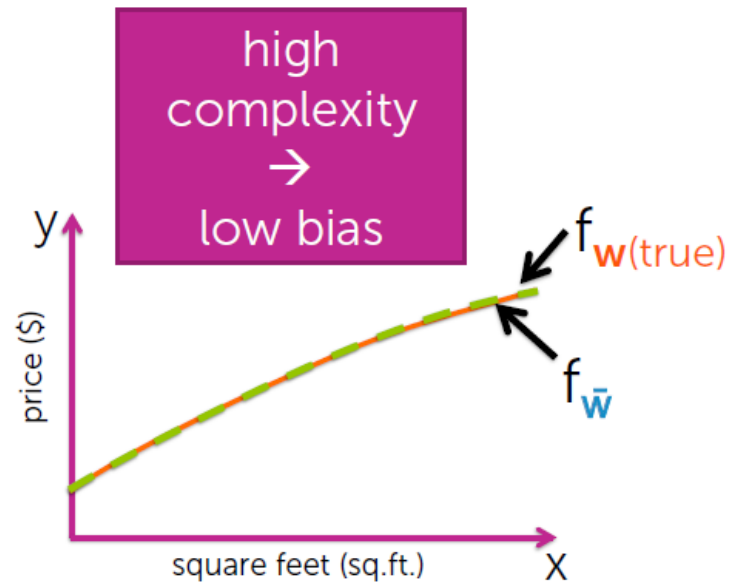
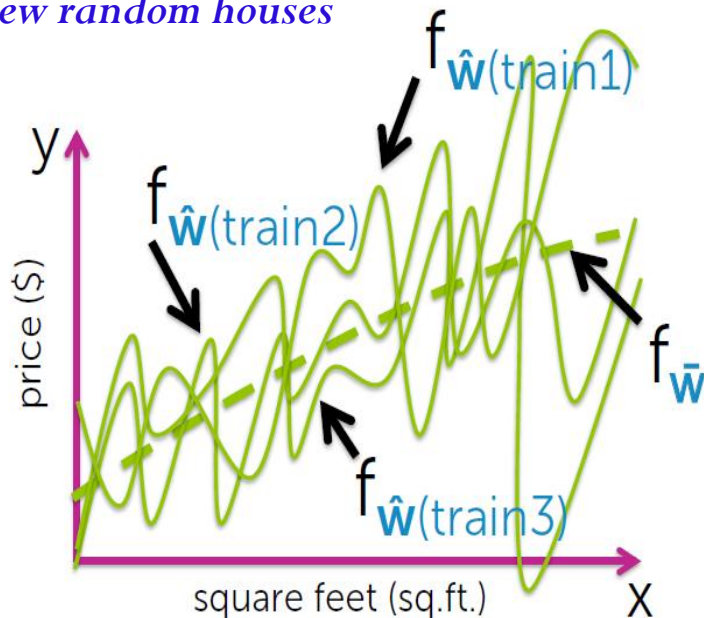


high
complexity
→
high variance

Bias of high complexity models

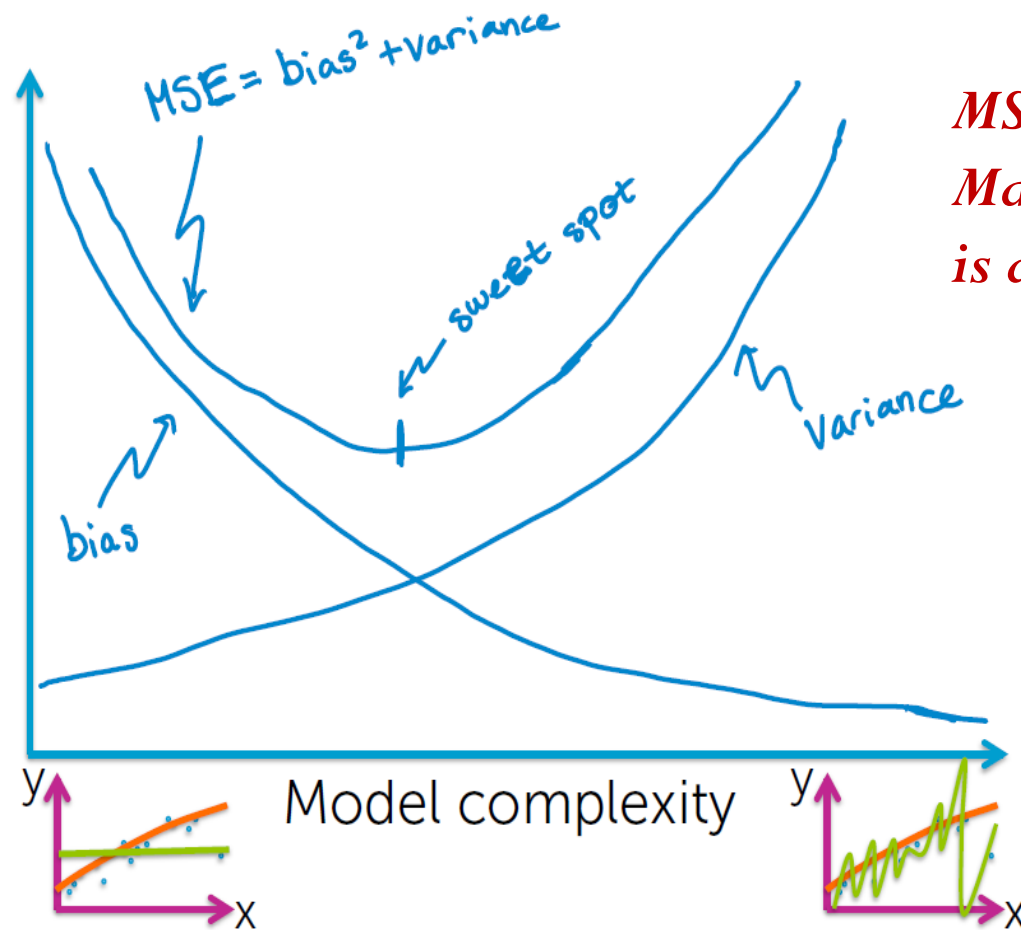
Assume we fit a high-order polynomial

*For each train remove
few random houses*



*High complexity models are very flexible,
pick better average trends.*

Bias – variance tradeoff

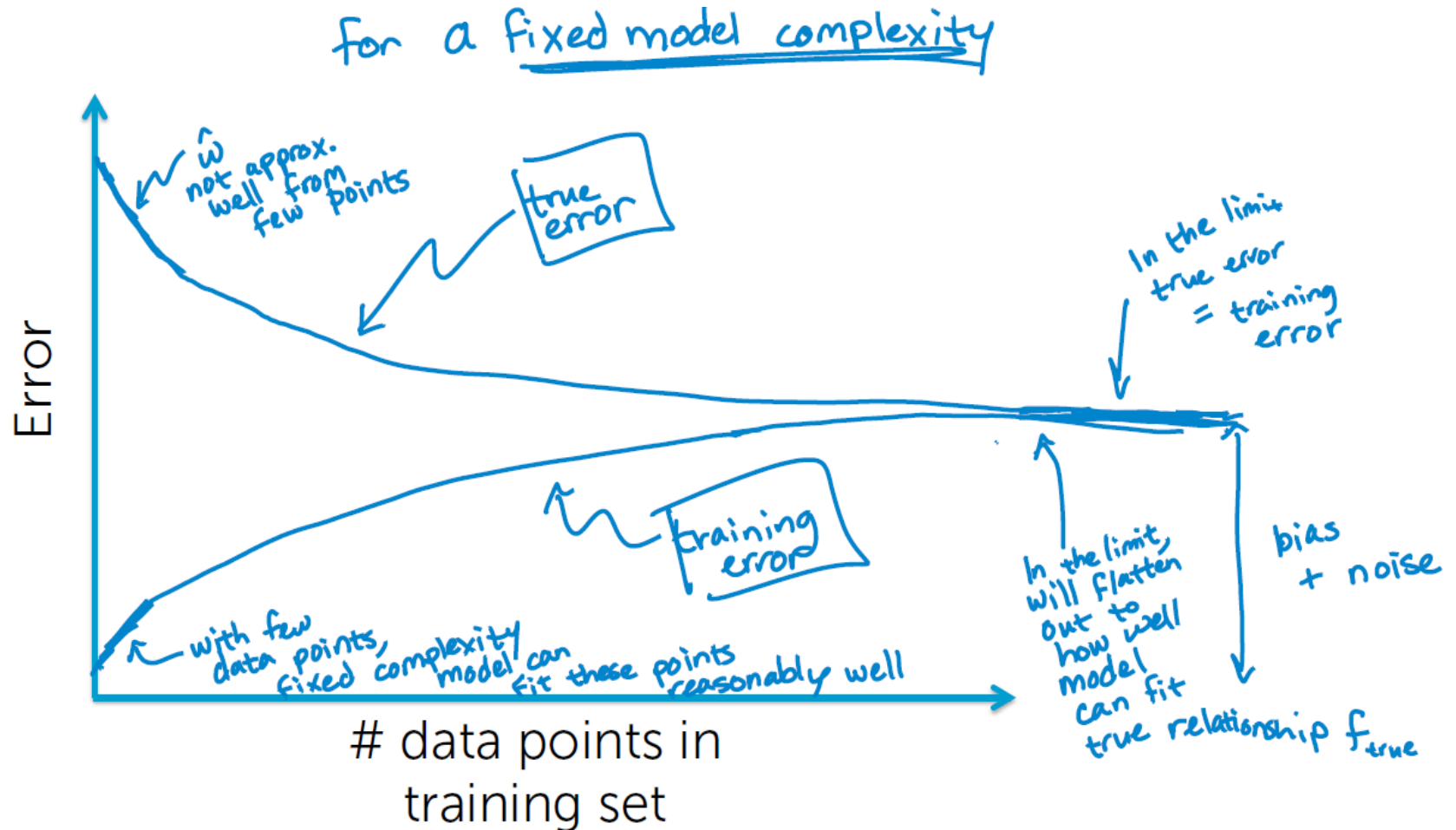


*MSE = mean square error
Machine Learning
is all about this tradeoff*

But....

*Just like with
generalization error,
we cannot compute
bias and variance*

Errors vs amount of data



The regression/ML workflow

1. Model selection

Often, need to choose tuning parameters λ controlling model complexity (e.g. degree of polynomial)

2. Model assessment

Having selected a model, assess the generalization error

Hypothetical implementation

Training set

Test set

1. Model selection

For each considered model complexity λ :

- i. Estimate parameters $\hat{\mathbf{w}}_\lambda$ on **training data**
- ii. Assess performance of $\hat{\mathbf{w}}_\lambda$ on **test data**
- iii. Choose λ^* to be λ with **lowest test error**

2. Model assessment

Compute test error of $\hat{\mathbf{w}}_{\lambda^*}$ (fitted model for selected complexity λ^*) to approx. generalization error

Hypothetical implementation

Training set

Test set

1. Model selection

For each considered model complexity λ :

- i. Estimate parameters $\hat{\mathbf{w}}_\lambda$ on **training data**
- ii. Assess performance of $\hat{\mathbf{w}}_\lambda$ on **test data**
- iii. Choose λ^* to be λ with **lowest test error**

2. Model assessment

Compute test error of $\hat{\mathbf{w}}_{\lambda^*}$ (fitted model for selected complexity λ^*) to approx. generalization error

Overly optimistic!

Hypothetical implementation



Issue: Just like fitting $\hat{\mathbf{w}}$ and assessing its performance both on training data

- λ^* was selected to minimize test error (i.e., λ^* was fit on test data)
- If test data is not representative of the whole world, then $\hat{\mathbf{w}}_{\lambda^*}$ will typically perform worse than test error indicates

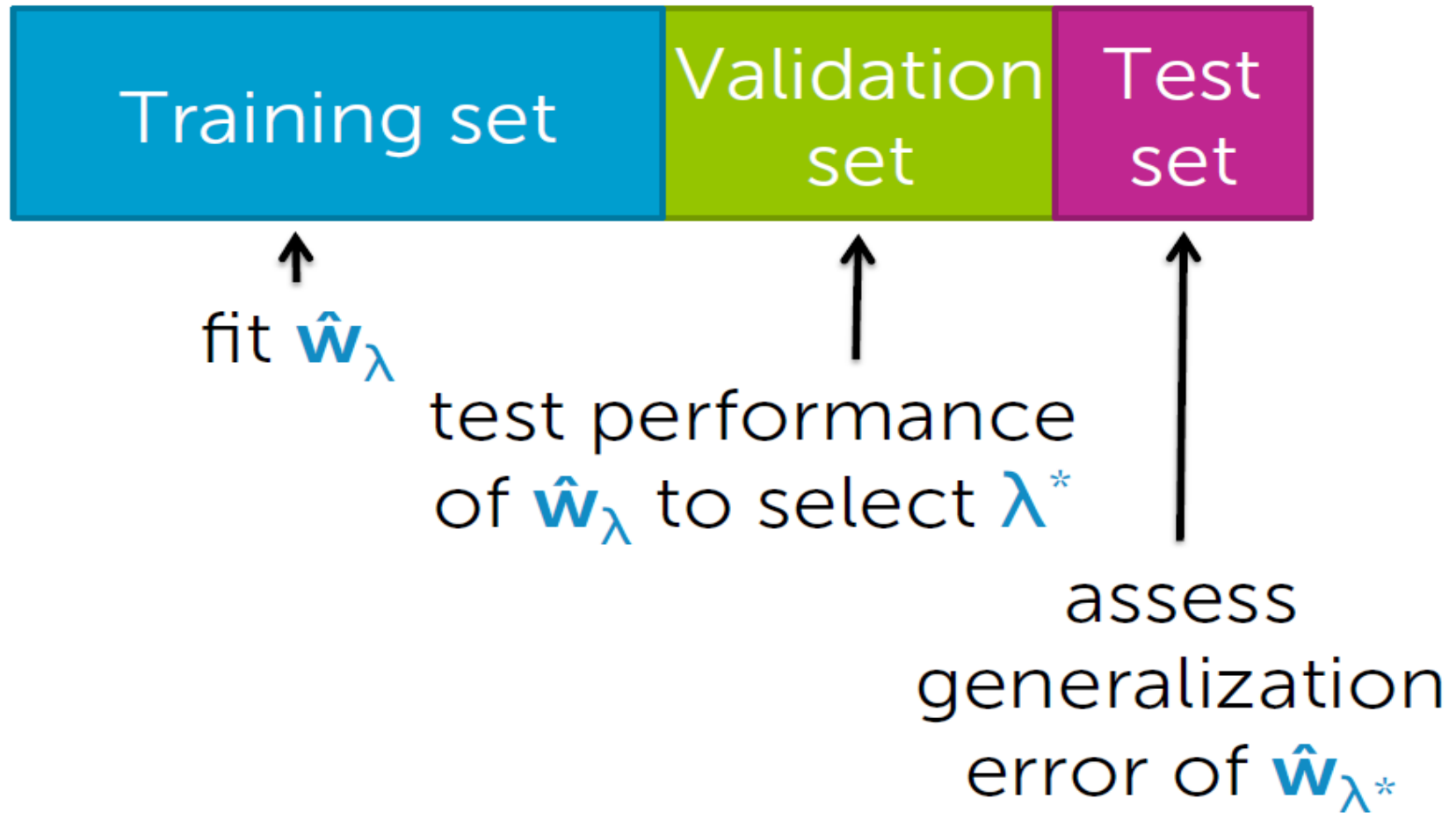
Practical implementation



Solution: Create two “test” sets!

1. Select λ^* such that $\hat{\mathbf{w}}_{\lambda^*}$ minimizes error on validation set
2. Approximate generalization error of $\hat{\mathbf{w}}_{\lambda^*}$ using test set

Practical implementation



Typical splits



80%

10%

10%

50%

25%

25%

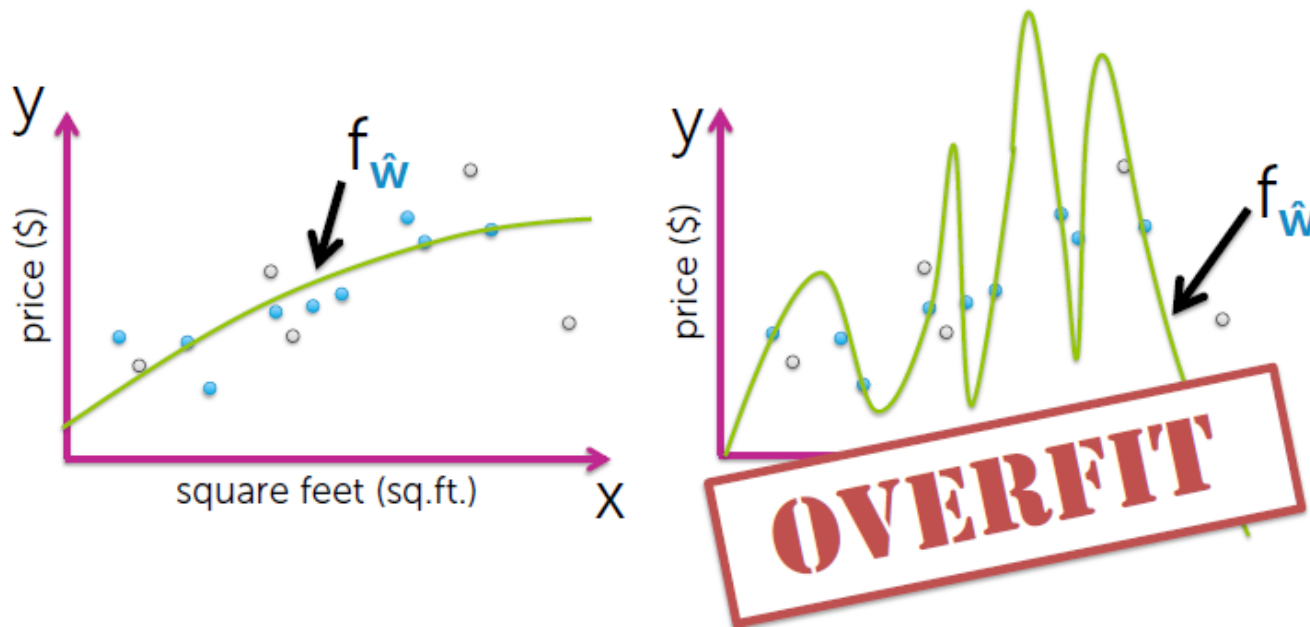
What you should be able to do now

- Describe what a loss function is and give examples
- Contrast training, generalization, and test error
- Compute training and test error given a loss function
- Discuss issue of assessing performance on training set
- Describe tradeoffs in forming training/test splits
- List and interpret the 3 sources of avg. prediction error
 - Irreducible error, bias, and variance
- Discuss issue of selecting model complexity on test data and then using test error to assess generalization error
- Motivate use of a validation set for selecting tuning parameters (e.g., model complexity)
- Describe overall regression workflow

RIDGE REGRESSION

Flexibility of high-order polynomials

$$y_i = w_0 + w_1 x_i + w_2 x_i^2 + \dots + w_p x_i^p + \varepsilon_i$$



Symptoms for overfitting: often associated with very large value of estimated parameters

$\hat{w}$

Overfitting with many features

Not unique to polynomial regression,
but also if **lots of inputs** (d large)

Or, generically,
lots of features (D large)

$$y_i = \sum_{j=0}^D w_j h_j(\mathbf{x}_i) + \varepsilon_i$$

- Square feet
- # bathrooms
- # bedrooms
- Lot size
- Year built
- ...

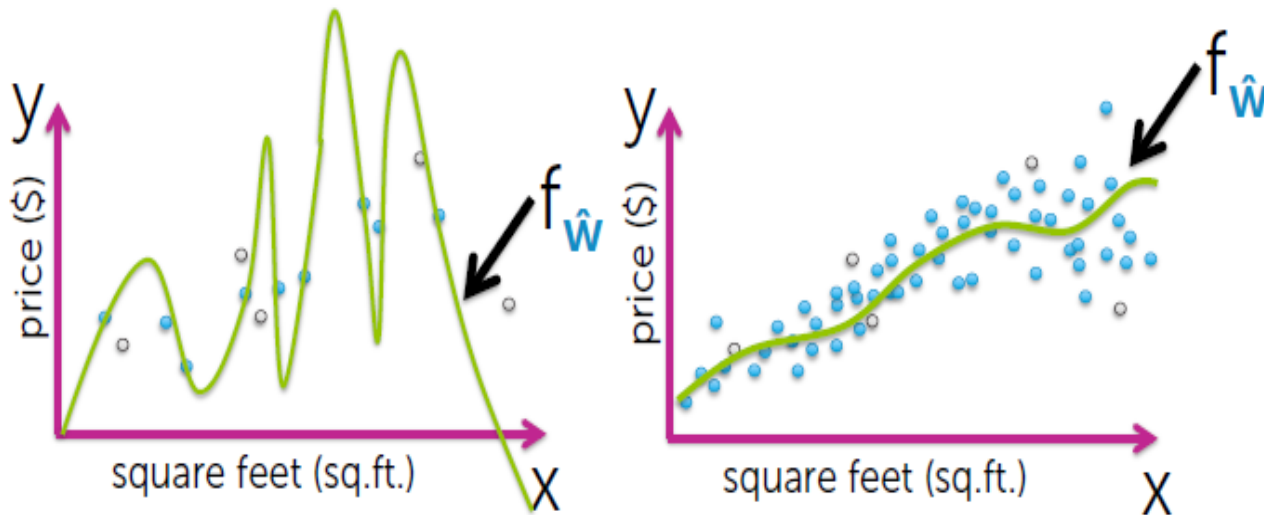
How does # of observations influence overfitting?

Few observations (N small)

→ rapidly overfit as model complexity increases

Many observations (N very large)

→ harder to overfit

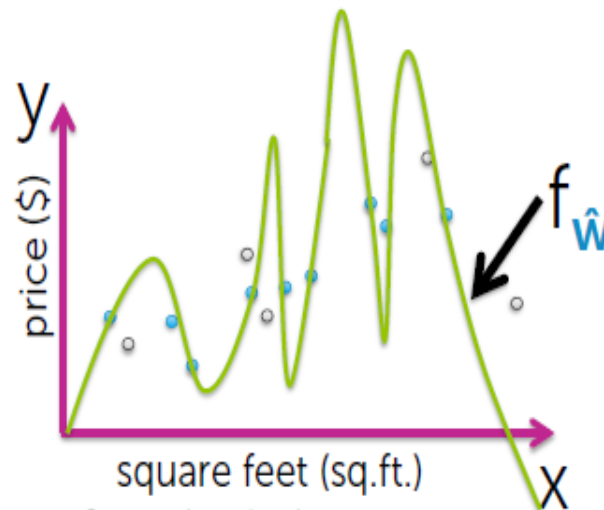


How does # of inputs influence overfitting?

