DATA SCIENCE WITH MACHINE LEARNING: REGRESSION

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

4/01/2022

WFAiS UJ, Informatyka Stosowana
I stopień studiów

What is Data Science?

Is mainly about extracting knowledge from data (terms "data mining" or "Knowledge Discovery in Databases" are highly related). It can be about analyzing trends, building predictive models, … etc.

Is an agglomerate of data collection, data modeling and analysis, a decision making, and everything you need to know to accomplish your goals. Eventually, it boils down to the following fields/skills:

- **Computer science:**

Algorithms, programming (patterns, languages etc.), understanding hardware & operating systems, high-performance computing'

- **Mathematical aspects:**

Linear algebra, differential equations for optimization problems, statistics

- **Few others:**

Machine learning, domain knowledge, and data visualization & communication skills

Data Science and Machine Learning?

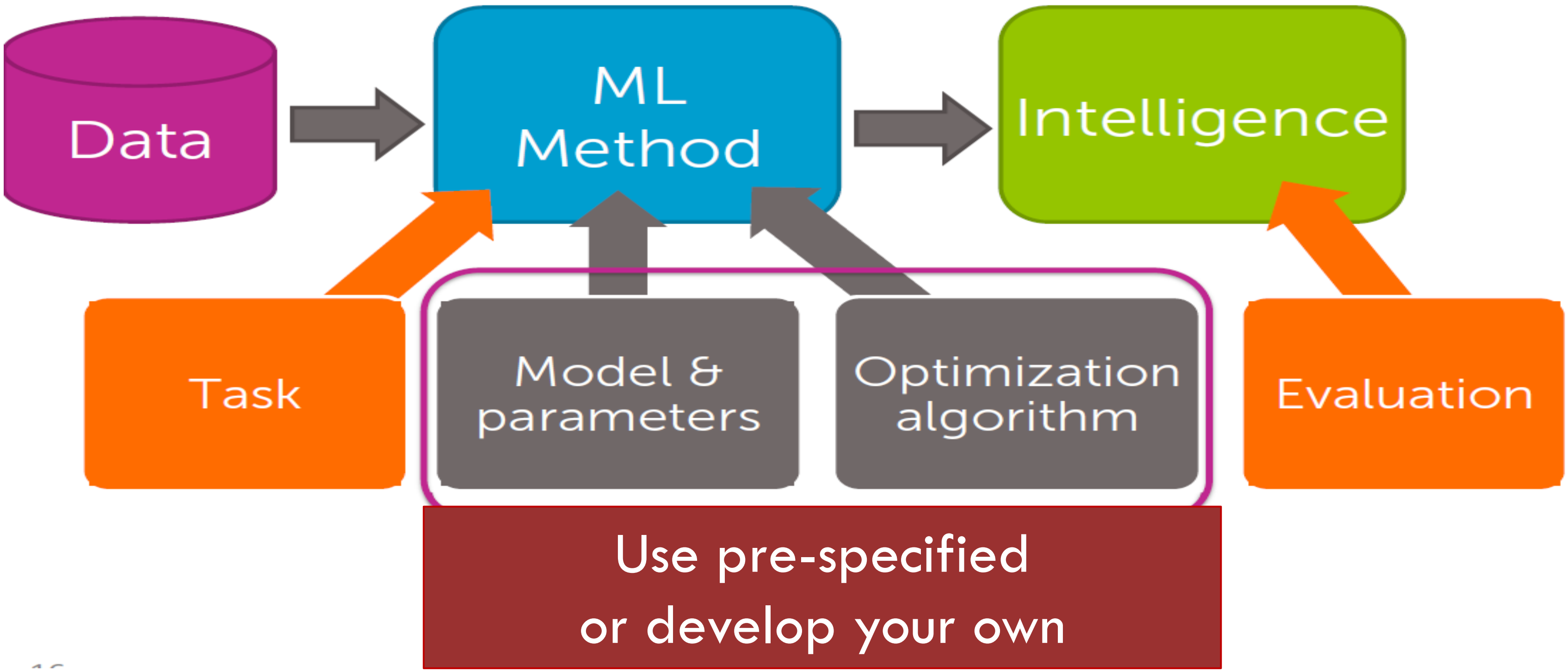
Machine learning algorithms are algorithms that learn (often predictive) models from data. I.e., instead of formulating "rules" manually, a machine learning algorithm will learn the model for you.

Machine learning - at its core - is about the use and development of these learning algorithms. Data science is more about the extraction of knowledge from data to answer particular question or solve particular problems.

Machine learning is often a big part of a "data science" project, e.g., it is often heavily used for exploratory analysis and discovery (clustering algorithms) and building predictive models (supervised learning algorithms). However, in data science, you often also worry about the collection, wrangling, and cleaning of your data (i.e., data engineering), and eventually, you want to draw conclusions from your data that helps you solve a particular problem.

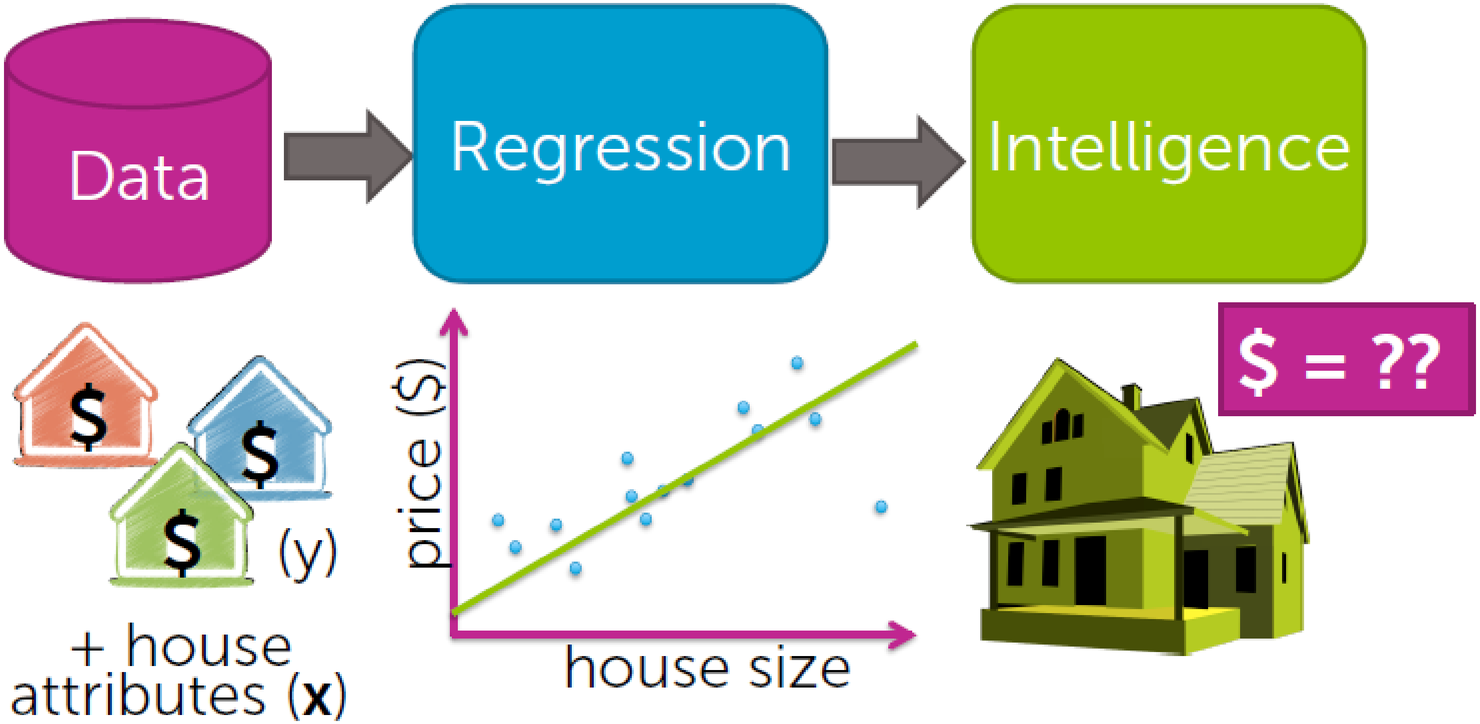
Deploing inteligence module

Case studied are about building, evaluating, deploying inteligence in data analysis.



Case study

Predicting house prices



Prediction: Predicting house prices

Models

- Linear regression
- Regularization:
  Ridge (L2), Lasso (L1)

Algorithms

- Gradient descent
- Coordinate descent

Concepts

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

Data

input *output*

$(x_1 = \text{sq.ft.}, \mathbf{y}_1 = \$)$

$(x_2 = \text{sq.ft.}, \mathbf{y}_2 = \$)$

$(x_3 = \text{sq.ft.}, \mathbf{y}_3 = \$)$

$(x_4 = \text{sq.ft.}, \mathbf{y}_4 = \$)$

$(x_5 = \text{sq.ft.}, \mathbf{y}_5 = \$)$

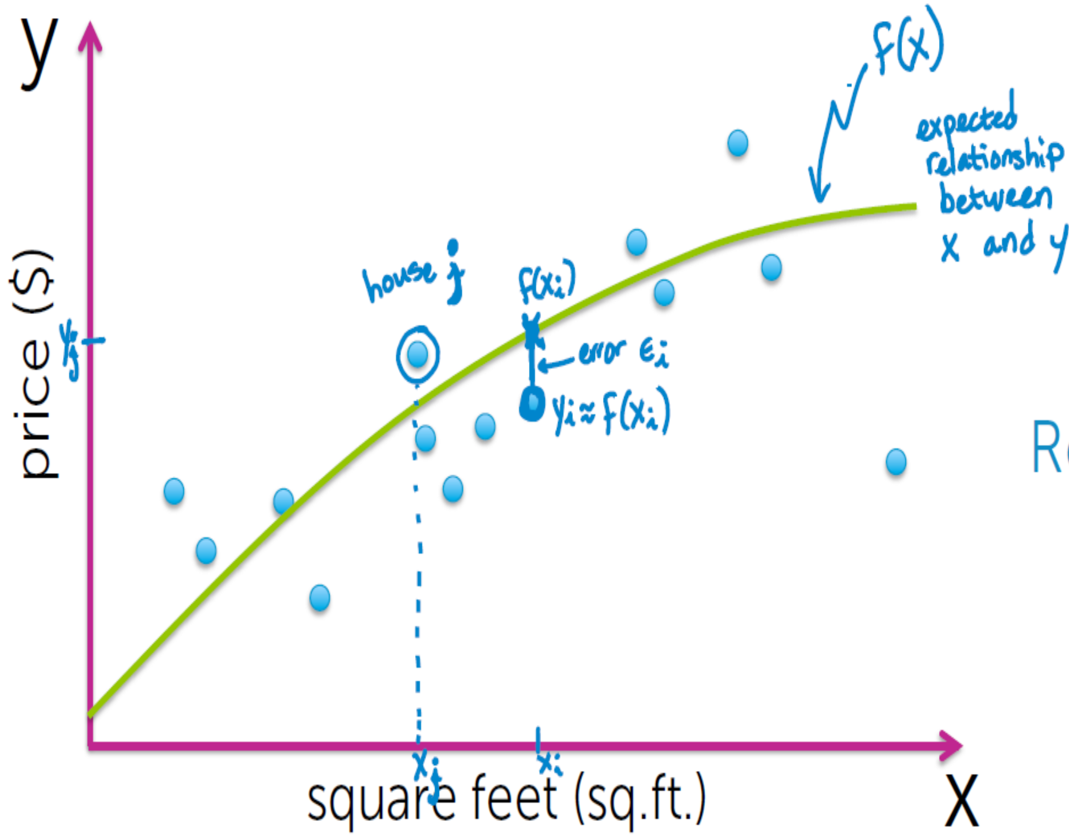
⋮

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 −

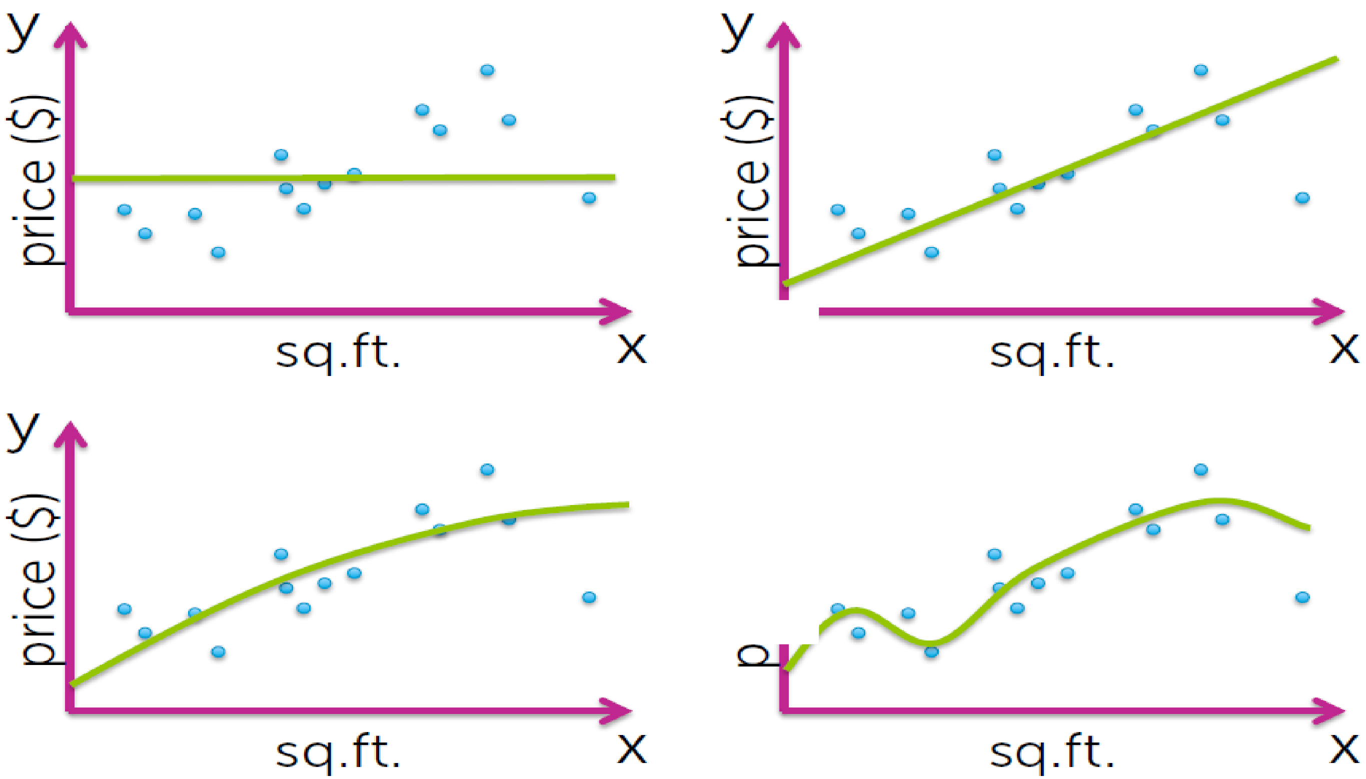
↑ expected value

⇓

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

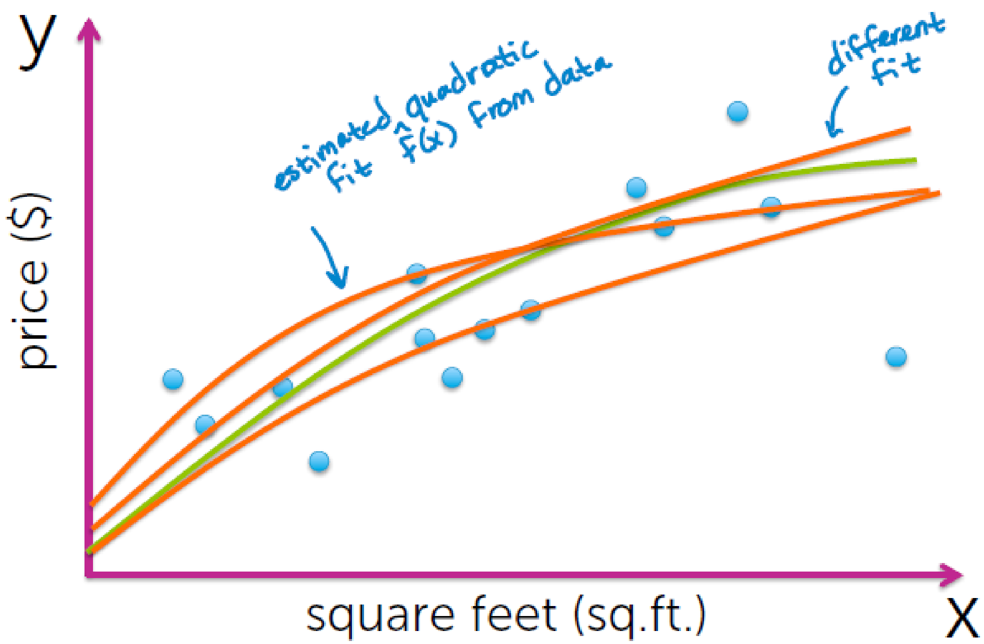
Task 1:

Which model to fit?



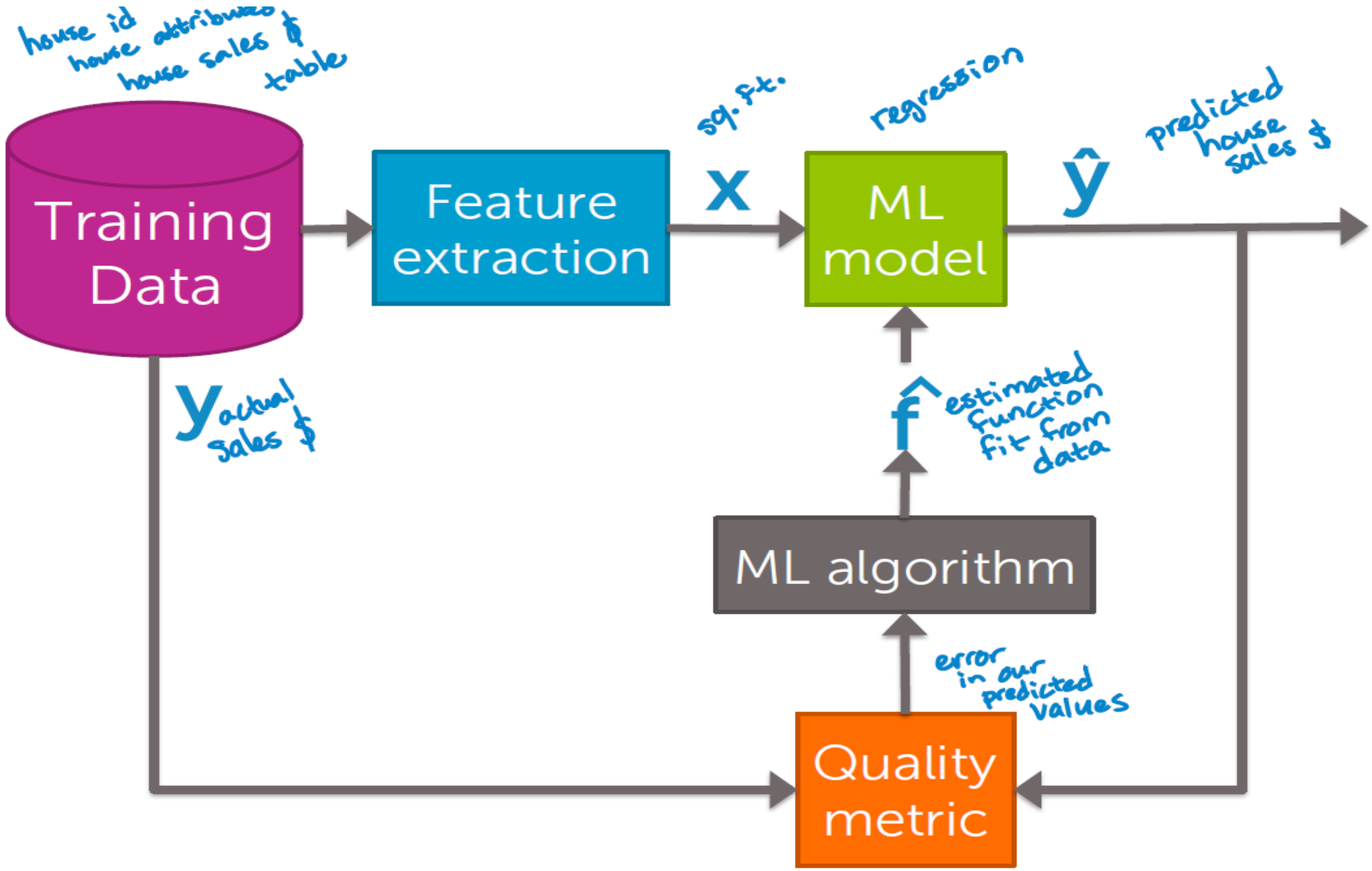
Task 2:

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



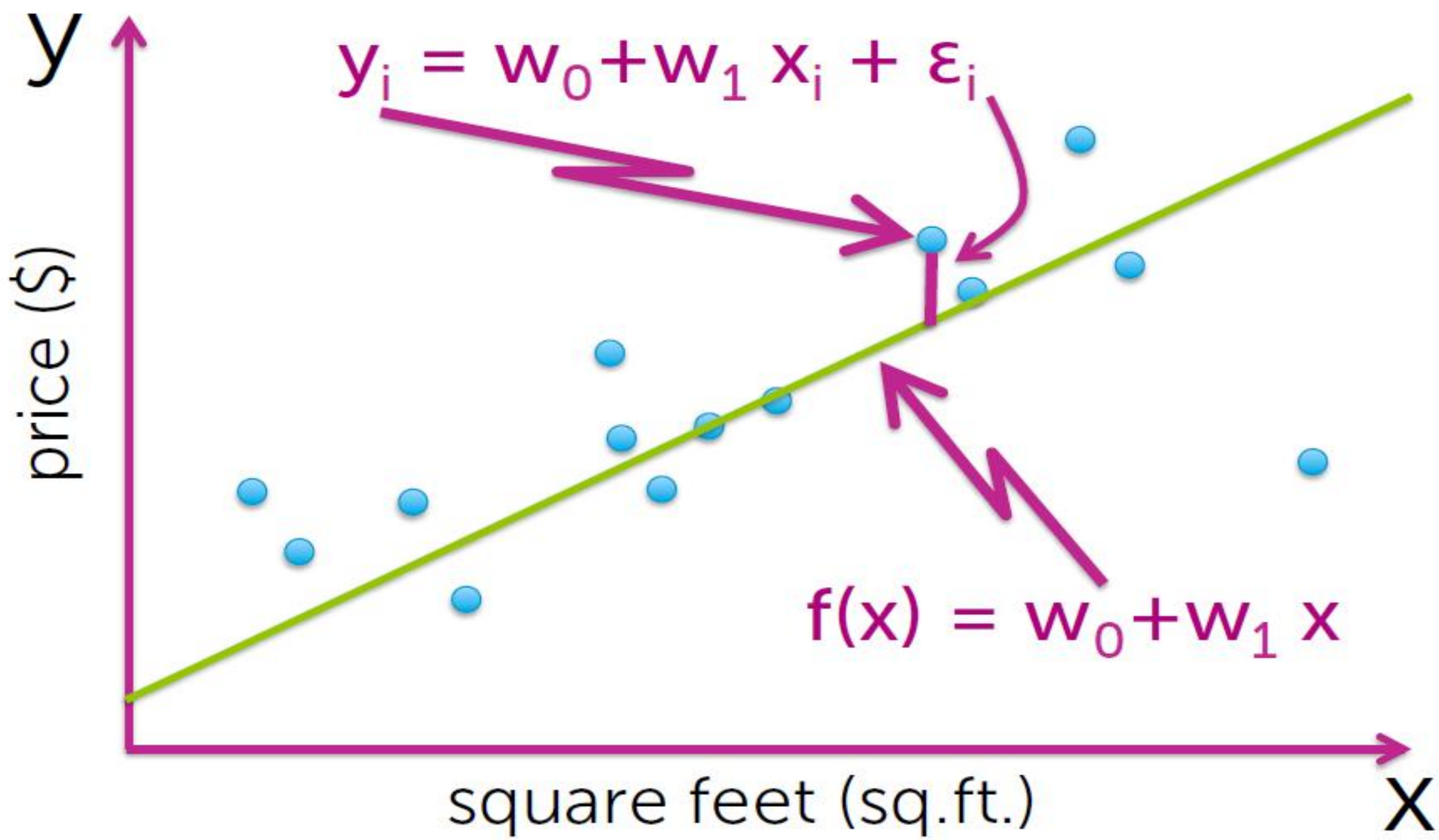
Assume model f(x) is a quadratic function