1 input (e.g., sq.ft.):

Data must include representative examples of all possible (sq.ft., \$) pairs to avoid overfitting

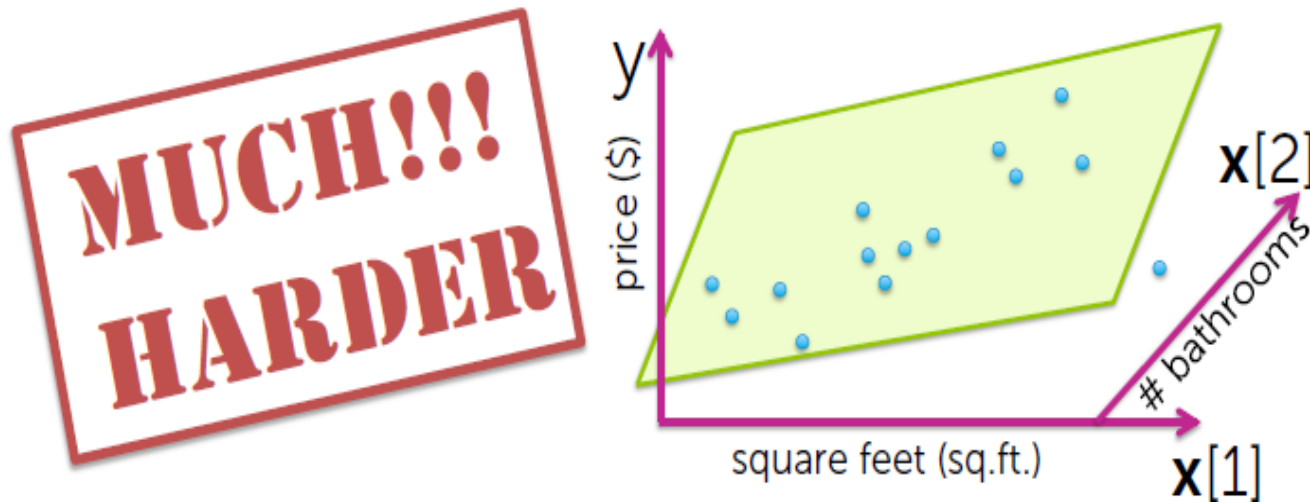
HARD



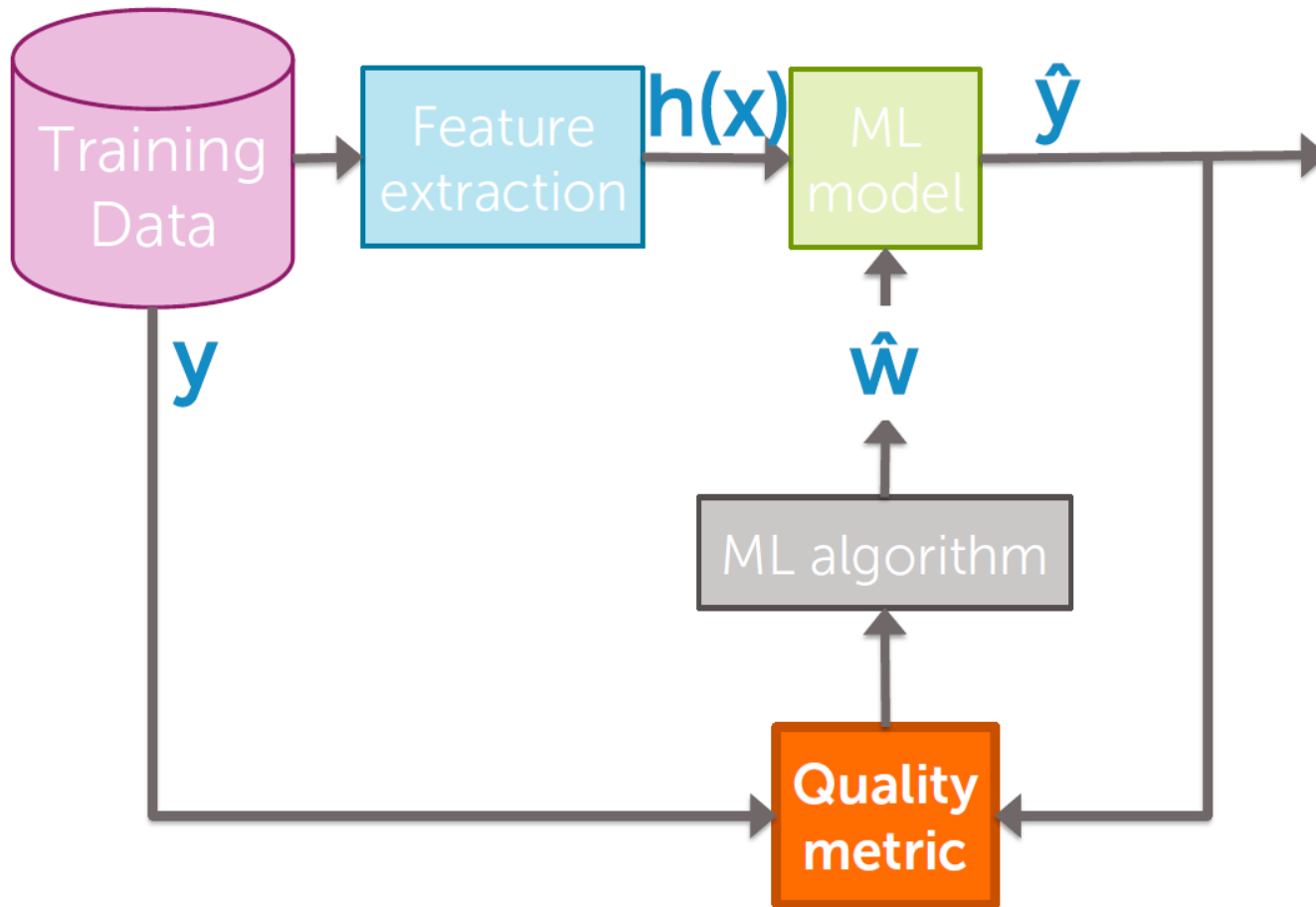
How does # of inputs influence overfitting?

d inputs (e.g., sq.ft., #bath, #bed, lot size, year,...):

Data must include examples of all possible
(sq.ft., #bath, #bed, lot size, year,..., \$) combos
to avoid overfitting



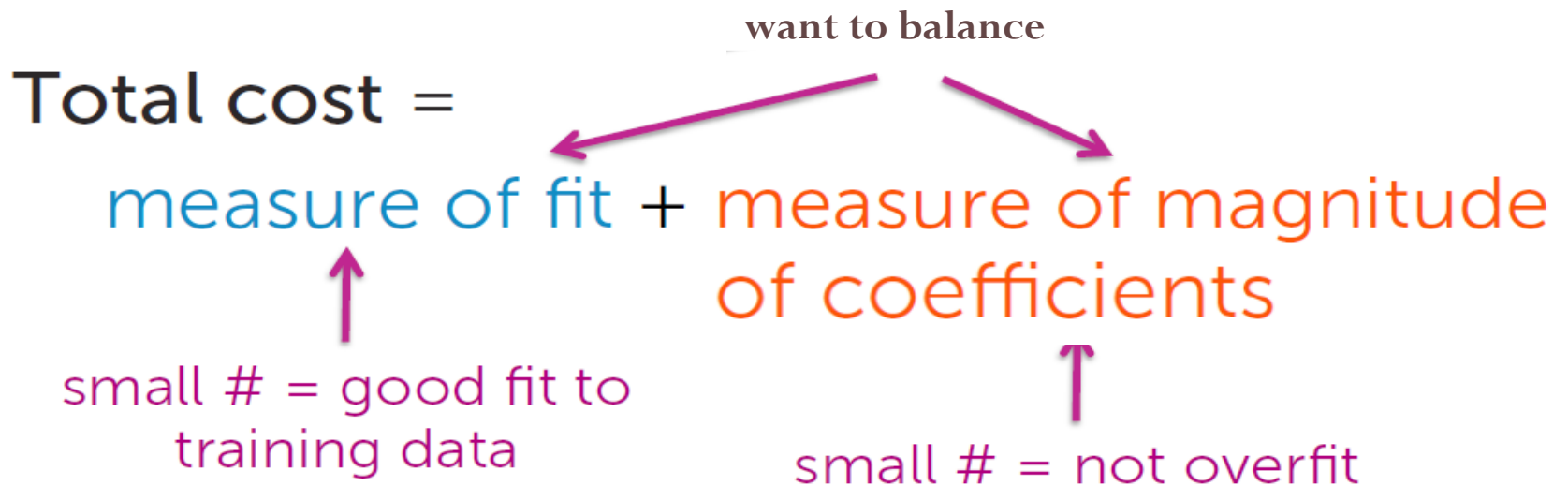
Lets improve quality metric blok



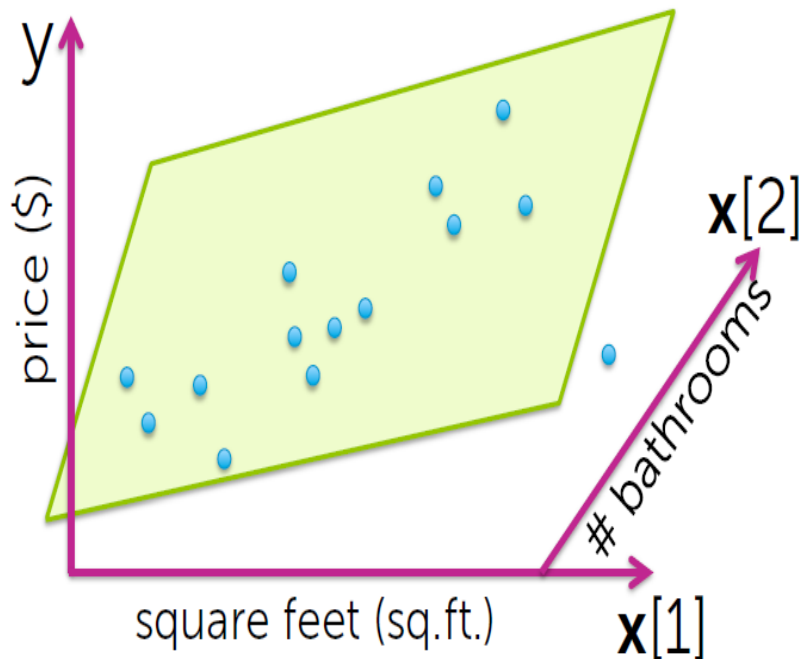
Desire total cost format

Want to balance:

- i. How well function fits data
- ii. Magnitude of coefficients



Measure of fit to training data



$$\begin{aligned} \text{RSS}(\mathbf{w}) &= \sum_{i=1}^N (y_i - h(\mathbf{x}_i)^T \mathbf{w})^2 \\ &= \sum_{i=1}^N (y_i - \hat{y}_i(\mathbf{w}))^2 \end{aligned}$$

pred. value using $\mathbf{w}$

small RSS $\rightarrow$ model fitting training data well

Measure of magnitude of regression coefficients

What summary # is indicative of size of regression coefficients?

- Sum? $w_0 = 1,527,301$ $w_1 = -1,605,253$

$$w_0 + w_1 = \text{small \#}$$

← But ... the coefficients are very large

- Sum of absolute value?

$$|w_0| + |w_1| + \dots + |w_D| = \sum_{j=0}^D |w_j| \triangleq \|w\|_1 \quad L_1 \text{ norm} \quad \dots \text{discuss more in next module}$$

- Sum of squares (L_2 norm)

$$w_0^2 + w_1^2 + \dots + w_D^2 = \sum_{j=0}^D w_j^2 \triangleq \|w\|_2^2 \quad L_2 \text{ norm} \quad \dots \text{focus of this module}$$

Consider specific total cost

Total cost =

$$\underbrace{\text{measure of fit}}_{\text{RSS}(\mathbf{w})} + \underbrace{\text{measure of magnitude of coefficients}}_{\|\mathbf{w}\|_2^2}$$

Consider resulting objectives

What if $\hat{\mathbf{w}}$ selected to minimize

$$\text{RSS}(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2$$

 tuning parameter = balance of fit and magnitude

Ridge regression
(a.k.a L_2 regularization)

If $\lambda=0$:

reduces to minimizing $\text{RSS}(\mathbf{w})$, as before (old solution) $\rightarrow \hat{\mathbf{w}}^{\text{LS}}$ ← least squares

If $\lambda=\infty$:

For solutions where $\hat{\mathbf{w}} \neq 0$, then total cost is ∞

If $\hat{\mathbf{w}}=0$, then total cost = $\text{RSS}(0)$ $\rightarrow$ solution is $\hat{\mathbf{w}}=0$

If λ in between: Then $0 \leq \|\hat{\mathbf{w}}\|_2^2 \leq \|\hat{\mathbf{w}}^{\text{LS}}\|_2^2$

Ridge regression: bias-variance tradeoff

Large λ :

high bias, low variance

(e.g., $\hat{\mathbf{w}} = 0$ for $\lambda = \infty$)

In essence, λ
controls model
complexity

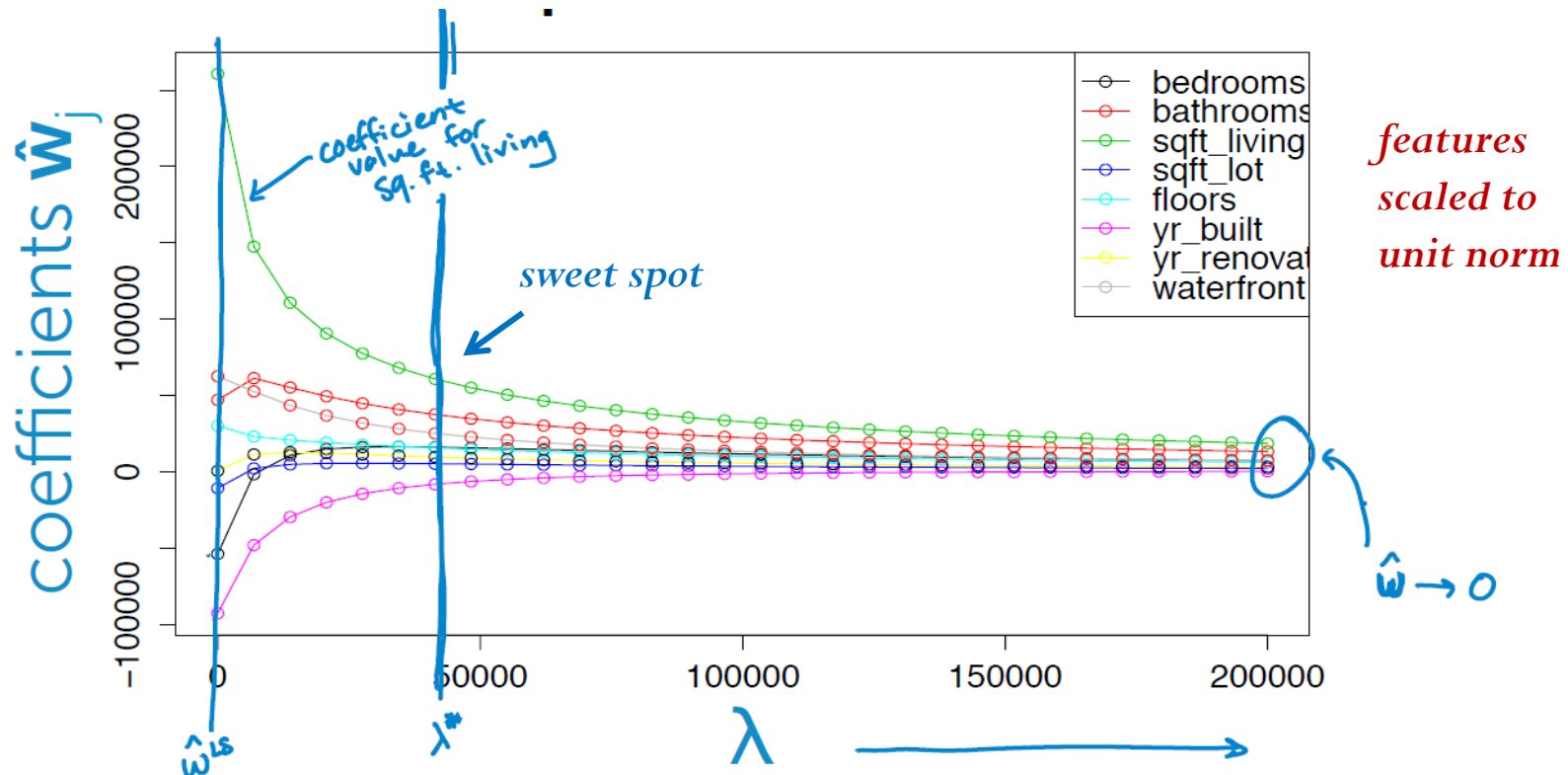
Small λ :

low bias, high variance

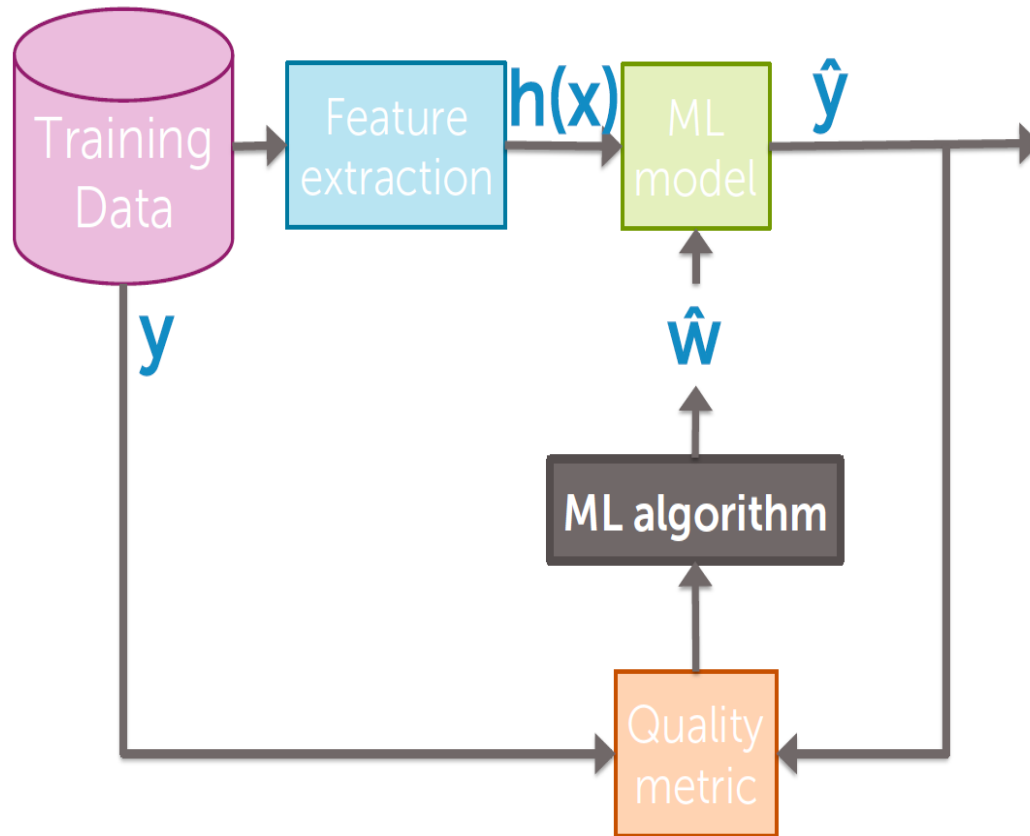
(e.g., standard least squares (RSS) fit of
high-order polynomial for $\lambda = 0$)

Ridge regression: coefficients path

What happens if we refit our high-order polynomial, but now using **ridge regression**?



Flow chart



Model for all N observations together

$$\mathbf{y} = \mathbf{H} \mathbf{w} + \boldsymbol{\varepsilon}$$

Ridge regression: cost in matrix notation

In matrix form, ridge regression cost is:

$$\begin{aligned} \text{RSS}(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2 \\ = (\mathbf{y} - \mathbf{H}\mathbf{w})^\top (\mathbf{y} - \mathbf{H}\mathbf{w}) + \lambda \mathbf{w}^\top \mathbf{w} \end{aligned}$$

$$\|\mathbf{w}\|_2^2 = w_0^2 + w_1^2 + w_2^2 + \dots + w_D^2$$

$$= \begin{bmatrix} \boxed{} & \boxed{} & \boxed{} & \boxed{} & \boxed{} \end{bmatrix} \begin{bmatrix} \boxed{} \\ \boxed{} \\ \boxed{} \\ \boxed{} \\ \boxed{} \end{bmatrix}$$

w_0
 w_1
 w_2
 $\vdots$
 w_D

$$= \mathbf{w}^\top \mathbf{w}$$

Gradient of ridge regression cost

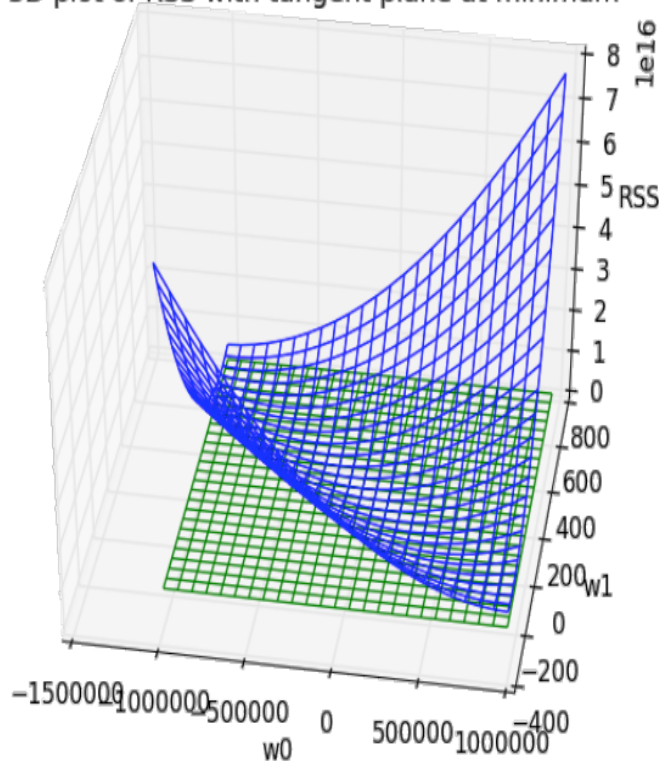
$$\begin{aligned}\nabla [\text{RSS}(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2] &= \nabla [(\mathbf{y} - \mathbf{H}\mathbf{w})^\top (\mathbf{y} - \mathbf{H}\mathbf{w}) + \lambda \mathbf{w}^\top \mathbf{w}] \\ &= \underbrace{\nabla [(\mathbf{y} - \mathbf{H}\mathbf{w})^\top (\mathbf{y} - \mathbf{H}\mathbf{w})]}_{-2\mathbf{H}^\top (\mathbf{y} - \mathbf{H}\mathbf{w})} + \lambda \underbrace{\nabla [\mathbf{w}^\top \mathbf{w}]}_{2\mathbf{w}}\end{aligned}$$

Why? By analogy to 1d case...

$\mathbf{w}^\top \mathbf{w}$ analogous to w^2 and derivative of $w^2 = 2w$

Ridge regression: closed-form solution

3D plot of RSS with tangent plane at minimum



$$\nabla \text{cost}(\mathbf{w}) = -2\mathbf{H}^T(\mathbf{y} - \mathbf{H}\mathbf{w}) + 2\lambda\mathbf{I}\mathbf{w} = 0$$

$$\text{Solve for } \mathbf{w}: -\mathbf{H}^T\mathbf{y} + \mathbf{H}^T\mathbf{H}\hat{\mathbf{w}} + \lambda\mathbf{I}\hat{\mathbf{w}} = 0$$

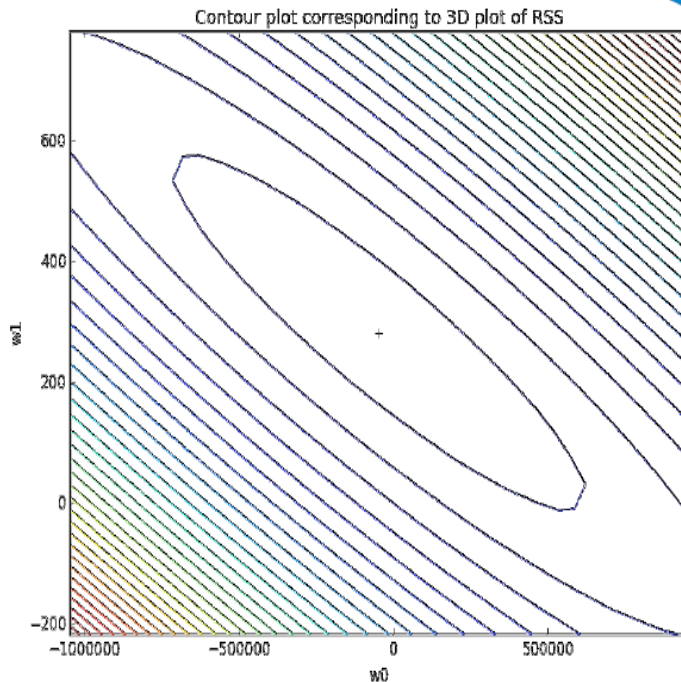
$$\mathbf{H}^T\mathbf{H}\hat{\mathbf{w}} + \lambda\mathbf{I}\hat{\mathbf{w}} = \mathbf{H}^T\mathbf{y}$$

$$(\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I})\hat{\mathbf{w}} = \mathbf{H}^T\mathbf{y}$$

$$\hat{\mathbf{w}}^{\text{ridge}} = (\mathbf{H}^T\mathbf{H} + \lambda\mathbf{I})^{-1}\mathbf{H}^T\mathbf{y}$$

Ridge regression: gradient descent

$$\nabla \text{cost}(\mathbf{w}) = -2\mathbf{H}^T(\mathbf{y} - \mathbf{H}\mathbf{w}) + 2\lambda\mathbf{w}$$



Update to j^{th} feature weight:

$$w_j^{(t+1)} \leftarrow \underline{w_j^{(t)}} - \eta *$$

Same as before (from RSS term) $\rightarrow -2 \eta_j \sum_{i=1}^N (\mathbf{x}_i)(y_i - \hat{y}_i(\mathbf{w}^{(t)}))$

$+ 2\lambda \underline{w_j^{(t)}}$
new term, comes from the j^{th} component of $2\lambda \mathbf{w}$

Summary of ridge regression algorithm

init $\mathbf{w}^{(1)} = 0$ (or randomly, or smartly), $t=1$

while $|| \nabla \text{RSS}(\mathbf{w}^{(t)}) || > \varepsilon$

for $j=0, \dots, D$

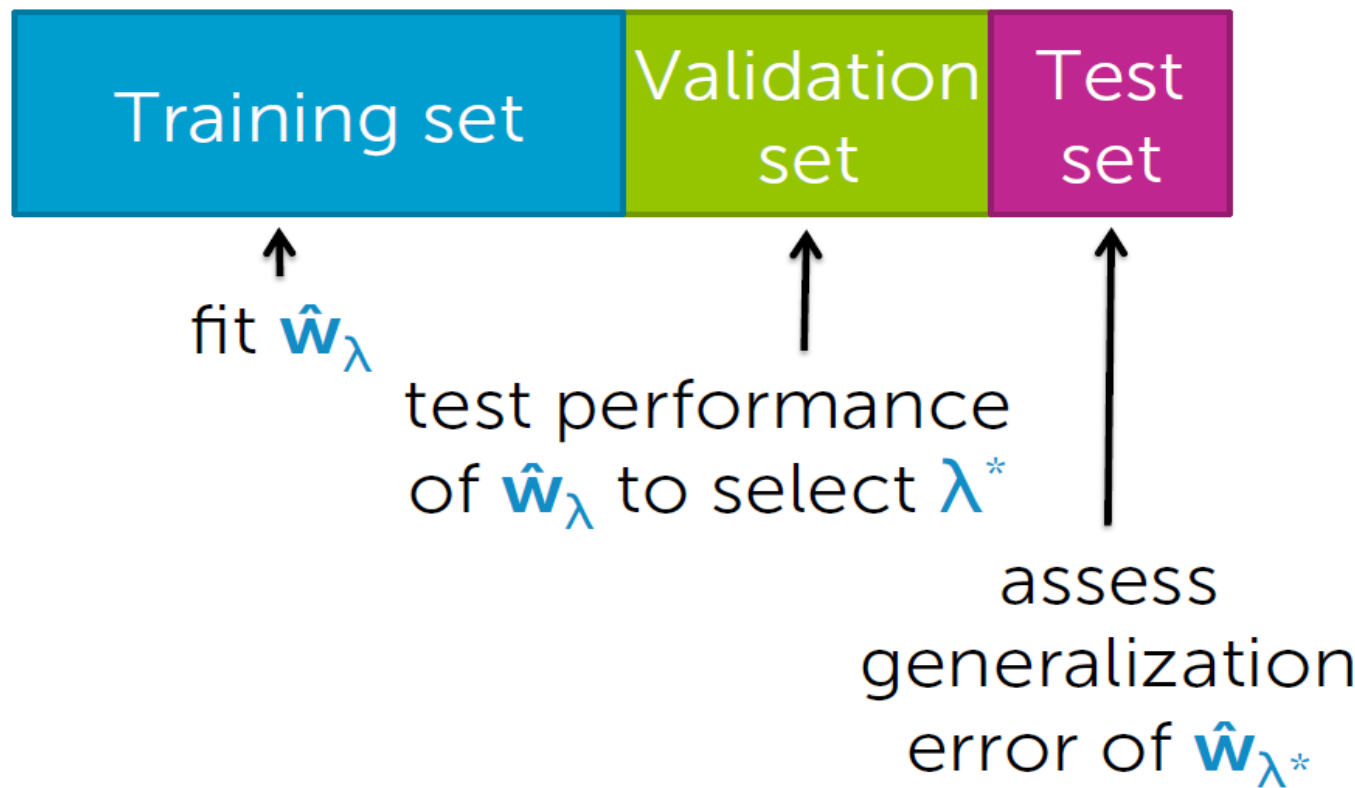
$$\text{partial}[j] = -2 h_j \sum_{i=1}^N (\mathbf{x}_i)(y_i - \hat{y}_i(\mathbf{w}^{(t)}))$$

$$\mathbf{w}_j^{(t+1)} \leftarrow (1 - 2\eta\lambda) \mathbf{w}_j^{(t)} - \eta \text{partial}[j]$$

$$t \leftarrow t + 1$$

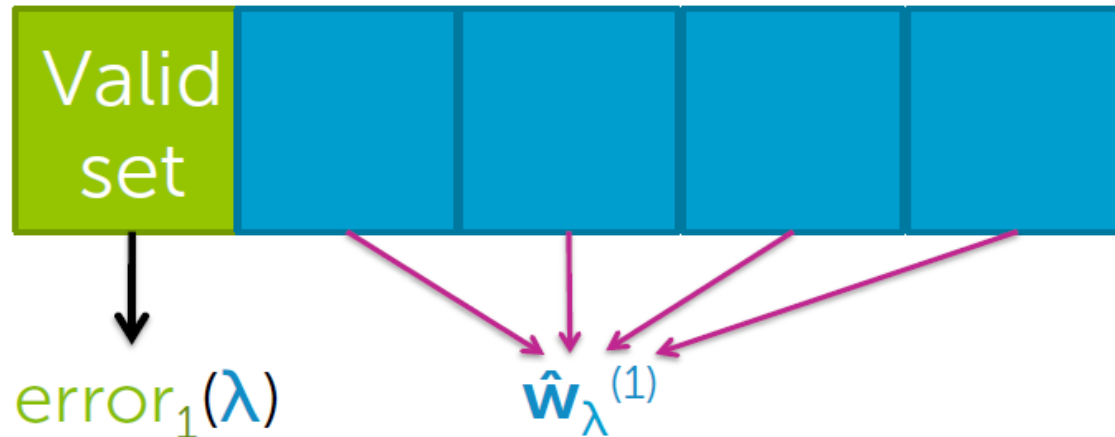
How to choose λ

If sufficient amount of data...



How to choose λ

K-fold cross validation

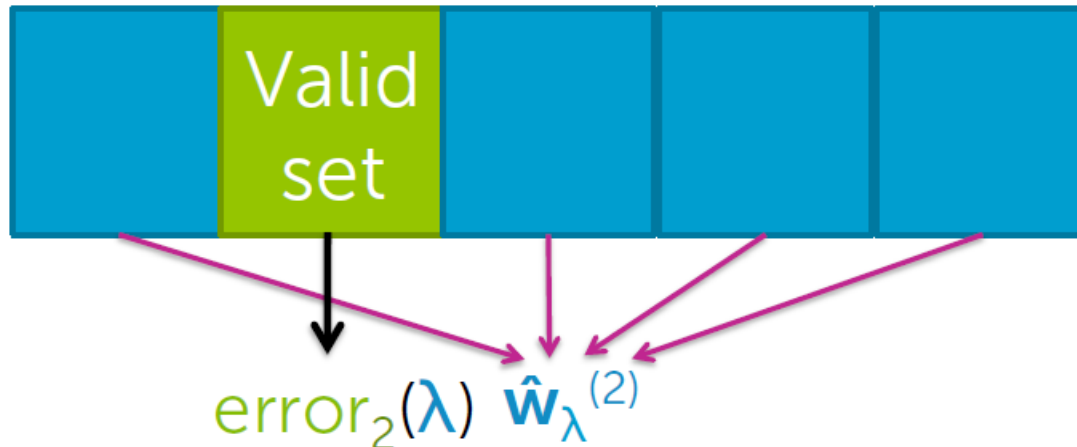


For $k=1, \dots, K$

1. Estimate $\hat{\mathbf{w}}_{\lambda}^{(k)}$ on the training blocks
2. Compute error on validation block: $\text{error}_k(\lambda)$

How to choose λ

K-fold cross validation

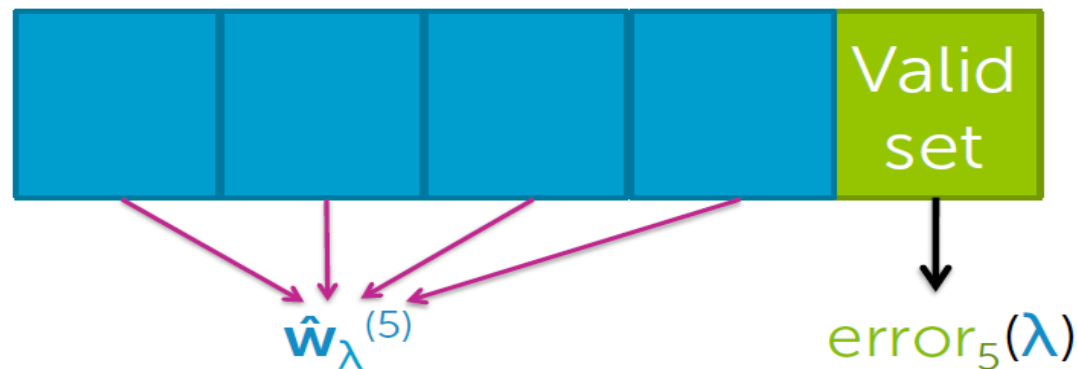


For $k=1, \dots, K$

1. Estimate $\hat{\mathbf{w}}_{\lambda}^{(k)}$ on the training blocks
2. Compute error on validation block: $\text{error}_k(\lambda)$

How to choose λ

K-fold cross validation



For $k=1, \dots, K$

1. Estimate $\hat{w}_{\lambda}^{(k)}$ on the training blocks
2. Compute error on validation block: $error_k(\lambda)$

Compute average error: $CV(\lambda) = \frac{1}{K} \sum_{k=1}^K error_k(\lambda)$

How to choose λ

K-fold cross validation



Repeat procedure for each choice of λ

Choose λ^* to minimize $CV(\lambda)$

What value of K

Formally, the **best approximation** occurs for validation sets of size 1 ($K=N$)

leave-one-out
cross validation

Computationally intensive

- requires computing N fits of model per λ

Typically, $K=5$ or 10

5-fold CV

10-fold CV

How to handle the intercept

Recall multiple regression model

Model:

$$\begin{aligned} y_i &= w_0 h_0(\mathbf{x}_i) + w_1 h_1(\mathbf{x}_i) + \dots + w_D h_D(\mathbf{x}_i) + \varepsilon_i \\ &= \sum_{j=0}^D w_j h_j(\mathbf{x}_i) + \varepsilon_i \end{aligned}$$

feature 1 = $h_0(\mathbf{x})$...often 1 (constant)

feature 2 = $h_1(\mathbf{x})$... e.g., $\mathbf{x}[1]$

feature 3 = $h_2(\mathbf{x})$... e.g., $\mathbf{x}[2]$

...

feature D+1 = $h_D(\mathbf{x})$... e.g., $\mathbf{x}[d]$

Do we penalize intercept?

Standard ridge regression cost:

$$\text{RSS}(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2$$

 strength of penalty

Encourages intercept w_0 to also be small

Do we want a small intercept?

Conceptually, not indicative of overfitting...

Do we penalize intercept?

- **Option 1: don't penalize intercept**

Modified ridge regression cost:

$$\text{RSS}(\mathbf{w}_0, \mathbf{w}_{\text{rest}}) + \lambda \|\mathbf{w}_{\text{rest}}\|_2^2$$

- **Option 2: center data first**

If data are first **centered about 0**, then favoring small intercept not so worrisome

Step 1: Transform y to have 0 mean

Step 2: Run ridge regression as normal
(closed-form or gradient algorithms)

What you should be able to do now

- Describe what happens to magnitude of estimated coefficients when model is overfit
- Motivate form of ridge regression cost function
- Describe what happens to estimated coefficients of ridge regression as tuning parameter λ is varied
- Interpret coefficient path plot
- Estimate ridge regression parameters:
 - In closed form
 - Using an iterative gradient descent algorithm
- Implement K-fold cross validation to select the ridge regression tuning parameter λ

FEATURES SELECTION & LASSO REGRESSION

Why features selection?

Efficiency:

- If $\text{size}(\mathbf{w}) = 100\text{B}$, each prediction is expensive
- If $\hat{\mathbf{w}}$ **sparse**, computation only depends on # of non-zeros

 many zeros


$$\hat{y}_i = \sum_{\hat{w}_j \neq 0} \hat{w}_j h_j(\mathbf{x}_i)$$

Interpretability:

- Which features are relevant for prediction?

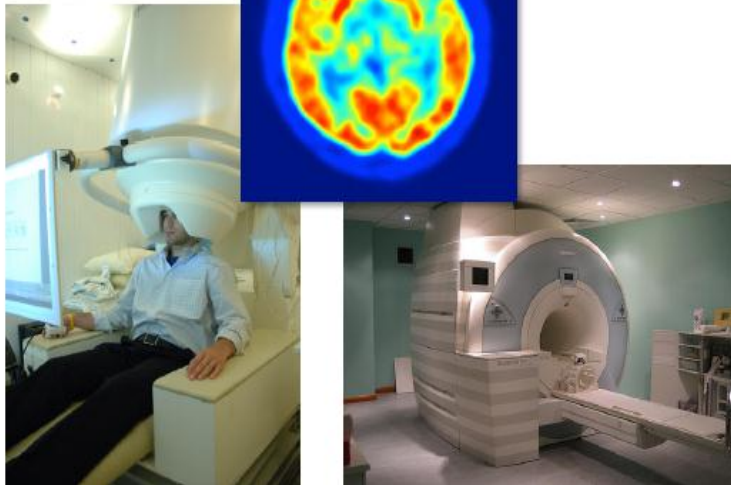
Housing application



Lot size	Dishwasher
Single Family	Garbage disposal
Year built	Microwave
Last sold price	Range / Oven
Last sale price/sqft	Refrigerator
Finished sqft	Washer
Unfinished sqft	Dryer
Finished basement sqft	Laundry location
# floors	Heating type
Flooring types	Jetted Tub
Parking type	Deck
Parking amount	Fenced Yard
Cooling	Lawn
Heating	Garden
Exterior materials	Sprinkler System
Roof type	⋮
Structure style	

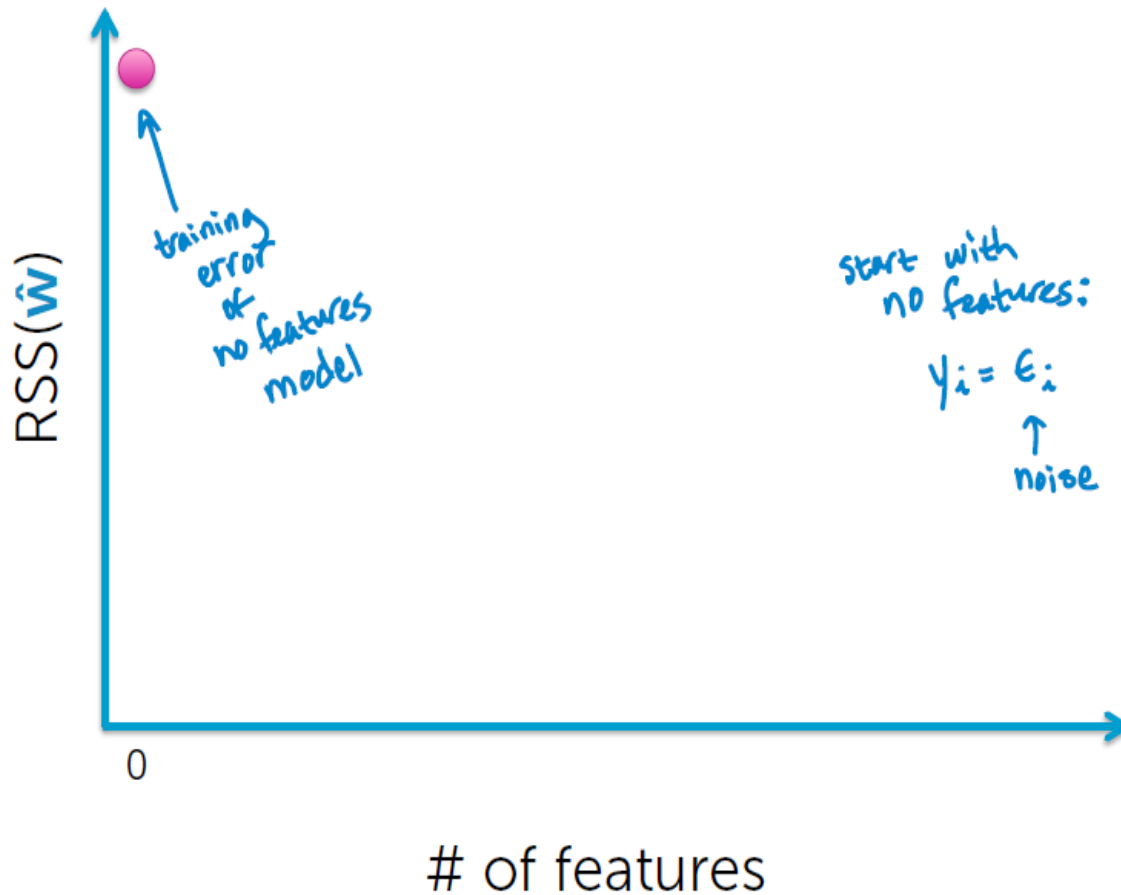
Sparcity

Reading your mind



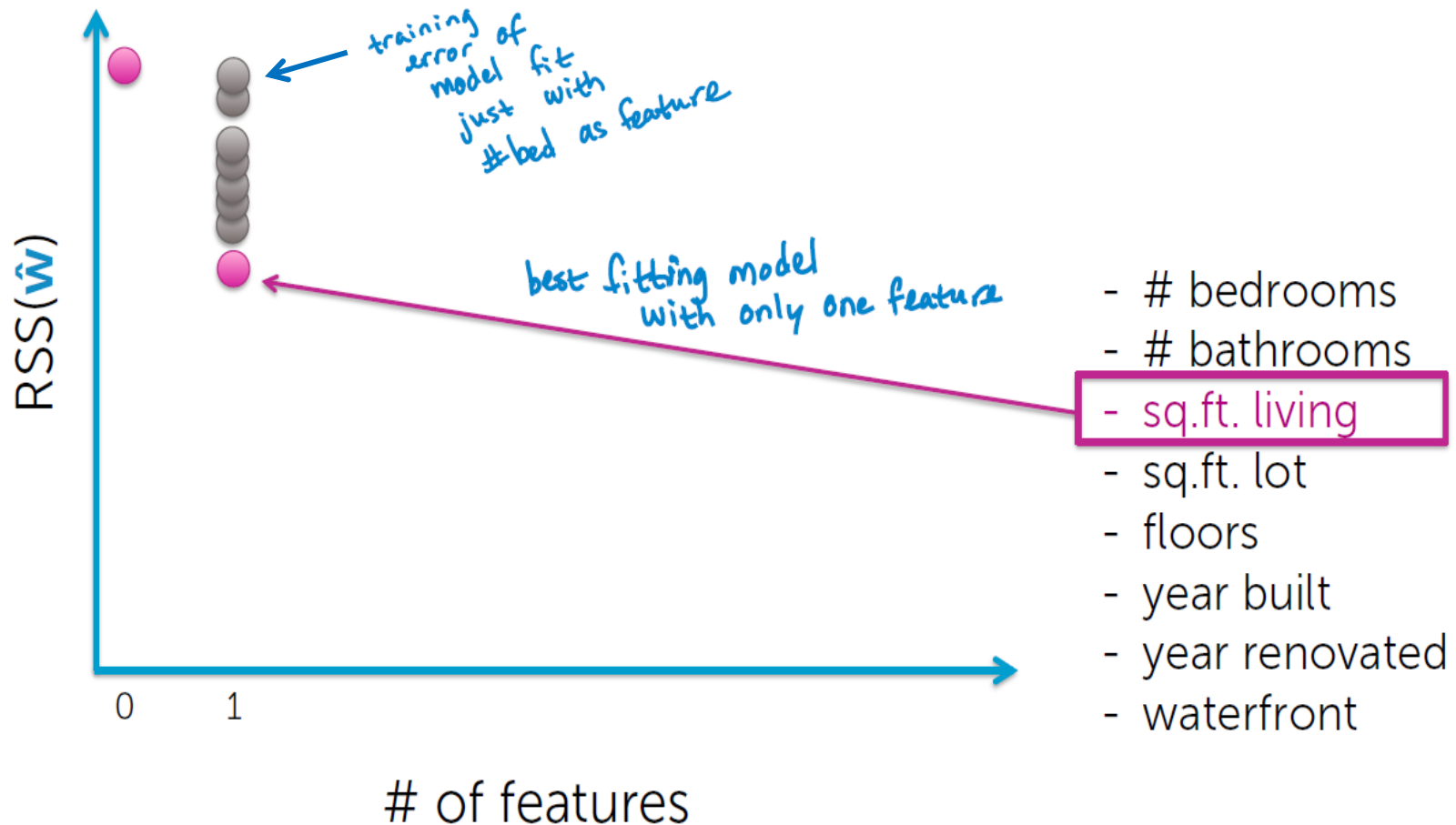
Activity in which
brain regions
can predict
happiness?

Find best model of size: 0



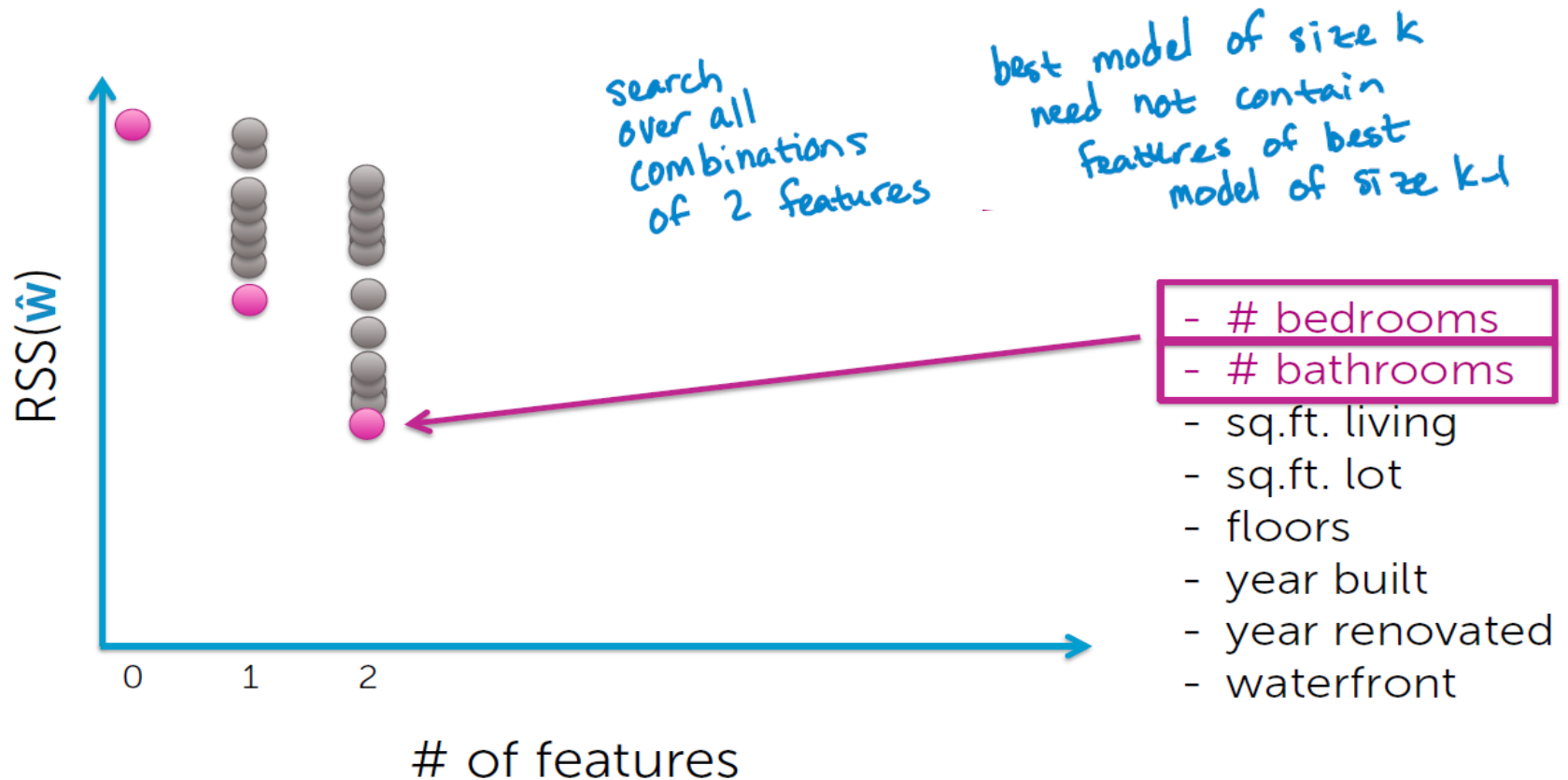
- # bedrooms
- # bathrooms
- sq.ft. living
- sq.ft. lot
- floors
- year built
- year renovated
- waterfront

Find best model of size: 1

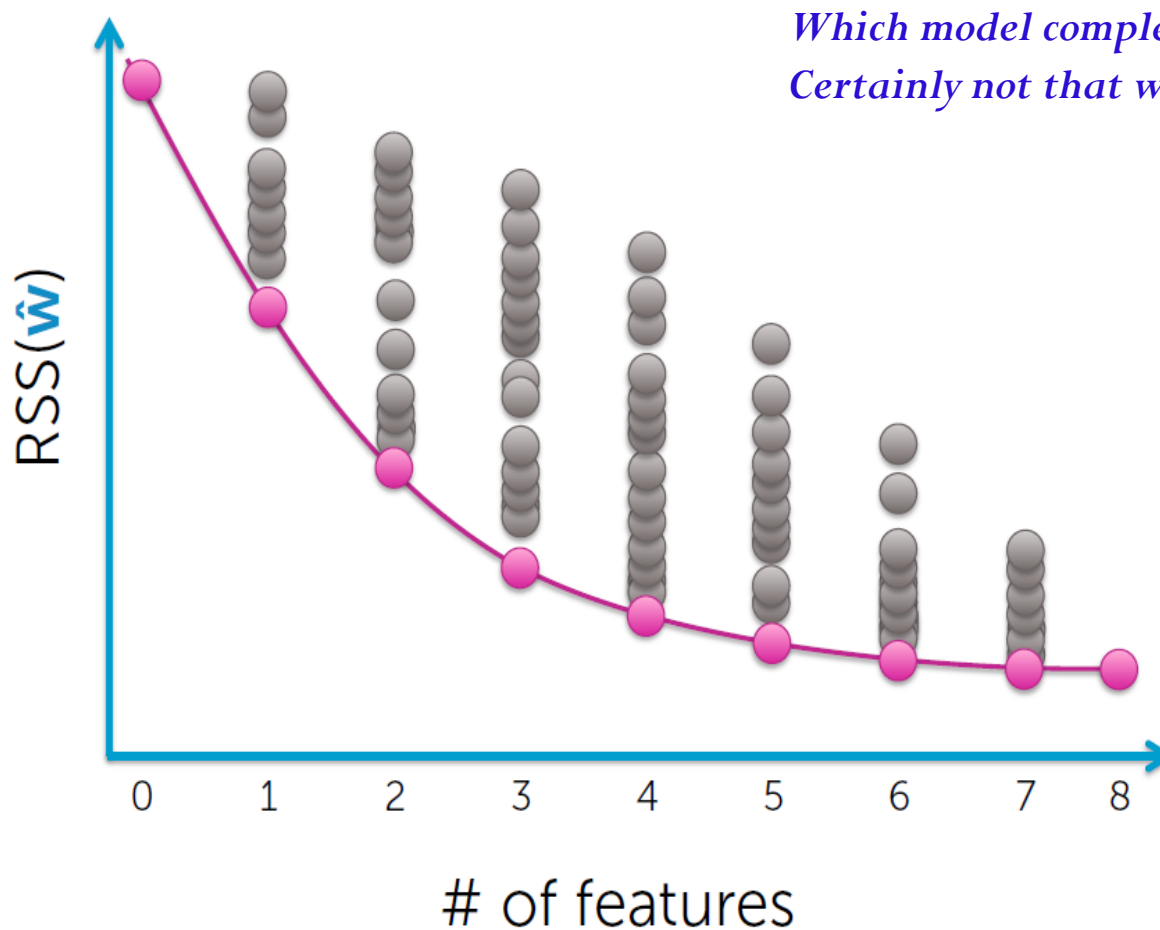


Find best model of size: 2

Note: not necessarily nested!



Find best model of size: N



*Which model complexity to choose?
Certainly not that with the smallest training error!*

- # bedrooms
- # bathrooms
- sq.ft. living
- sq.ft. lot
- floors
- year built
- year renovated
- waterfront

Choosing model complexity

Option 1: Assess on validation set

Option 2: Cross validation

Option 3+: Other metrics for penalizing model complexity like BIC...

Complexity of „all subsets”

How many models were evaluated?