How it works: baseline flow chart



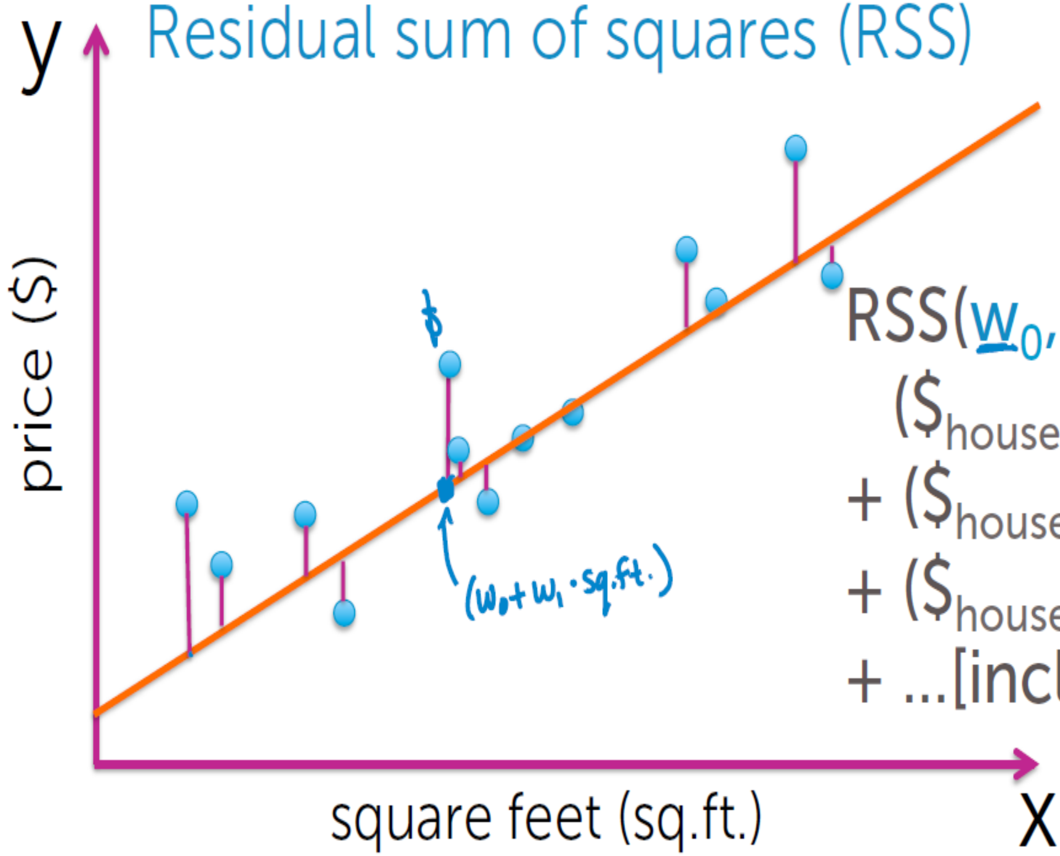
SIMPLE LINEAR REGRESSION

Simple linear regression model



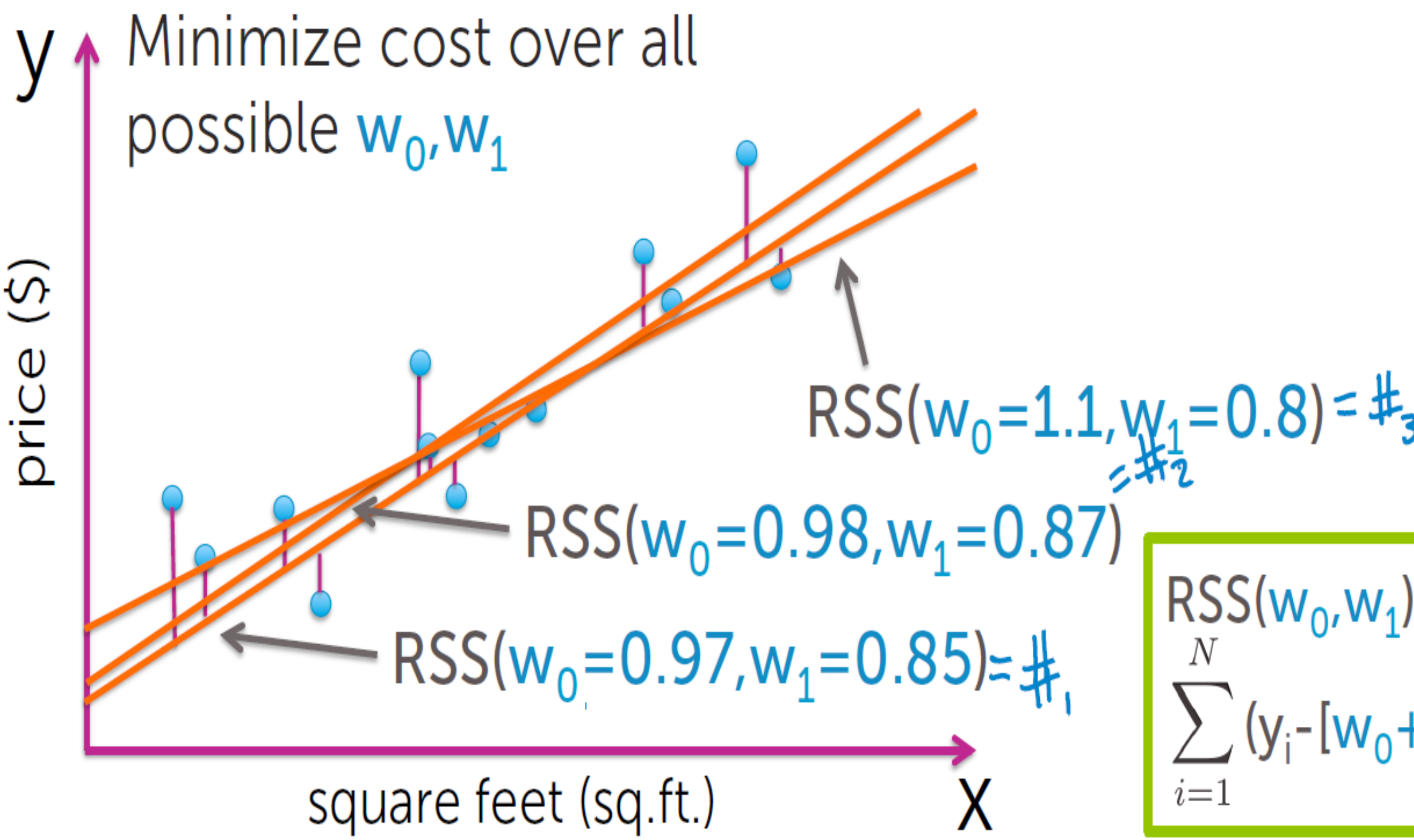
The cost of using a given line

Residual sum of squares (RSS)



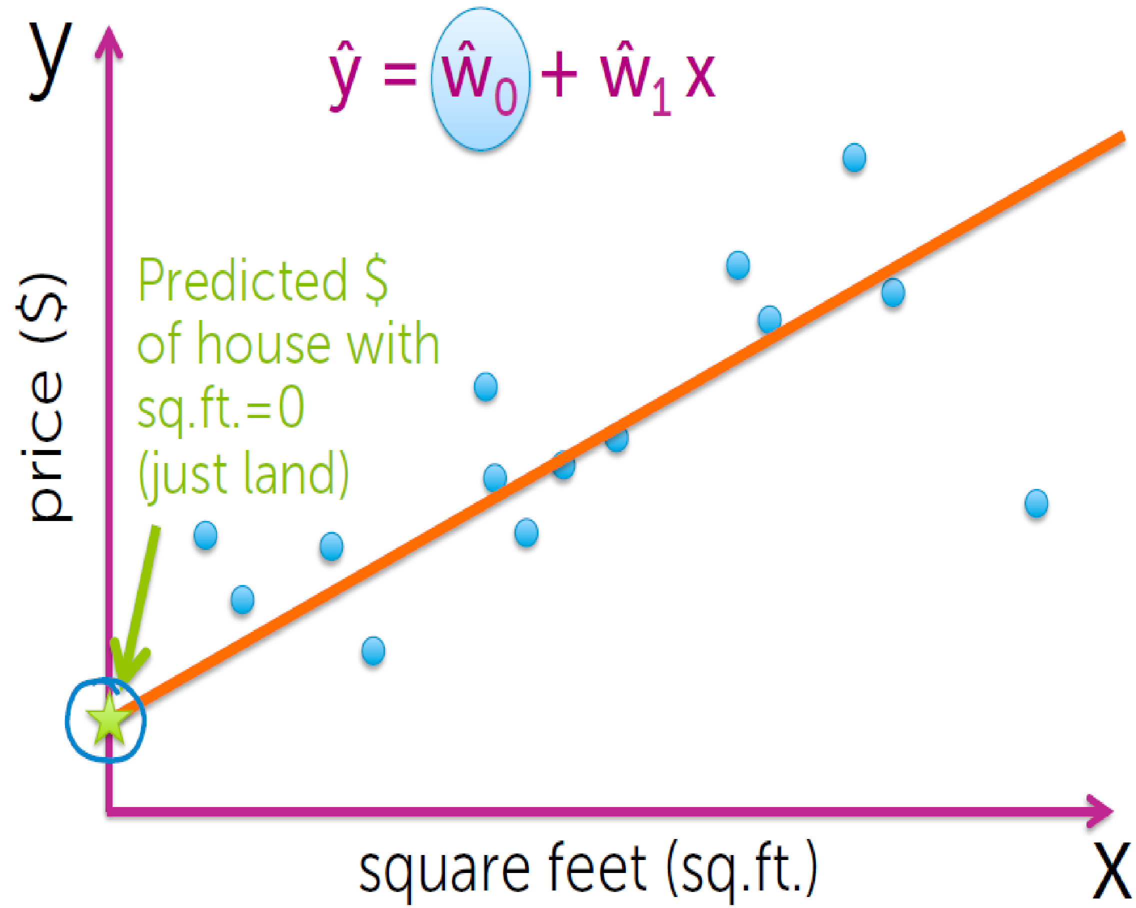
$$
\begin{aligned}
RSS(\underline{w_0},\underline{w_1}) = \ & (\$_{house\ 1}-[w_0+w_1 sq.ft._{house\ 1}])^2 \\
& + (\$_{house\ 2}-[w_0+w_1 sq.ft._{house\ 2}])^2 \\
& + (\$_{house\ 3}-[w_0+w_1 sq.ft._{house\ 3}])^2 \\
& + \ldots[\text{include all training houses}]
\end{aligned}
$$

Find „best" line



$$RSS(w_0,w_1) = \sum_{i=1}^{N}(y_i-[w_0+w_1x_i])^2$$

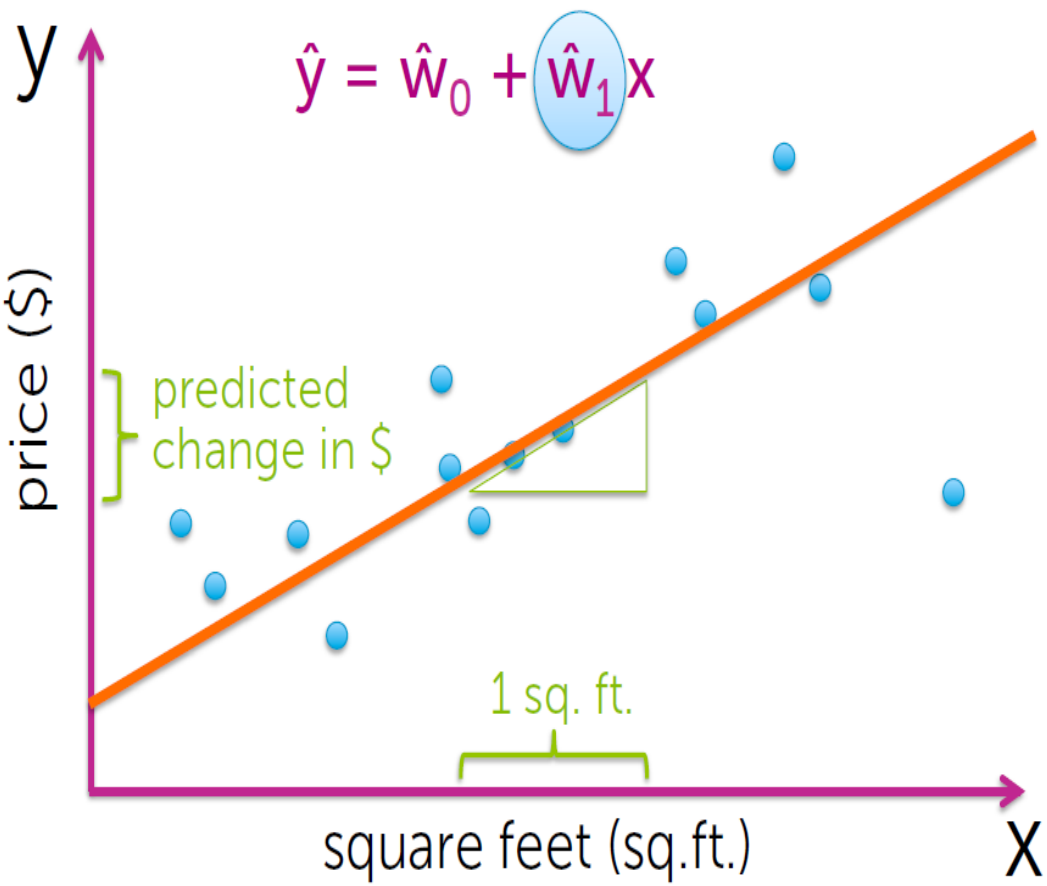
Interpreting the coefficients



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

not very meaningful

Interpreting the coefficients

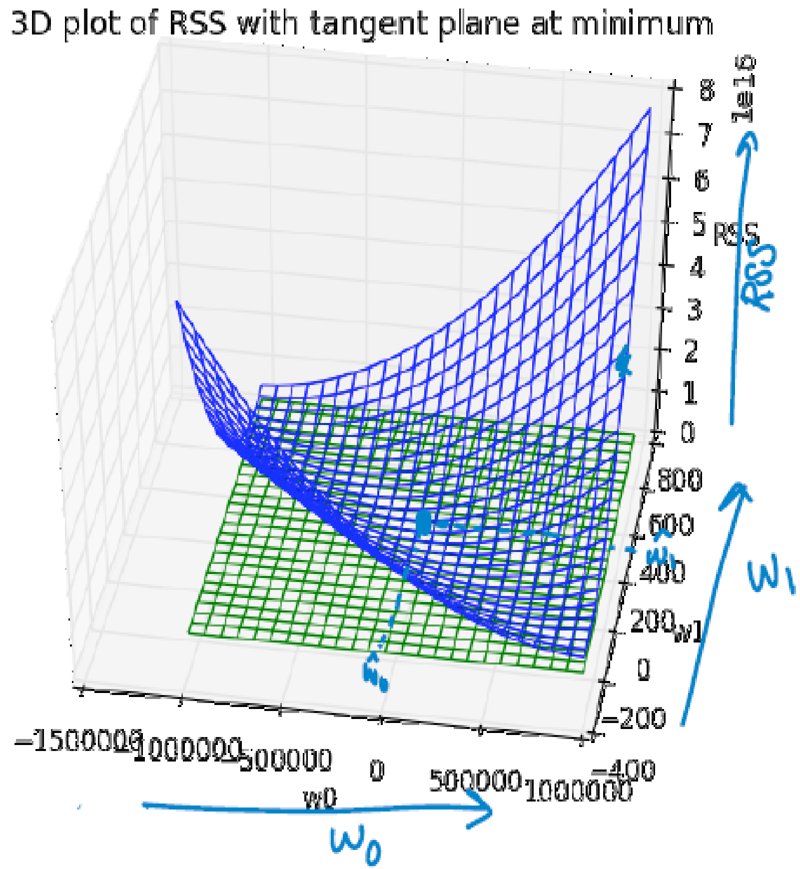


Magnitude of fit parameters depend on the units of both features and observations

$$\hat{\$}_{1001\ sq.ft.} - \hat{\$}_{1000\ sq.ft.}$$
$$= \cancel{\hat{w}_0} + \hat{w}_1 \cdot 1001\ sq.ft.$$
$$- (\cancel{\hat{w}_0} + \hat{w}_1 1000\ sq.ft.)$$
$$= \hat{w}_1$$

predicted change in the output per unit change in input

ML algorithm: minimasing the cost



Minimize function over all possible w_0, w_1

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

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

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"

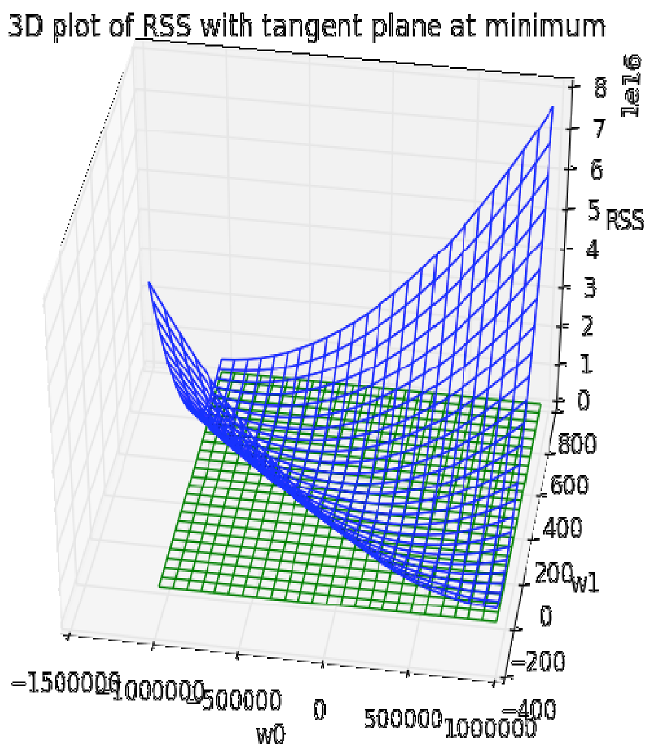
value of ε depends on the data we are looking at

Algorithm:

while not converged

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

Moving to multiple dimensions



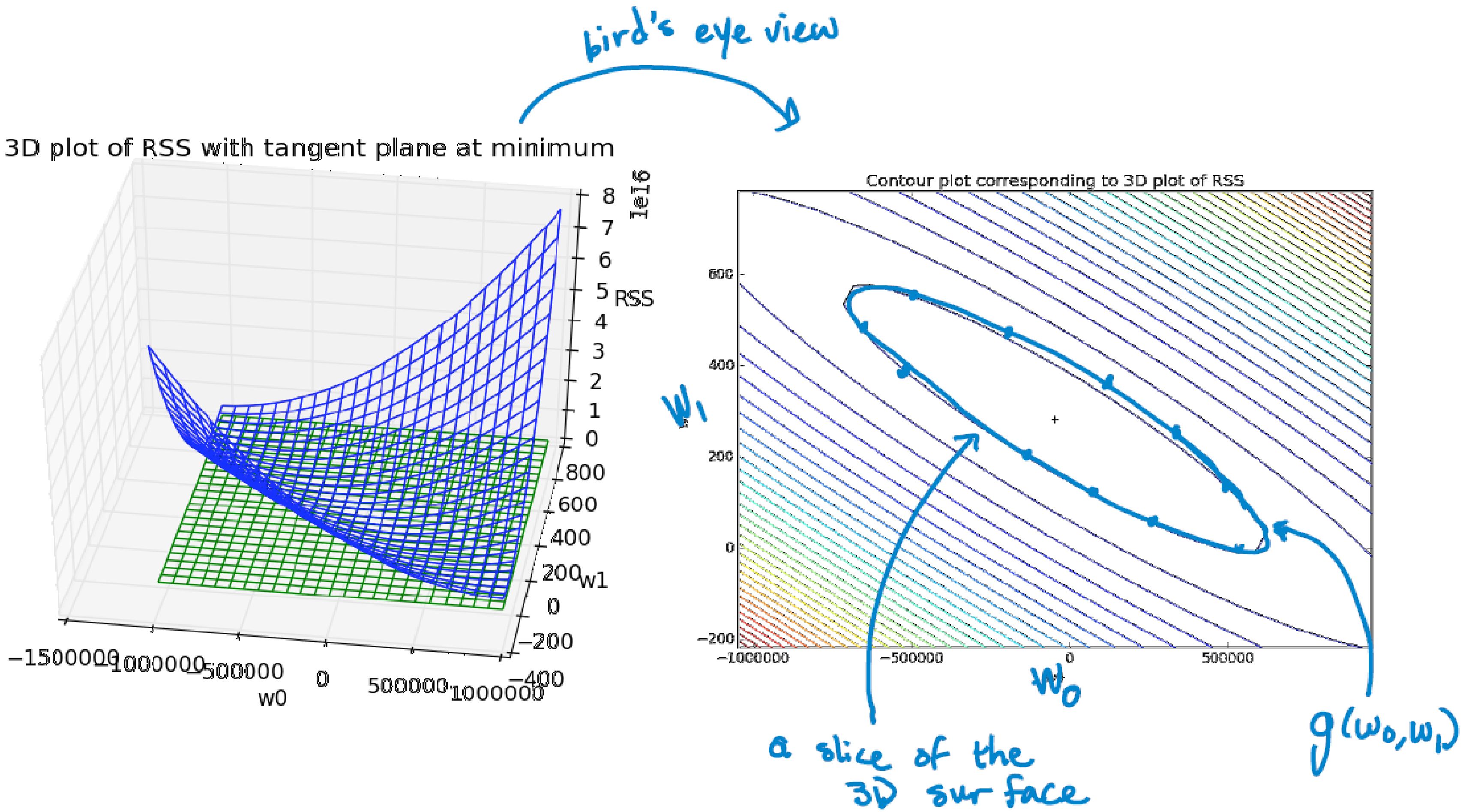
$$\nabla g(w) = \begin{bmatrix} \frac{\partial g}{\partial w_0} \\ \frac{\partial g}{\partial w_1} \\ \vdots \\ \frac{\partial g}{\partial w_p} \end{bmatrix} \Bigg\} \text{(p+1)-dimensional vector}$$

gradient

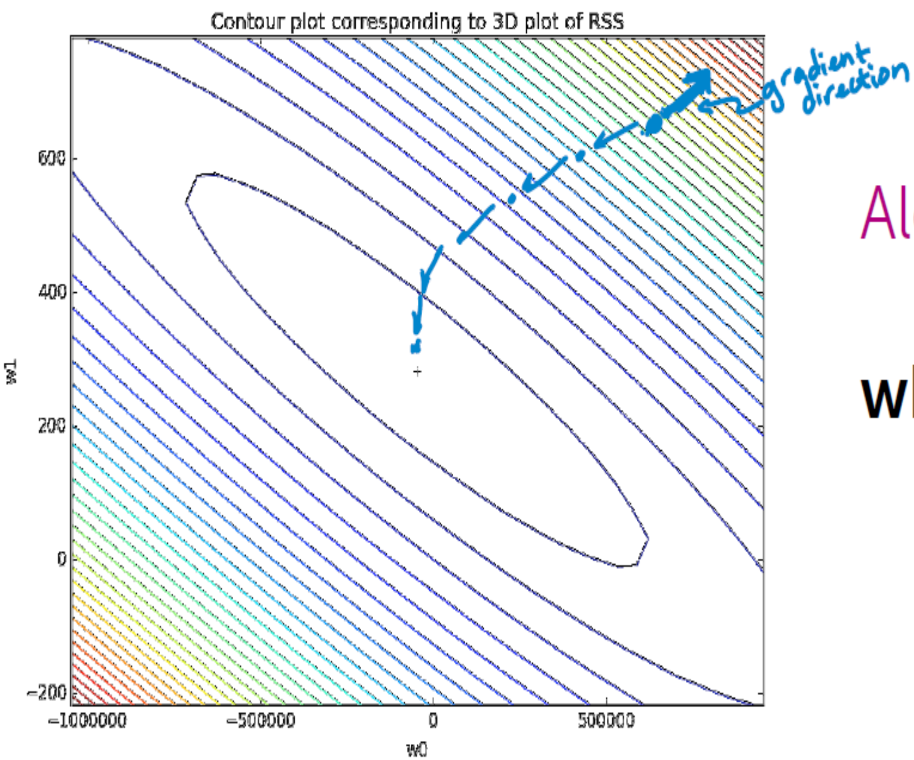
$[w_0, w_1, \dots, w_p]$

partial derivative is like a derivate with respect to w_1 treating all other variables as constants

Contour plots



Gradient descent



Algorithm:

while not converged

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

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

Compute the gradient

$$RSS(w_0,w_1) = \sum_{i=1}^{N}(y_i-[w_0+w_1x_i])^2$$

Taking the derivative w.r.t. w_0

$$\sum_{i=1}^{N} 2(y_i-[w_0+w_1x_i])^1\cdot(-1)$$

$$= -2\sum_{i=1}^{N}(y_i-[w_0+w_1x_i])$$

Putting it together:

$$\nabla RSS(w_0,w_1) = \begin{bmatrix} -2\sum_{i=1}^{N}[y_i - (w_0+w_1x_i)] \\ -2\sum_{i=1}^{N}[y_i - (w_0+w_1x_i)]x_i \end{bmatrix}$$

Taking the derivative w.r.t. w_1

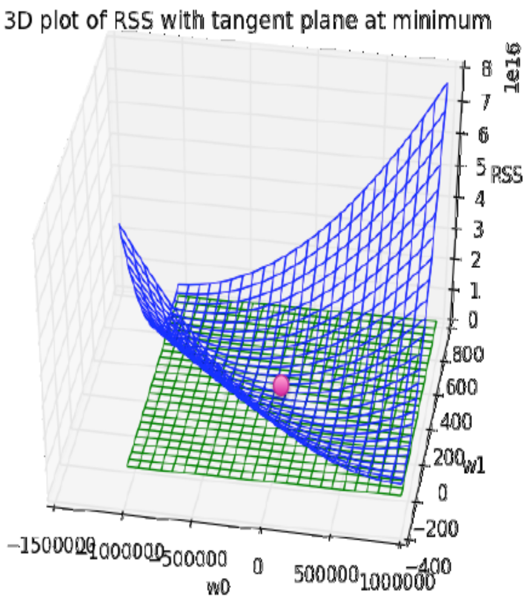
$$\sum_{i=1}^{N} 2(\underline{y_i-[w_0+w_1x_i]})^1\cdot(-x_i)$$

$$= -2\sum_{i=1}^{N}(y_i-[w_0+w_1x_i])\underline{x_i}$$

Approach 1: set gradient to 0

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

This method is called „Closed form solution"



top term:

average house sales price

estimate of the slope

average sq.ft.

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

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:

$\sum_{i=1}^{N} y_i$

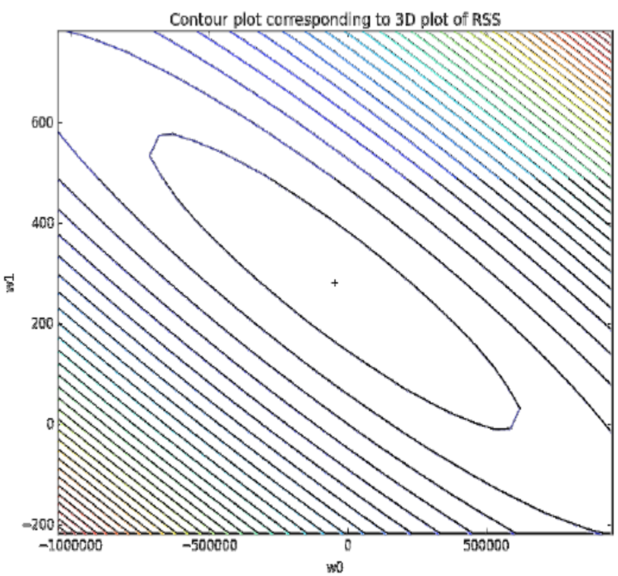
$\sum_{i=1}^{N} x_i$

$\sum_{i=1}^{N} y_i x_i$

$\sum_{i=1}^{N} x_i^2$

Approach 2: gradient descent

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



while not converged

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

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

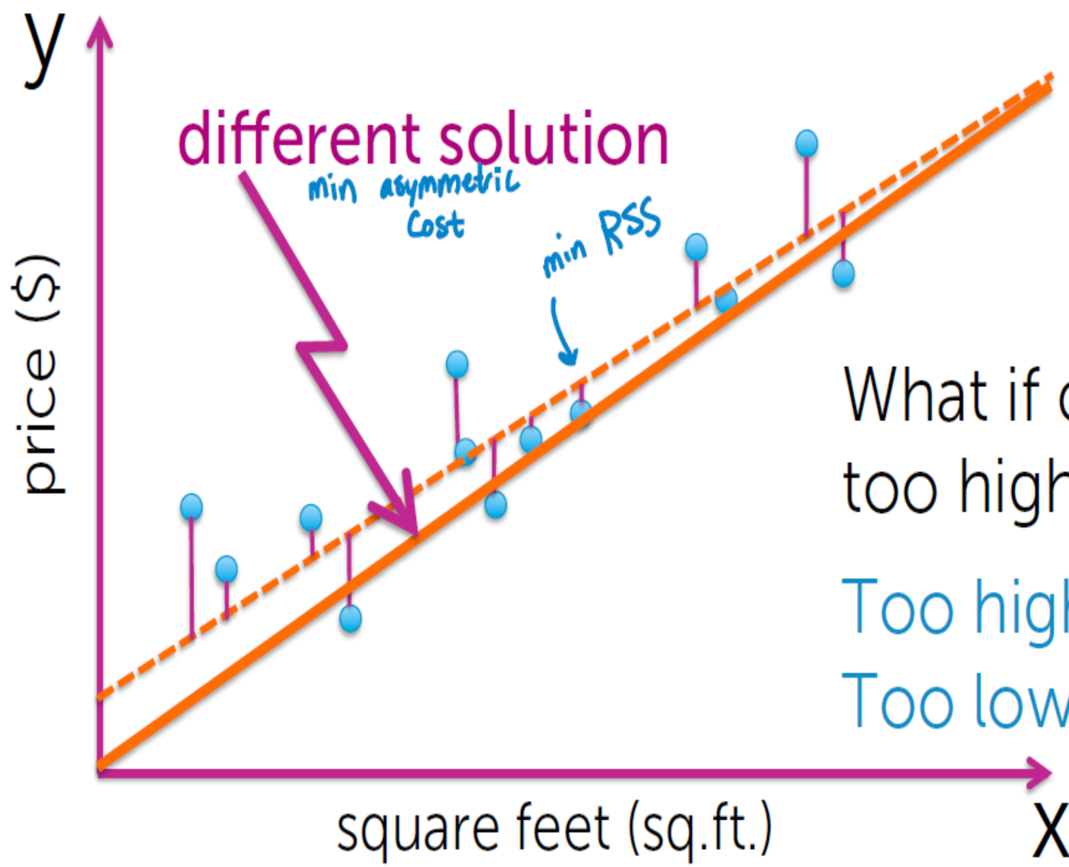
$\rightarrow$ w_0 is going to increase

similar intuition for w_1, but multiply by x_i

Comparing the approaches

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

Asymmetric cost functions



We can weight differently positive and negative errors in RSS calculations.

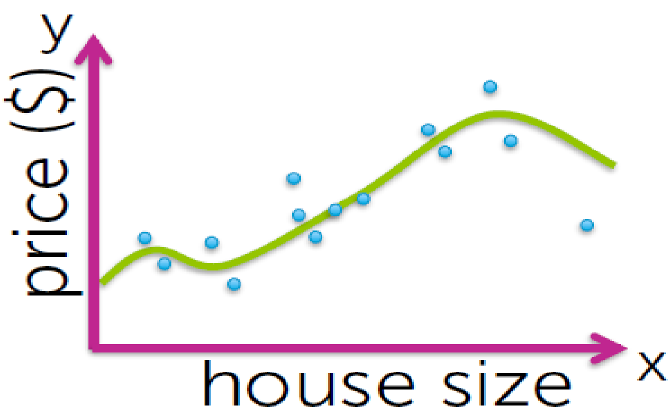
What if cost of listing house too high has **bigger cost**?

Too high → no offers ($=0)

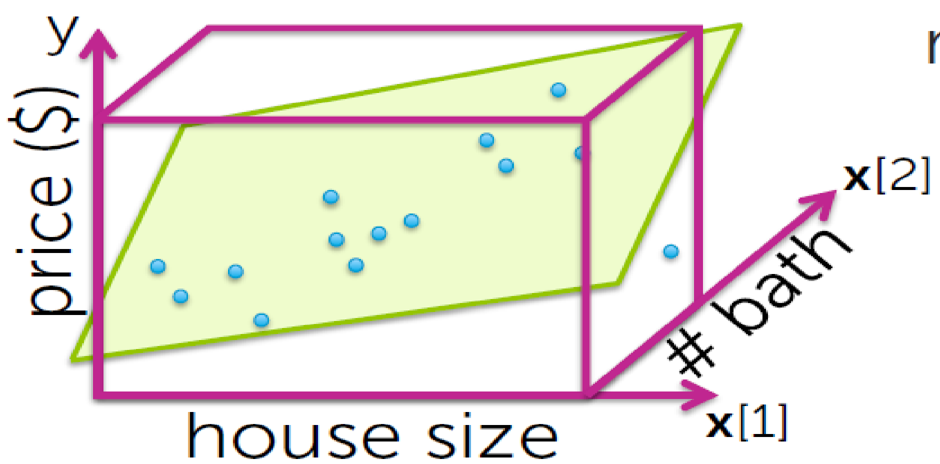
Too low → offers for lower $

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:

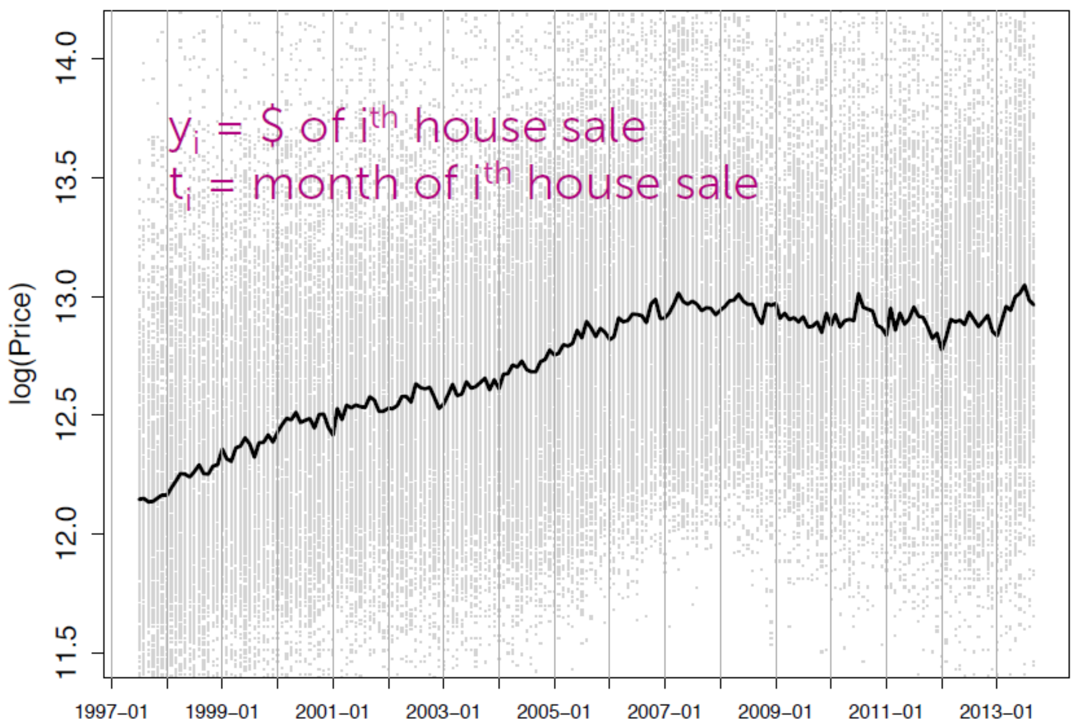
$y_i = w_0 + w_1 x_i + w_2 x_i^2 + \ldots + w_p x_i^p + \varepsilon_i$

treat as different **features**

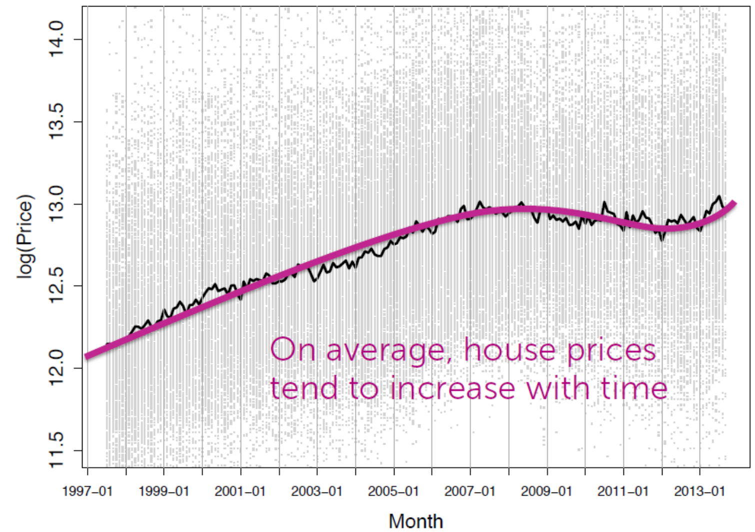
feature 1 = 1 (constant)	*parameter* 1 = w_0
feature 2 = x	*parameter* 2 = w_1
feature 3 = x^2	*parameter* 3 = w_2
…	…
feature p+1 = x^p	*parameter* p+1 = w_p

Other functional forms of one input

- Trends in time series



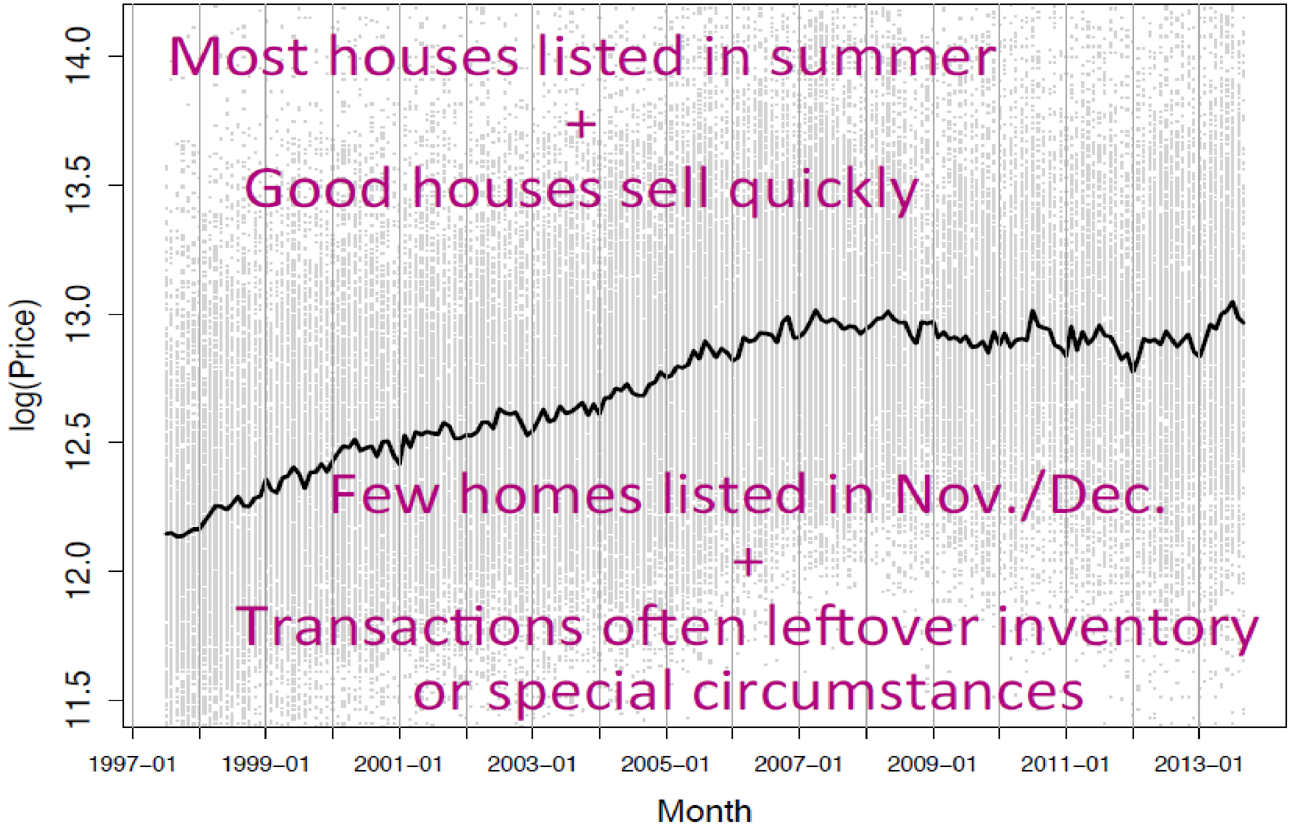
House sales recorded monthly



This trend can be modeled with polynomial function.

Other functional forms of one input

- **Seasonality**



Example of detrending

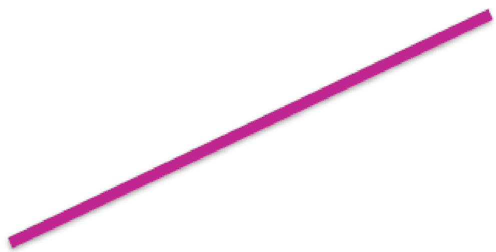
Model:

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

Unknown phase/shift

Linear trend

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,

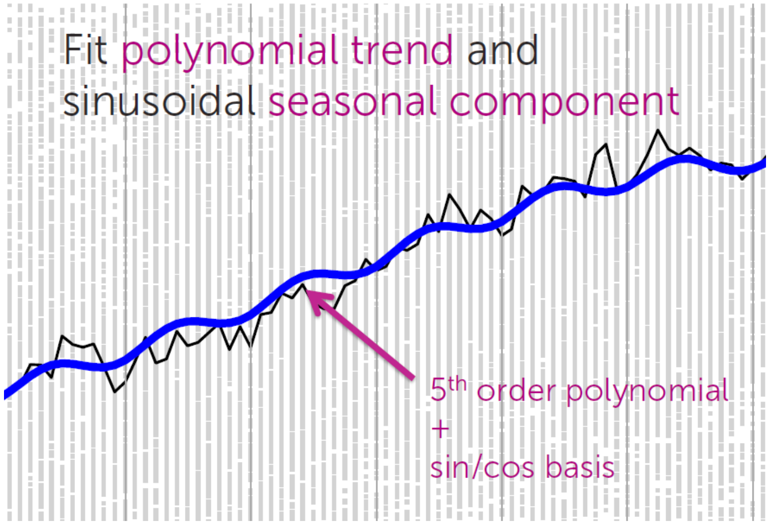
$$y_i = w_0 + w_1 t_i + w_2 \sin(2\pi t_i / 12) + w_3 \cos(2\pi t_i / 12) + \varepsilon_i$$

feature 1 = 1 (constant)

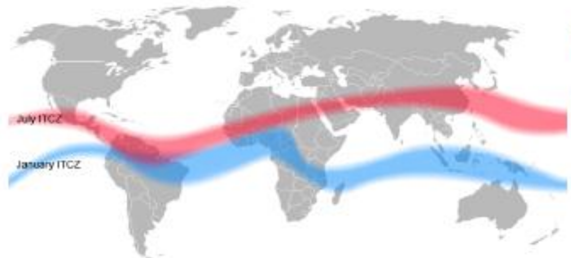
feature 2 = t

feature 3 = sin(2πt/12)

feature 4 = cos(2πt/12)

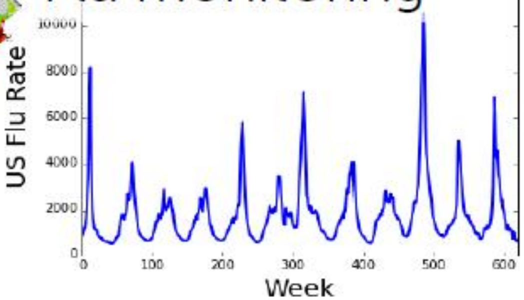


Other examples of seasonality

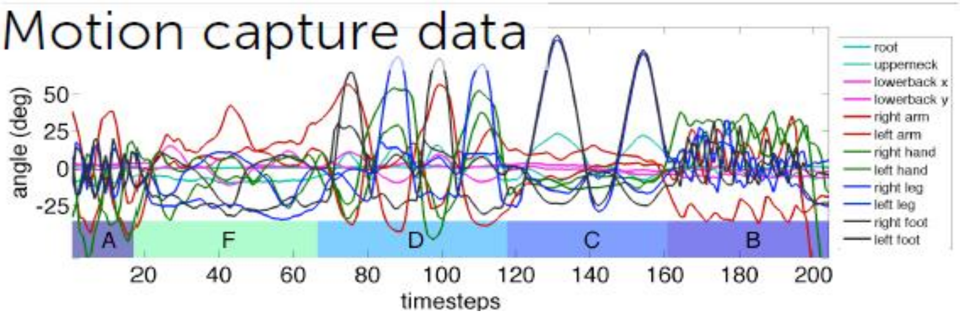


Weather modeling
(e.g., temperature, rainfall)

Flu monitoring



Motion capture data



Generic basic expansion

Model:

$$y_i = w_0 h_0(x_i) + w_1 h_1(x_i) + \dots + w_D h_D(x_i) + \varepsilon_i$$

$$= \sum_{j=0}^{D} w_j h_j(x_i) + \varepsilon_i$$

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

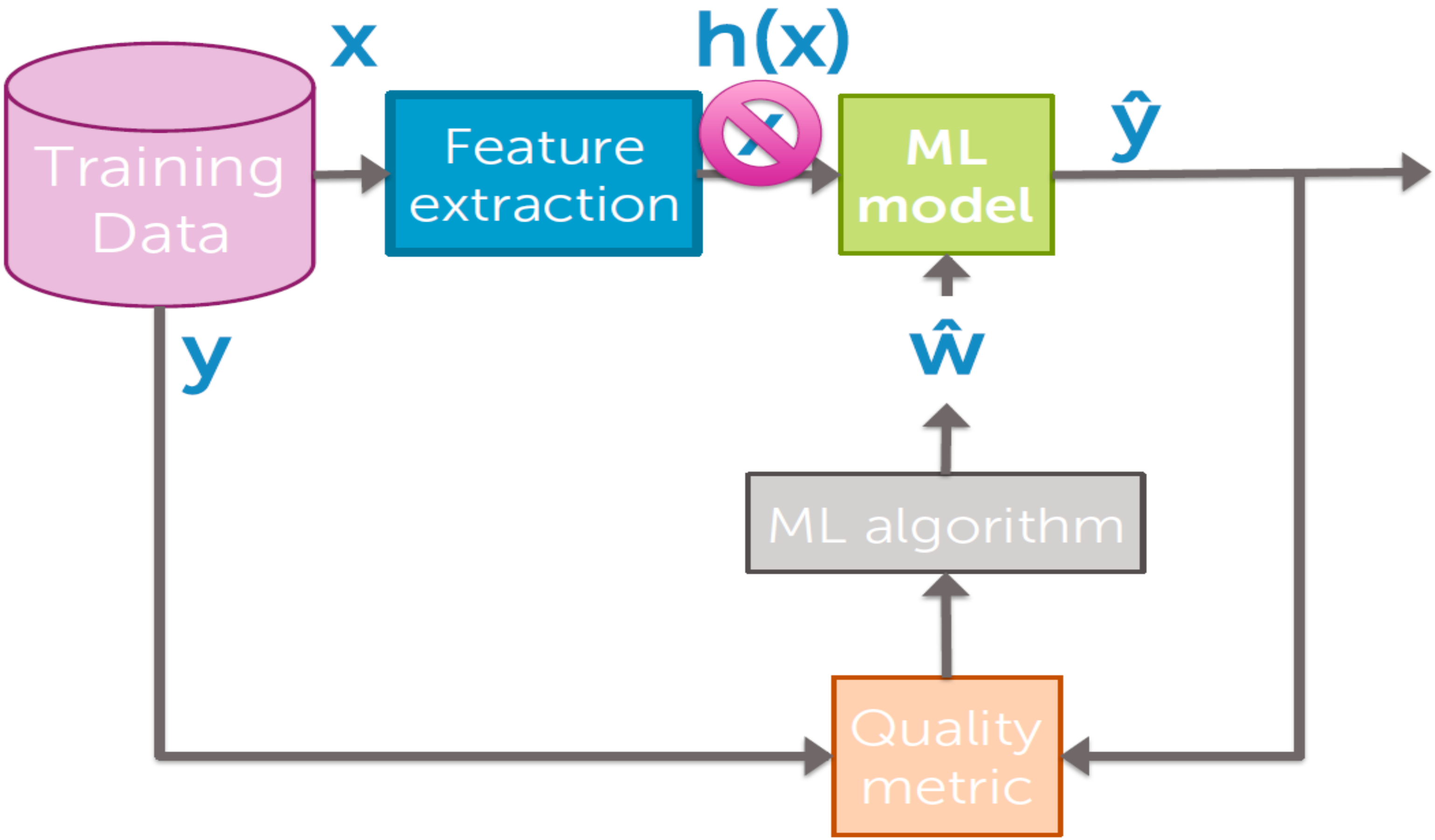
feature 2 = $h_1(x)$... e.g., x

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

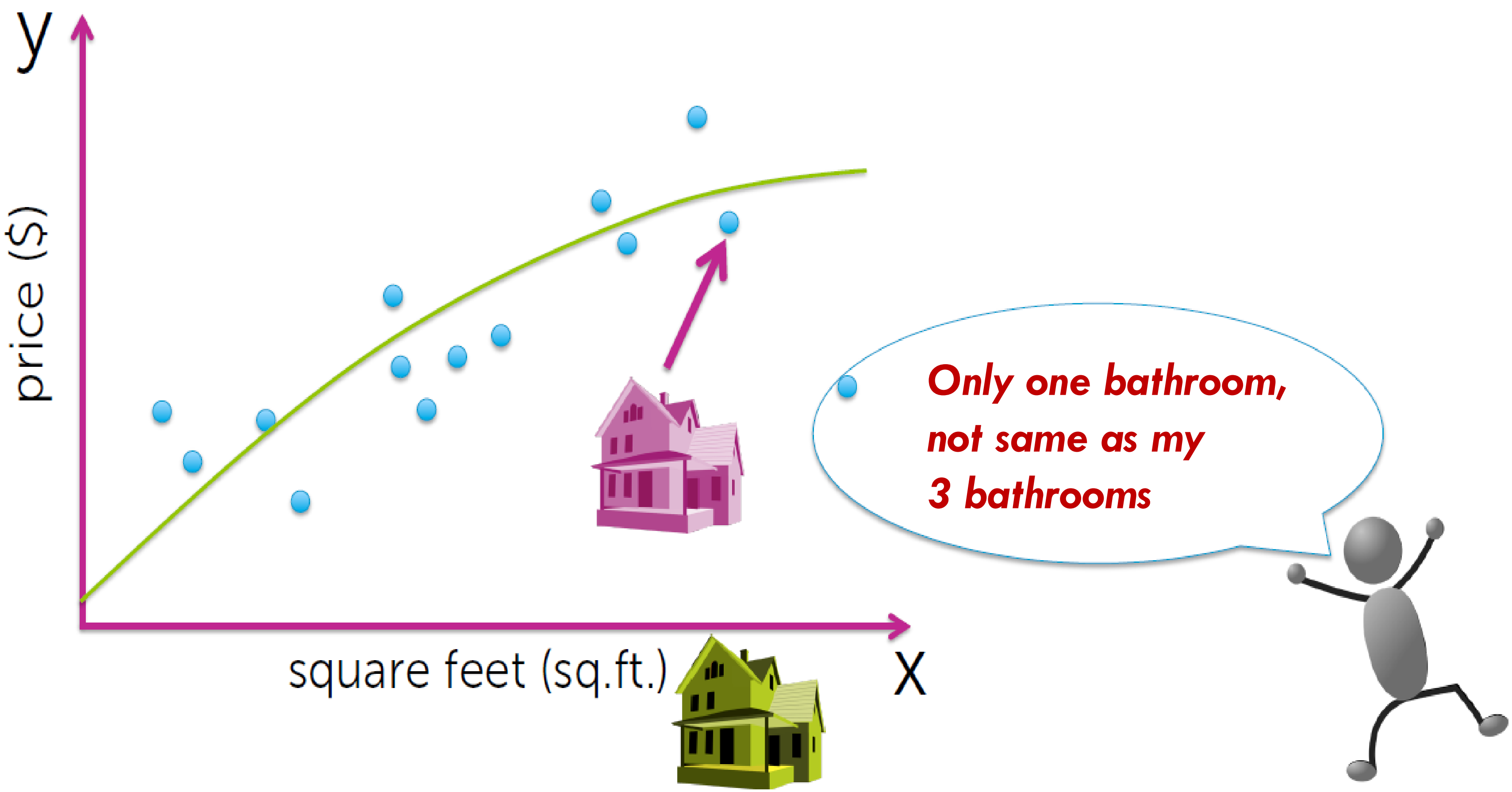
...