- each indexed by features included

$$y_i = \varepsilon_i$$

$$y_i = w_0 h_0(\mathbf{x}_i) + \varepsilon_i$$

$$y_i = w_1 h_1(\mathbf{x}_i) + \varepsilon_i$$

⋮

$$y_i = w_0 h_0(\mathbf{x}_i) + w_1 h_1(\mathbf{x}_i) + \varepsilon_i$$

⋮

$$y_i = w_0 h_0(\mathbf{x}_i) + w_1 h_1(\mathbf{x}_i) + \dots + w_D h_D(\mathbf{x}_i) + \varepsilon_i$$

feature 0 ← 0 if "no"
 1 if "yes"
feature 1 ... feature D

[0 0 0 ... 0 0 0]

[1 0 0 ... 0 0 0]

[0 1 0 ... 0 0 0]

⋮

[1 1 0 ... 0 0 0]

⋮

[1 1 1 ... 1 1 1]

2 2 2 ··· 2

2^{D+1}

$$2^8 = 256$$

$$2^{30} = 1,073,741,824$$

$$2^{1000} = 1.071509 \times 10^{301}$$

$$2^{100B} = \text{HUGE!!!!!!}$$

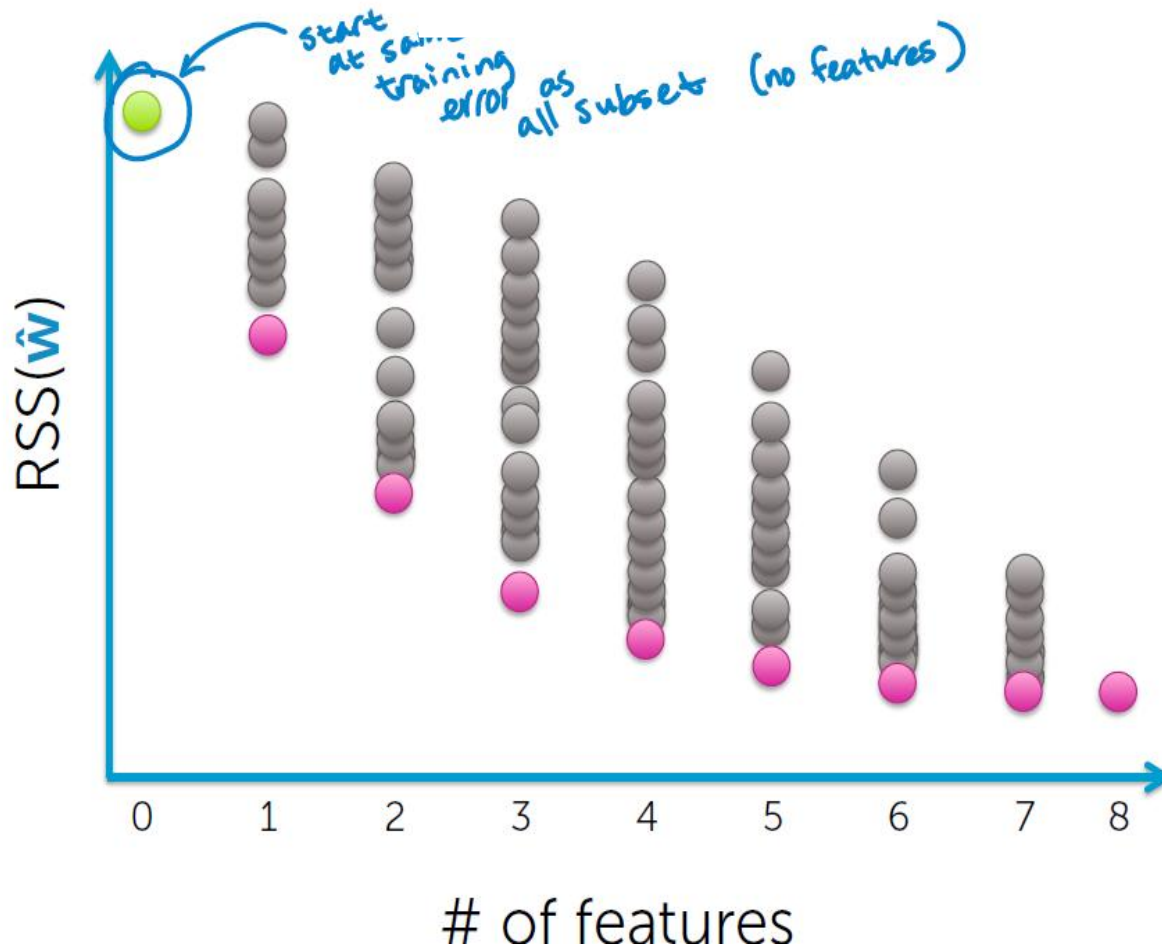
Typically,
computationally
infeasible

Greedy algorithm

Forward stepwise algorithm

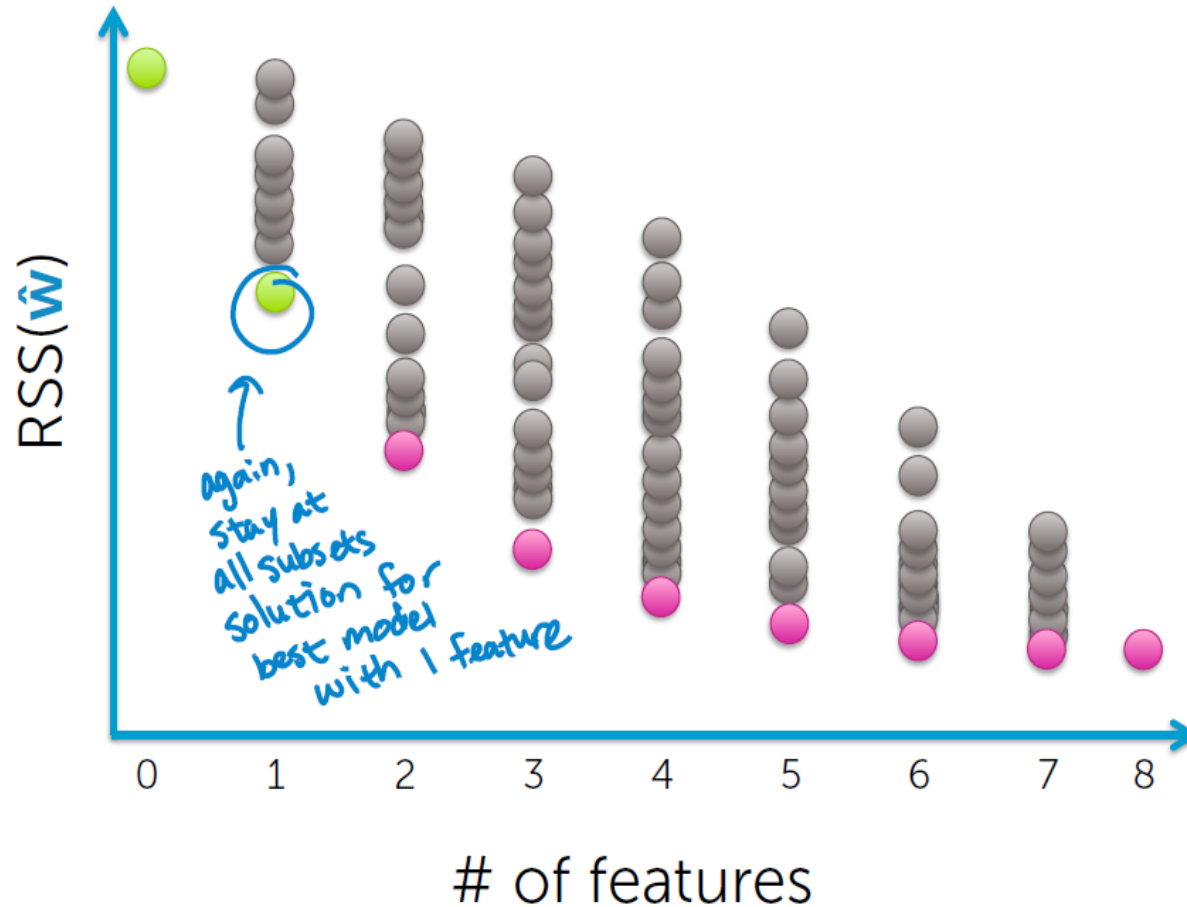
1. Pick a dictionary of features $\{h_0(\mathbf{x}), \dots, h_D(\mathbf{x})\}$
 - e.g., polynomials for linear regression
2. Greedy heuristic:
 - i. Start with empty set of features $F_0 = \emptyset$
(or simple set, like just $h_0(\mathbf{x})=1 \rightarrow y_i = w_0 + \varepsilon_i$)
 - ii. Fit model using current feature set F_t to get $\hat{\mathbf{w}}^{(t)}$
 - iii. Select next best feature $h_{j^*}(\mathbf{x})$
 - e.g., $h_j(\mathbf{x})$ resulting in lowest training error
when learning with $F_t + \{h_j(\mathbf{x})\}$
 - iv. Set $F_{t+1} \leftarrow F_t + \{h_{j^*}(\mathbf{x})\}$
 - v. Recurse

Visualizing greedy algorithm



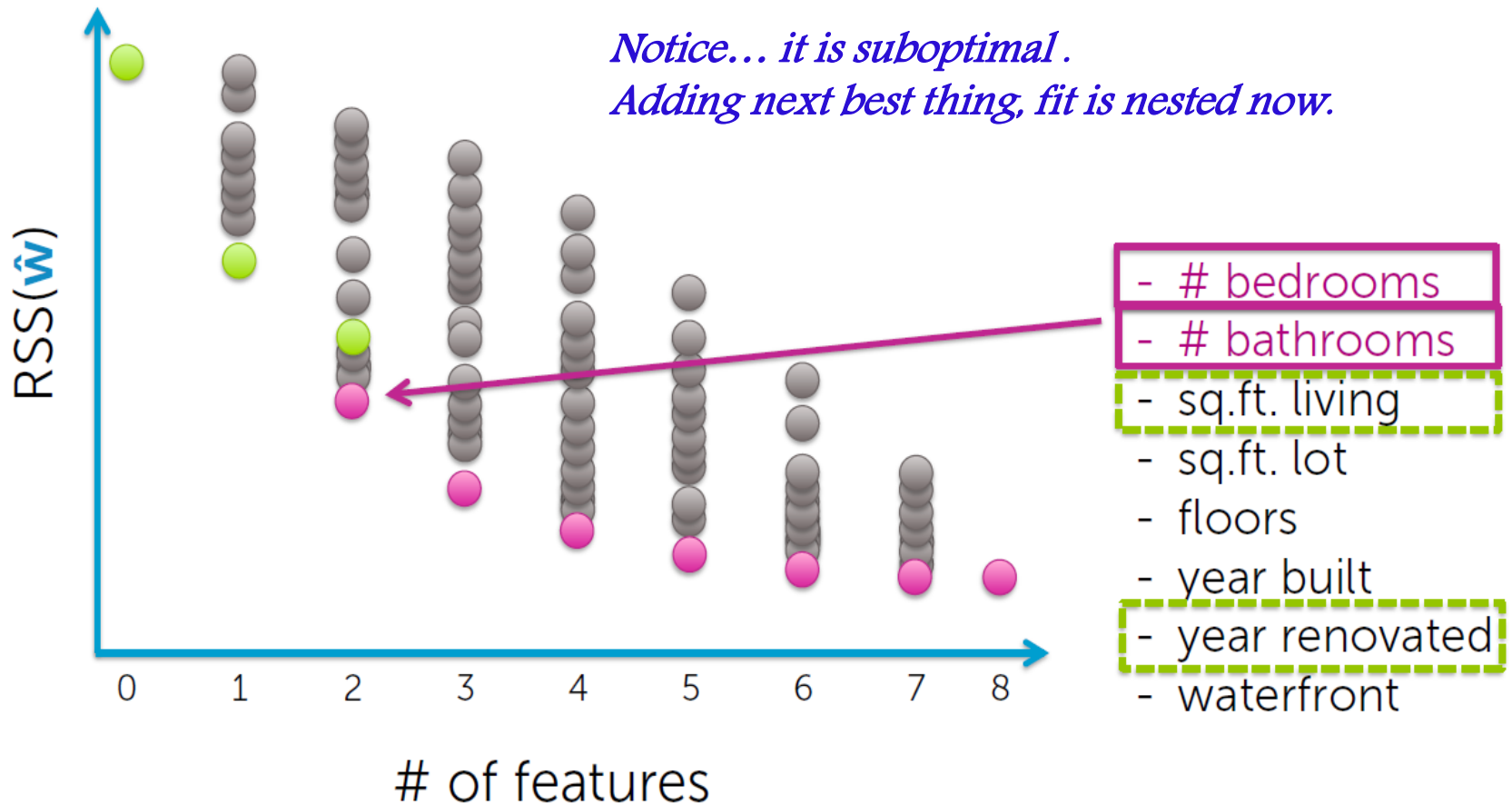
- # bedrooms
- # bathrooms
- sq.ft. living
- sq.ft. lot
- floors
- year built
- year renovated
- waterfront

Visualizing greedy algorithm

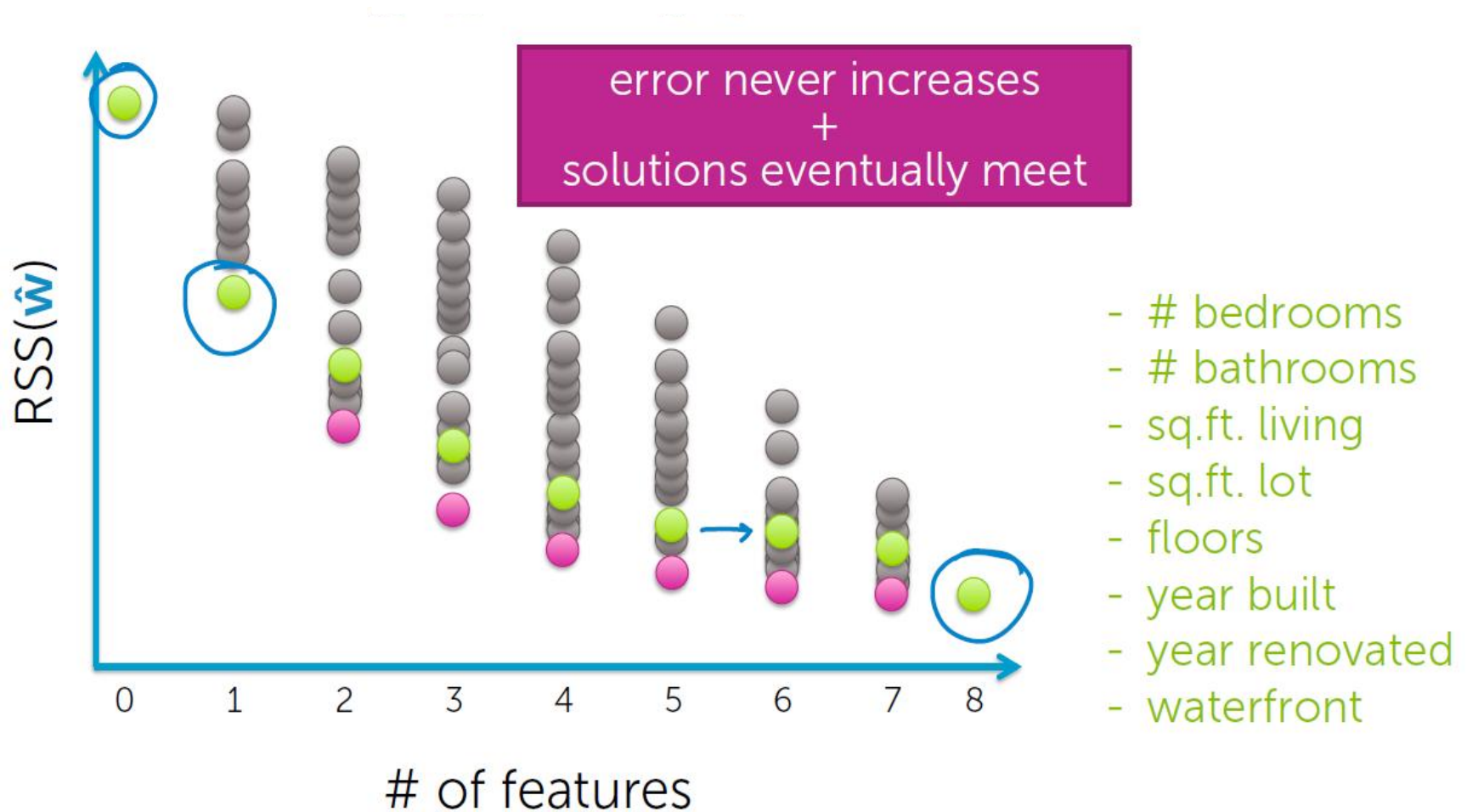


- # bedrooms
- # bathrooms
- sq.ft. living
- sq.ft. lot
- floors
- year built
- year renovated
- waterfront

Visualizing greedy algorithm



Visualizing greedy algorithm



When do we stop?

When **training error** is low enough?

No!

When **test error** is low enough?

No!

Use *validation set* or *cross validation*!

Complexity of forward stepwise

How many models were evaluated?

- 1st step, D models
- 2nd step, $D-1$ models (add 1 feature out of $D-1$ possible)
- 3rd step, $D-2$ models (add 1 feature out of $D-2$ possible)
- ...

How many steps?

- Depends
- At most D steps (to full model)

$$O(D^2) \ll 2^D$$

for large D

Other greedy algorithms

Instead of starting from simple model and always growing...

Backward stepwise:

Start with full model and iteratively remove features least useful to fit

Combining forward and backward steps:

In forward algorithm, insert steps to remove features no longer as important

Lots of other variants, too.

Using regularisation for features selection

Instead of searching over a **discrete** set of solutions, can we use **regularization**?

- Start with full model (all possible features)
- “Shrink” some coefficients **exactly to 0**
 - i.e., knock out certain features
- Non-zero coefficients indicate “selected” features

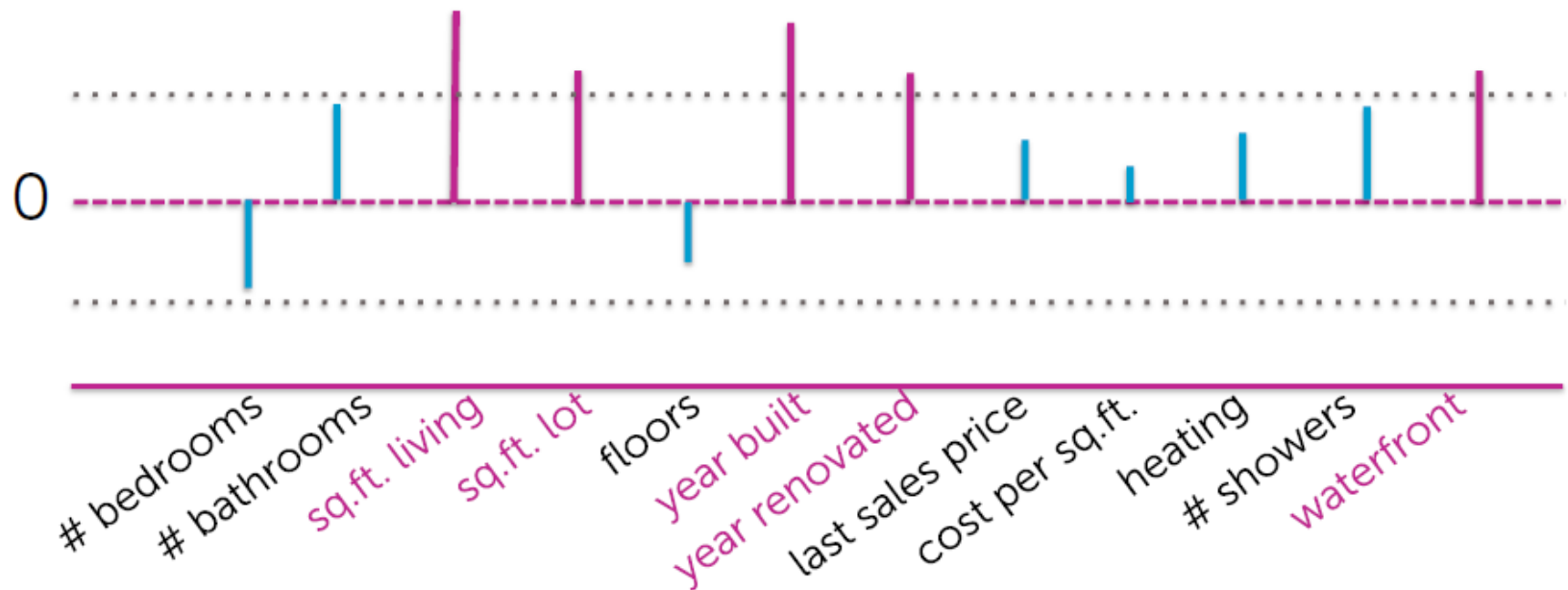
Thresholding ridge coefficients?

Why don't we just set small ridge coefficients to 0?



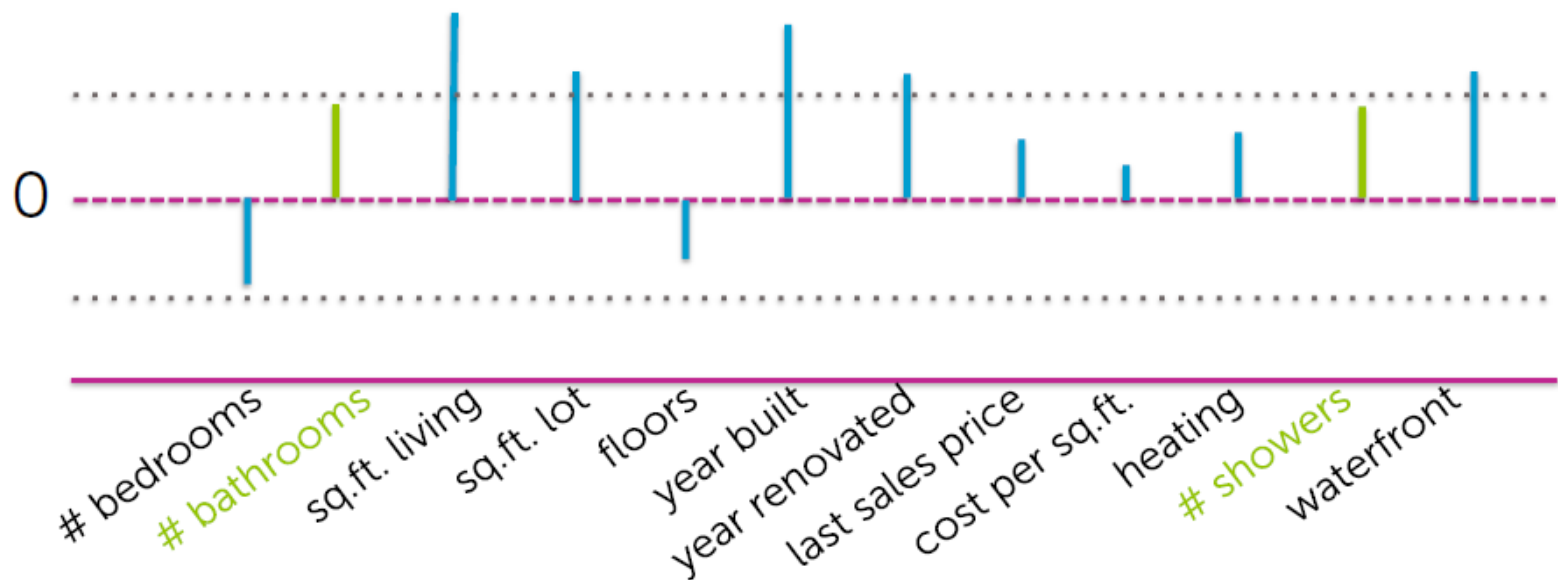
Thresholding ridge coefficients?

Selected features for a given threshold value



Thresholding ridge coefficients?

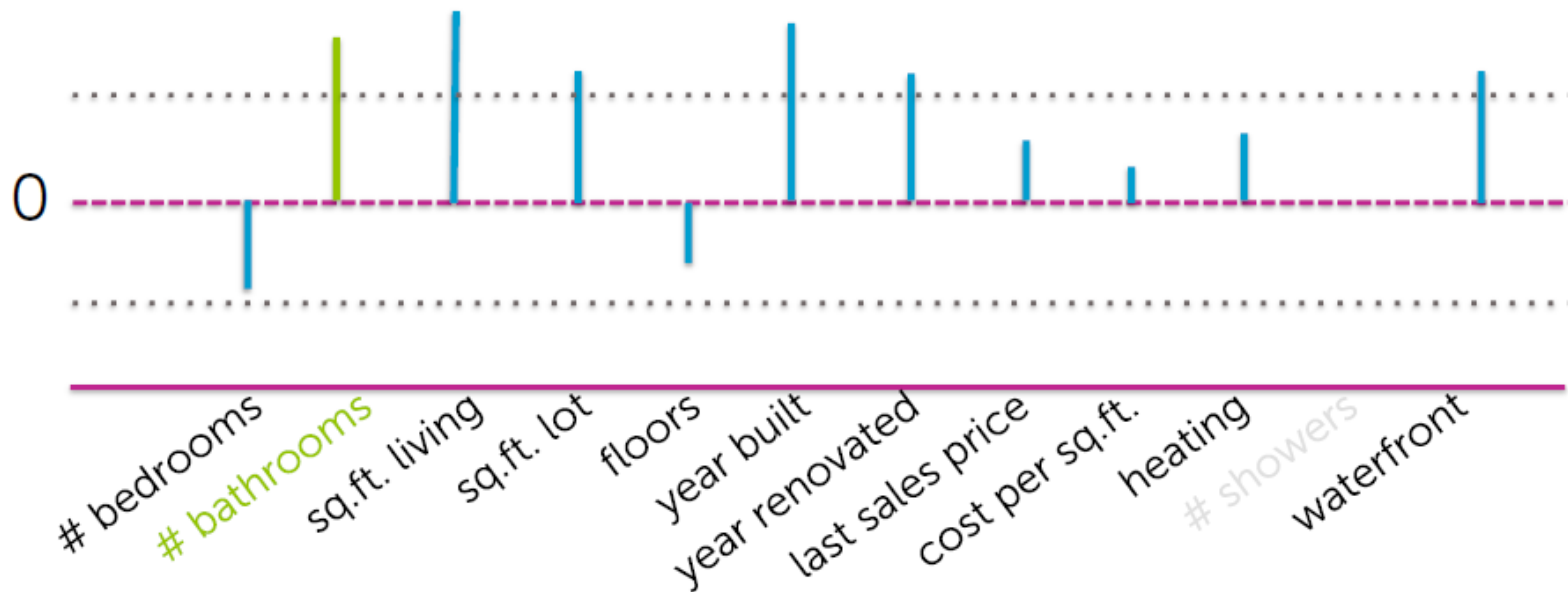
Let's look at two related features...



Nothing measuring bathrooms was included!

Thresholding ridge coefficients?

If only one of the features had been included...