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

More realistic flow chart

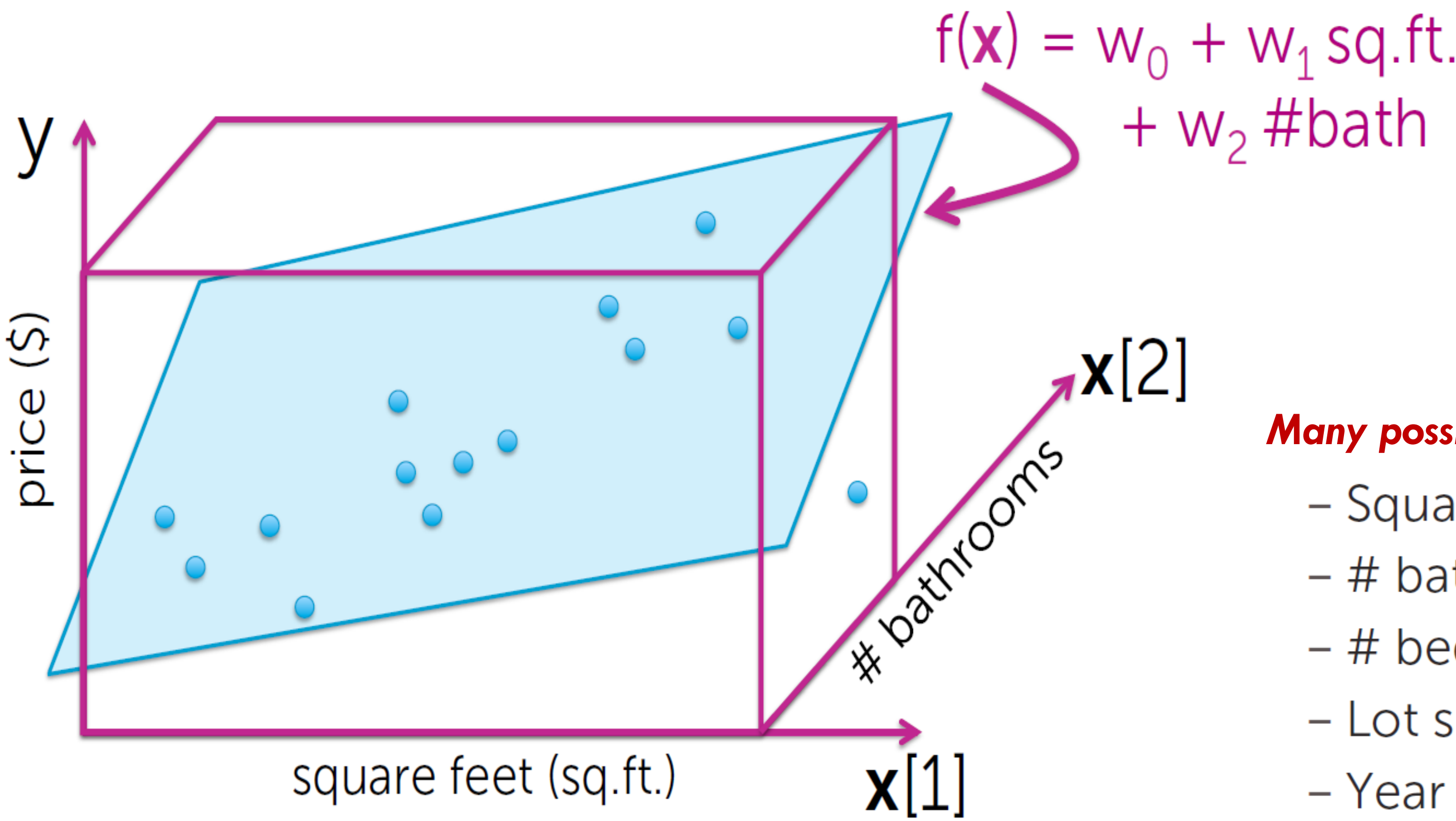


Incorporating multiple inputs



– ...

Incorporating multiple inputs



Many possible inputs

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

General notation

Output: y scalar

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

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

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

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:

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

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

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

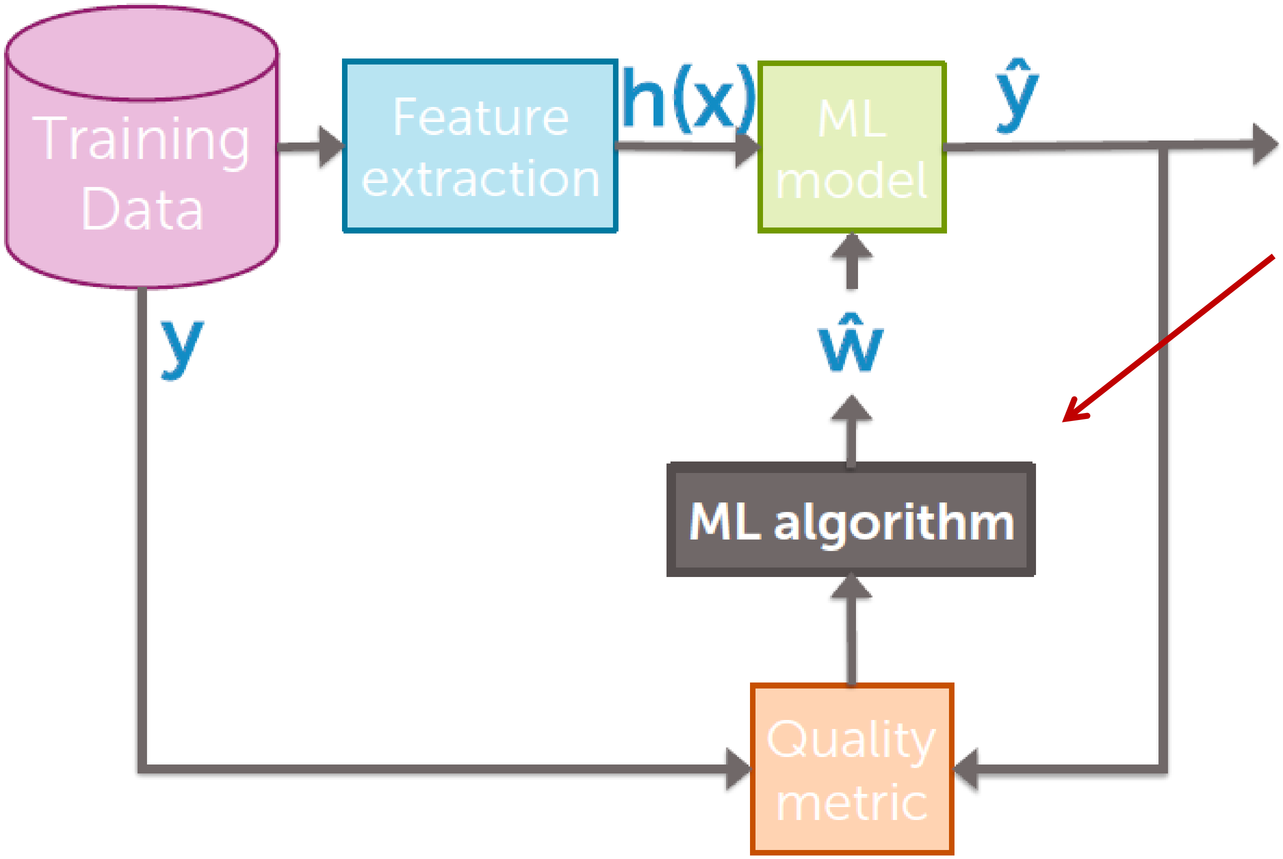
feature 2 = $h_1(\mathbf{x})$ … e.g., $\mathbf{x}[1]$ = sq. ft.

feature 3 = $h_2(\mathbf{x})$ … e.g., $\mathbf{x}[2]$ = #bath
or, $\log(\mathbf{x}[7])\, \mathbf{x}[2]$ = log(#bed) x #bath

…

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

Fitting in D-dimmensions

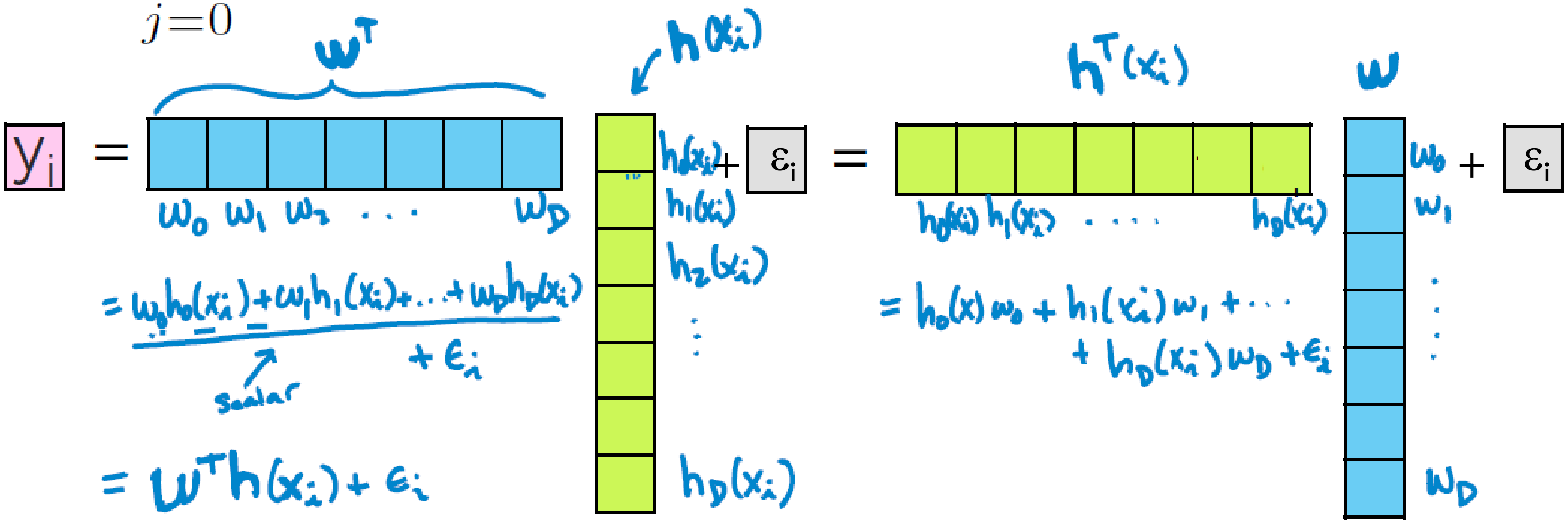


Look now at this block

Rewriting in vector notation

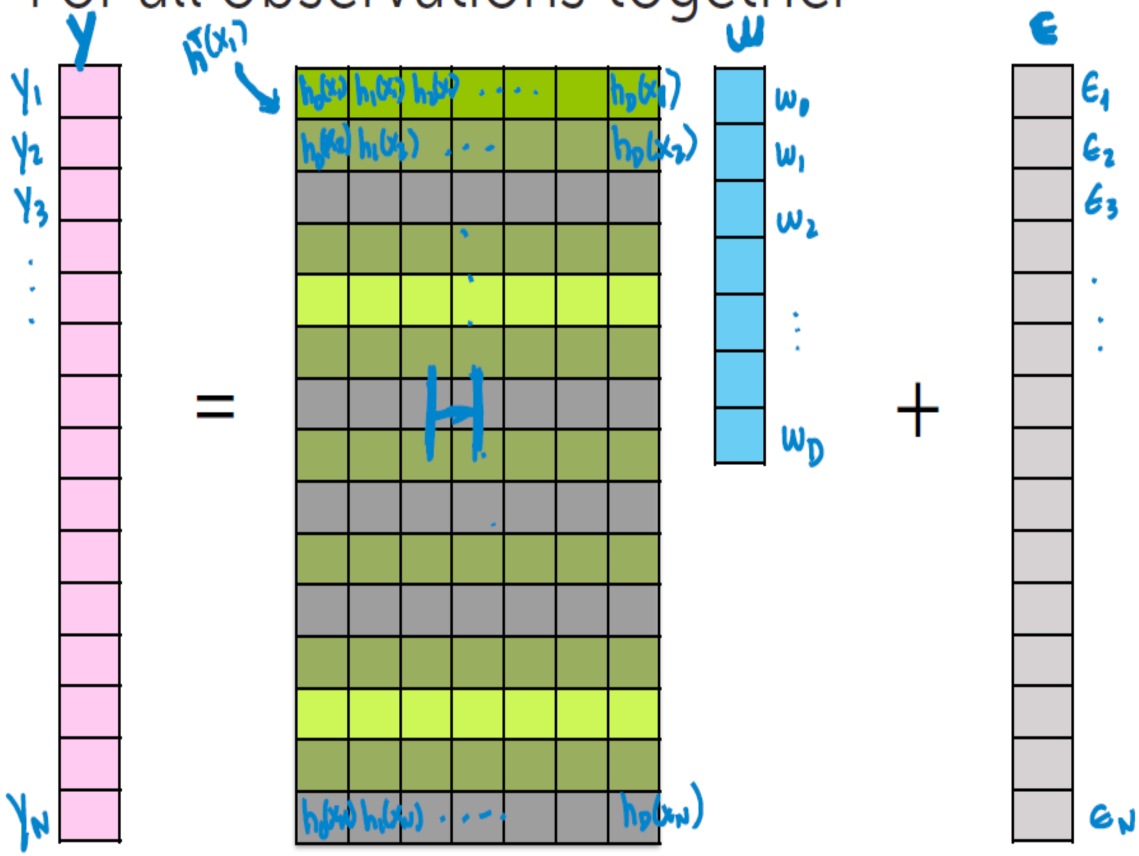
For observation i

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



Rewriting in matrix notation

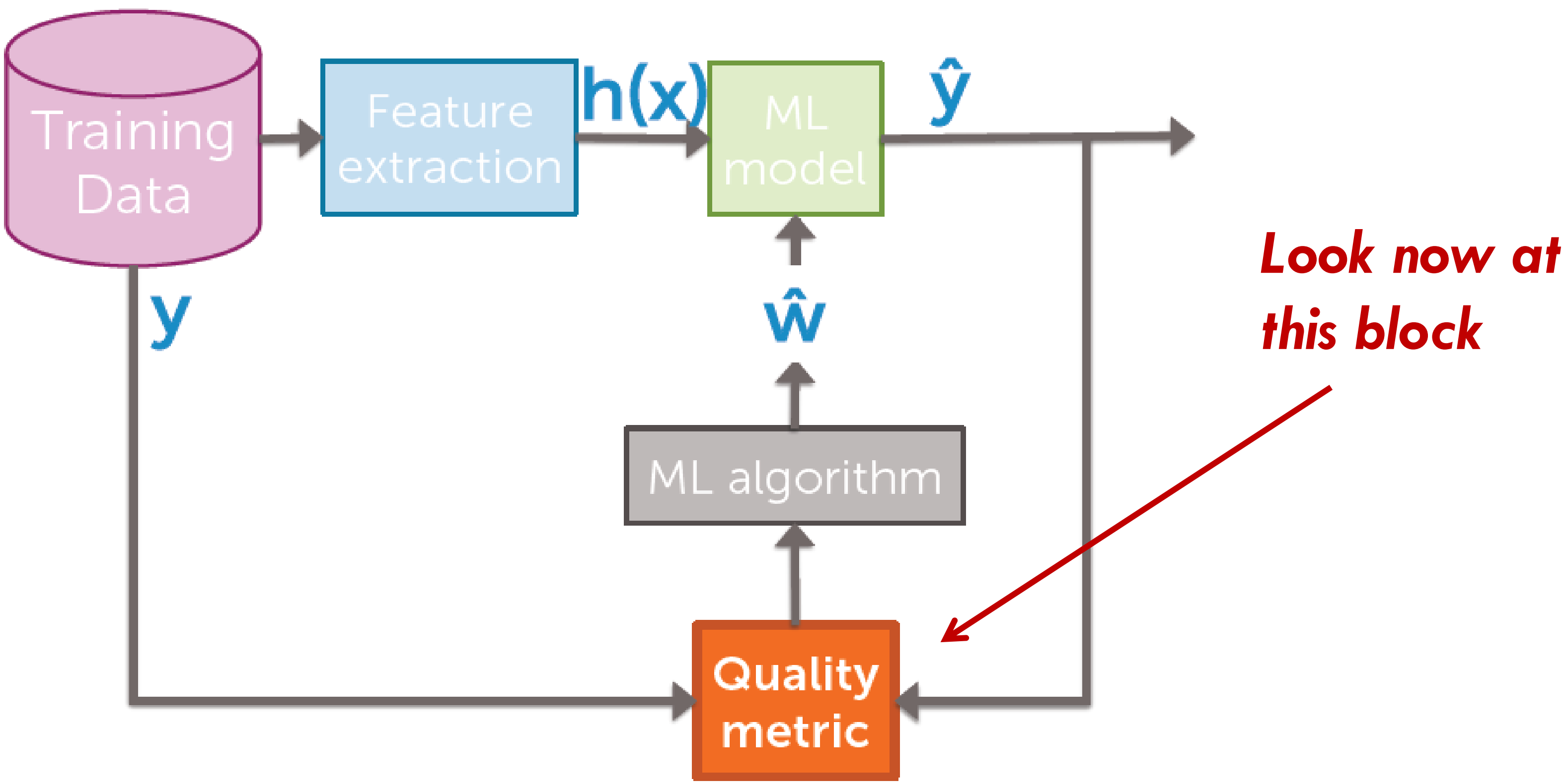
For all observations together



Here is our ML algorithm

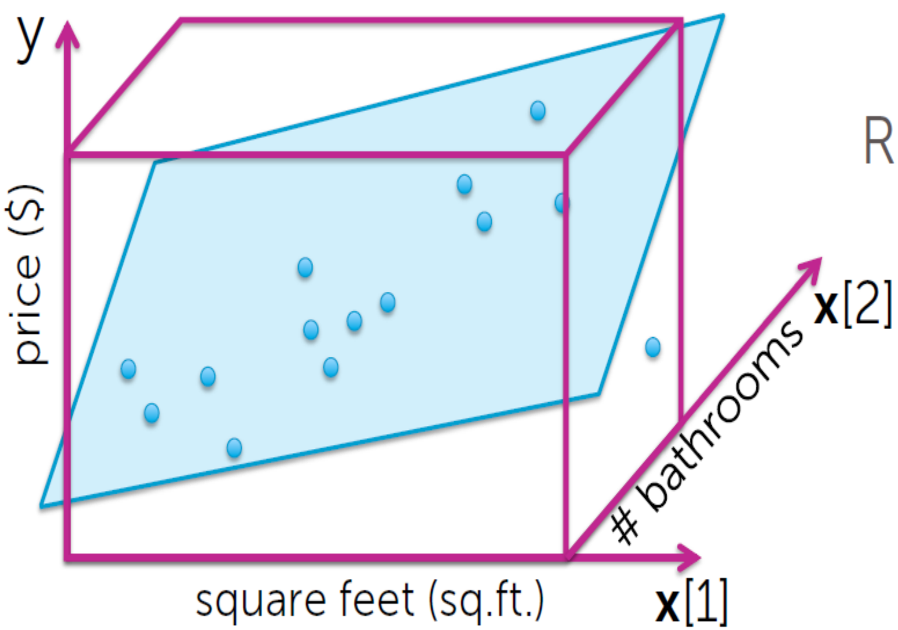
$$\Rightarrow \boxed{y = Hw + \epsilon}$$

Fitting in D-dimmensions

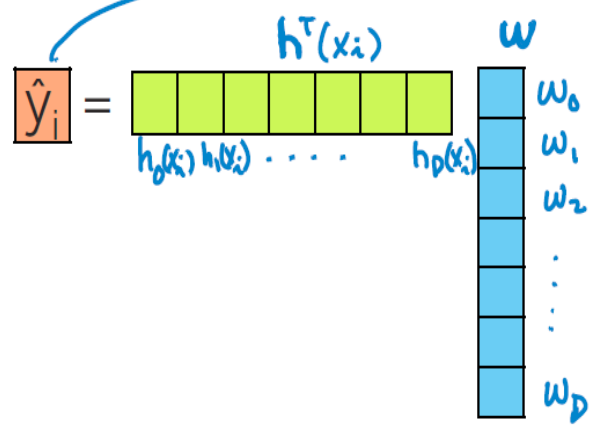


Cost function in D-dimmension

RSS in vector notation



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



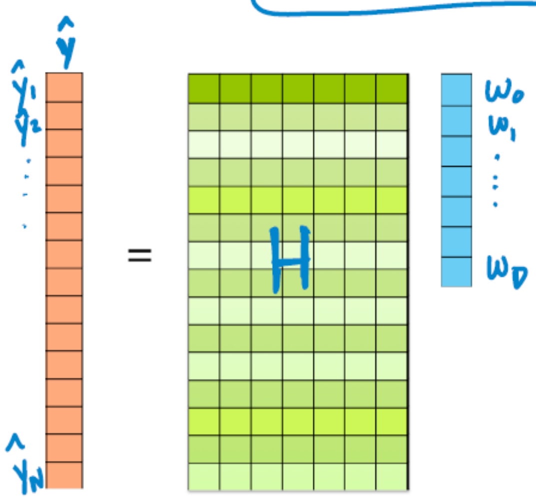
Cost function in D-dimmension

RSS in matrix notation

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

$$= (y - Hw)^T (y - Hw)$$

Why? (part 1)



$$\hat{y} = Hw$$

$$(y - \underbrace{Hw}) = (y - \hat{y}) = \begin{bmatrix} \text{residual}_1 \\ \text{residual}_2 \\ \vdots \\ \text{residual}_N \end{bmatrix}$$

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

Regression model for D-dimmension

Gradient of RSS

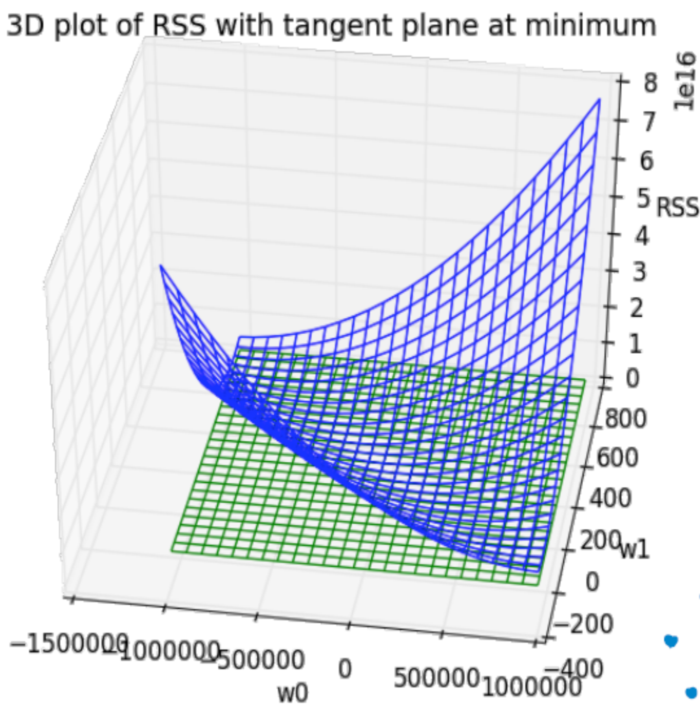
$$\nabla \text{RSS}(\mathbf{w}) = \nabla[(\mathbf{y}-\mathbf{H}\mathbf{w})^T(\mathbf{y}-\mathbf{H}\mathbf{w})]$$
$$= -2\mathbf{H}^T(\mathbf{y}-\mathbf{H}\mathbf{w})$$

Why? By analogy to 1D case:

$$\frac{d}{dw}(\underbrace{y}_{\text{scalars}}-\underbrace{hw}_{\text{scalars}})(y-hw) = \frac{d}{dw}(y-hw)^2 = 2\cdot(y-hw)^1(-h)$$
$$= -2h(y-hw)$$

Regression model for D-dimmension

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 **w**:

$$-\cancel{2}H^Ty + \cancel{2}H^TH\hat{w} = 0$$

$$H^TH\hat{w} = H^Ty$$

$$\underbrace{(H^TH)^{-1}H^TH}_{I}\hat{w} = (H^TH)^{-1}H^Ty$$

$$\hat{w} = (H^TH)^{-1}H^Ty$$

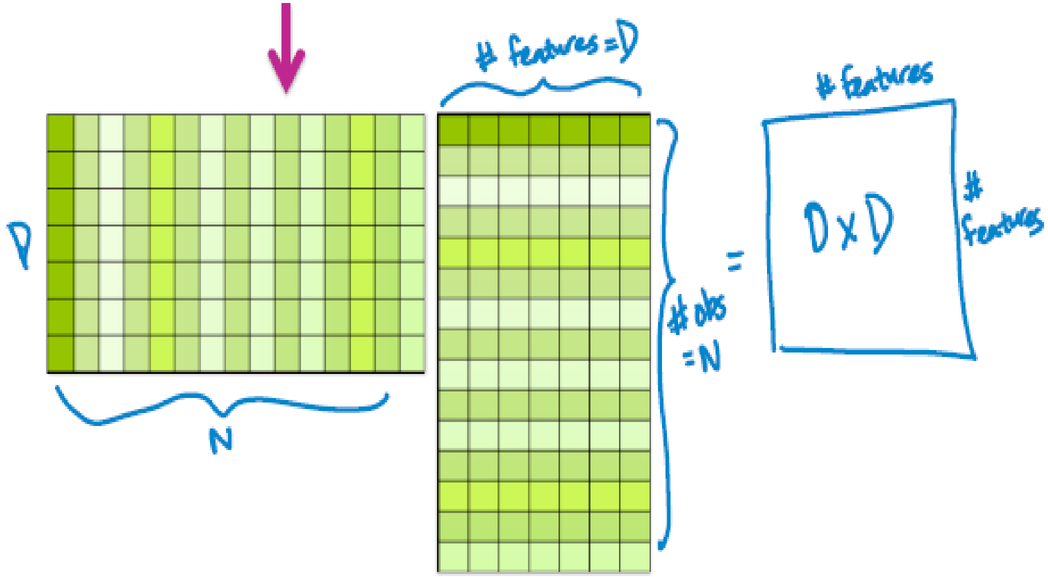
- $A^{-1}A = I$
- $Iv = v$

$IV = V$

Closed-form solution

This matrix might not be invertible.