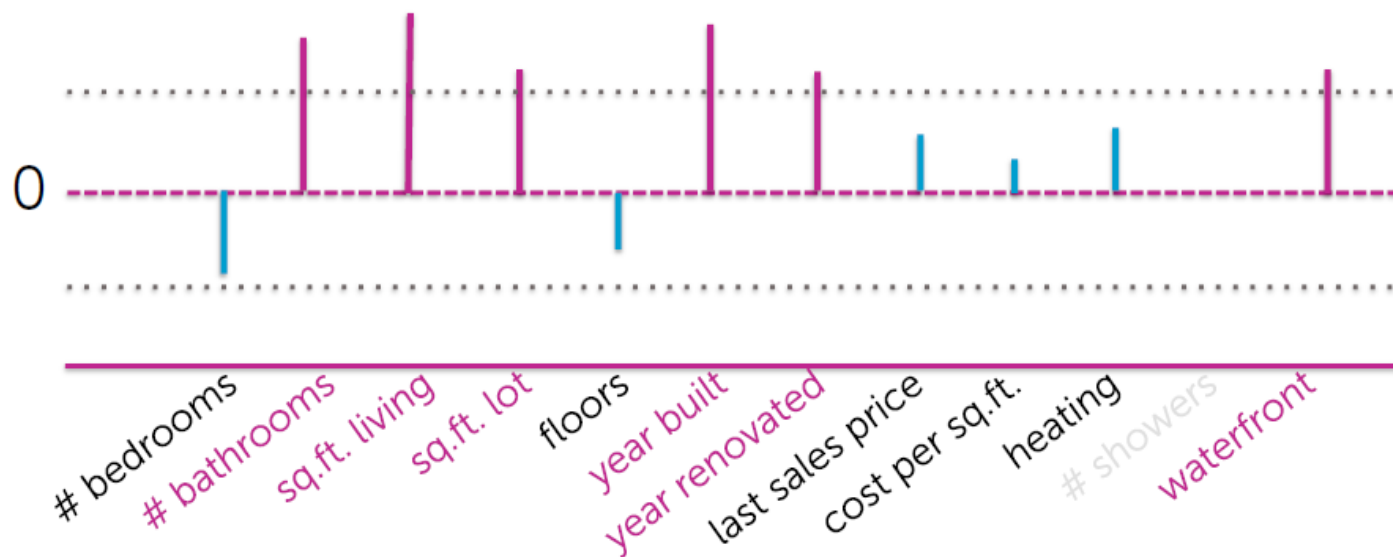


Remember:

this is linear model. If we assume that #showers = #bathrooms and remove one of them from the model, coefficients will sum up.

Thresholding ridge coefficients?

Would have included bathrooms in selected model



Can regularization lead directly to sparsity?

Try this cost instead of ridge ...

Total cost =

measure of fit + λ measure of magnitude of coefficients
RSS($\mathbf{w}$)

$$\|\mathbf{w}\|_1 = |w_0| + \dots + |w_D|$$

Lasso regression
(a.k.a. L_1 regularized regression)

Leads to
sparse
solutions!

Lasso regression

Just like ridge regression, solution is governed by a continuous parameter λ

$$\text{RSS}(\mathbf{w}) + \lambda \|\mathbf{w}\|_1$$

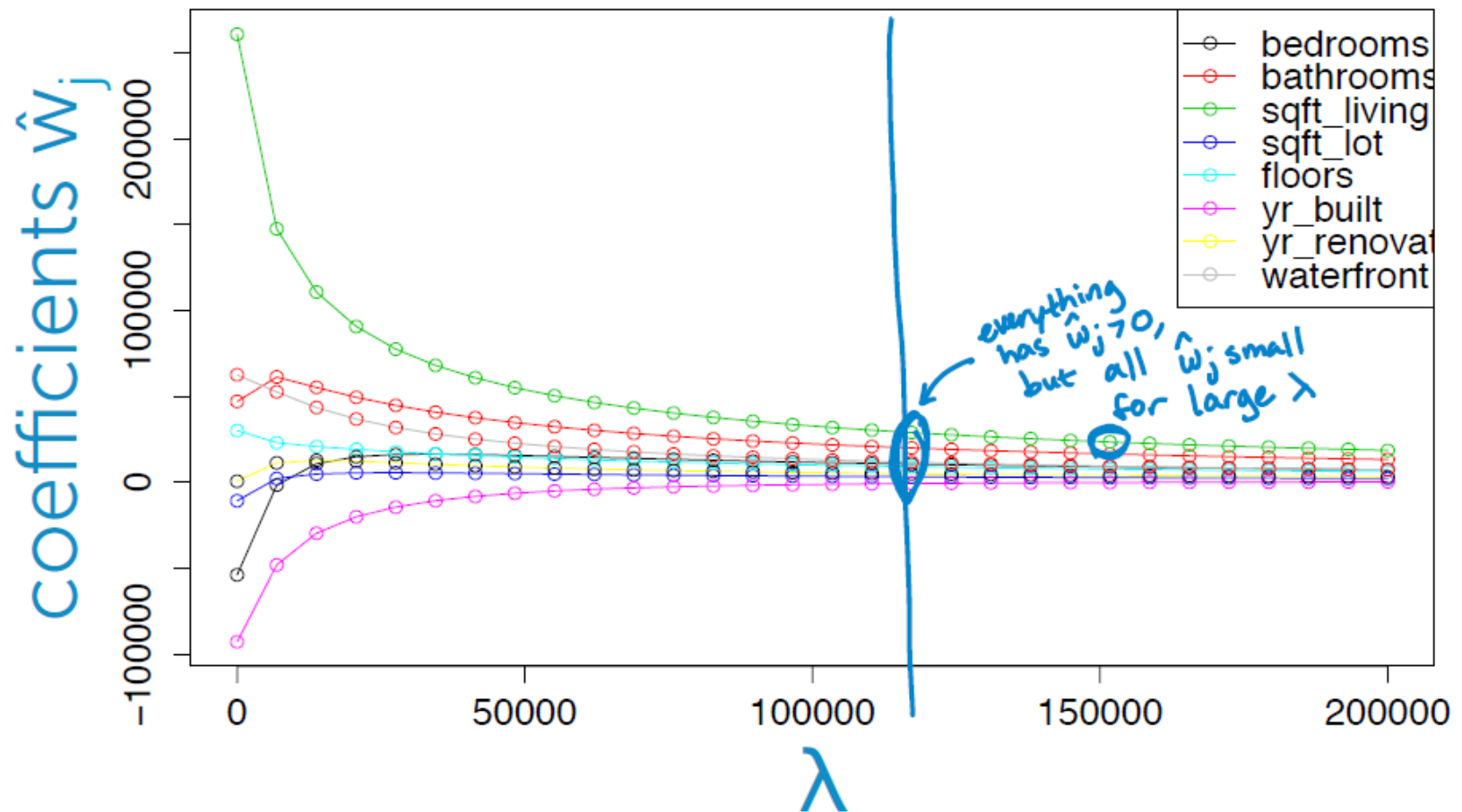
 tuning parameter = balance of fit and sparsity

If $\lambda=0$: $\hat{\mathbf{w}}^{\text{lasso}} = \hat{\mathbf{w}}^{\text{LS}}$ (unregularized solution)

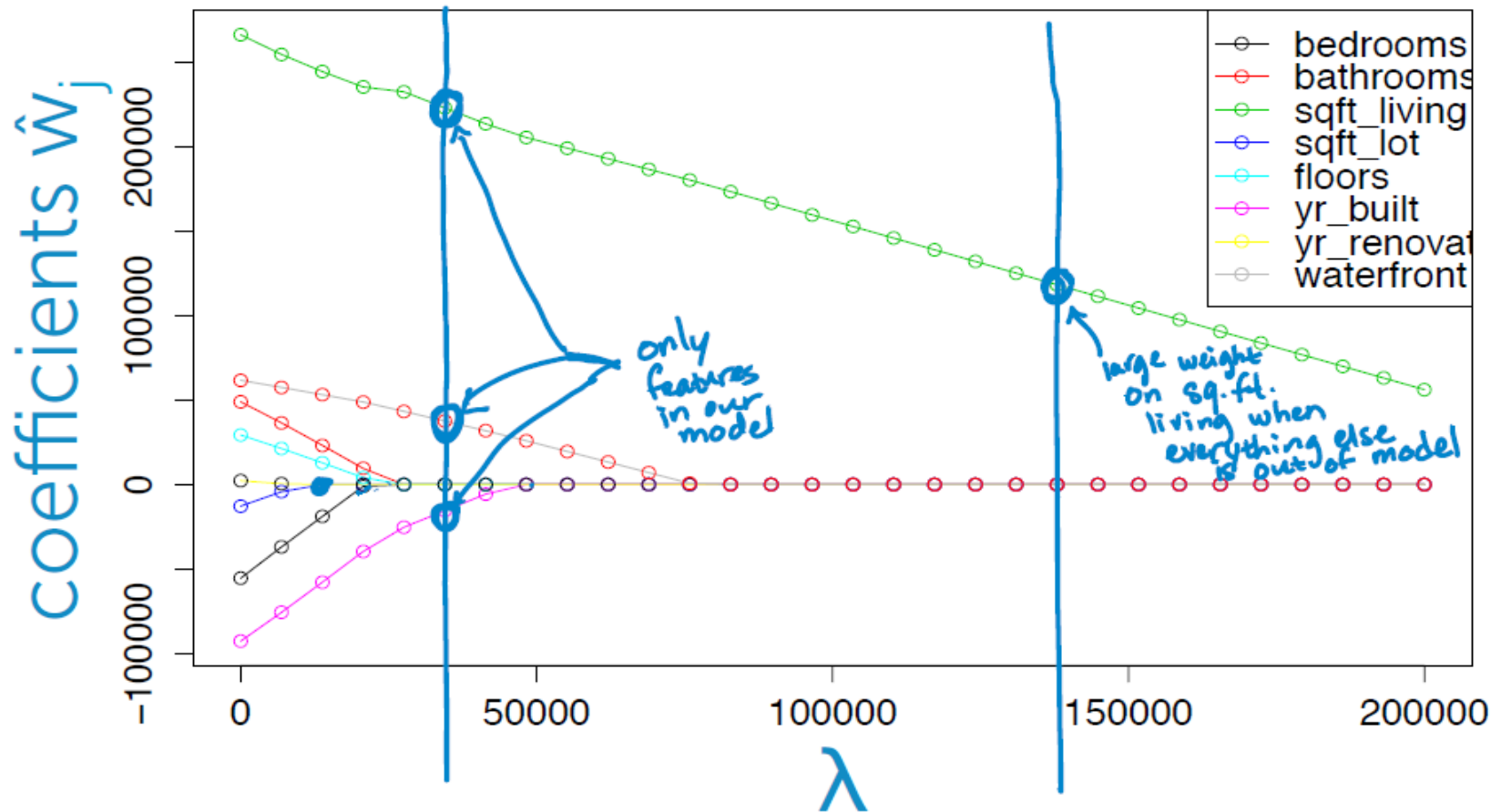
If $\lambda=\infty$: $\hat{\mathbf{w}}^{\text{lasso}} = \mathbf{0}$

If λ in between: $0 \leq \|\hat{\mathbf{w}}^{\text{lasso}}\|_1 \leq \|\hat{\mathbf{w}}^{\text{LS}}\|_1$

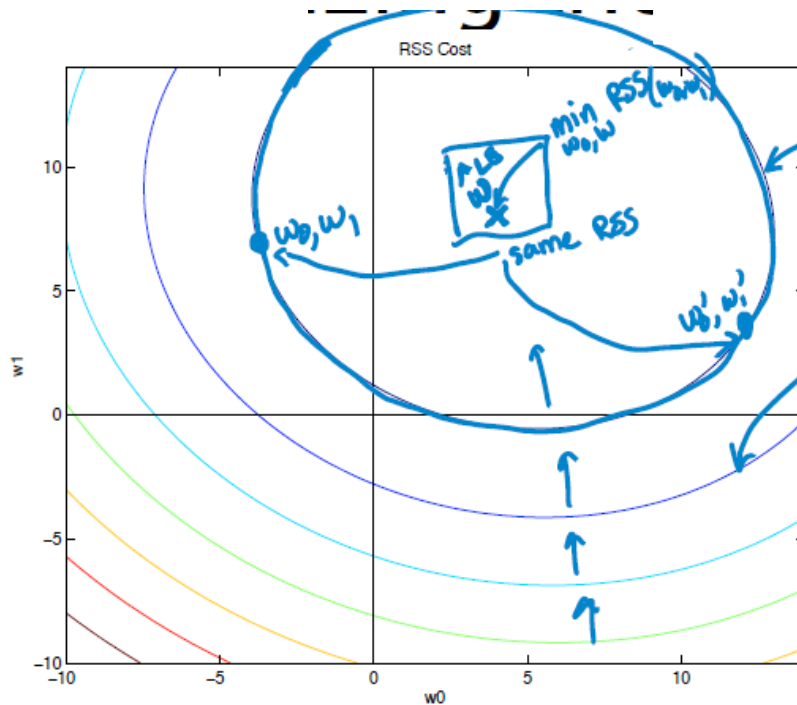
Coefficient path: ridge



Coefficient path: lasso



Visualising ridge cost in 2D



$RSS(w_0, w_1) = \text{const}_1$

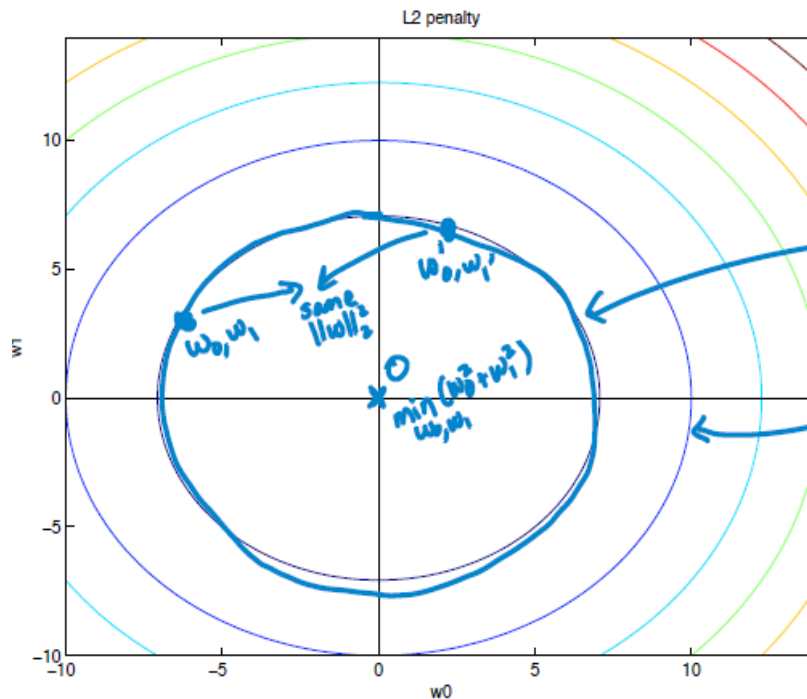
$RSS(w_0, w_1) = \text{const}_2 > \text{const}_1$
 $\vdots$

$\sum_i y_i^2 + \underline{w_0^2 h_0^2} + \underline{w_1^2 h_1^2} + \text{cross terms} = \text{constant} +$
ellipse

2 features for visualization sake

$$RSS(\mathbf{w}) + \lambda ||\mathbf{w}||_2^2 = \sum_{i=1}^N (y_i - w_0 h_0(\mathbf{x}_i) - w_1 h_1(\mathbf{x}_i))^2 + \lambda (w_0^2 + w_1^2)$$

Visualising ridge cost in 2D



$$\|w\|_2^2 = \text{const}_1$$

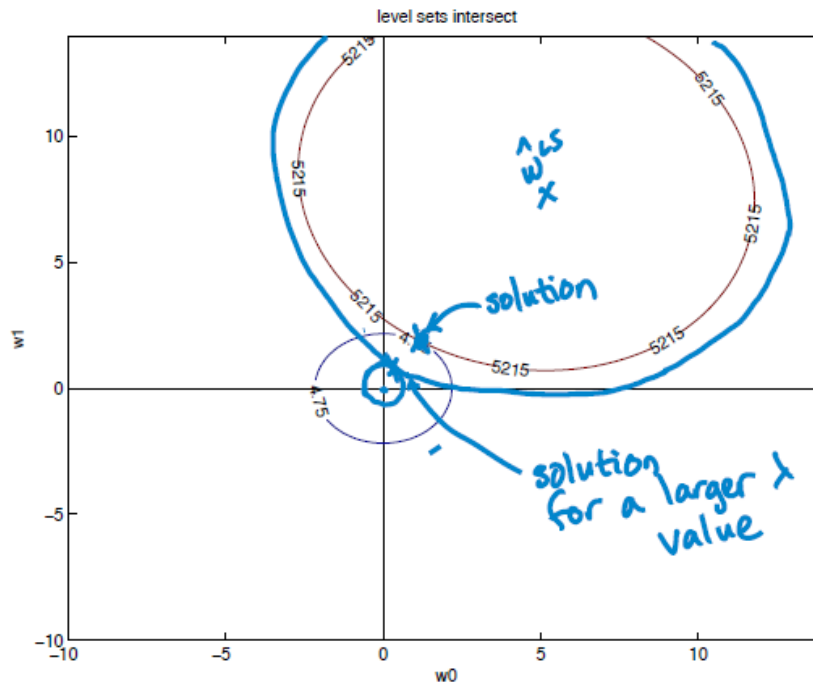
$$\|w\|_2^2 = \text{const}_2 > \text{const}_1$$

$$w_0^2 + w_1^2 = \text{constant}$$

circle

$$\text{RSS}(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2 = \sum_{i=1}^N (y_i - w_0 h_0(\mathbf{x}_i) - w_1 h_1(\mathbf{x}_i))^2 + \lambda (w_0^2 + w_1^2)$$

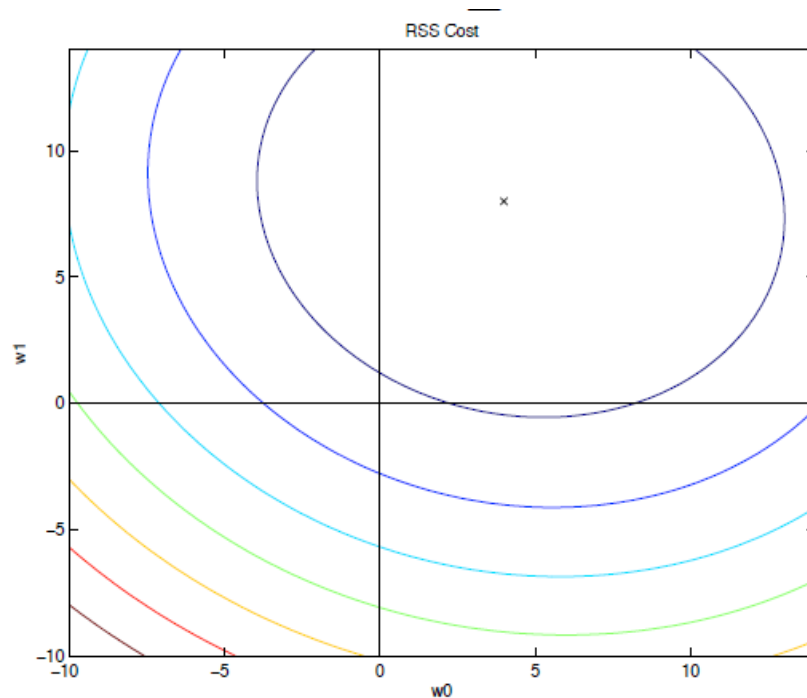
Visualising ridge cost in 2D



For a specific λ value,
some balance between
RSS and $\|w\|_2^2$

$$\text{RSS}(\mathbf{w}) + \lambda \|\mathbf{w}\|_2^2 = \sum_{i=1}^N (y_i - w_0 h_0(\mathbf{x}_i) - w_1 h_1(\mathbf{x}_i))^2 + \lambda (w_0^2 + w_1^2)$$

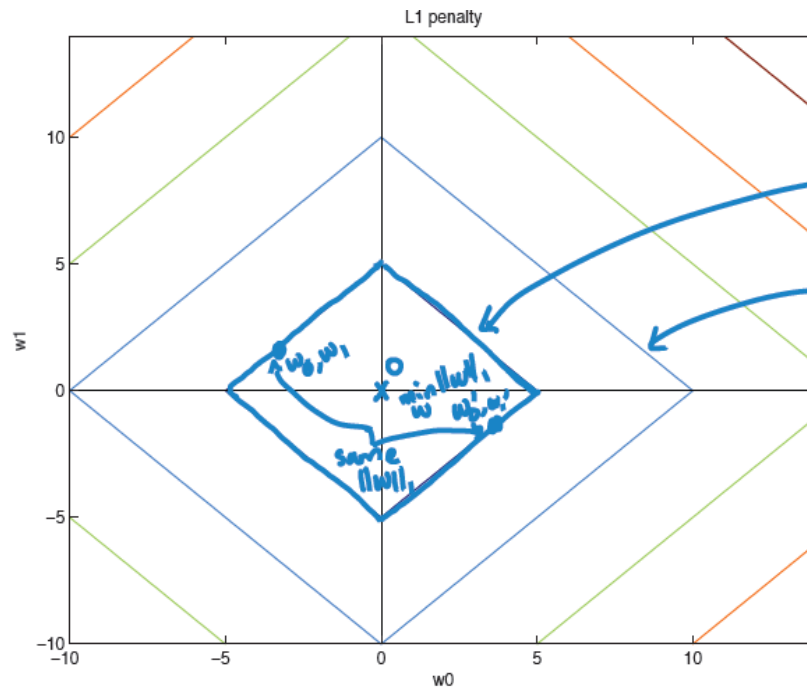
Visualising lasso cost in 2D



RSS contours for
lasso are exactly
the same as
those for ridge!

$$\text{RSS}(\mathbf{w}) + \lambda \|\mathbf{w}\|_1 = \sum_{i=1}^N (y_i - w_0 h_0(\mathbf{x}_i) - w_1 h_1(\mathbf{x}_i))^2 + \lambda (|w_0| + |w_1|)$$

Visualising lasso cost in 2D

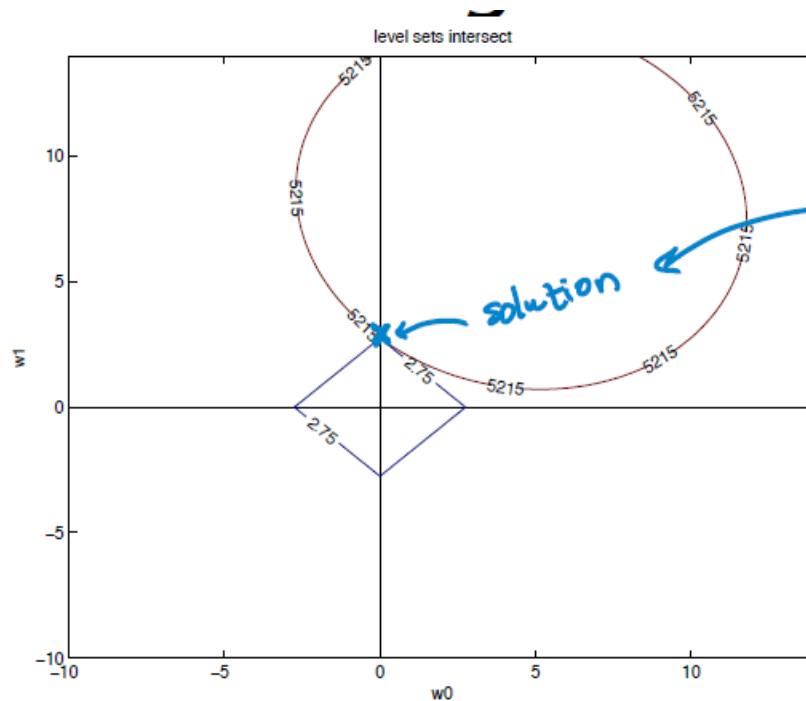


$\|w\|_1 = \text{const}_1$
 $\|w\|_1 = \text{const}_2 > \text{const}_1$
 $\vdots$

$|w_0| + |w_1| = \text{constant}$
diamond

$$\text{RSS}(\mathbf{w}) + \lambda \|\mathbf{w}\|_1 = \sum_{i=1}^N (y_i - w_0 h_0(\mathbf{x}_i) - w_1 h_1(\mathbf{x}_i))^2 + \lambda (|w_0| + |w_1|)$$

Visualising lasso cost in 2D



For a specific value of λ ,

We are getting sparse solution,
the $w_0 = 0$

$$\text{RSS}(\mathbf{w}) + \lambda \|\mathbf{w}\|_1 = \sum_{i=1}^N (y_i - w_0 h_0(\mathbf{x}_i) - w_1 h_1(\mathbf{x}_i))^2 + \lambda (|w_0| + |w_1|)$$

How we optimise for objective

To solve for $\hat{\mathbf{w}}$, previously took gradient of total cost objective and either:

- 1) Derived closed-form solution
- 2) Used in gradient descent algorithm

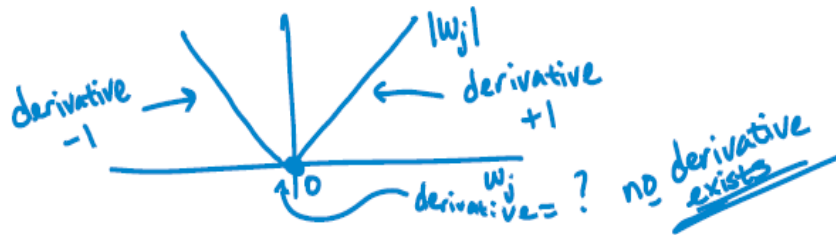
Optimise for lasso objective

Lasso total cost: $\text{RSS}(\mathbf{w}) + \lambda \|\mathbf{w}\|_1$

$$\sum_{j=0}^p |w_j|$$

Issues:

- 1) What's the derivative of $|w_j|$?



gradients $\rightarrow$ subgradients

- 2) Even if we could compute derivative, no closed-form solution

can use subgradient descent

Coordinate descent

Goal: Minimize some function g

$$\min_w g(w)$$

$$g(w) = g(w_0, w_1, \dots, w_D)$$

when keeping others fixed

Often, hard to find minimum for all coordinates, but **easy for each coordinate**

Coordinate descent:

Initialize $\hat{w} = 0$ (or smartly...)

while not converged

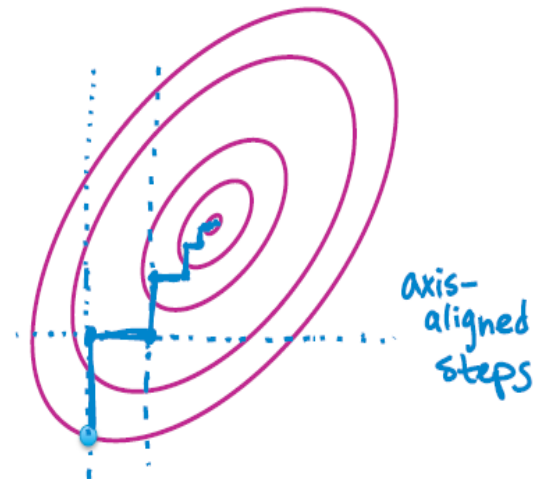
pick a coordinate j

$$\hat{w}_j \leftarrow$$

values from previous iterations

$$\min_w g(\hat{w}_0, \dots, \hat{w}_{j-1}, w, \hat{w}_{j+1}, \dots, \hat{w}_D)$$

just min over j th coordinate



Comments on coordinate descent

How do we pick next coordinate?