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



Invertible if:

really, # of linearly ind. observations

In most cases is $N > D$

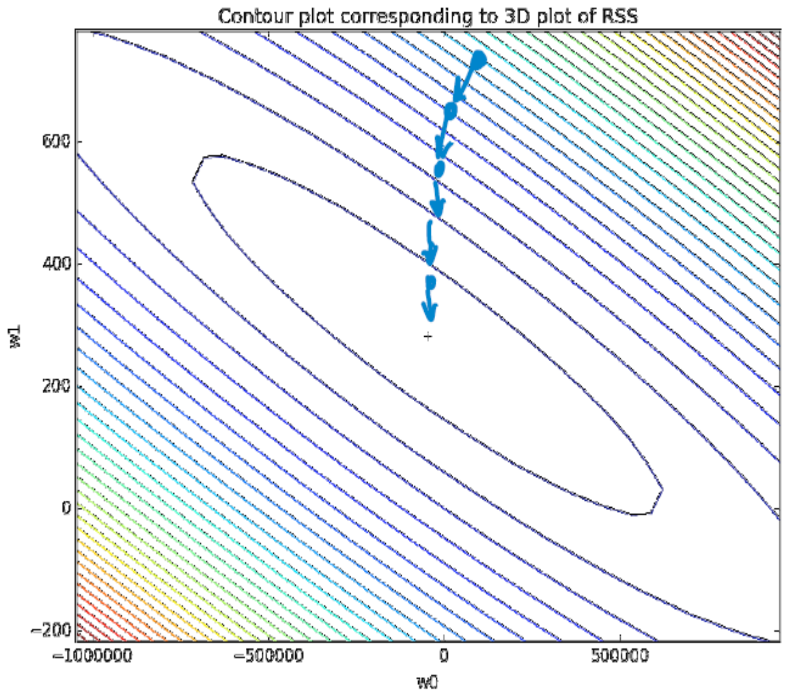
Complexity of inverse:

$O(D^3)$

This might not be CPU feasible.

Regression model for D-dimmension

Approach 2: gradient descent



We initialise our solution somewhere and then …

while not converged

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

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

Gradient descent

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

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

Partial with respect to w_j

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

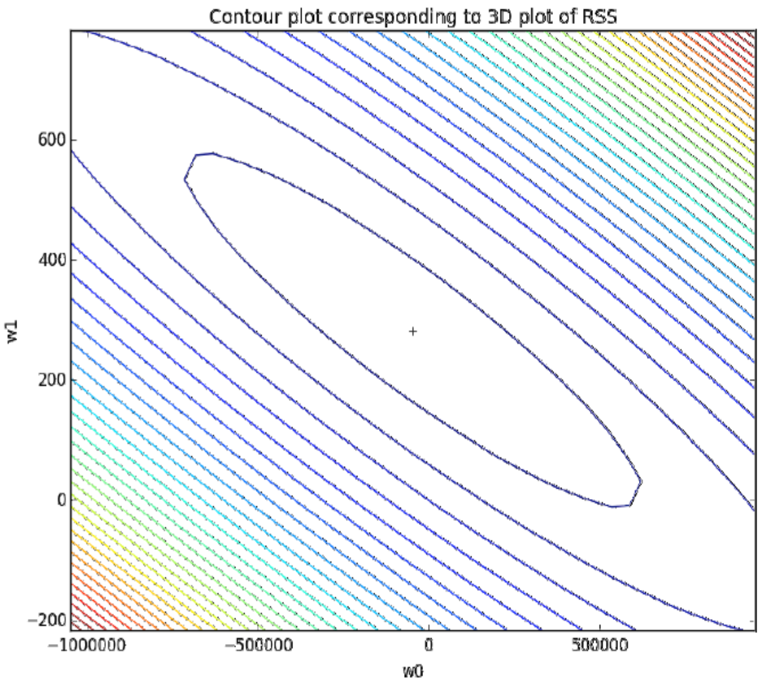
$$= -2\sum_{i=1}^{N} h_j(x_i)\left(y_i - h(x_i)^T w\right)$$

Update to j^{th} feature weight:

$$w_j^{(t+1)} \leftarrow w_j^{(t)} - \eta\Big(-2\sum_{i=1}^{N} h_j(x_i)\big(y_i - \underbrace{h^T(x_i)\,w^{(t)}}_{\hat{y}_i(w^{(t)})}\big)\quad\Big)$$

Summary of gradient descent

Extremely useful algorithm in several applications



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

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

$\sqrt{\text{partial}[0]^2 + \ldots + \text{partial}[D]^2}$

for $j=0,\ldots,D$

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

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

$t \leftarrow t + 1$

ACCESSING PERFORMANCE

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

$L(\underset{\text{actual value}}{y}, \underbrace{f_{\hat{w}}(\mathbf{x})}_{\hat{f}(\mathbf{x}) = \text{predicted value } \hat{y}})$

Examples:
(assuming loss for underpredicting = overpredicting)

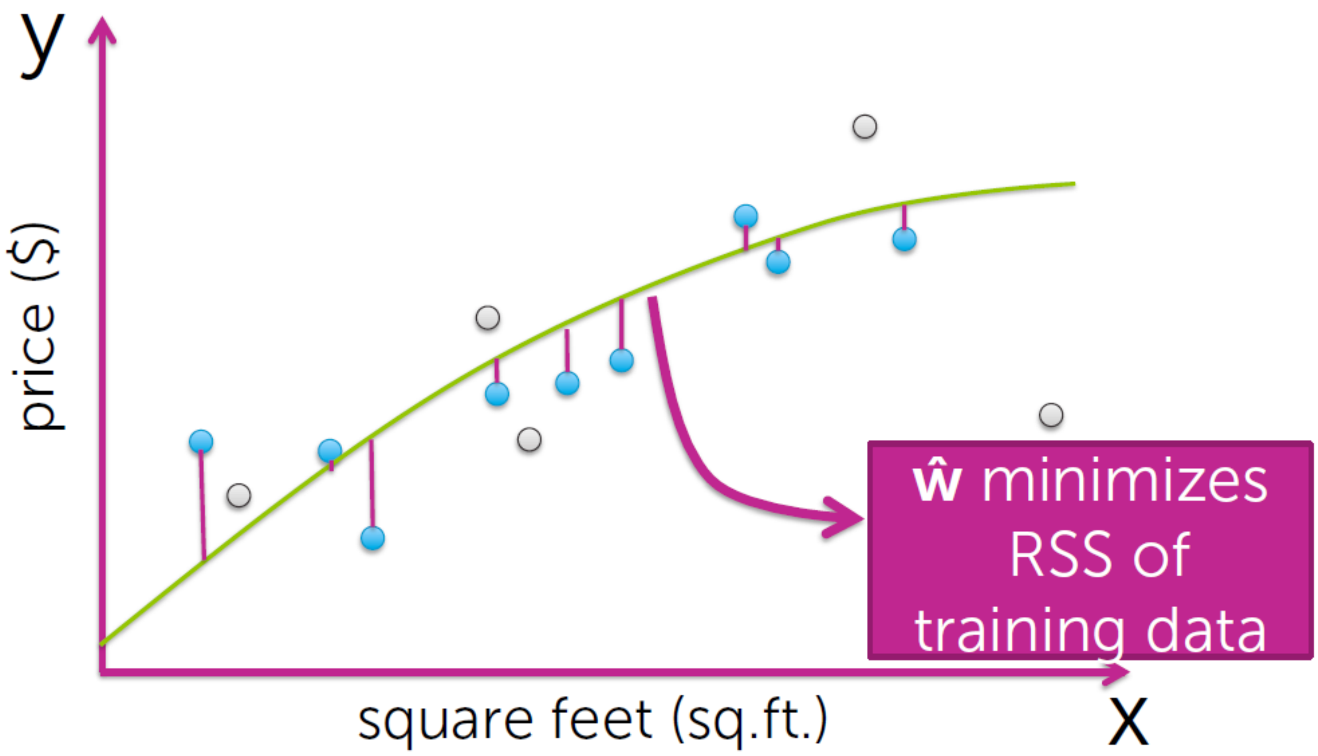
Symmetric loss functions

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

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

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,…

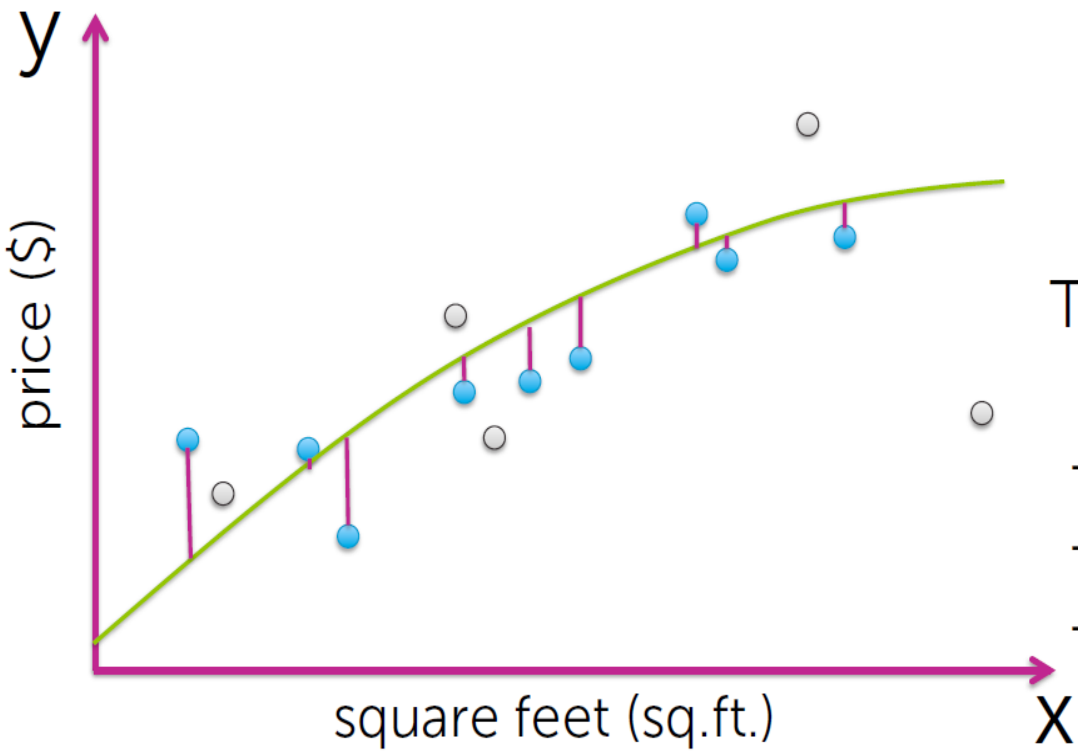
2. Training error
   = avg. loss on houses in training set

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

fit using training data

Training error

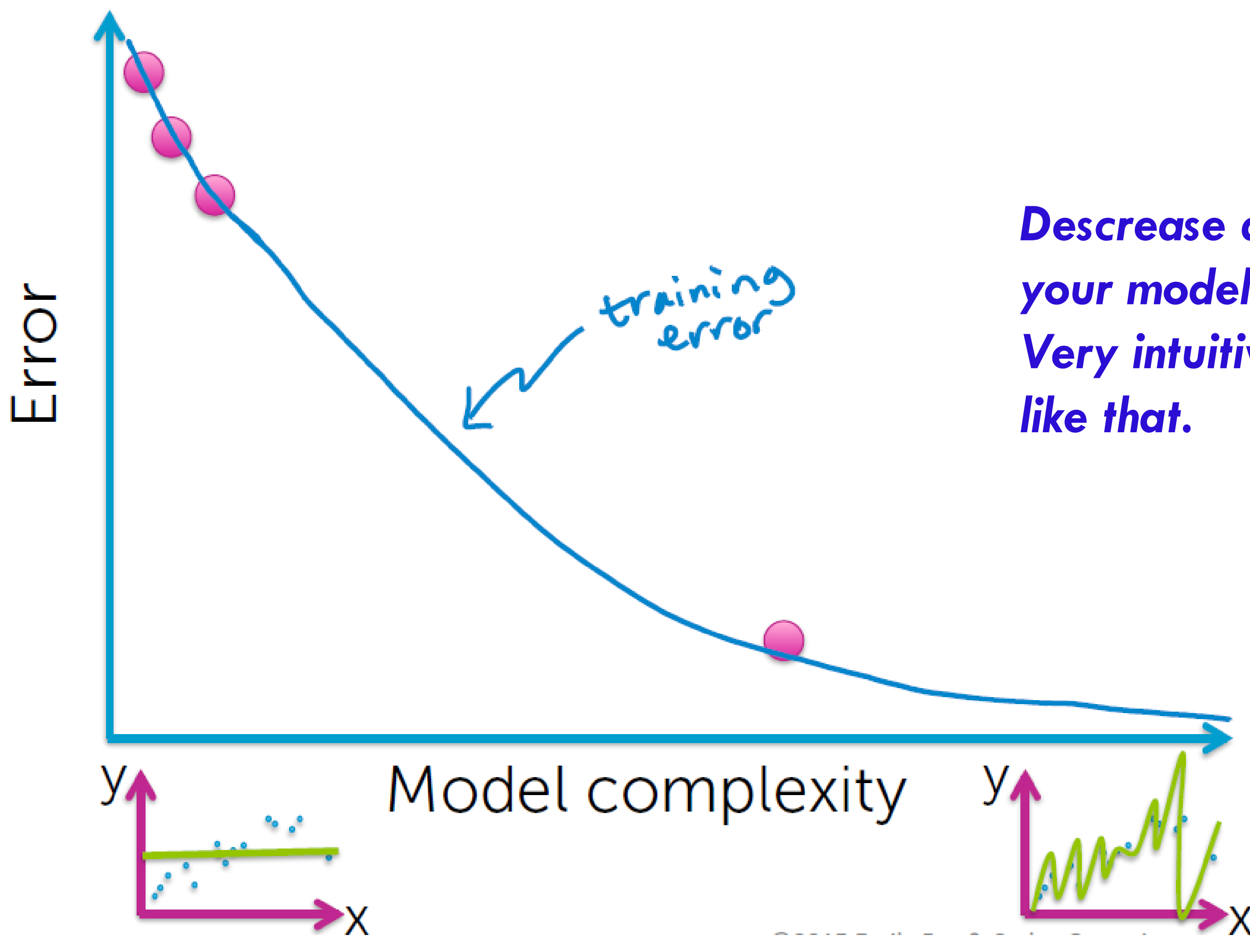
Use squared error loss $(y-f_{\hat{w}}(x))^2$



Convention is to take average here

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

Training error vs. model complexity

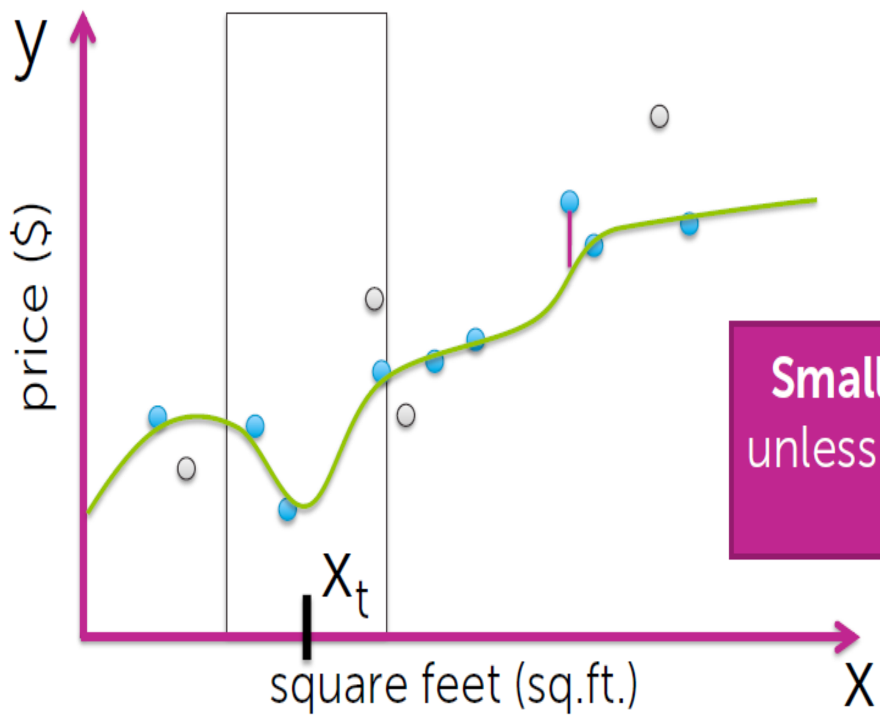


Descrease as you increase your model complexity. Very intuitive why it is like that.

Is training error a good measure?

Issue: Training error is overly optimistic

because $\hat{\mathbf{w}}$ was fit to training data



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

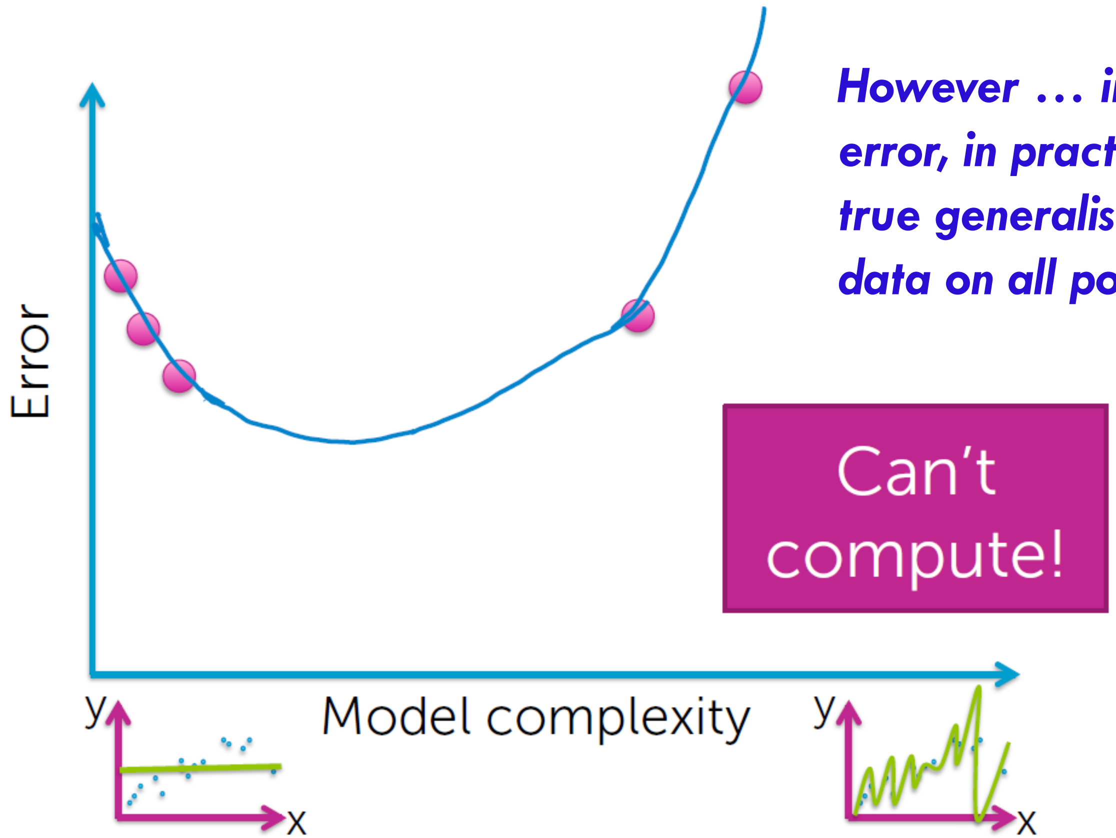
Small training error ≠> good predictions
unless training data includes everything you might ever see

Generalisation (true) error

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

Lots of houses
in neighborhood,
but not in dataset

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.

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"

Compute test error

Test error

= avg. loss on houses in test set

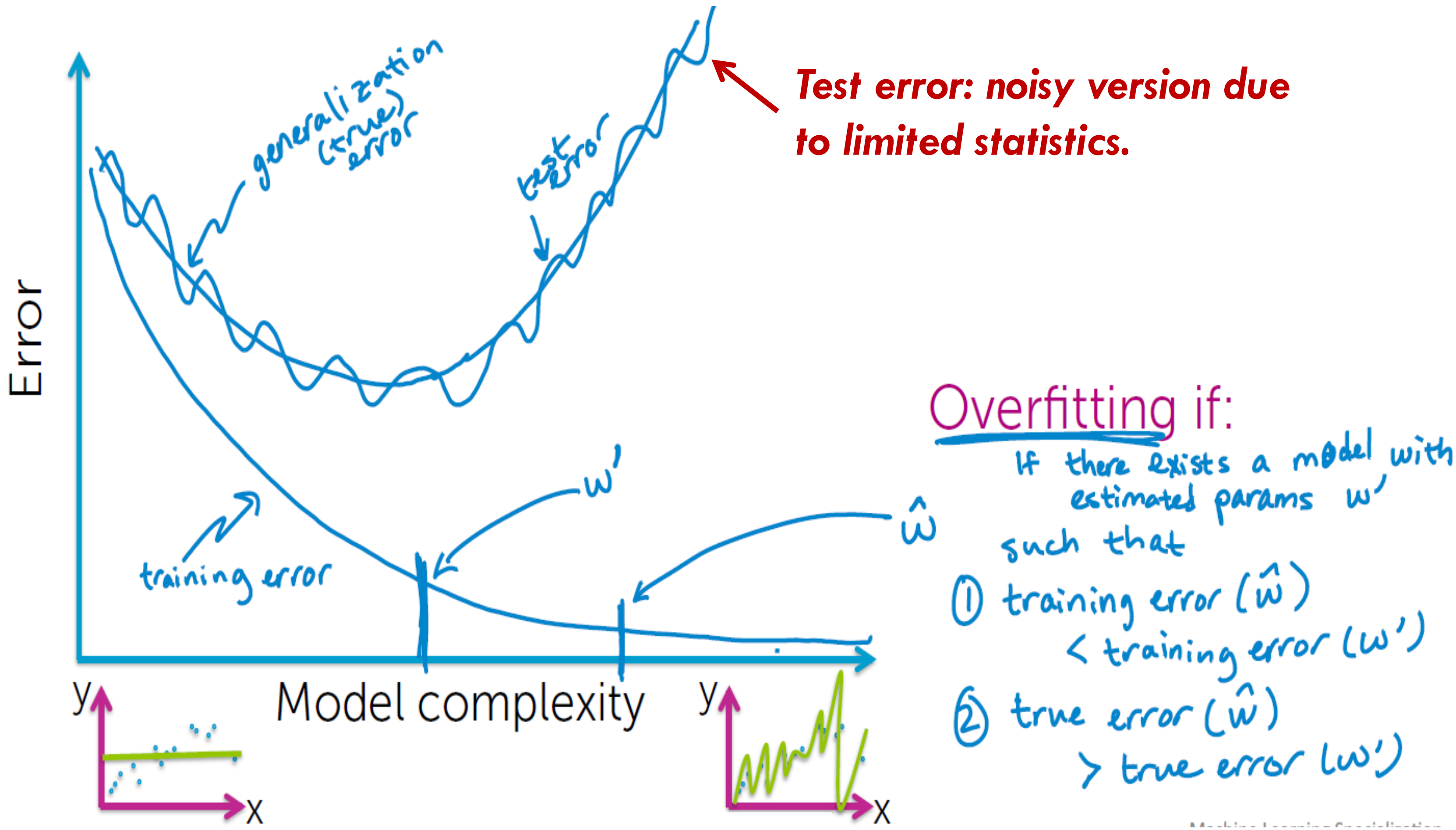
$$= \frac{1}{N_{test}} \sum_{i \text{ in test set}} L(y_i, f_{\hat{\mathbf{w}}}(\mathbf{x}_i))$$

\# test points

fit using training data

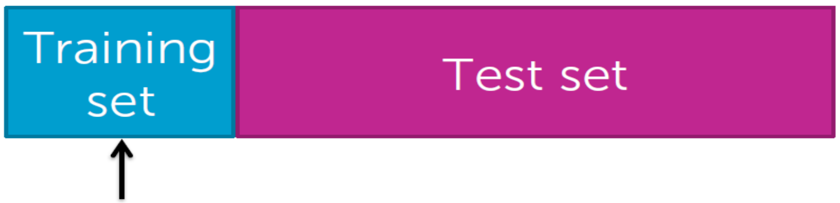
has never seen test data!

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



Test error: noisy version due to limited statistics.

Training/test splits



Too few → $\hat{\mathbf{w}}$ 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

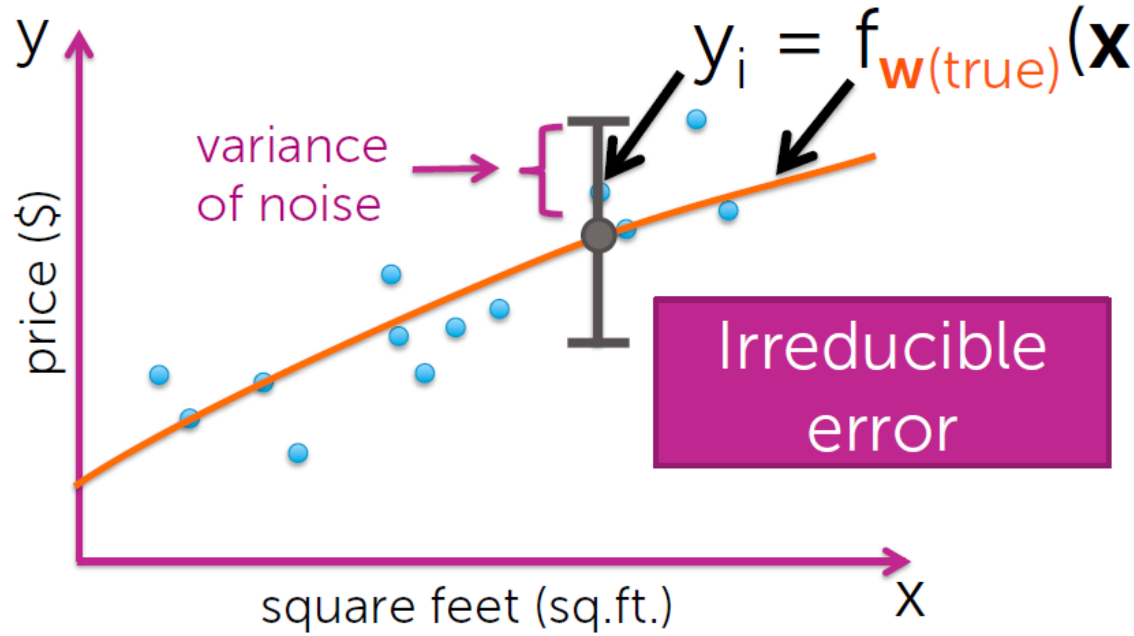
In forming predictions, there are 3 sources of error:

1. Noise
2. Bias
3. Variance

Data are inherently noisy

There is some true relatioship between sq.ft and value of the house, specific to the given house.

$$y_i = f_{\mathbf{w}(\mathrm{true})}(\mathbf{x}_i) + \varepsilon_i$$

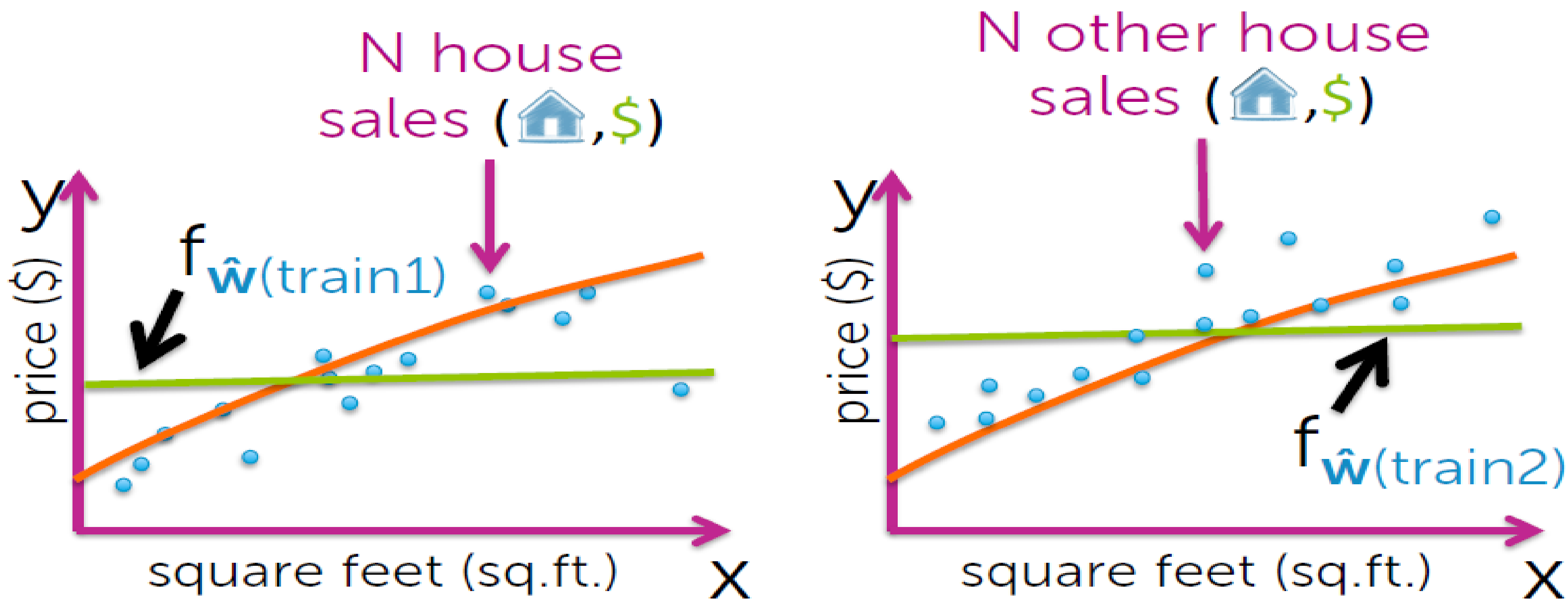


We cannot reduce it by chosing better model or procedure, It is beyond our control.

Bias contribution

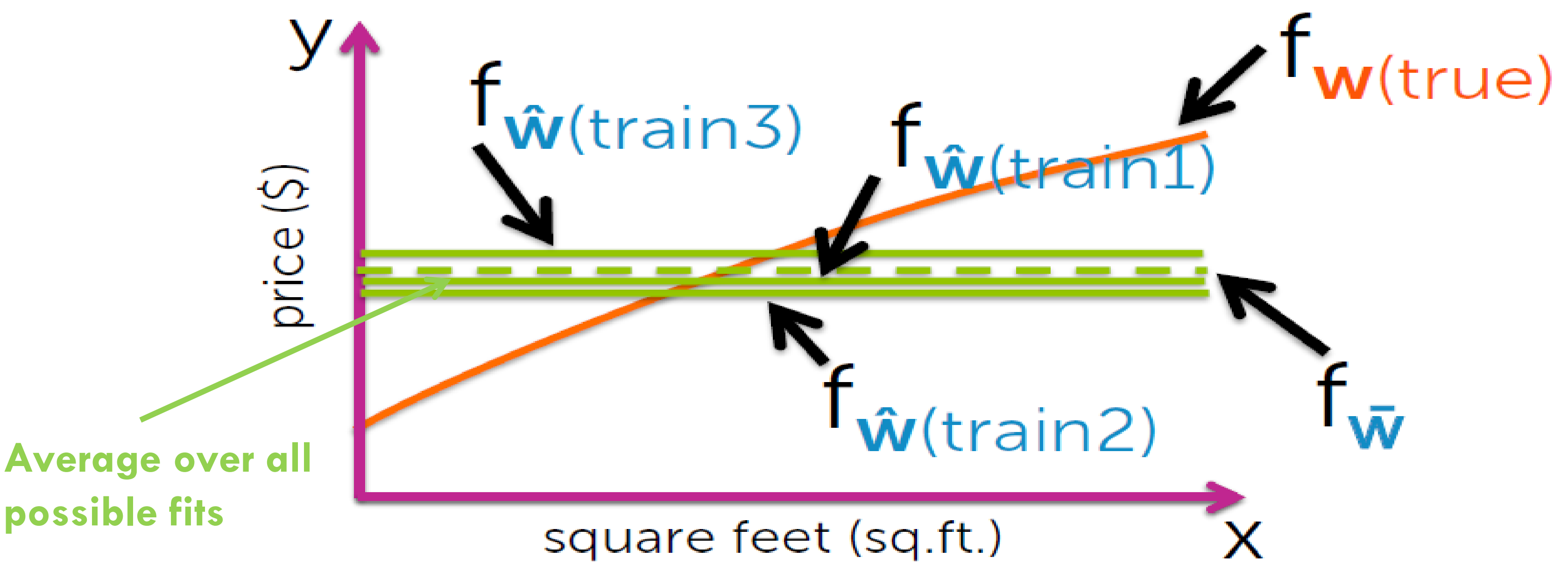
This contribution we can control.

Assume we fit a constant function



Bias contribution

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



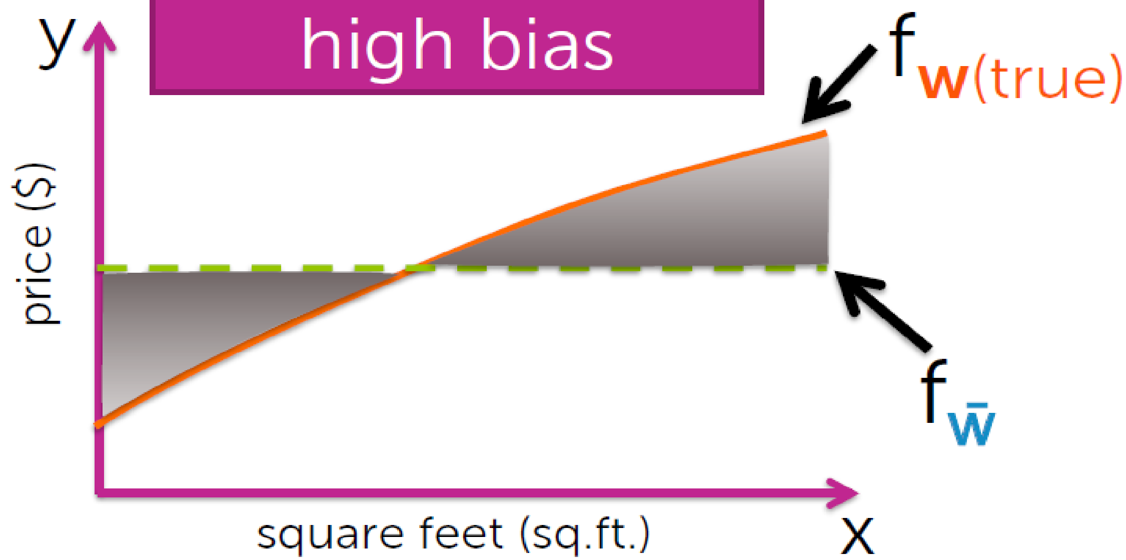
Bias contribution

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

Is our approach flexible enough to capture $f_{\mathbf{w}(\text{true})}$?

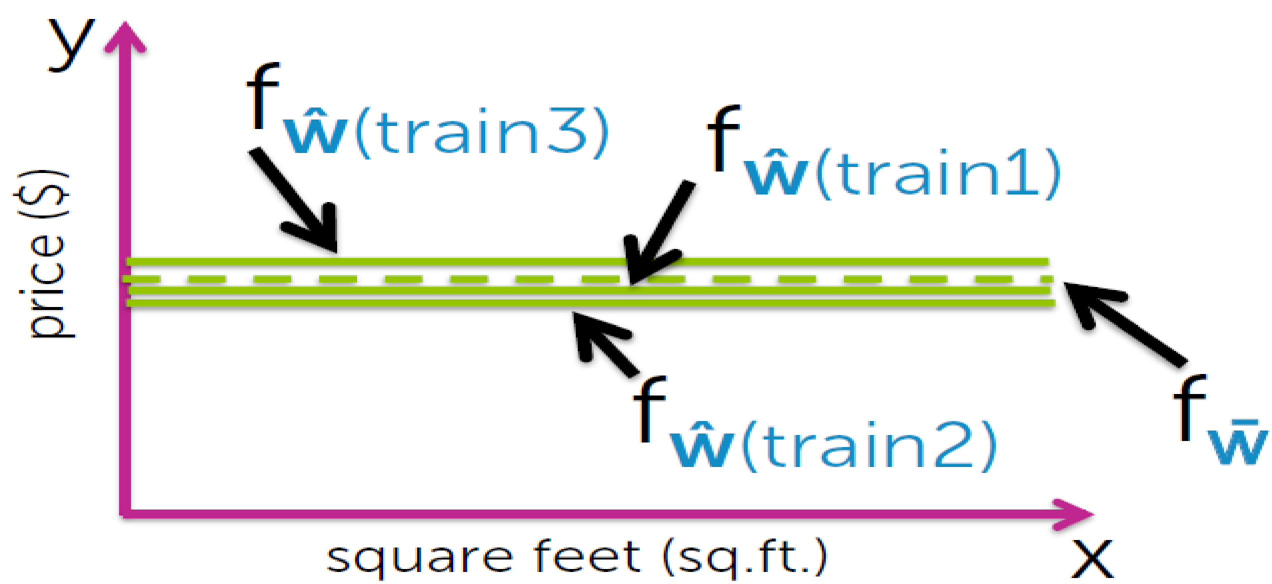
If not, error in predictions.

low complexity → high bias



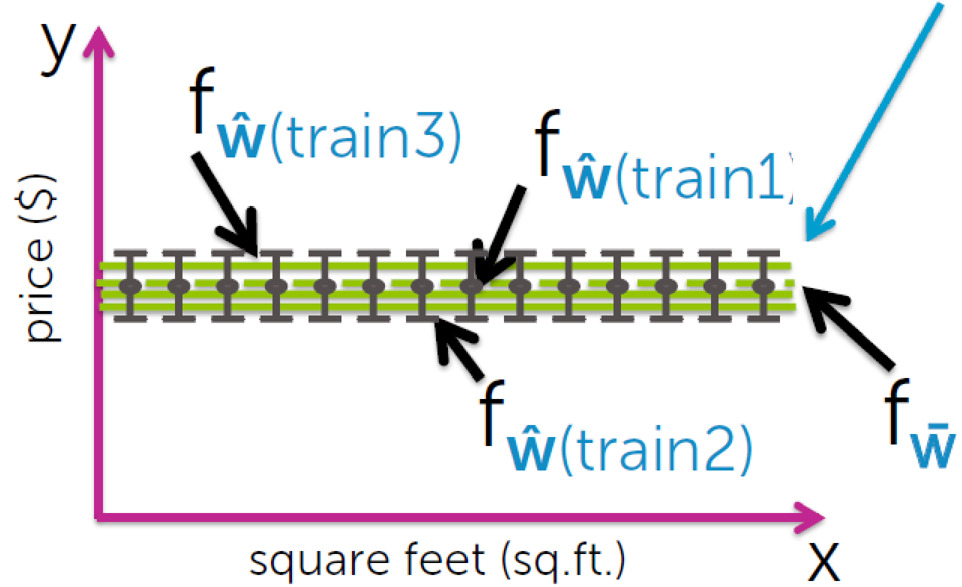
Variance contribution

How much do specific fits vary from the expected fit?



Variance contribution

How much do specific fits vary from the expected fit?



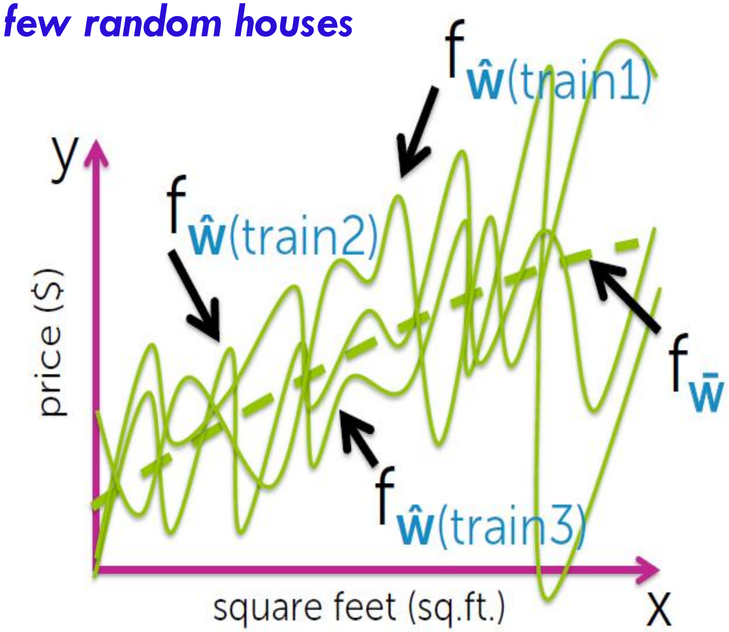
Can specific fits vary widely?
If so, erratic predictions