- At random ("random" or "stochastic" coordinate descent), round robin, ...

No stepsize to choose!

Super useful approach for *many* problems

- Converges to optimum in some cases (e.g., "strongly convex")
- Converges for lasso objective

Normalizing features

Normalizing features

Scale training columns (not rows!) as:

$$\underline{h}_j(\mathbf{x}_k) = \frac{h_j(\mathbf{x}_k)}{\sqrt{\sum_{i=1}^N h_j(\mathbf{x}_i)^2}}$$

Normalizer: z_j

Apply same training scale factors to test data:

$$\underline{h}_j(\mathbf{x}_k) = \frac{h_j(\mathbf{x}_k)}{\sqrt{\sum_{i=1}^N h_j(\mathbf{x}_i)^2}}$$

apply to test point

Normalizer: z_j

summing over training points



Optimising least squares objective

One coordinate at a time

$$\text{RSS}(\mathbf{w}) = \sum_{i=1}^N \left(y_i - \sum_{j=0}^D w_j \underline{h}_j(\mathbf{x}_i) \right)^2$$

normalized features

Fix all coordinates $\mathbf{w}_{-j}$ and take partial w.r.t. w_j

all w_k for $k \neq j$

1d optimization coordinate by coordinate

$$\begin{aligned} \frac{\partial}{\partial w_j} \text{RSS}(\mathbf{w}) &= -2 \sum_{i=1}^N \underline{h}_j(\mathbf{x}_i) \left(y_i - \sum_{j=0}^D w_j \underline{h}_j(\mathbf{x}_i) \right) \\ &= -2 \sum_{i=1}^N \underline{h}_j(\mathbf{x}_i) \left(y_i - \sum_{k \neq j} w_k \underline{h}_k(\mathbf{x}_i) - w_j \underline{h}_j(\mathbf{x}_i) \right) \\ &= -2 \underbrace{\sum_{i=1}^N \underline{h}_j(\mathbf{x}_i) \left(y_i - \sum_{k \neq j} w_k \underline{h}_k(\mathbf{x}_i) \right)}_{\triangleq p_j} + 2 w_j \underbrace{\sum_{i=1}^N \underline{h}_j(\mathbf{x}_i)^2}_{\substack{\text{by definition of normalized features,} \\ = 1}} \\ &= -2 p_j + 2 w_j \end{aligned}$$

Optimising least squares objective

One coordinate at a time

$$\text{RSS}(\mathbf{w}) = \sum_{i=1}^N \left(y_i - \sum_{j=0}^D w_j h_j(\mathbf{x}_i) \right)^2$$

Set partial = 0 and solve

$$\frac{\partial}{\partial w_j} \text{RSS}(\mathbf{w}) = -2\rho_j + 2w_j = 0$$

$\hat{w}_j = \rho_j$

Coordinate descent for least squares regression

Initialize $\hat{\mathbf{w}} = 0$ (or smartly...)

while not converged

for $j=0,1,\dots,D$

compute:

$$\rho_j = \sum_{i=1}^N \underline{h}_j(\mathbf{x}_i) \overbrace{(y_i - \hat{y}_i(\hat{\mathbf{w}}_{-j}))}^{\text{residual without feature } j}$$

set: $\hat{\mathbf{w}}_j = \rho_j$

prediction
without feature j

Measure of the correlation between w_j
and the residual without this feature.

How to access convergence

Initialize $\hat{\mathbf{w}} = 0$ (or smartly...)

while not converged

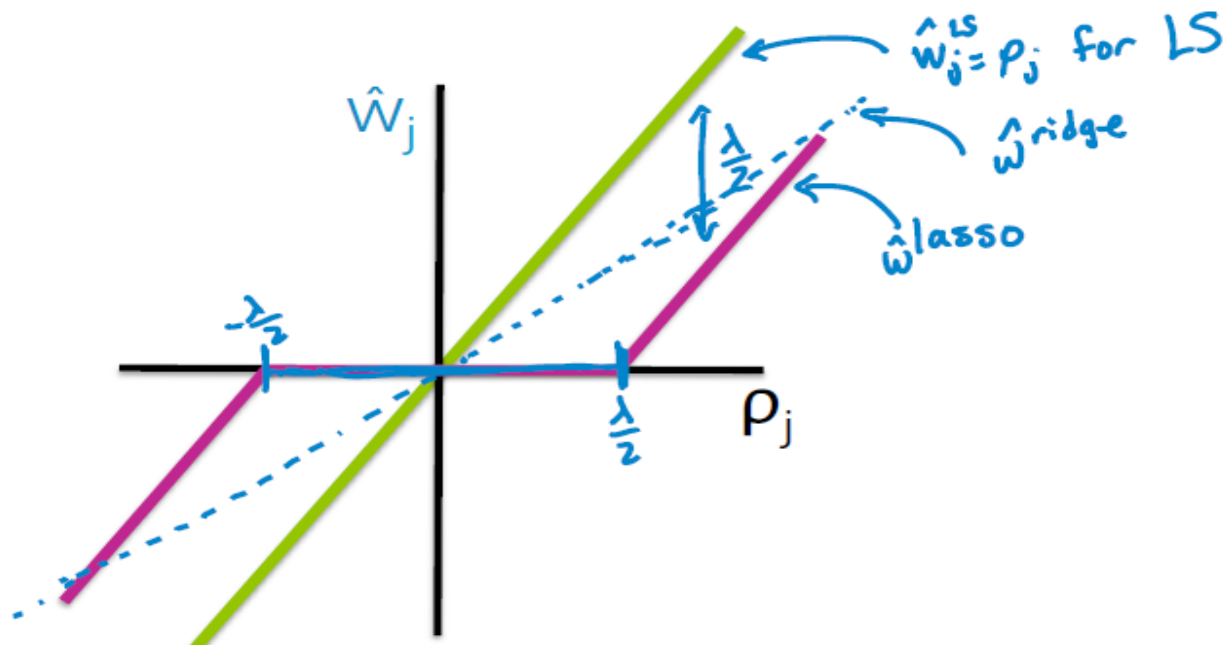
for $j=0,1,\dots,D$

compute: $\rho_j = \sum_{i=1}^N \underline{h}_j(\mathbf{x}_i)(y_i - \hat{y}_i(\hat{\mathbf{w}}_{-j}))$

set: $\hat{w}_j = \begin{cases} \rho_j + \lambda/2 & \text{if } \rho_j < -\lambda/2 \\ 0 & \text{if } \rho_j \text{ in } [-\lambda/2, \lambda/2] \\ \rho_j - \lambda/2 & \text{if } \rho_j > \lambda/2 \end{cases}$

Soft thresholding

$$\hat{w}_j = \begin{cases} \rho_j + \lambda/2 & \text{if } \rho_j < -\lambda/2 \\ 0 & \text{if } \rho_j \text{ in } [-\lambda/2, \lambda/2] \\ \rho_j - \lambda/2 & \text{if } \rho_j > \lambda/2 \end{cases}$$



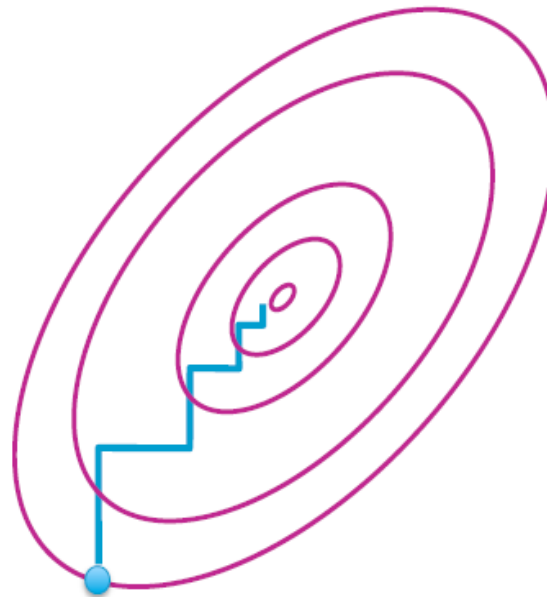
Convergence criteria

When to stop?

For convex problems, will start to take **smaller and smaller steps**

Measure size of steps taken in a full loop over all features

– stop when **$\max \text{ step} < \epsilon$**



Other lasso solvers

Classically: Least angle regression (LARS) [Efron et al. '04]

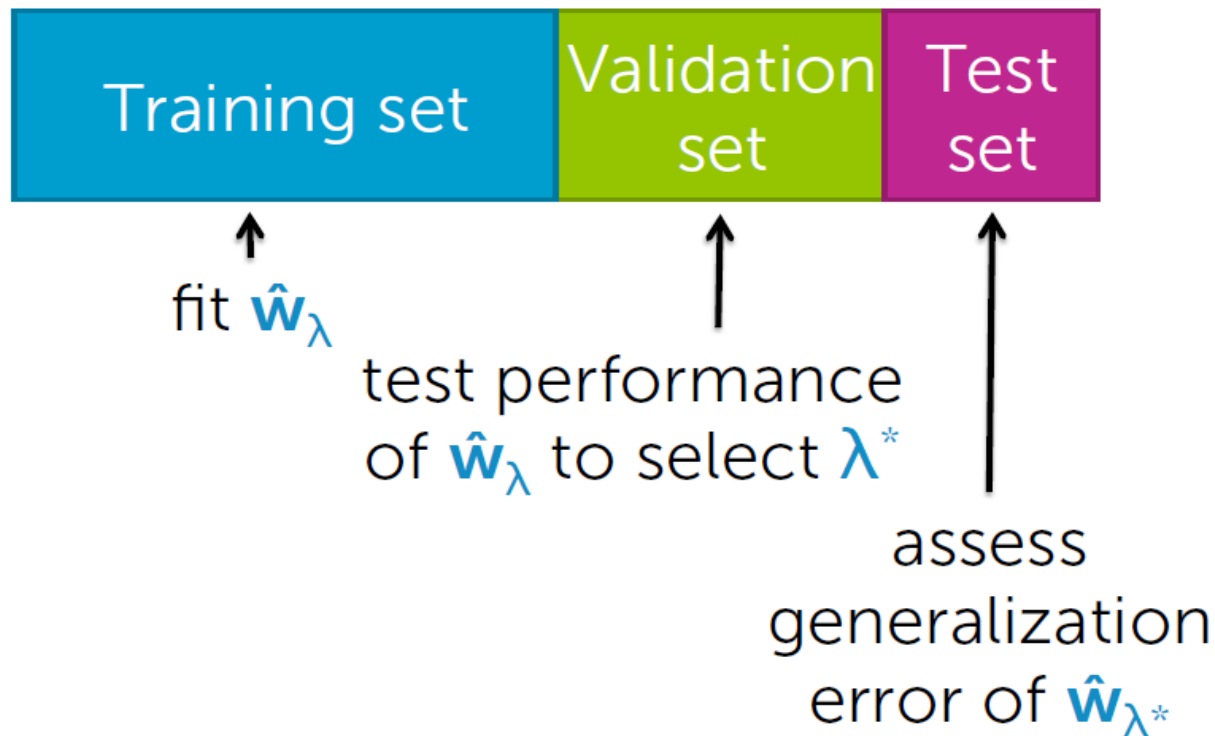
Then: Coordinate descent algorithm
[Fu '98, Friedman, Hastie, & Tibshirani '08]

Now:

- Parallel CD (e.g., Shotgun, [Bradley et al. '11])
- Other parallel learning approaches for linear models
 - Parallel stochastic gradient descent (SGD) (e.g., Hogwild! [Niu et al. '11])
 - Parallel independent solutions then averaging [Zhang et al. '12]
- Alternating directions method of multipliers (ADMM) [Boyd et al. '11]

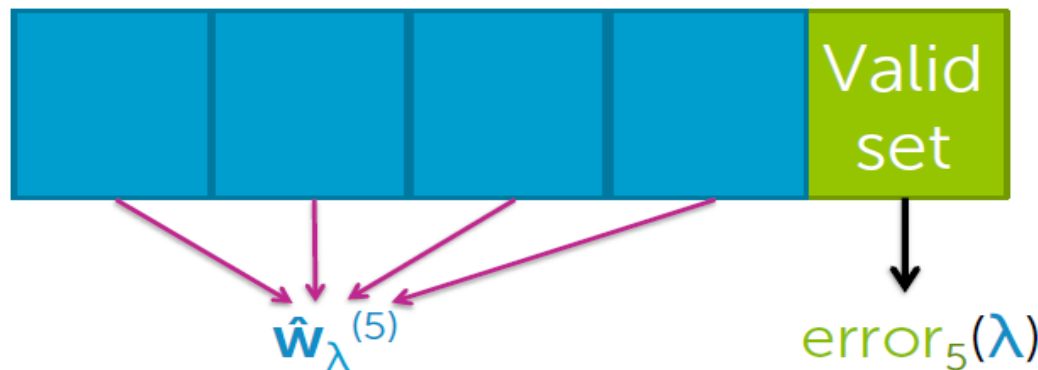
How do we choose λ

If sufficient amount of data...



How do we choose λ

K-fold cross validation



For $k=1, \dots, K$

1. Estimate $\hat{\mathbf{w}}_{\lambda}^{(k)}$ on the training blocks
2. Compute error on validation block: $\text{error}_k(\lambda)$

Compute average error: $\text{CV}(\lambda) = \frac{1}{K} \sum_{k=1}^K \text{error}_k(\lambda)$

How do we chose λ

Choosing λ via cross validation

Cross validation is choosing the λ that provides best predictive accuracy

Tends to favor less sparse solutions, and thus smaller λ , than optimal choice for feature selection

c.f., "Machine Learning: A Probabilistic Perspective",
Murphy, 2012 for further discussion

Impact of feature selection and lasso

Lasso has changed machine learning, statistics, & electrical engineering

But, for feature selection in general, be careful about interpreting selected features

- selection only considers features included
- sensitive to correlations between features
- result depends on algorithm used
- there are theoretical guarantees for lasso under certain conditions

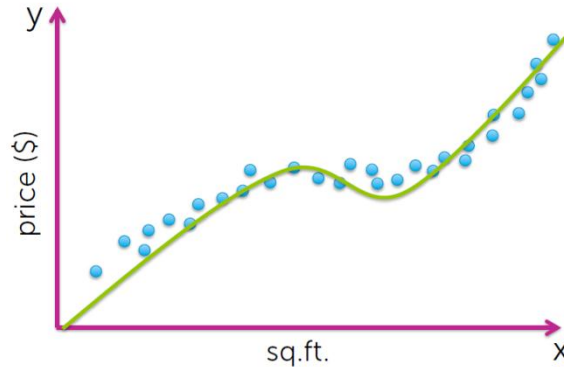
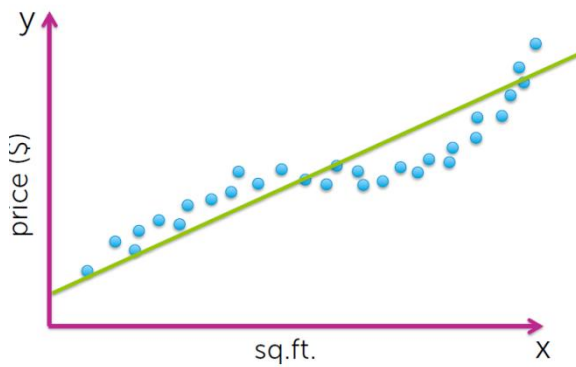
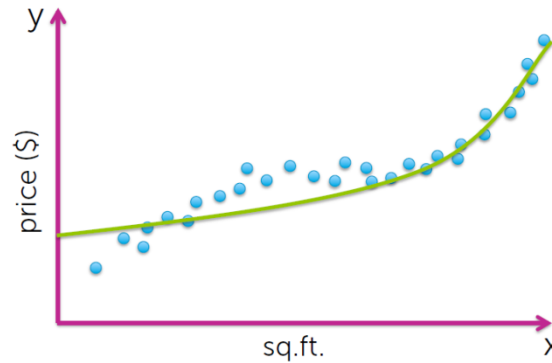
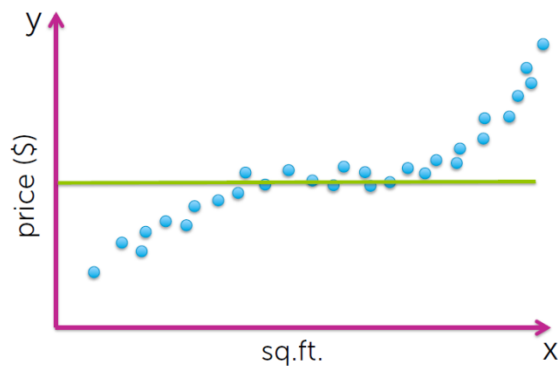
What you can do now

- Perform feature selection using “all subsets” and “forward stepwise” algorithms
- Analyze computational costs of these algorithms
- Contrast greedy and optimal algorithms
- Formulate lasso objective
- Describe what happens to estimated lasso coefficients as tuning parameter λ is varied
- Interpret lasso coefficient path plot
- Contrast ridge and lasso regression
- Describe geometrically why L1 penalty leads to sparsity
- Estimate lasso regression parameters using an iterative coordinate descent algorithm
- Implement K-fold cross validation to select lasso tuning parameter λ

NONPARAMETRIC REGRESSION

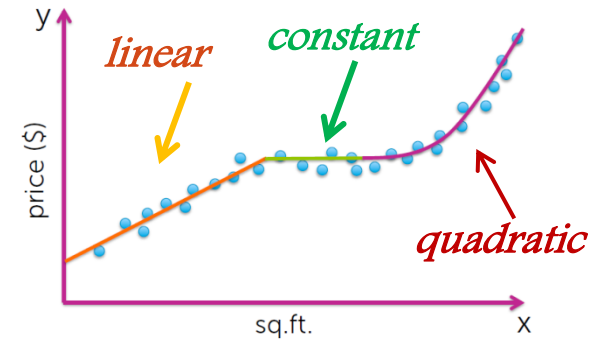
Fit globally vs fit locally

Parametric models



Below ...

*$f(x)$ is not really
a polynomial function*



What alternative do we have?

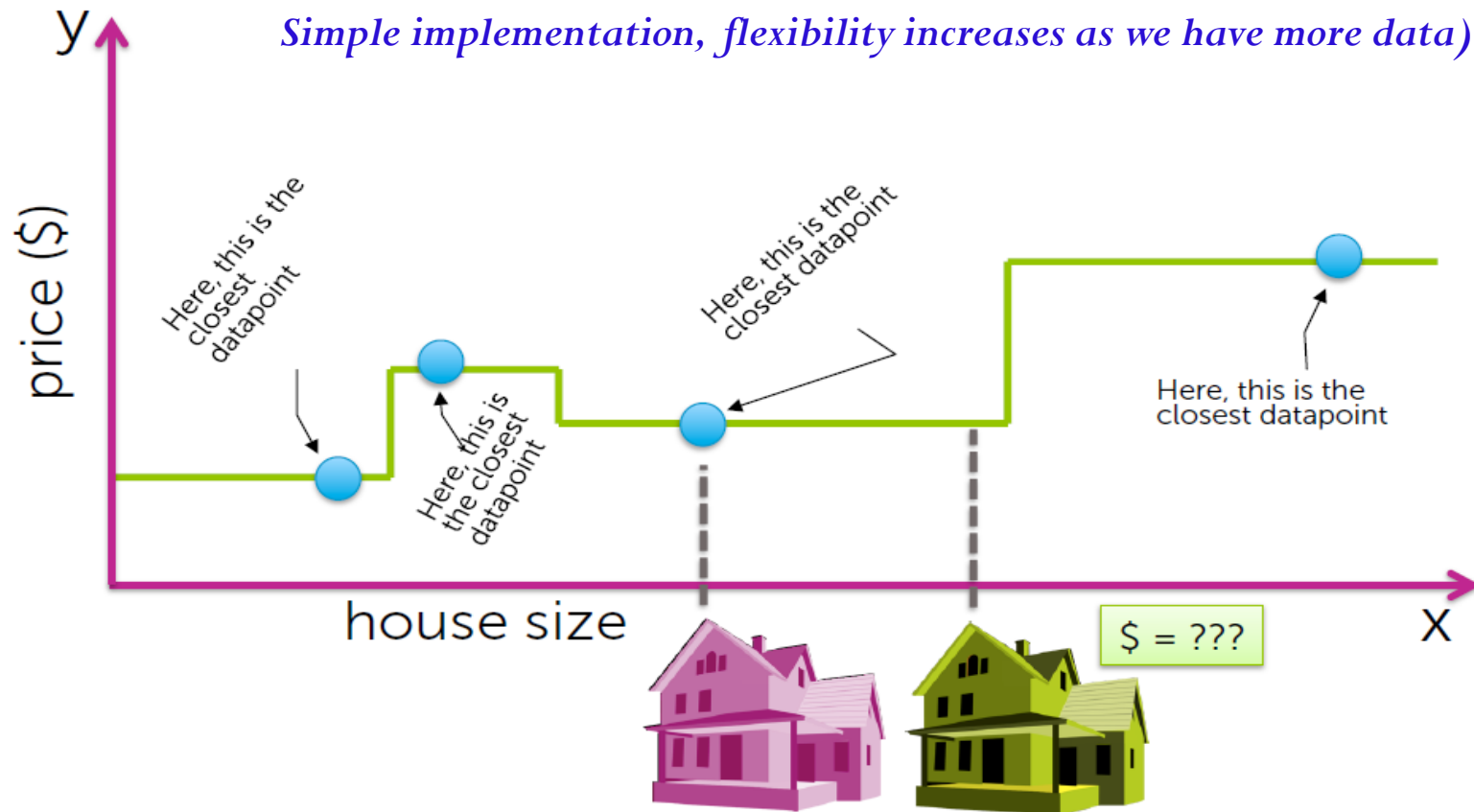
If we:

- Want to allow flexibility in $f(\mathbf{x})$ having local structure
- Don't want to infer “structural breaks”

What's a simple option we have?

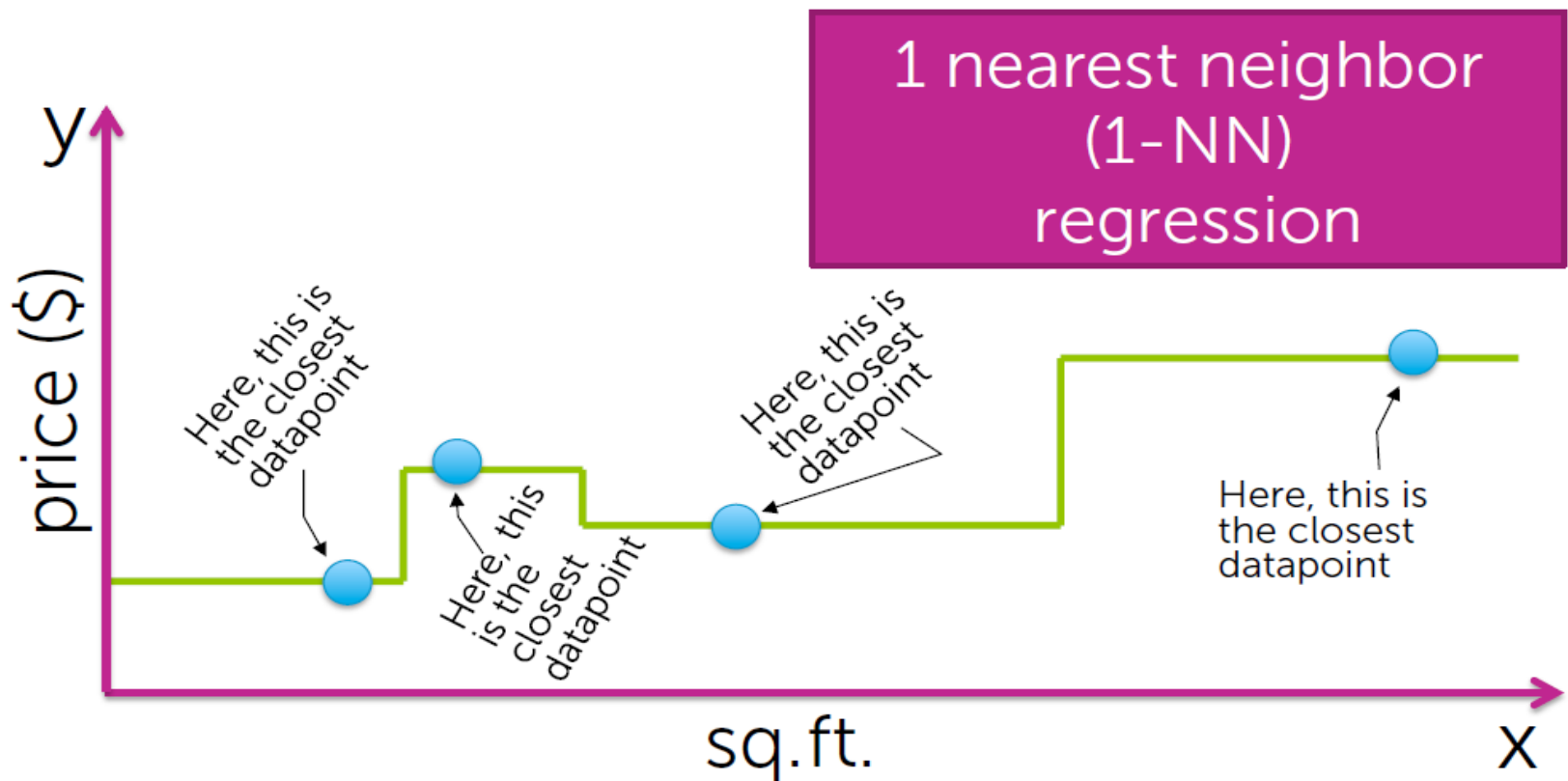
- Assuming we have plenty of data...