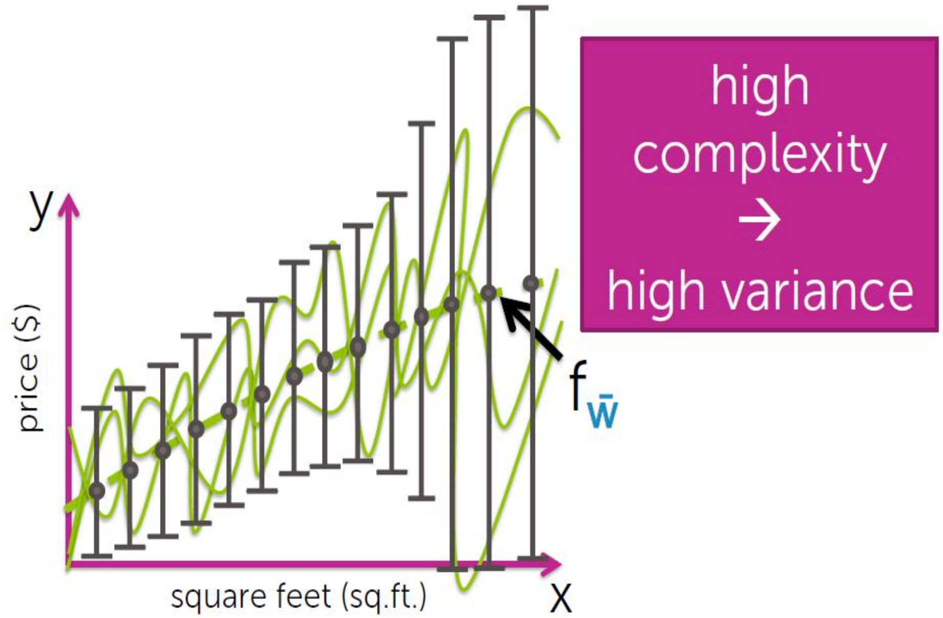
low complexity → low variance

Variance of high complexity models

Assume we fit a high-order polynomial

For each train remove few random houses

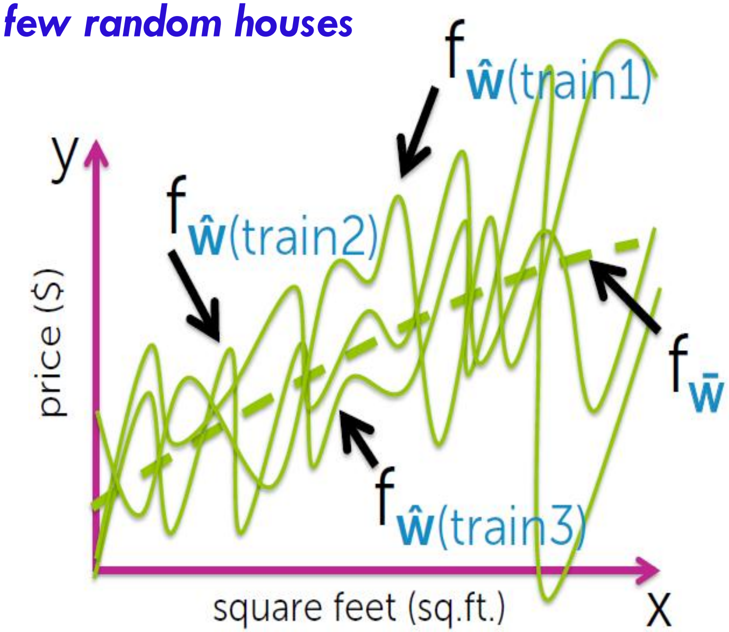


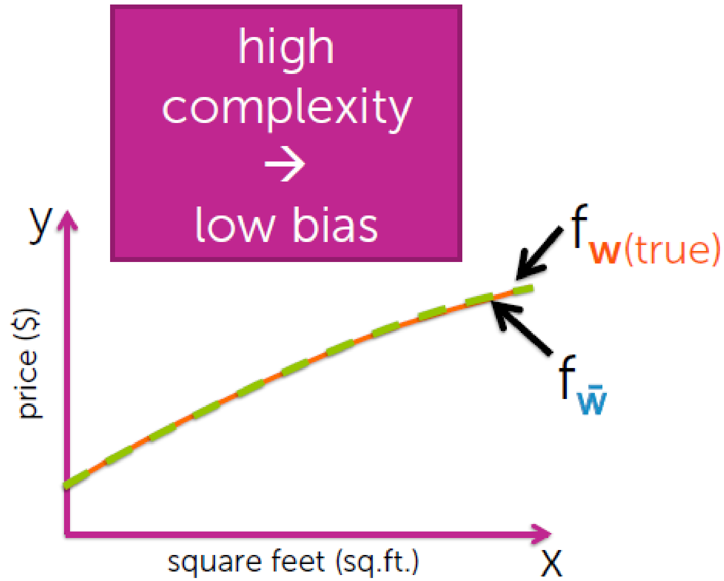


Bias of high complexity models

Assume we fit a high-order polynomial

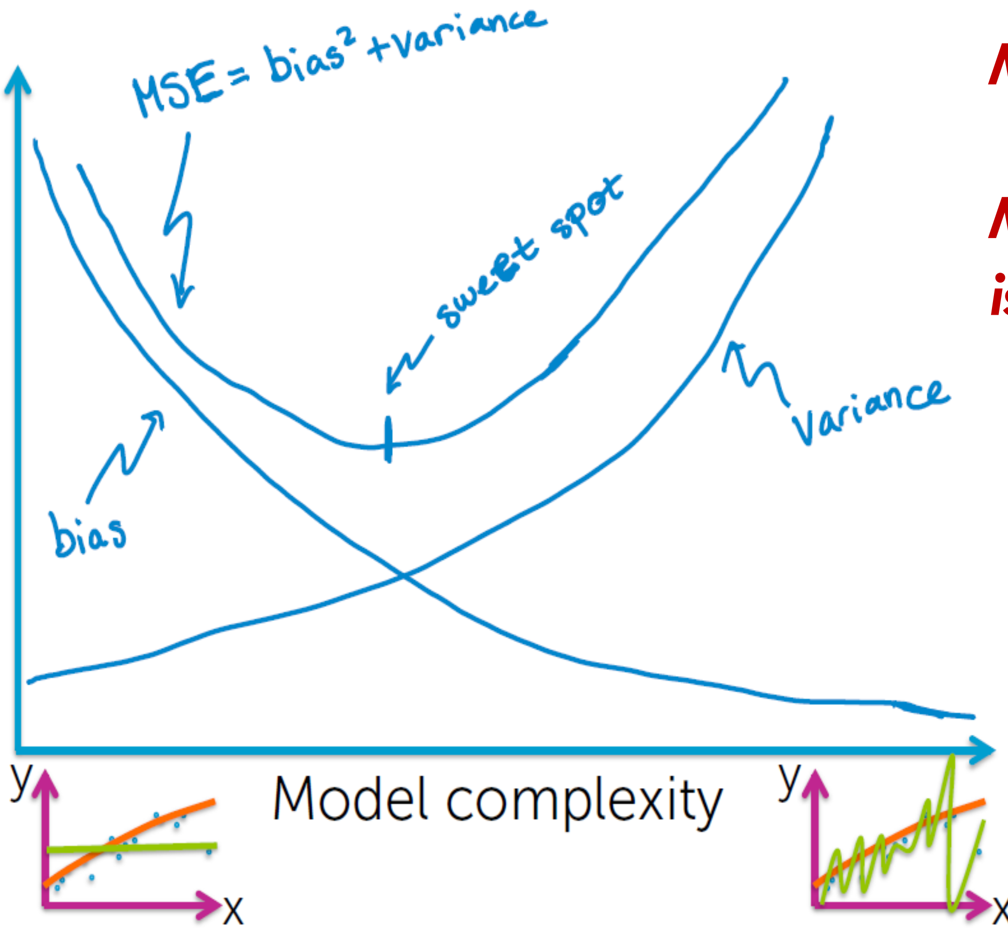
For each train remove few random houses





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

Bias –variance tradeoff



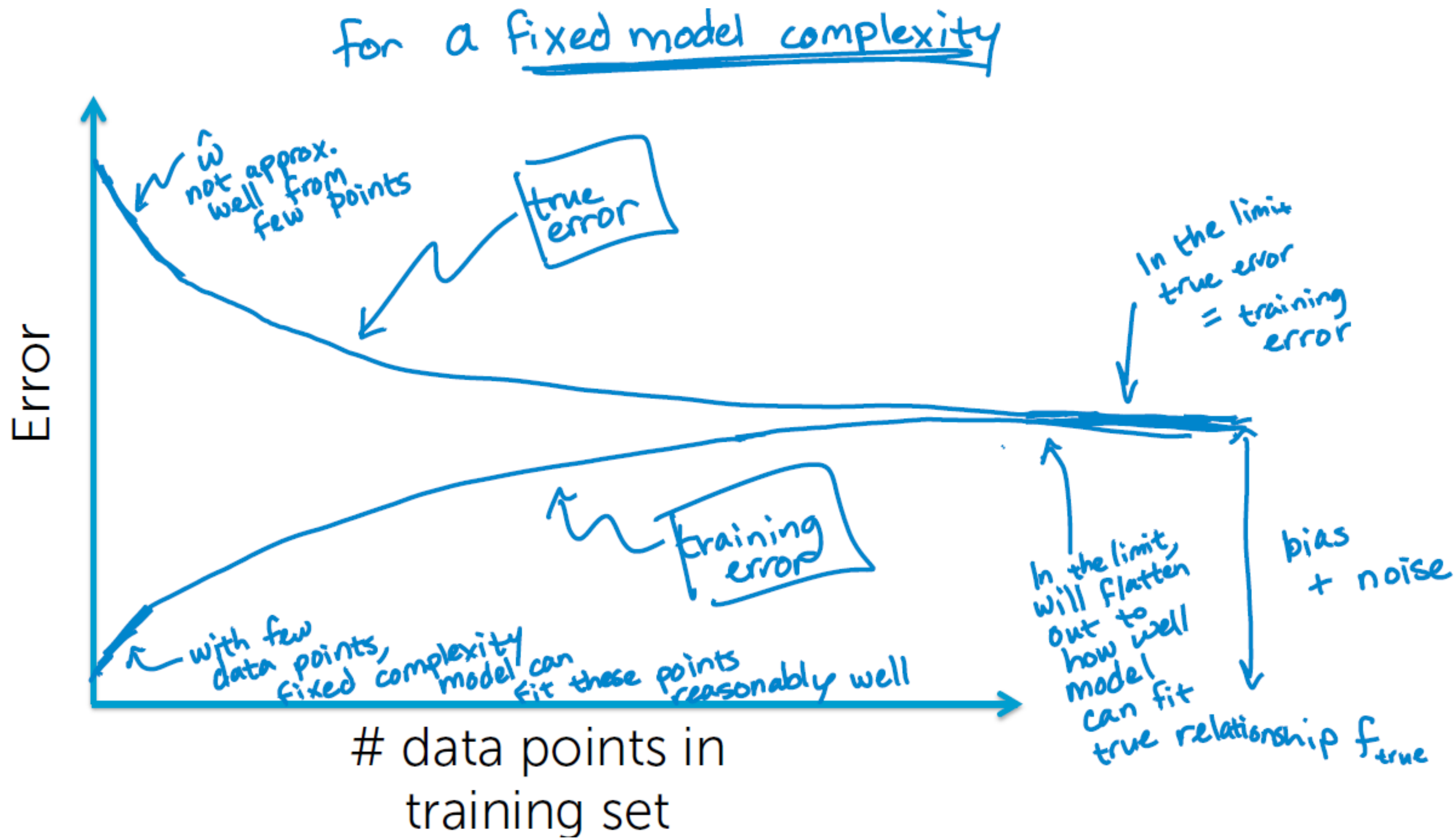
MSE = mean square error

Machine Learing
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**

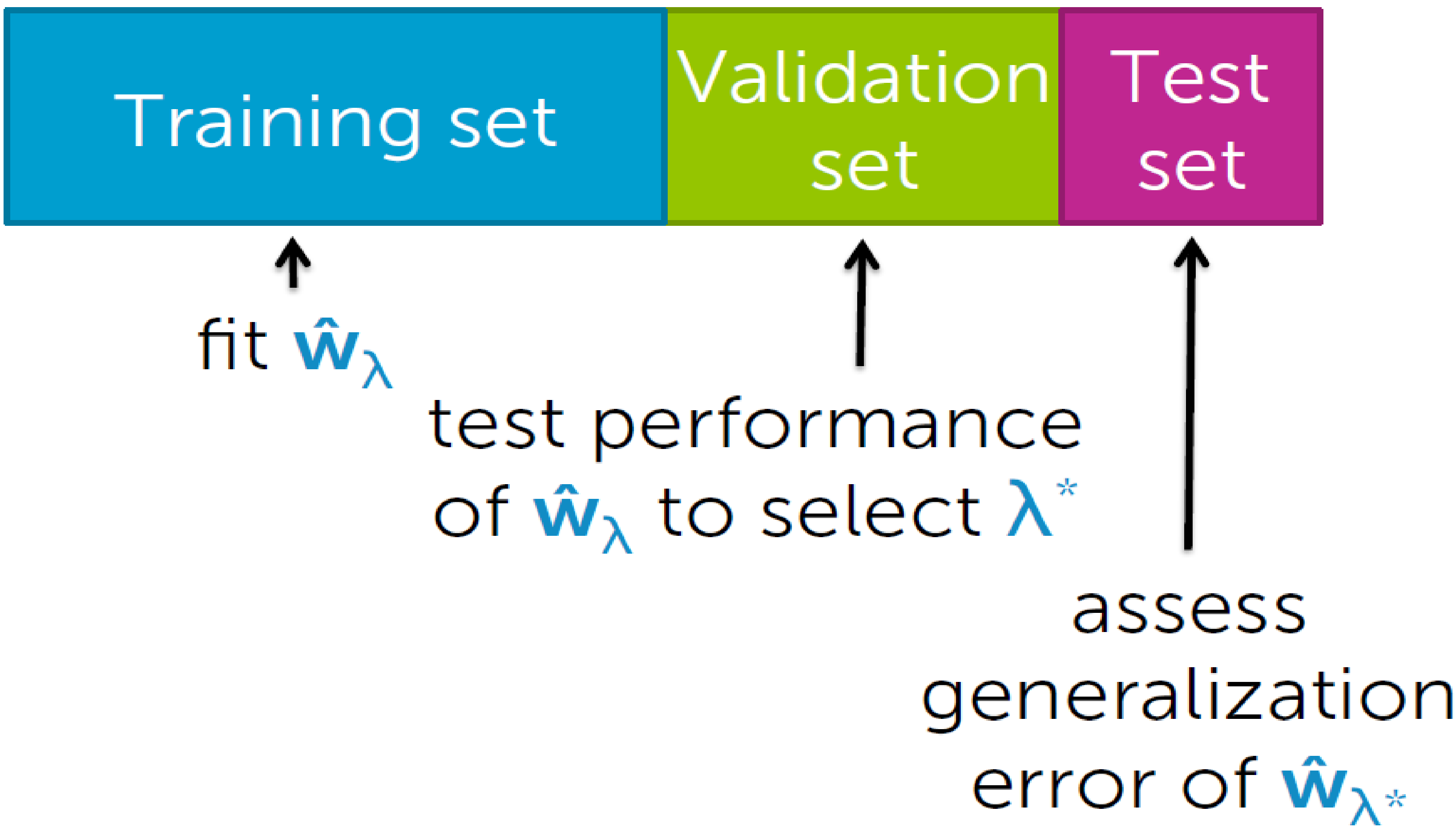
iii. Choose λ^* to be λ with lowest test error

Overly optimistic!

2. **Model assessment**

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

Practical implementation

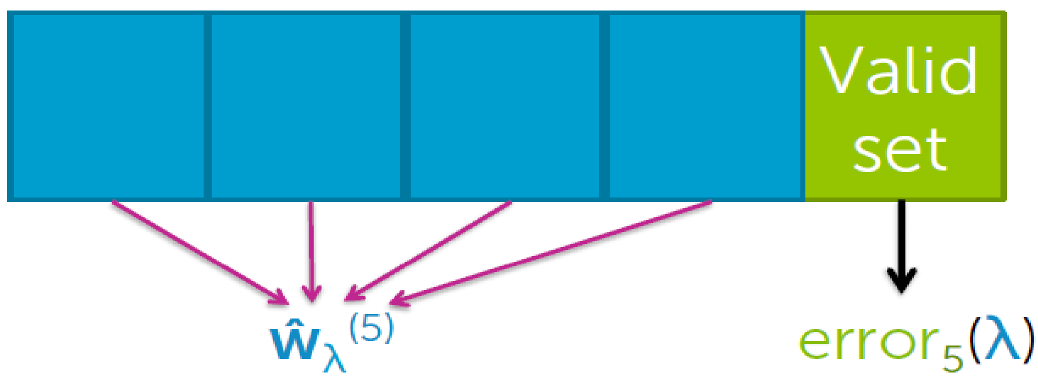


Typical splits

Training set	Validation set	Test set
80%	10%	10%
50%	25%	25%

K-fold cross validation

K-fold cross validation



For k=1,...,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: $CV(\lambda) = \frac{1}{K}\sum_{k=1}^{K} \text{error}_k(\lambda)$

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

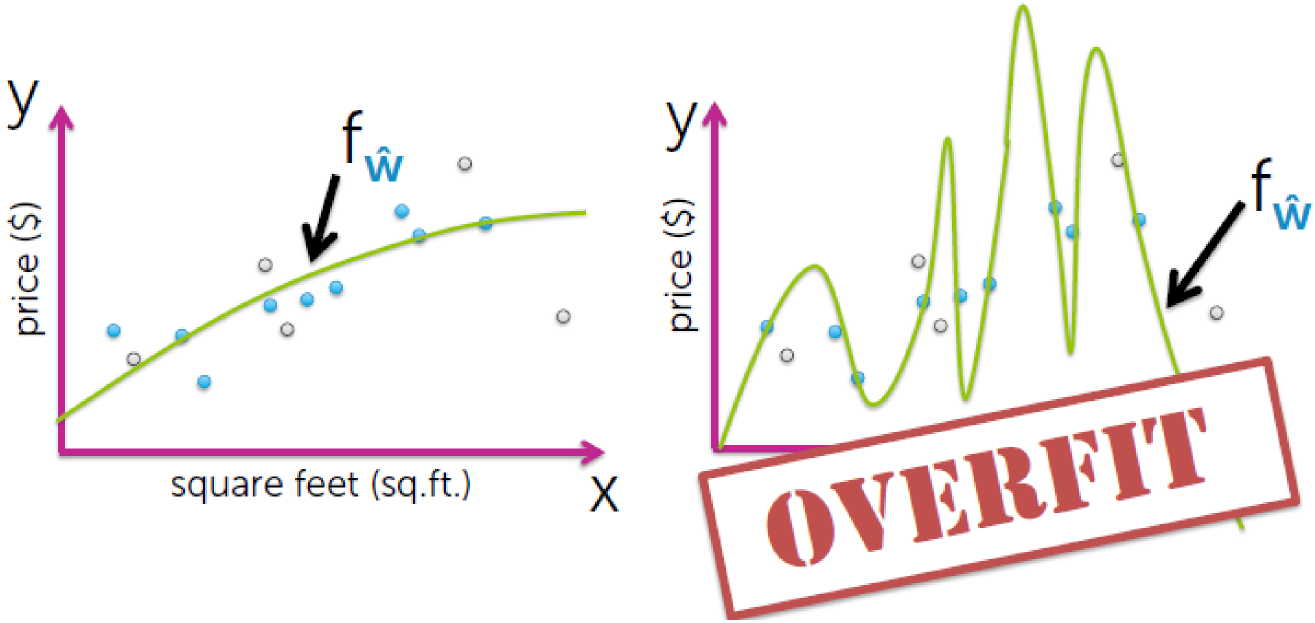
5-fold CV

10-fold CV

RIDGE REGRESSION

Flexibility of high-order polynomials

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

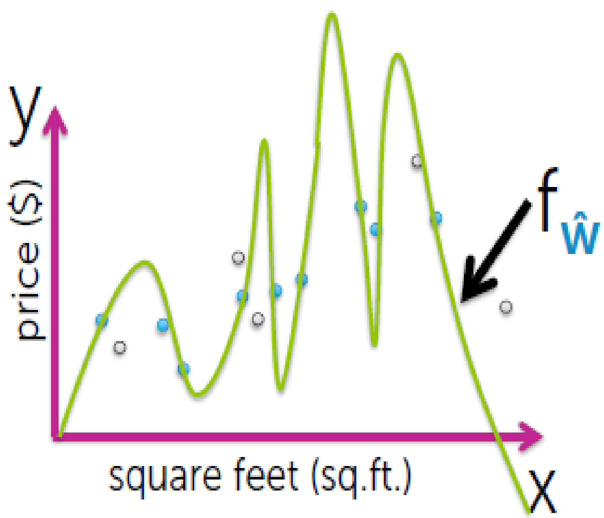


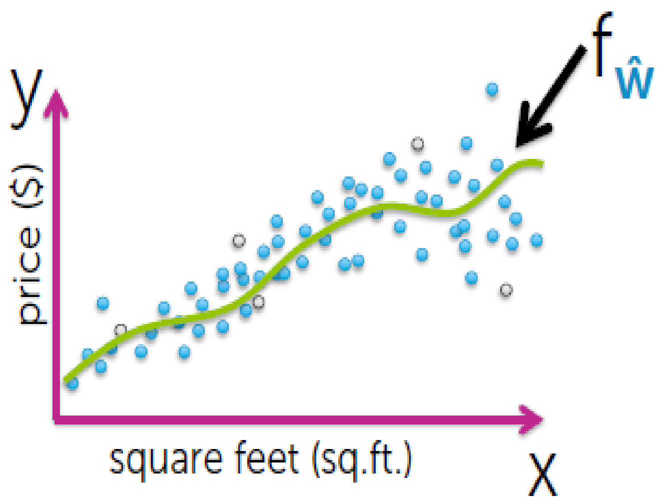
Symptoms for overfitting: often associated with very large value of estimated parameters $\hat{w}$

How does # of observations influence overfitting?

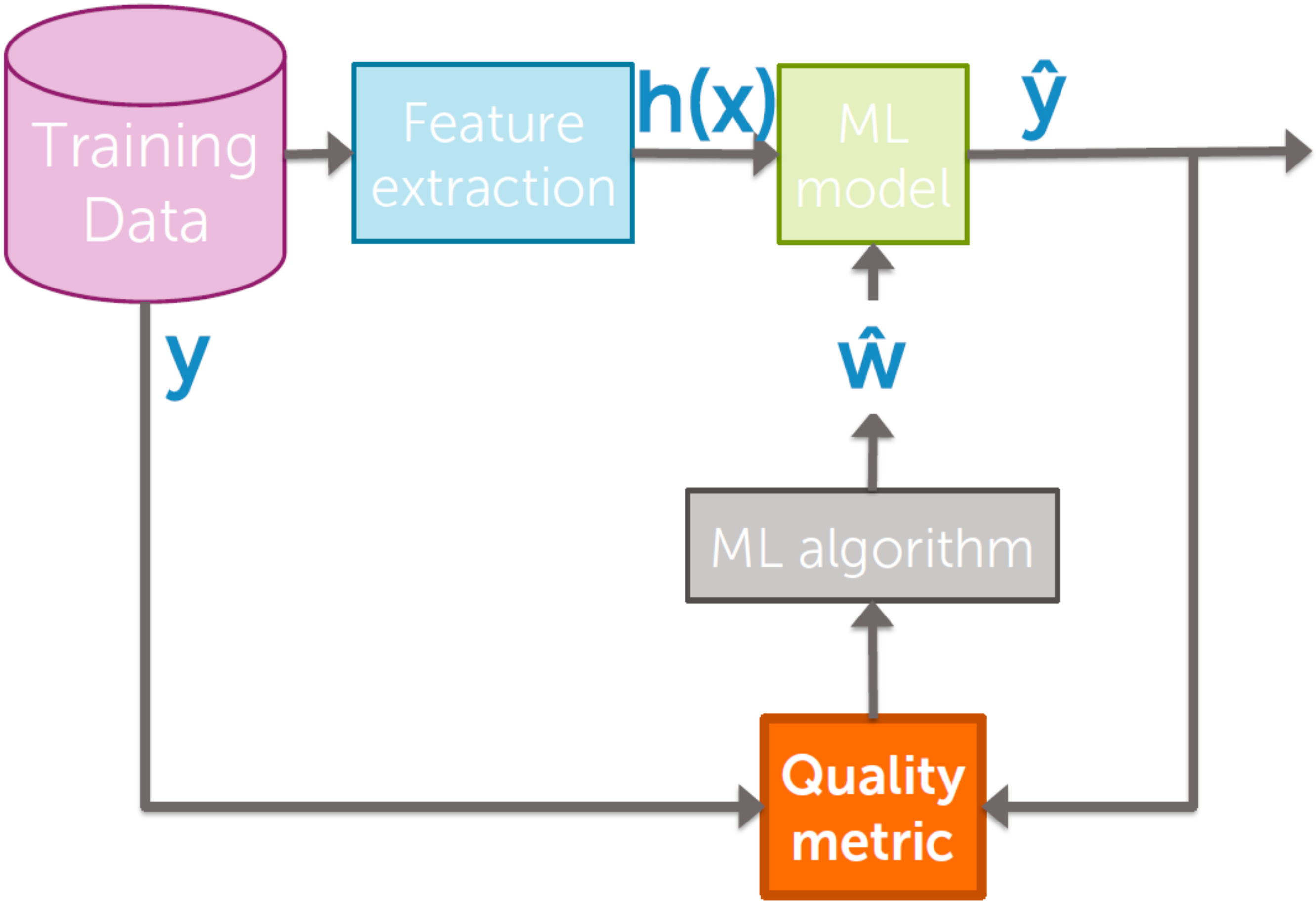
Few observations (N small)
→ rapidly overfit as model complexity increases

Many observations (N very large)
→ harder to overfit





Lets improve quality metric blok



Desire total cost format

Want to balance:

i. How well function fits data

ii. Magnitude of coefficients

want to balance

Total cost =

measure of fit + measure of magnitude of coefficients

small # = good fit to training data

small # = not overfit

Measure of magnitude of regression coefficients

What summary # is indicative of size of regression coefficients?

- Sum? $w_0 = 1{,}527{,}301 \quad w_1 = -1{,}605{,}253$

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

  But … the coefficients are very large

- Sum of absolute value?

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

- Sum of squares (L_2 norm)

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

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

Ridge regression
(a.k.a L_2 regularization)

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

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

tuning parameter = balance of fit and magnitude

If λ=0:

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

If λ=∞:

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

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

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

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

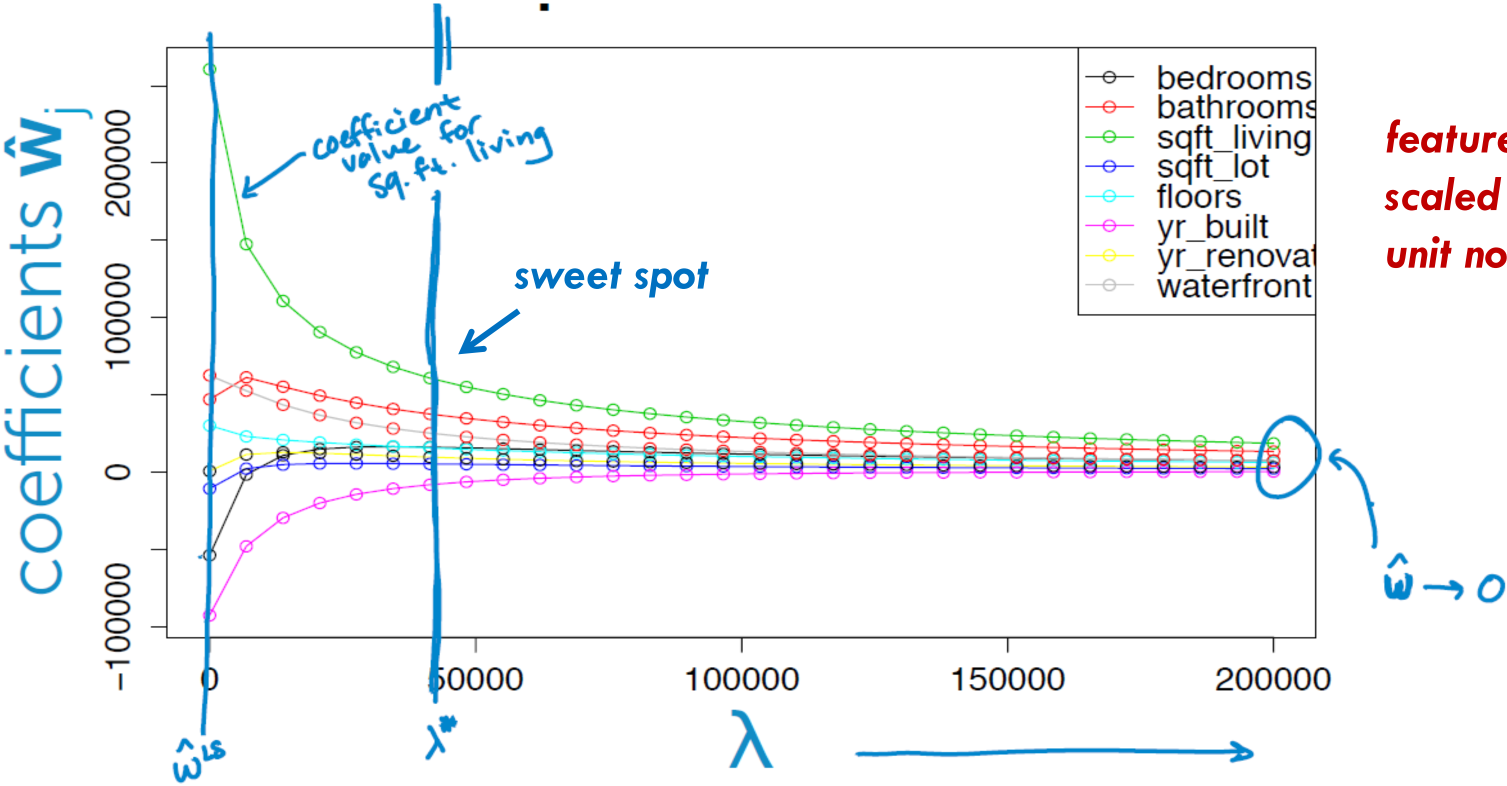
Small λ:

low bias, high variance

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

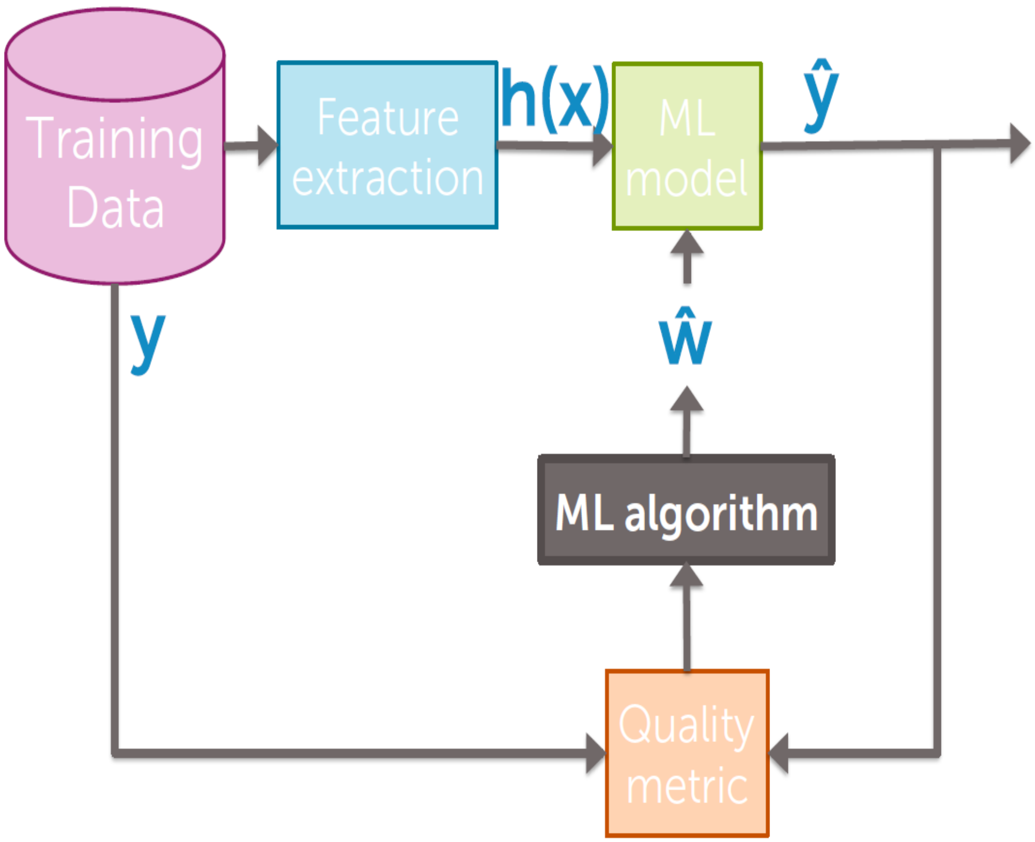
Ridge regression: coefficients path

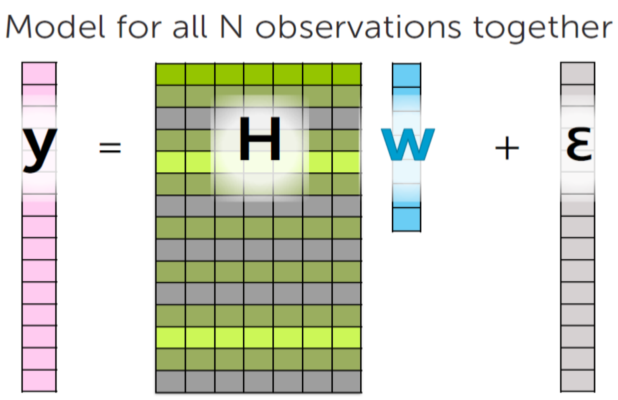
What happens if we refit our high-order polynomial, but now using **ridge regression**?



features scaled to unit norm

Flow chart


Training Data
Feature extraction
h(x)
ML model
ŷ
y
ŵ
ML algorithm
Quality metric


Model for all N observations together
y
=
H
w
+
ε

Ridge regression: cost in matrix notation

In matrix form, ridge regression cost is:

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

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

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

$$= \begin{bmatrix} w_0 & w_1 & w_2 & \cdots & w_D \end{bmatrix} \begin{bmatrix} w_0 \\ w_1 \\ w_2 \\ \vdots \\ w_D \end{bmatrix}$$

$$= w^T w$$

Gradient of ridge regression cost

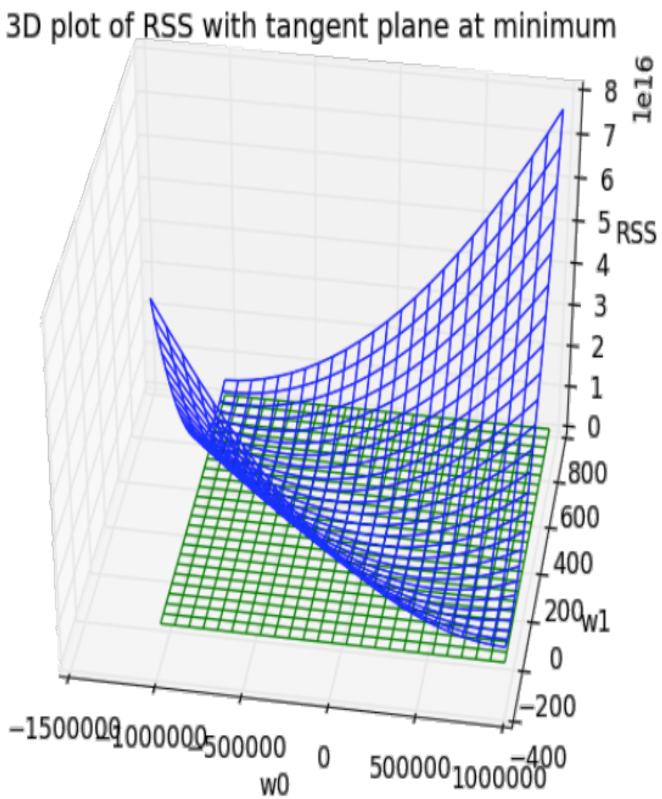
$$\nabla[\mathrm{RSS}(\mathbf{w}) + \lambda\|\mathbf{w}\|_2^2] = \nabla[(\mathbf{y}-\mathbf{H}\mathbf{w})^T(\mathbf{y}-\mathbf{H}\mathbf{w}) + \lambda\mathbf{w}^T\mathbf{w}]$$

$$= \underbrace{\nabla[(\mathbf{y}-\mathbf{H}\mathbf{w})^T(\mathbf{y}-\mathbf{H}\mathbf{w})]}_{-2\mathbf{H}^T(\mathbf{y}-\mathbf{H}\mathbf{w})} + \lambda \underbrace{\nabla[\mathbf{w}^T\mathbf{w}]}_{2\mathbf{w}}$$

Why? By analogy to 1d case...