Nearest Neighbor & Kernel Regression (nonparametric approach)



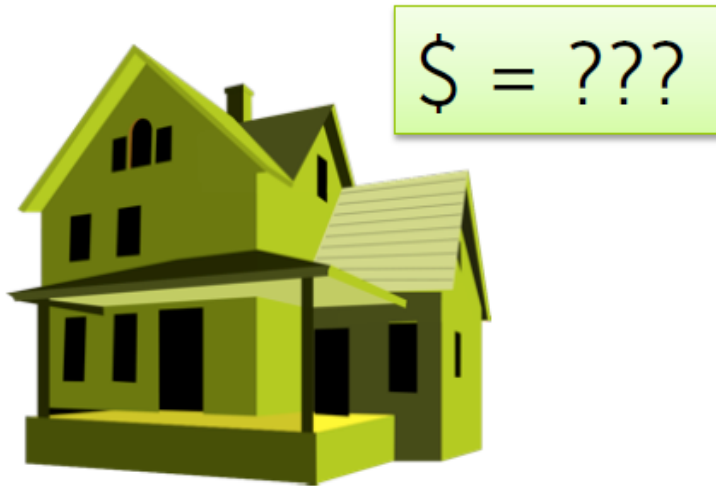
Fit locally to each data point

Predicted value = "closest" y_i



What people do naturally...

Real estate agent assesses value by finding sale of most similar house



1-NN regression more formally

Dataset of (🏠, \$) pairs: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$

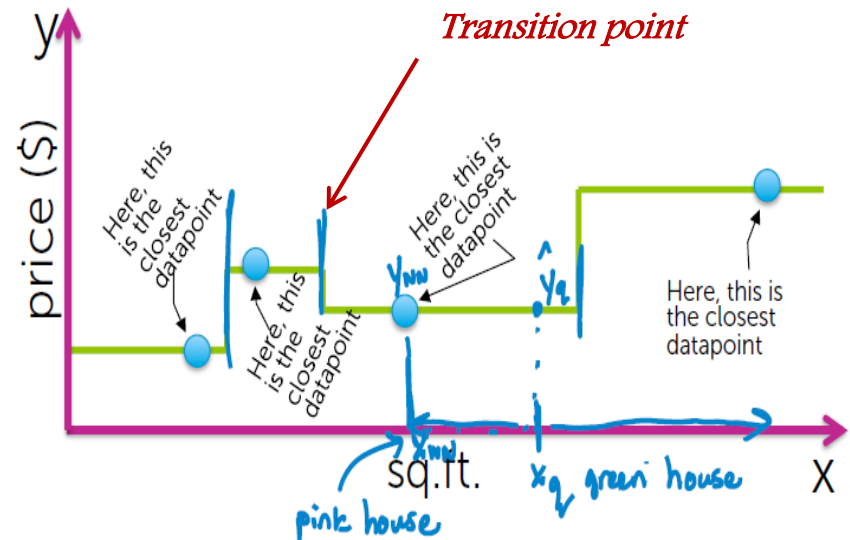
Query point: $\mathbf{x}_q$ ← 🏠 \$?
big lime green house

1. Find "closest" $\mathbf{x}_i$ in dataset

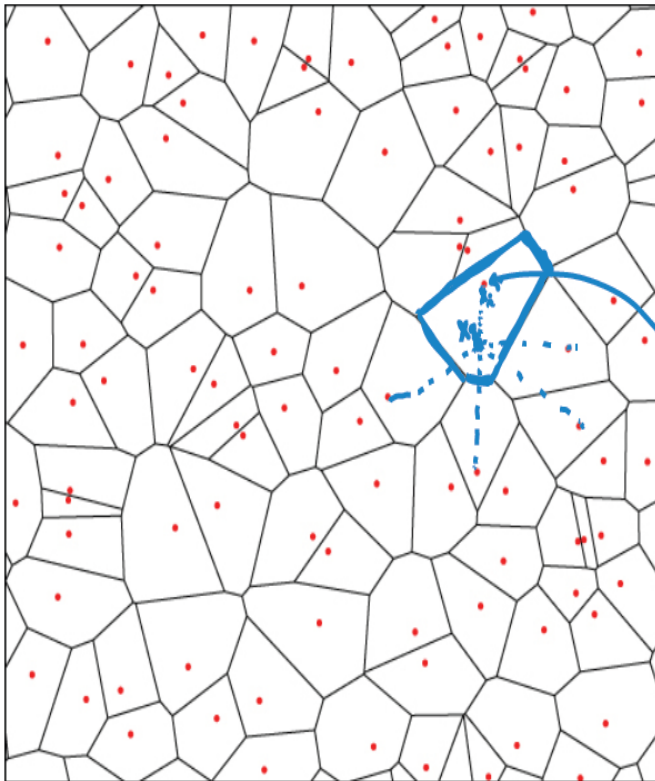
$\mathbf{x}_{NN}$ ← $\min_i \text{distance}(\mathbf{x}_i, \mathbf{x}_q)$
big pink house

2. Predict

$\hat{y}_q = y_{NN}$
sales price of big pink house



Visualizing 1-NN in multiple dimensions



Voronoi tessellation (or diagram):

- Divide space into N regions, each containing 1 datapoint
- Defined such that any $\mathbf{x}$ in region is "closest" to region's datapoint

x_i closer to x_i
than any other
 x_j for $j \neq i$.

Don't explicitly form!

Distance metrics: Notion of „closest”

In 1D, just Euclidean distance:

$$\text{distance}(x_j, x_q) = |x_j - x_q|$$

In multiple dimensions:

- can define many interesting distance functions
- most straightforwardly, might want to weight different dimensions differently

Weighting housing inputs

Some inputs are more relevant than others



bedrooms
bathrooms
sq.ft. living
sq.ft. lot
floors
year built
year renovated
waterfront



Scaled Euclidan distance

Formally, this is achieved via

$$\text{distance}(\mathbf{x}_j, \mathbf{x}_q) =$$

$$\sqrt{a_1(\mathbf{x}_j[1] - \mathbf{x}_q[1])^2 + \dots + a_d(\mathbf{x}_j[d] - \mathbf{x}_q[d])^2}$$

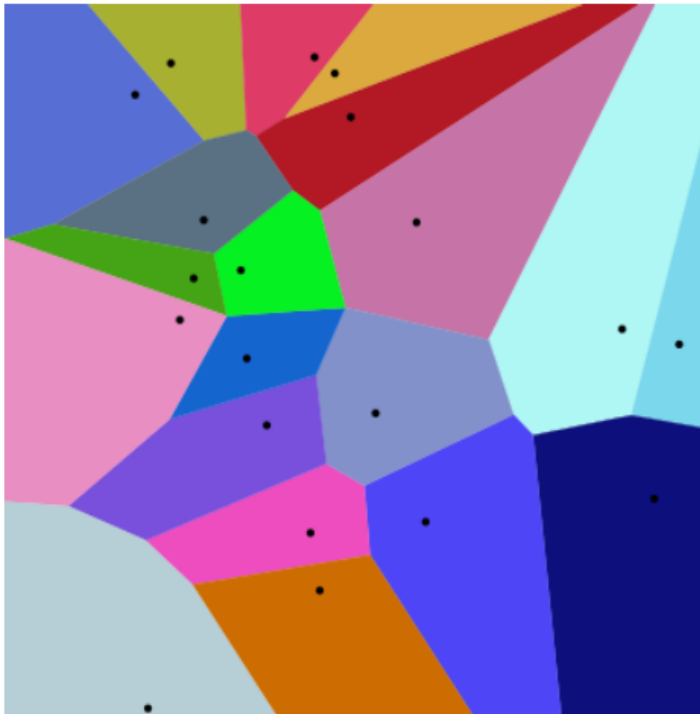
weight on each input
(defining relative importance)

Other example distance metrics:

- Mahalanobis, rank-based, correlation-based, cosine similarity, Manhattan, Hamming, ...

Different distance metrics

Euclidean distance



Manhattan distance



Performing 1-NN search

- Query house:



- Dataset:



- **Specify:** Distance metric
- **Output:** Most similar house



1-NN algorithm

Initialize **Dist2NN** = ∞ ,  = $\emptyset$ ← closest house

For $i=1,2,\dots,N$

 Compute: $\delta = \text{distance}(\text{img alt="house icon" data-bbox="535 445 590 515"}, \text{img alt="house icon" data-bbox="620 445 675 515"}_q)$ ← query house 

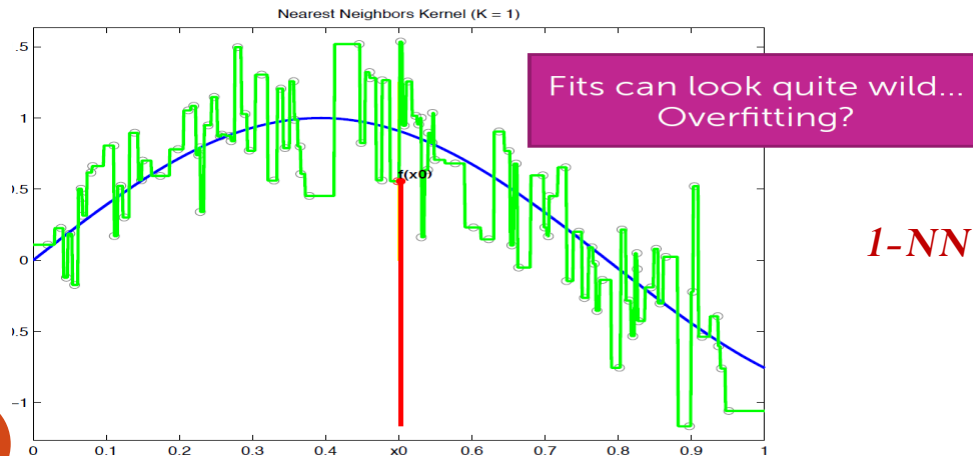
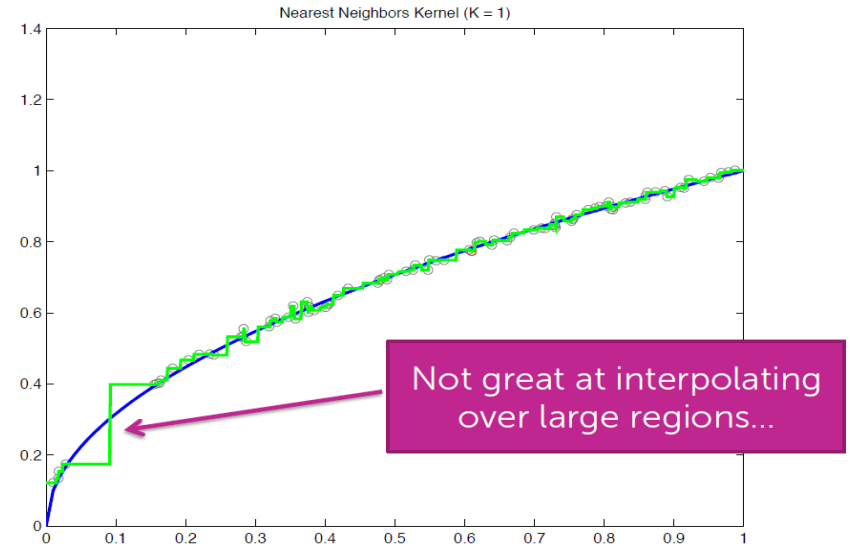
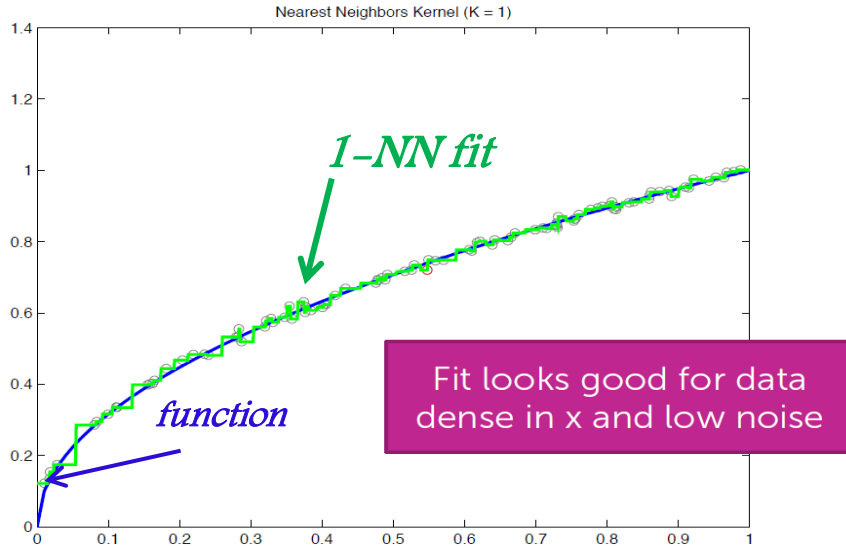
 If $\delta < \text{Dist2NN}$

 set   _{i}

 set **Dist2NN** = δ

Return most similar house  ← closest house to query house 

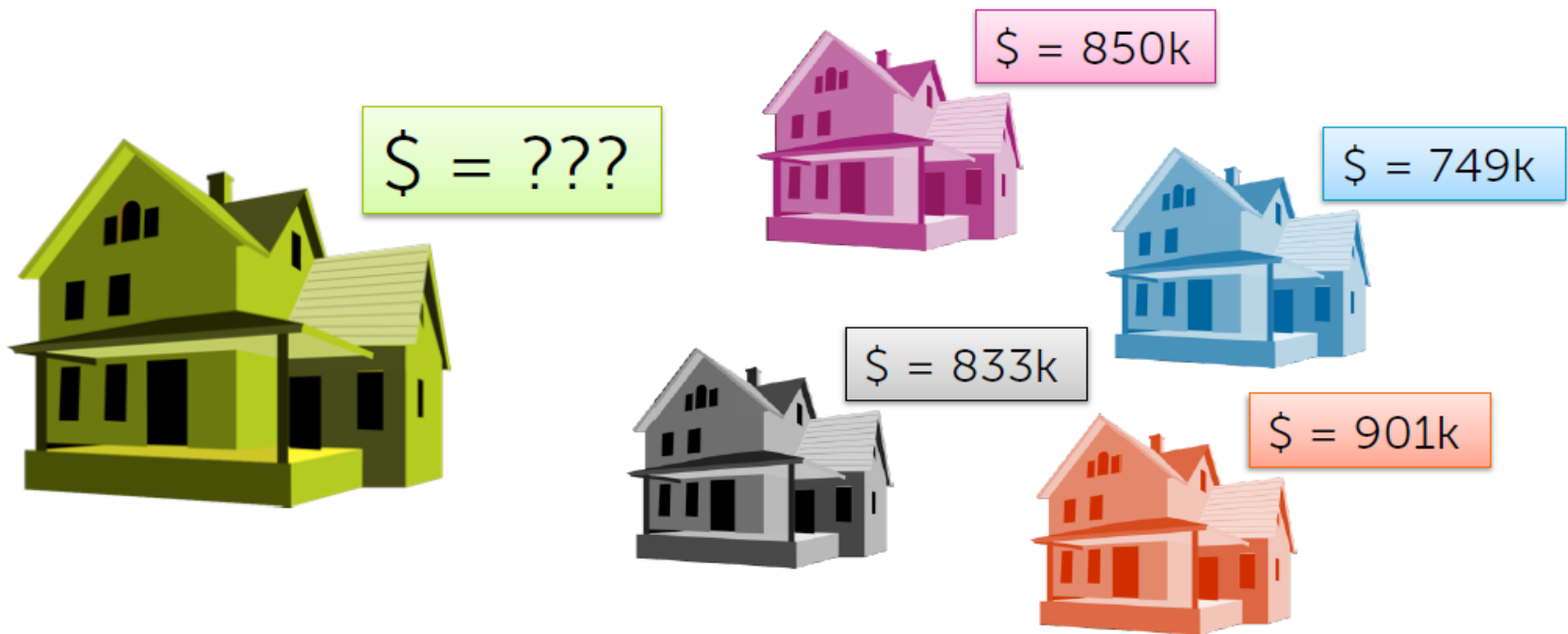
1-NN in practice



1-NN sensitive to noise in the data

Get more „comps”

More reliable estimate if you base estimate off of a larger set of comparable homes



K-NN regression more formally

Dataset of (, \$) pairs: $(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), \dots, (\mathbf{x}_N, y_N)$

Query point: $\mathbf{x}_q$

1. Find k closest $\mathbf{x}_i$ in dataset

$(x_{NN1}, x_{NN2}, \dots, x_{NNk})$ such that for any x_i not in nearest neighbor set,
 $\text{distance}(x_i, x_q) \geq \text{distance}(x_{NNk}, x_q)$

2. Predict

$$\begin{aligned}\hat{y}_q &= \frac{1}{k} (y_{NN1} + y_{NN2} + \dots + y_{NNk}) \\ &= \frac{1}{k} \sum_{j=1}^k y_{NNj}\end{aligned}$$

K-NN more formally

- Query house:



- Dataset:



- **Specify:** Distance metric
- **Output:** Most similar houses



K-NN algorithm

Initialize **Dist2kNN** = $\text{sort}(\delta_1, \dots, \delta_k)$  $\dots$  $\dots$ 
 = $\text{sort}(\text{house}_1, \dots, \text{house}_k)$  $\dots$  $\dots$ 
 For $i = k+1, \dots, N$
 Compute: $\delta = \text{distance}(\text{house}_i, \text{house}_q)$   
 If $\delta < \text{Dist2kNN}[k]$
 find j such that $\delta > \text{Dist2kNN}[j-1]$ but $\delta < \text{Dist2kNN}[j]$
 remove furthest house and shift queue:
 $[j+1:k] = [j:k]$
 $\text{Dist2kNN}[j+1:k] = \text{Dist2kNN}[j:k-1]$
 insert new NN
 set $\text{Dist2kNN}[j] = \delta$ and $\text{house}_j = \text{house}_i$
 Return k most similar houses      

sort first k houses by distance to query house

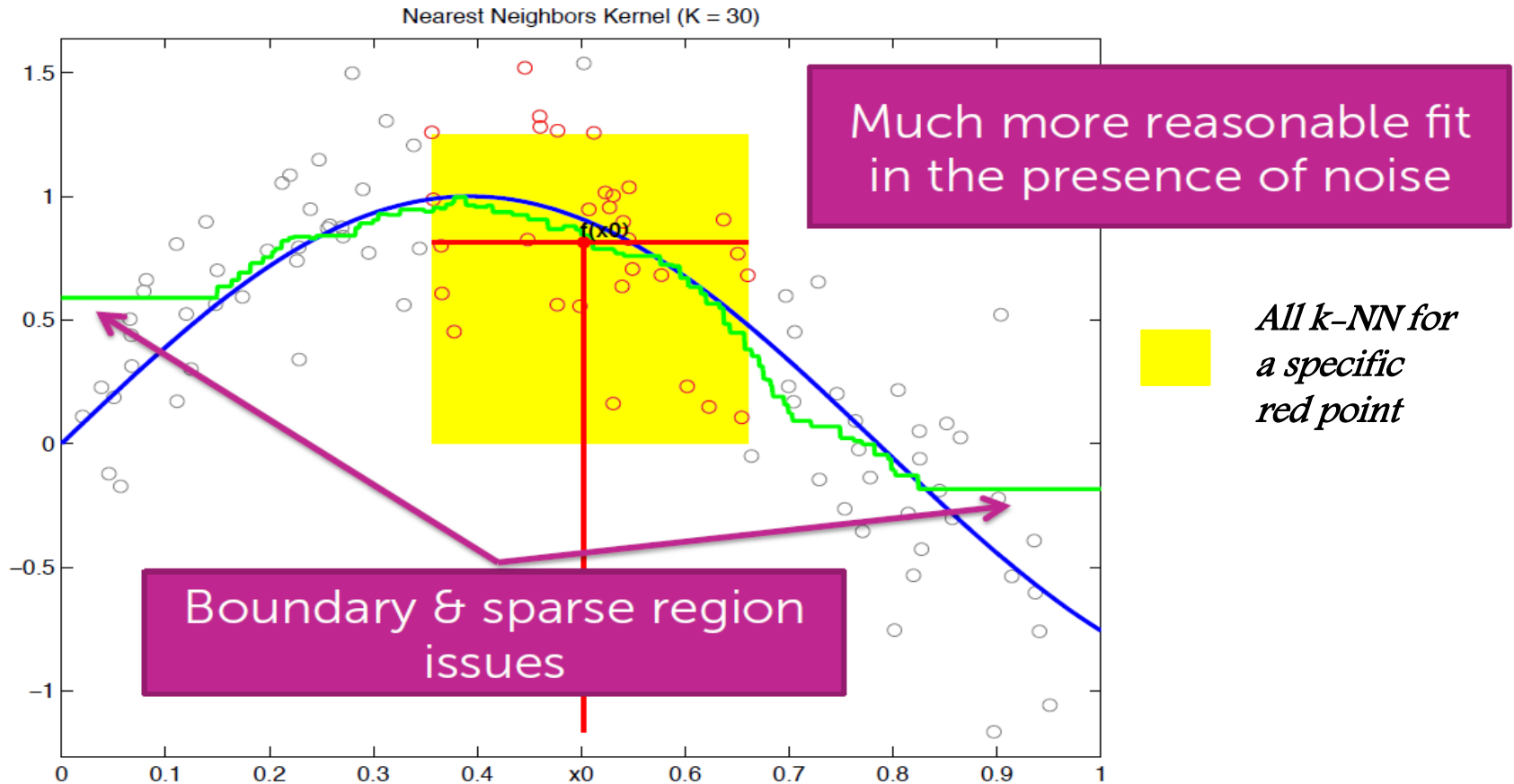
list of sorted distances

list of sorted houses

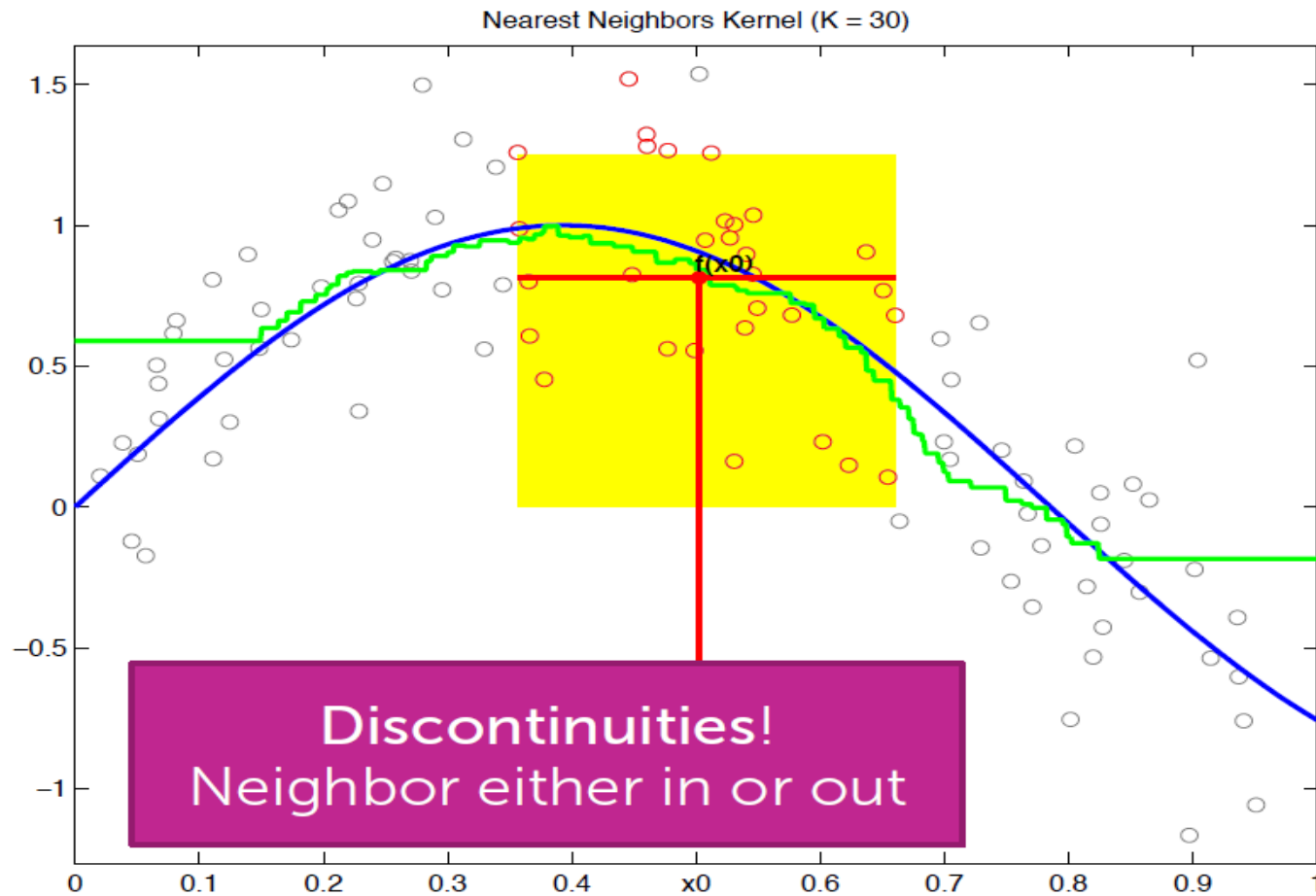
query house

closest houses to query house

K-NN in practice



K-NN in practice



Issues with discontinuities

Overall predictive accuracy might be okay, but...

For example, in housing application:

- If you are a buyer or seller, this matters
- Can be a jump in estimated value of house going just from 2640 sq.ft. to 2641 sq.ft.
- Don't really believe this type of fit

Weighted k-NN

Weigh more similar houses more than those less similar in list of k-NN

Predict:

weights on NN


$$\hat{y}_q = \frac{c_{qNN1}y_{NN1} + c_{qNN2}y_{NN2} + c_{qNN3}y_{NN3} + \dots + c_{qNNk}y_{NNk}}{\sum_{j=1}^k c_{qNNj}}$$

How to define weights

Want weight c_{qNNj} to be small when
 $\text{distance}(\mathbf{x}_{NNj}, \mathbf{x}_q)$ large

and c_{qNNj} to be large when
 $\text{distance}(\mathbf{x}_{NNj}, \mathbf{x}_q)$ small

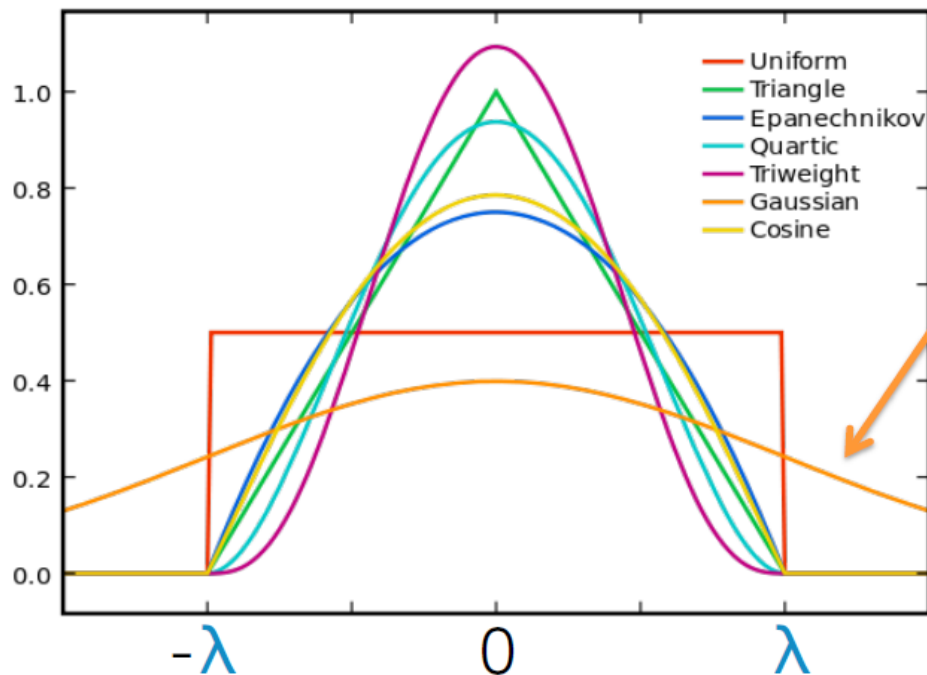
Simple method :

$$c_{qNNj} = \frac{1}{\text{distance}(\mathbf{x}_j, \mathbf{x}_q)}$$

Kernel weights for d=1

Define: $c_{qNNj} = \text{Kernel}_\lambda(|x_{NNj} - x_q|)$

simple
isotropic case



Gaussian kernel:

$$\text{Kernel}_\lambda(|x_i - x_q|) = \exp(-(x_i - x_q)^2 / \lambda)$$

Note: never exactly 0!

Kernel drives how the weights will decay, if at all, as a function of the distance.

Kernel regression

Nadaraya-Watson
kernel weighted average

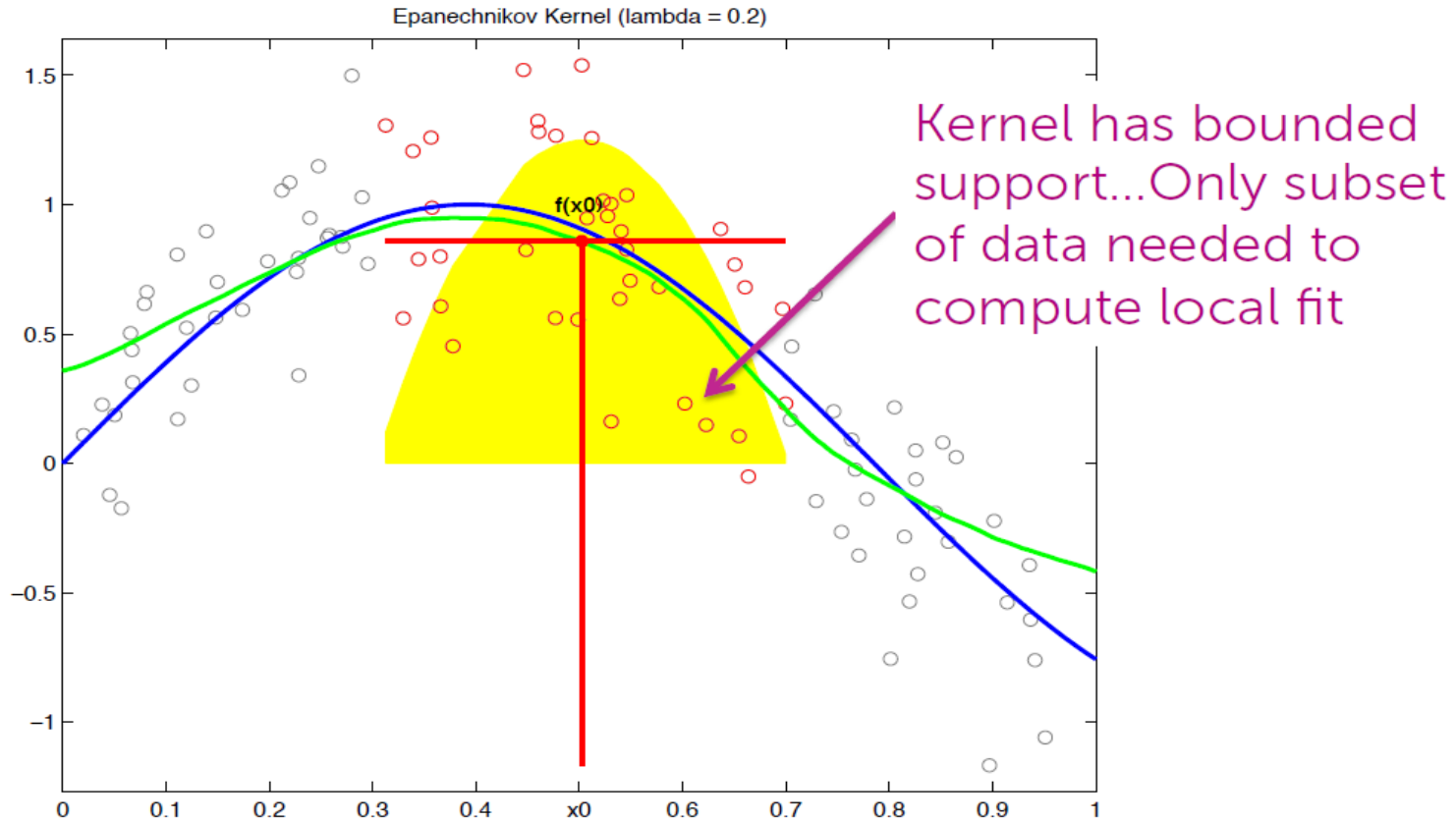
Instead of just weighting NN, weight *all* points

Predict:

weight on each datapoint

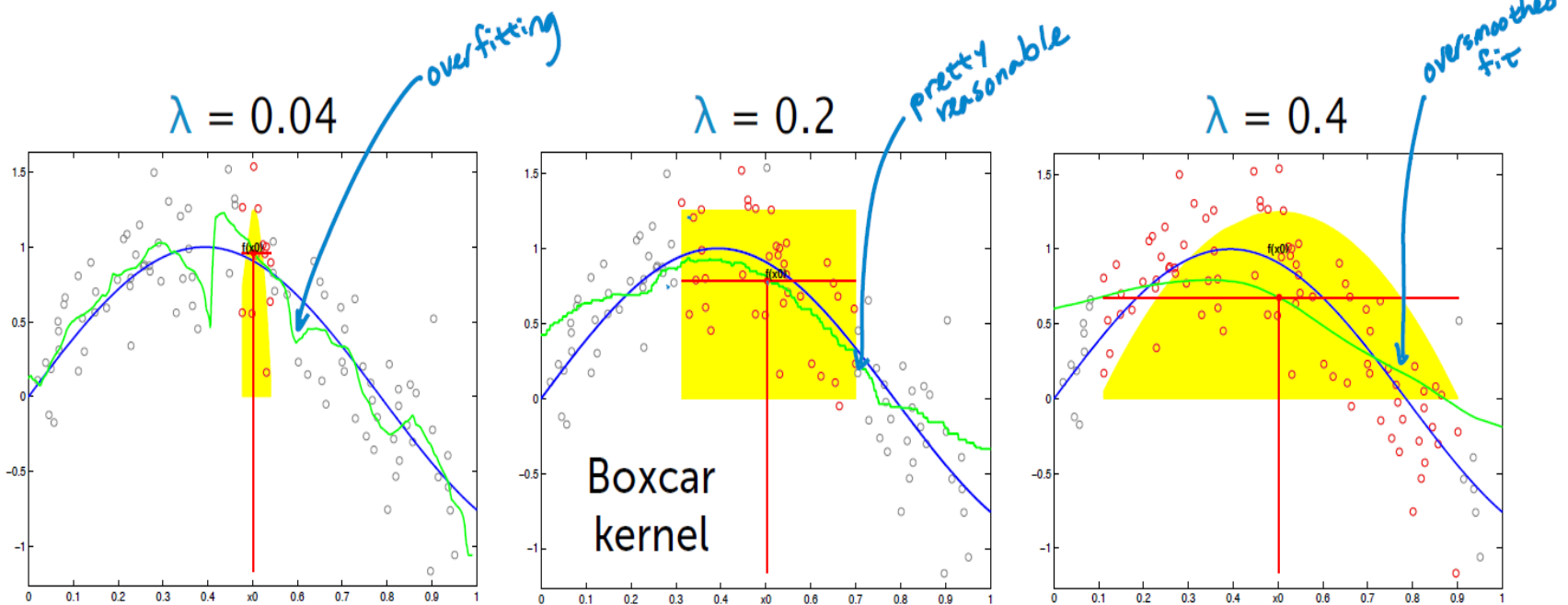
$$\hat{y}_q = \frac{\sum_{i=1}^N c_{qi} y_i}{\sum_{i=1}^N c_{qi}} = \frac{\sum_{i=1}^N \text{Kernel}_{\lambda}(\text{distance}(\mathbf{x}_i, \mathbf{x}_q)) * y_i}{\sum_{i=1}^N \text{Kernel}_{\lambda}(\text{distance}(\mathbf{x}_i, \mathbf{x}_q))}$$

Kernel regression in practice



Choice of bandwidth λ

Often, choice of kernel matters much less than choice of λ



Choosing λ (or k on k-NN)

How to choose? Same story as always...

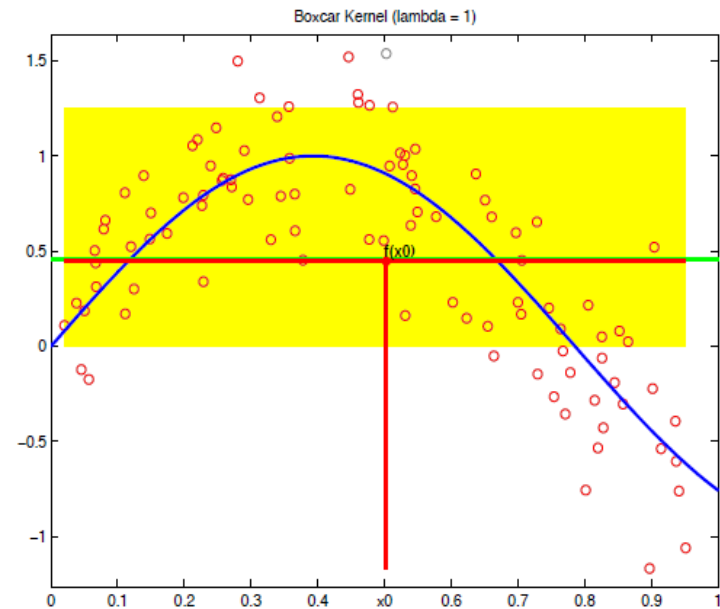
Cross Validation

Contrasting with global average

A globally constant fit weights all points equally

$$\hat{y}_q = \frac{1}{N} \sum_{i=1}^N y_i = \frac{\sum_{i=1}^N c y_i}{\sum_{i=1}^N c}$$

equal weight on each datapoint

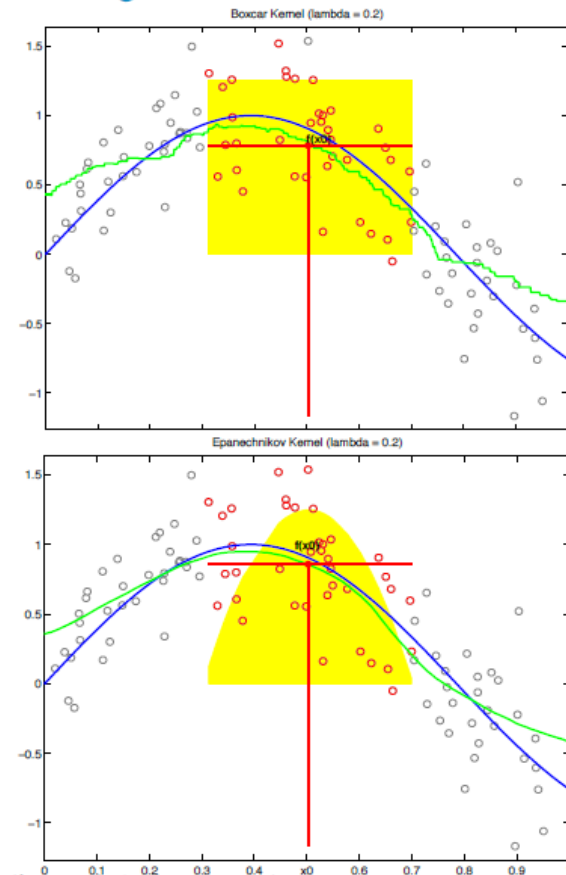


Contrasting with global average

Kernel regression leads to **locally constant fit**

- slowly add in some points and
and let others gradually die off

$$\hat{y}_q = \frac{\sum_{i=1}^N \text{Kernel}_{\lambda}(\text{distance}(\mathbf{x}_i, \mathbf{x}_q)) * y_i}{\sum_{i=1}^N \text{Kernel}_{\lambda}(\text{distance}(\mathbf{x}_i, \mathbf{x}_q))}$$



Local linear regression

So far, discussed fitting **constant function locally** at each point

→ “locally weighted averages”

Can instead fit a **line or polynomial locally** at each point

→ “locally weighted linear regression”

Local regression rules of thumb

- Local linear fit reduces bias at boundaries with minimum increase in variance
- Local quadratic fit doesn't help at boundaries and increases variance, but does help capture curvature in the interior
- With sufficient data, local polynomials of odd degree dominate those of even degree

Recommended default choice:
local linear regression

Nonparametric approaches

k-NN and kernel regression are examples of **nonparametric** regression

General goals of nonparametrics:

- Flexibility
- Make few assumptions about $f(\mathbf{x})$
- **Complexity can grow with the number of observations N**

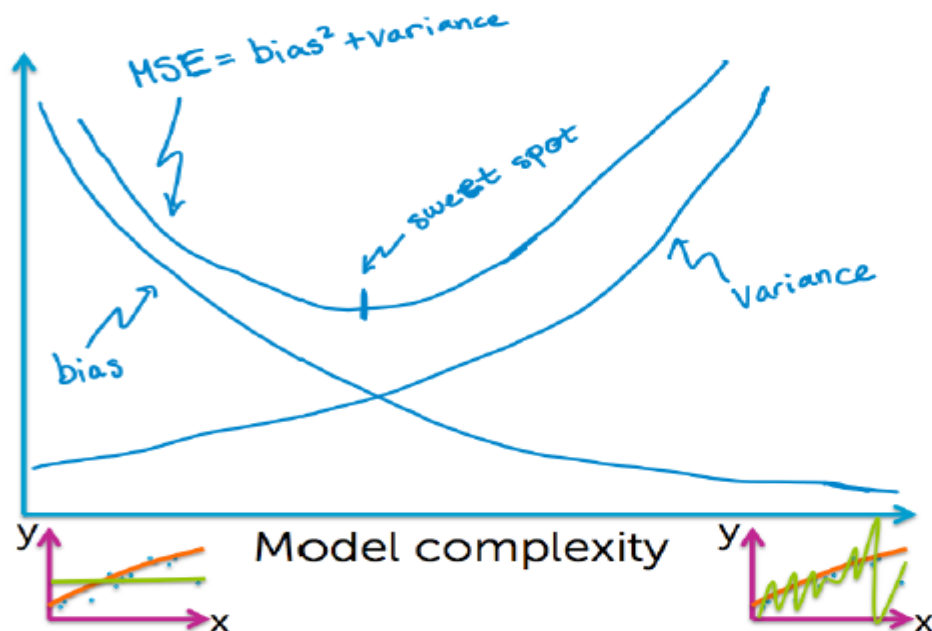
Lots of other choices:

- Splines, trees, locally weighted structured regression models...

Limiting behaviour of NN

Noiseless setting ($\varepsilon_i=0$)

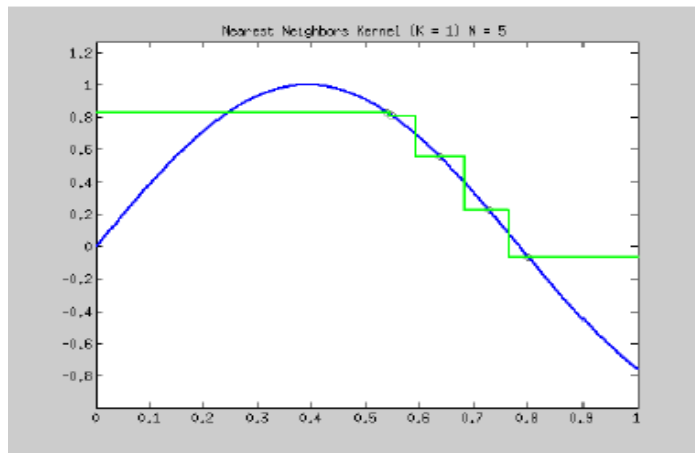
In the limit of getting an infinite amount of noiseless data, the MSE of 1-NN fit goes to 0



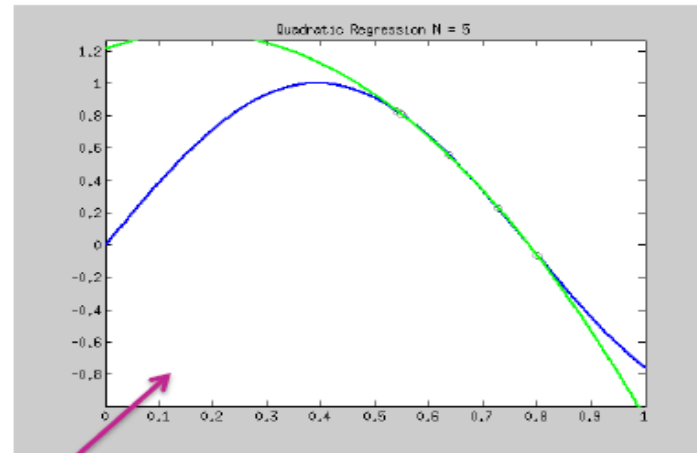
Limiting behaviour of NN

Noiseless setting ($\varepsilon_i=0$)

In the limit of getting an infinite amount of noiseless data, the **MSE of 1-NN fit goes to 0**



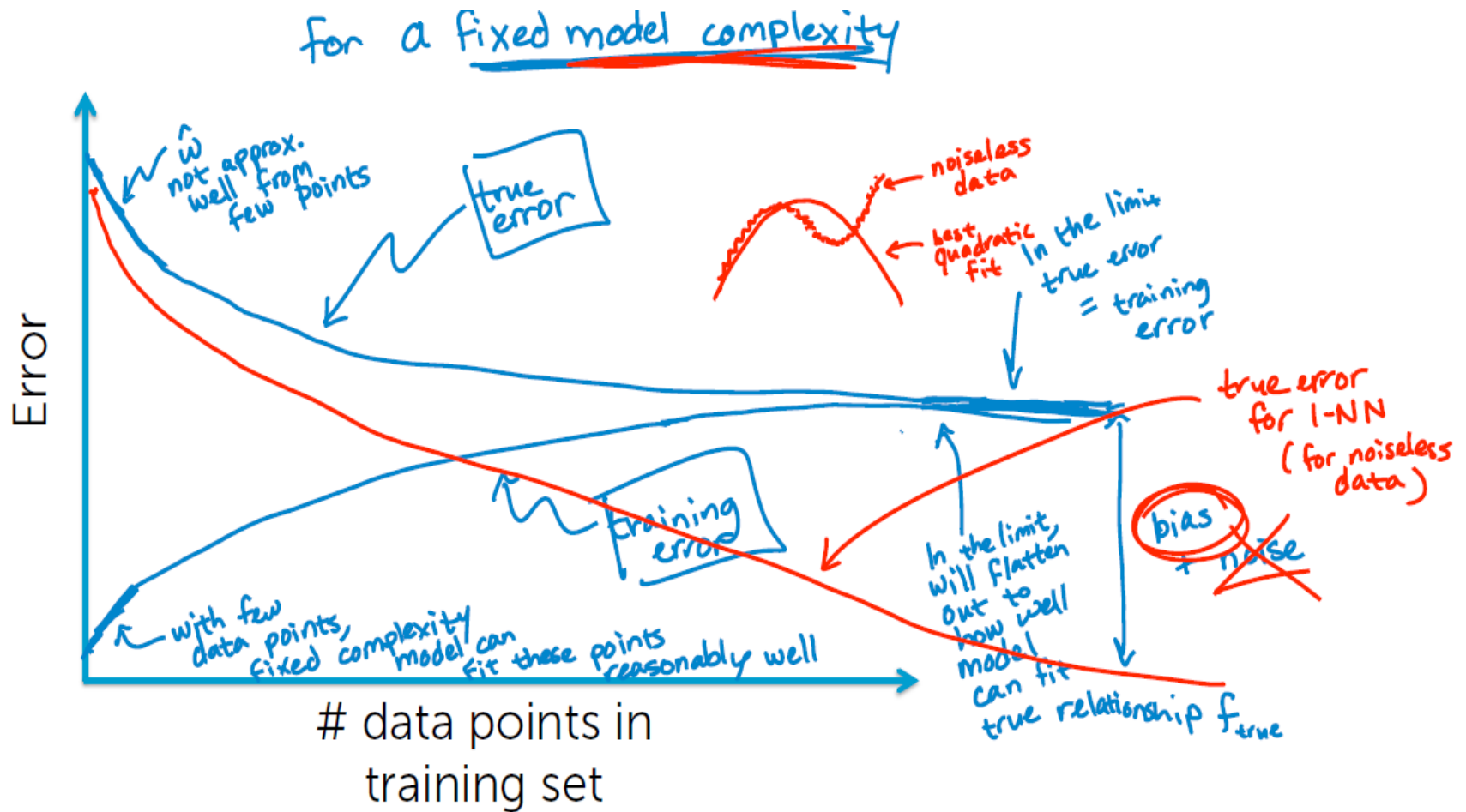
1-NN fit



Quadratic fit

Not true for parametric models!

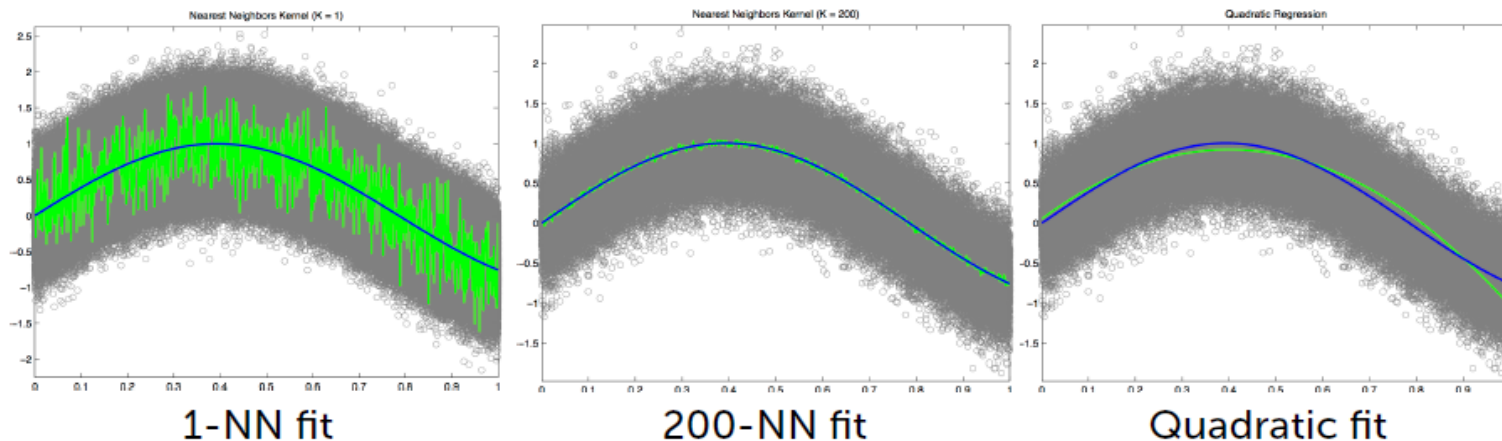
Error vs amount of data



Limiting behaviour of NN

Noisy data setting

In the limit of getting an infinite amount of data, the **MSE of NN fit goes to 0** if **k grows, too**



Issues: NN and kernel methods

NN and kernel methods work well when the data cover the space, but...

- the more dimensions d you have, the more points N you need to cover the space
- need $N = O(\exp(d))$ data points for good performance

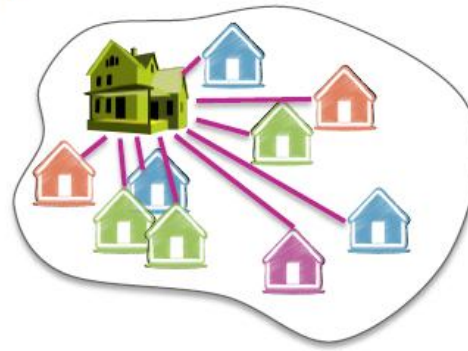
This is where parametric models become useful...

Issues: Complexity of NN search

Naïve approach: **Brute force search**

- Given a query point $\mathbf{x}_q$
- Scan through each point $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_N$
- $O(N)$ distance computations per 1-NN query!
- $O(N \log k)$ per k-NN query!

What if N is huge???
(and many queries)



Will talk more about efficient methods in
Clustering & Retrieval course

We you should be able to do now

- Motivate the use of nearest neighbor (NN) regression
- Define distance metrics in 1D and multiple dimensions
- Perform NN and k-NN regression
- Analyze computational costs of these algorithms
- Discuss sensitivity of NN to lack of data, dimensionality, and noise
- Perform weighted k-NN and define weights using a kernel
- Define and implement kernel regression
- Describe the effect of varying the kernel bandwidth λ or # of nearest neighbors k
- Select λ or k using cross validation
- Compare and contrast kernel regression with a global average fit
- Define what makes an approach nonparametric and why NN and kernel regression are considered nonparametric methods
- Analyze the limiting behavior of NN regression

Summarising

Models

- Linear regression
- Regularization: Ridge (L2), Lasso (L1)
- Nearest neighbor and kernel regression

Algorithms

- Gradient descent
- Coordinate descent

Concepts

- Loss functions, bias-variance tradeoff, cross-validation, sparsity, overfitting, model selection, feature selection