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

Ridge regression: closed-form solution



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

Solve for **w**:

$$-H^T y + H^T H \hat{w} + \lambda I \hat{w} = 0$$

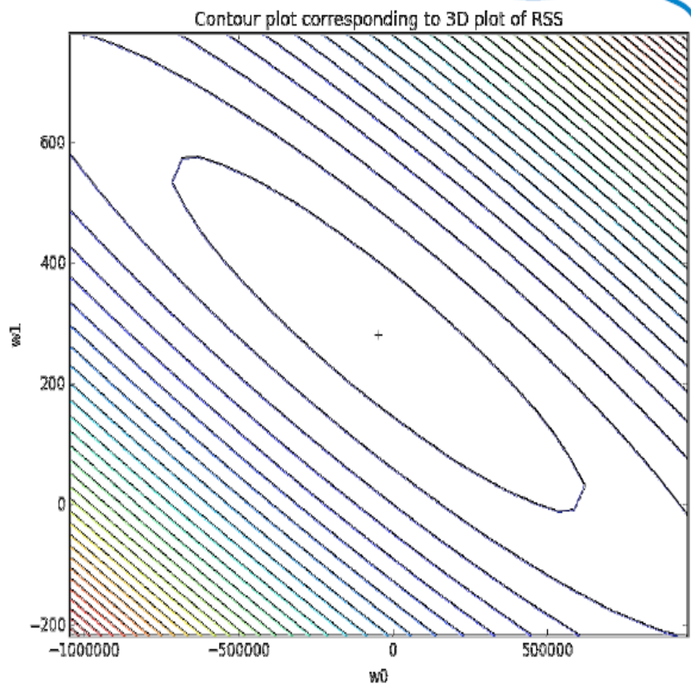
$$H^T H \hat{w} + \lambda I \hat{w} = H^T y$$

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

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

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 jth feature weight:

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

Same as before (from RSS term)

new term, comes from the j^{th} component of $2\lambda w$

Summary of ridge regression algorithm

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

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

for $j=0,\ldots,D$

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

$w_j^{(t+1)} \leftarrow (1-2\eta\lambda)w_j^{(t)} - \eta$ partial[j]

$t \leftarrow t + 1$

How to handle the intercept

Recall multiple regression model

Model:

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

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

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:

  $$RSS(w_{0,}\mathbf{w}_{rest}) + \lambda||\mathbf{w}_{rest}||_2^2$$

- **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)

FEATURES SELECTION

&

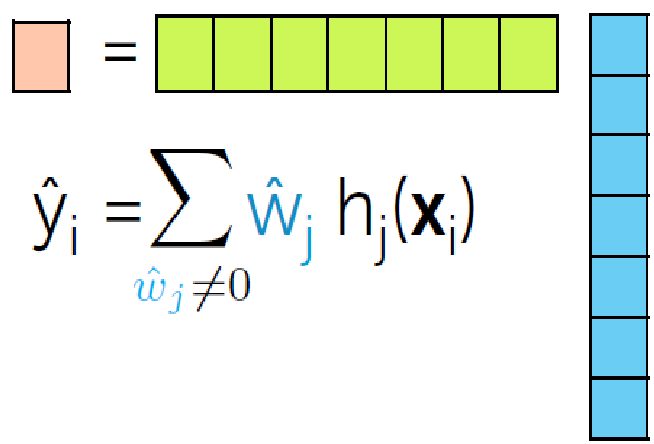
LASSO REGRESSION

Why features selection?

Efficiency:

- If size(**w**) = 100B, each prediction is expensive
- If $\hat{\mathbf{w}}$ $\boxed{\text{sparse}}$, 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?

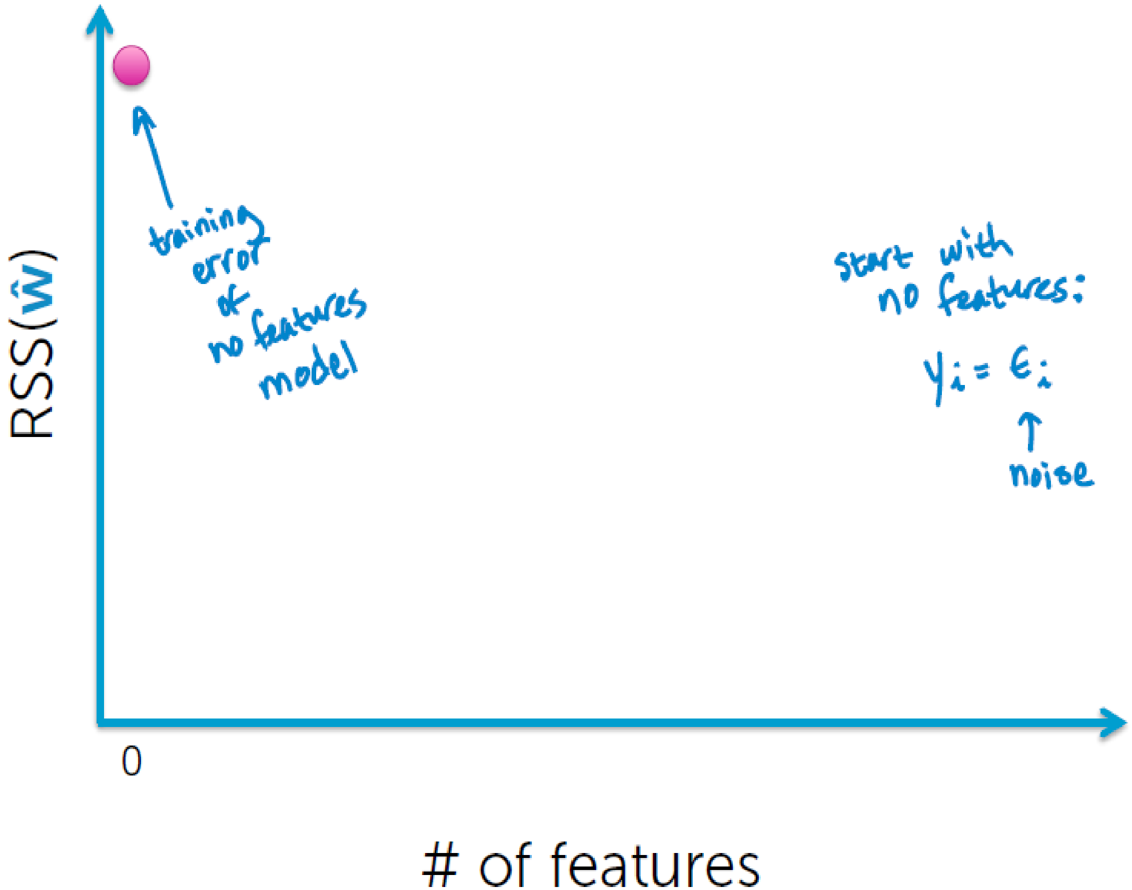
Sparcity

Housing application

Lot size
Single Family
Year built
Last sold price
Last sale price/sqft
Finished sqft
Unfinished sqft
Finished basement sqft
floors
Flooring types
Parking type
Parking amount
Cooling
Heating
Exterior materials
Roof type
Structure style

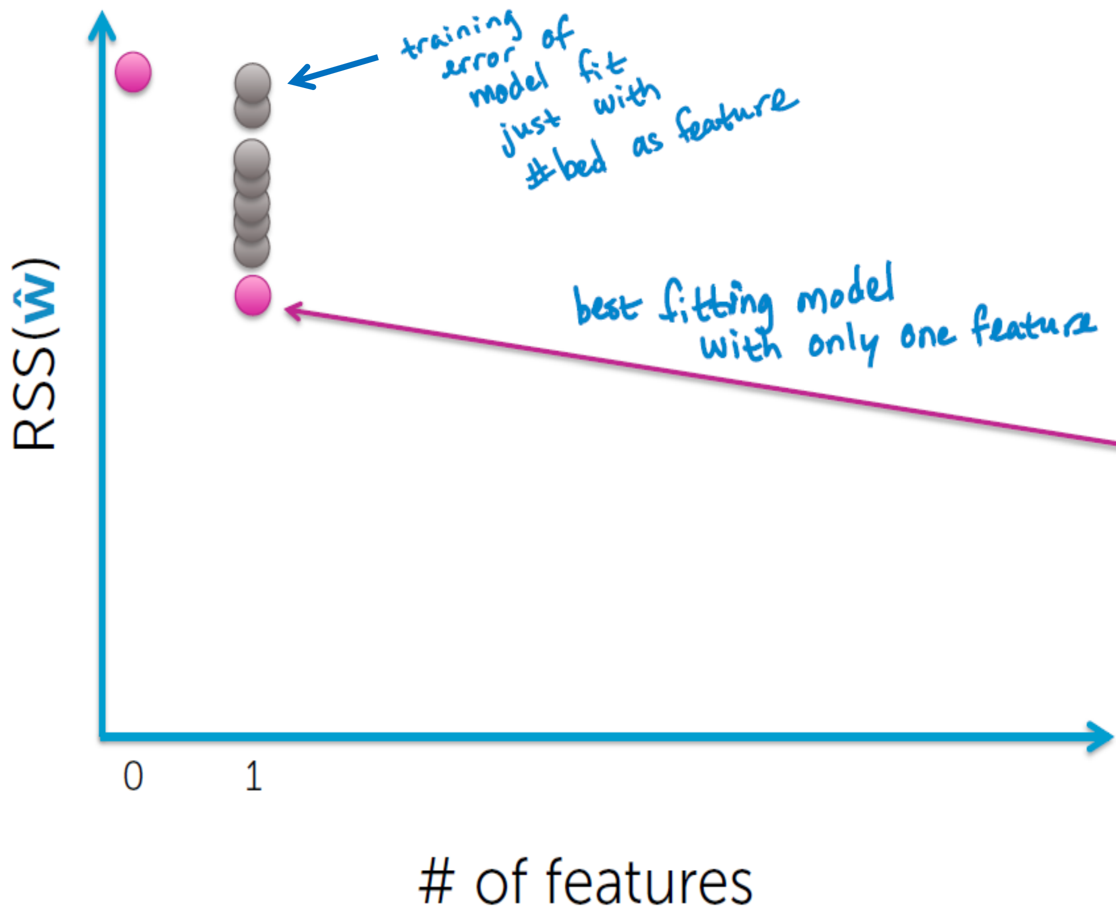
Dishwasher
Garbage disposal
Microwave
Range / Oven
Refrigerator
Washer
Dryer
Laundry location
Heating type
Jetted Tub
Deck
Fenced Yard
Lawn
Garden
Sprinkler System
⋮

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



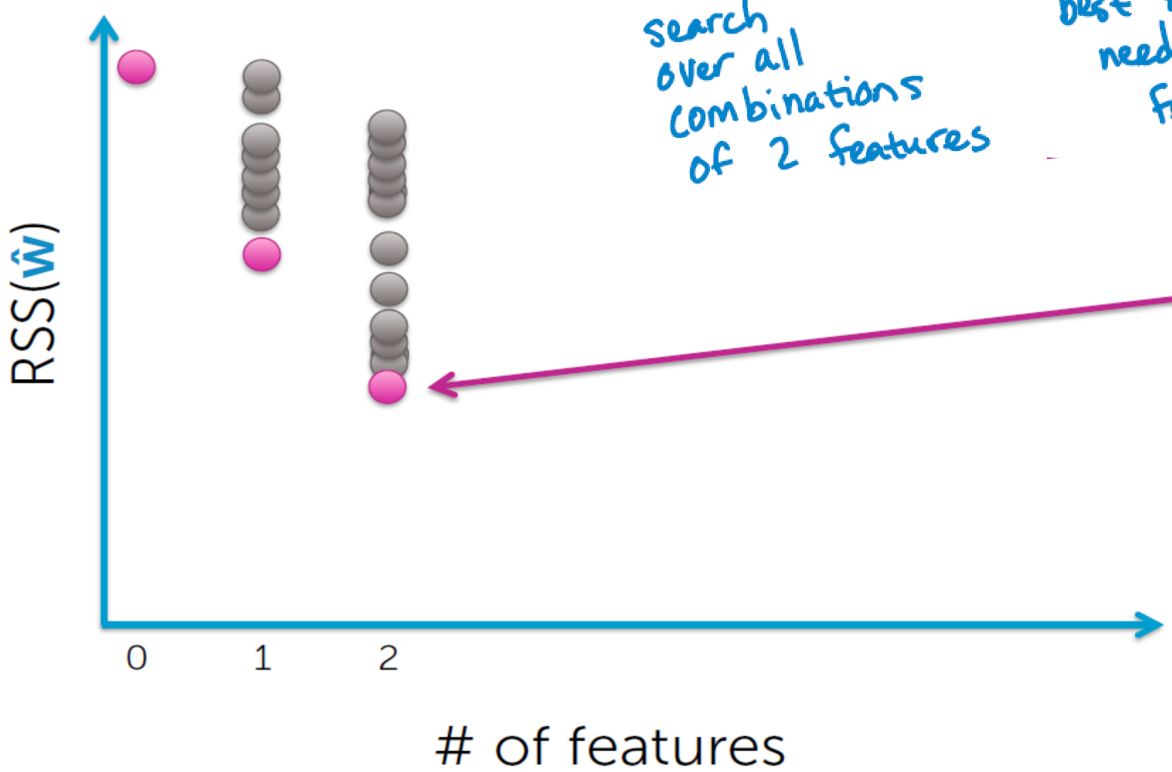
- # bedrooms
- # bathrooms
- sq.ft. living
- sq.ft. lot
- floors
- year built
- year renovated
- waterfront

Find best model of size: 2

Note: not necessarily nested!

search over all combinations of 2 features

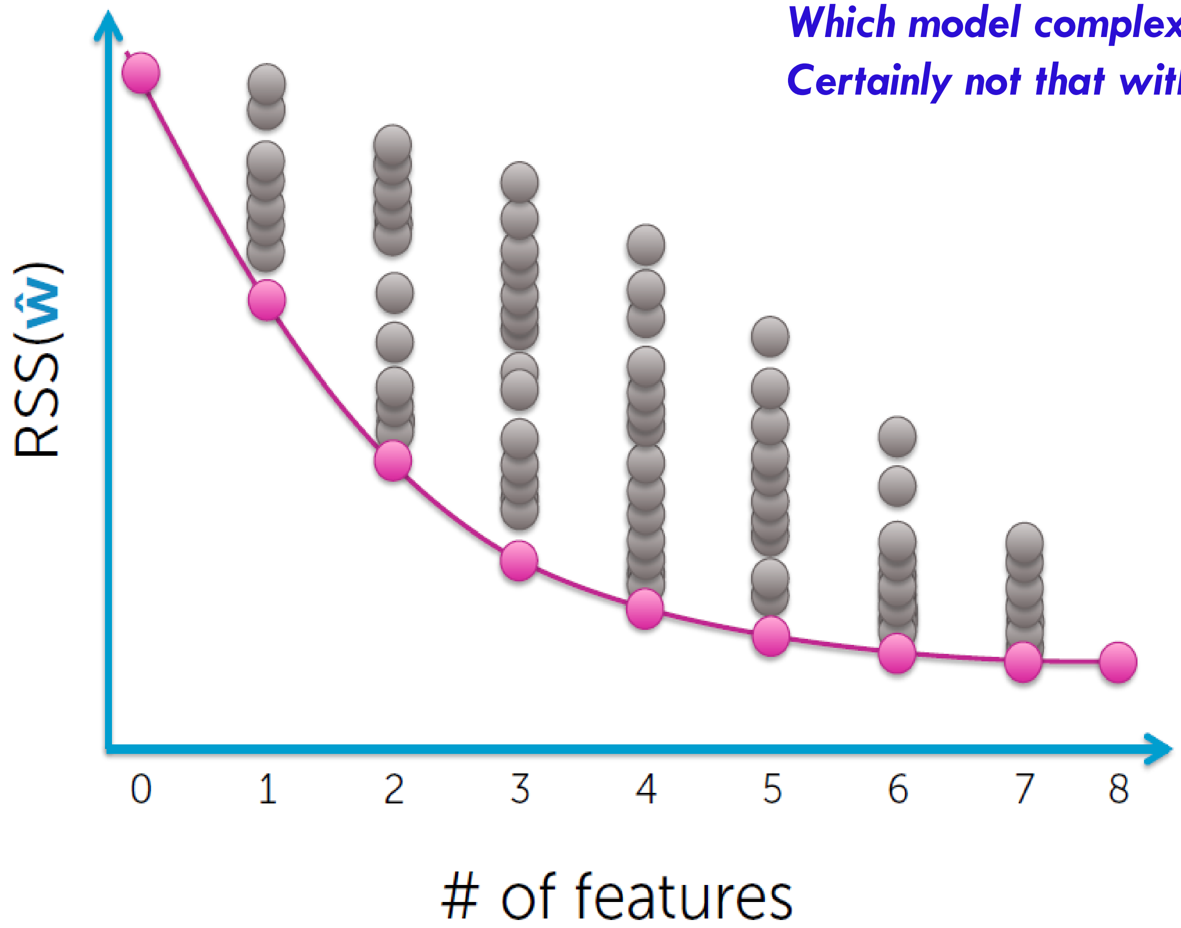
best model of size k need not contain features of best model of size k-1



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

Find best model of size: N

Which model complexity to choose?
Certainly not that with the smalest 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

feature 0 ← 0 if "no", 1 if "yes"
feature 1 … feature D

$y_i = \varepsilon_i$	[0 0 0 … 0 0 0]
$y_i = w_0 h_0(\mathbf{x}_i) + \varepsilon_i$	[1 0 0 … 0 0 0]
$y_i = w_1 h_1(\mathbf{x}_i) + \varepsilon_i$	[0 1 0 … 0 0 0]
$\vdots$	$\vdots$
$y_i = w_0 h_0(\mathbf{x}_i) + w_1 h_1(\mathbf{x}_i) + \varepsilon_i$	[1 1 0 … 0 0 0]
$\vdots$	$\vdots$
$y_i = w_0 h_0(\mathbf{x}_i) + w_1 h_1(\mathbf{x}_i) + \ldots + w_D h_D(\mathbf{x}_i) + \varepsilon_i$	[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} = HUGE!!!!!!

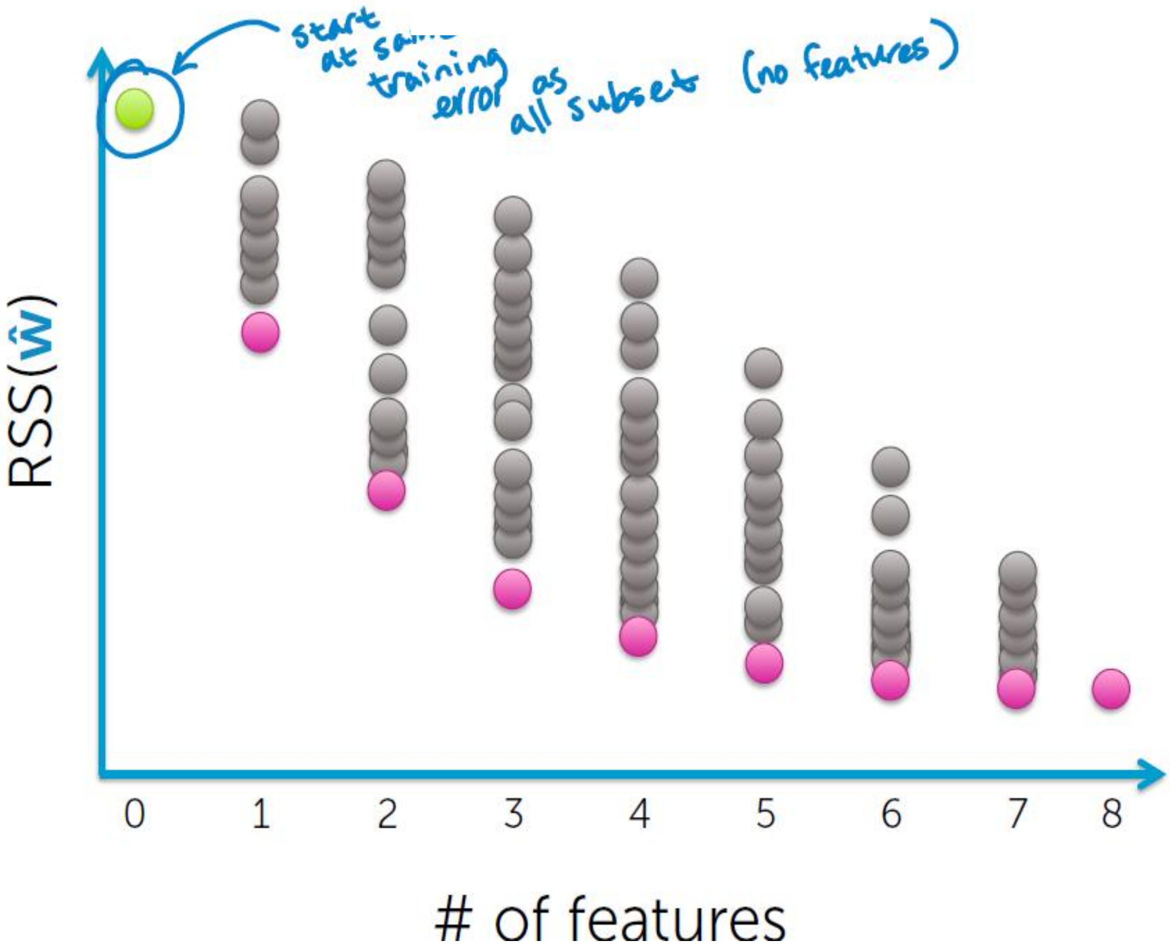
Typically, computationally infeasible

Greedy algorithm

Forward stepwise algorithm

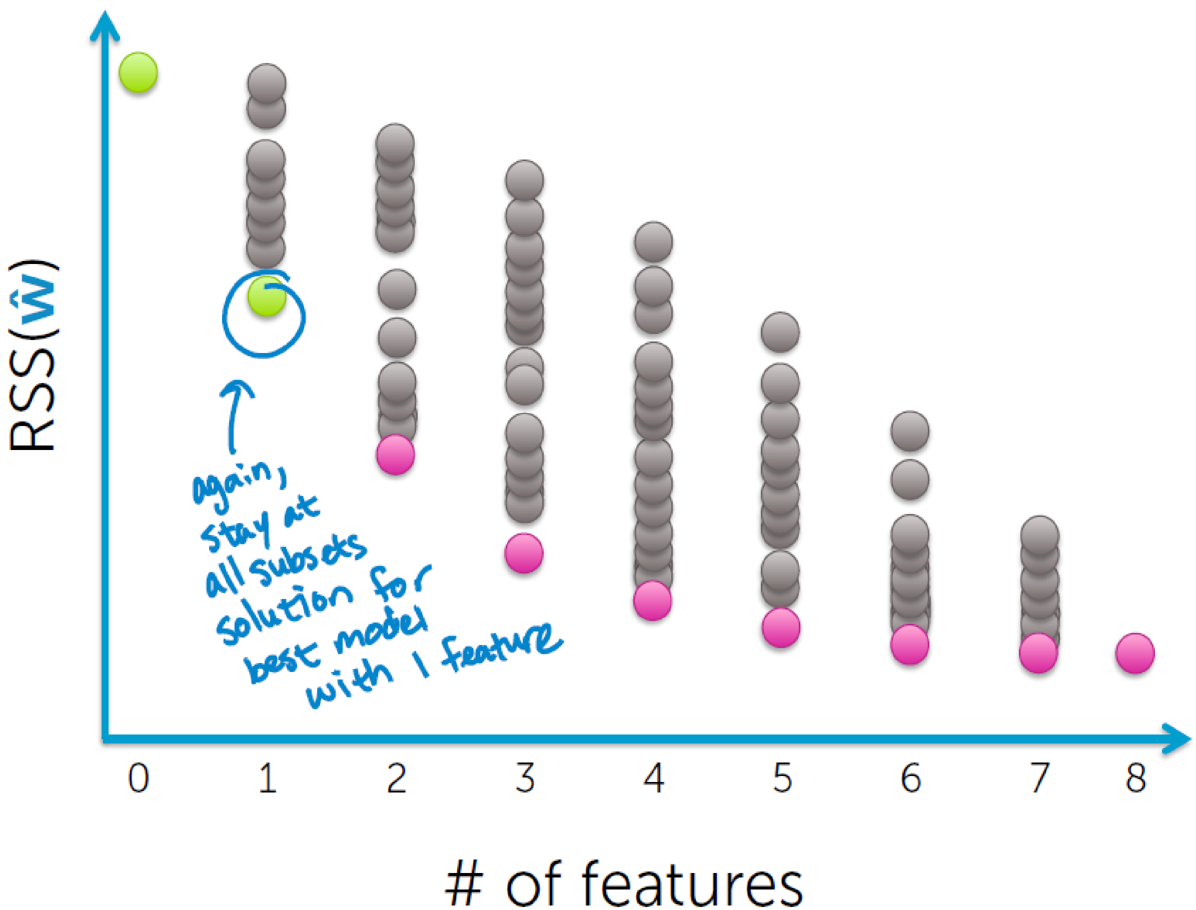
1. Pick a dictionary of features $\{h_0(\mathbf{x}),\ldots,h_D(\mathbf{x})\}$
   – e.g., polynomials for linear regression
2. Greedy heuristic:
   i. Start with empty set of features $F_0 = \varnothing$
   (or simple set, like just $h_0(\mathbf{x})=1 \rightarrow y_i = w_0+\varepsilon_i$)
   ii. Fit model 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



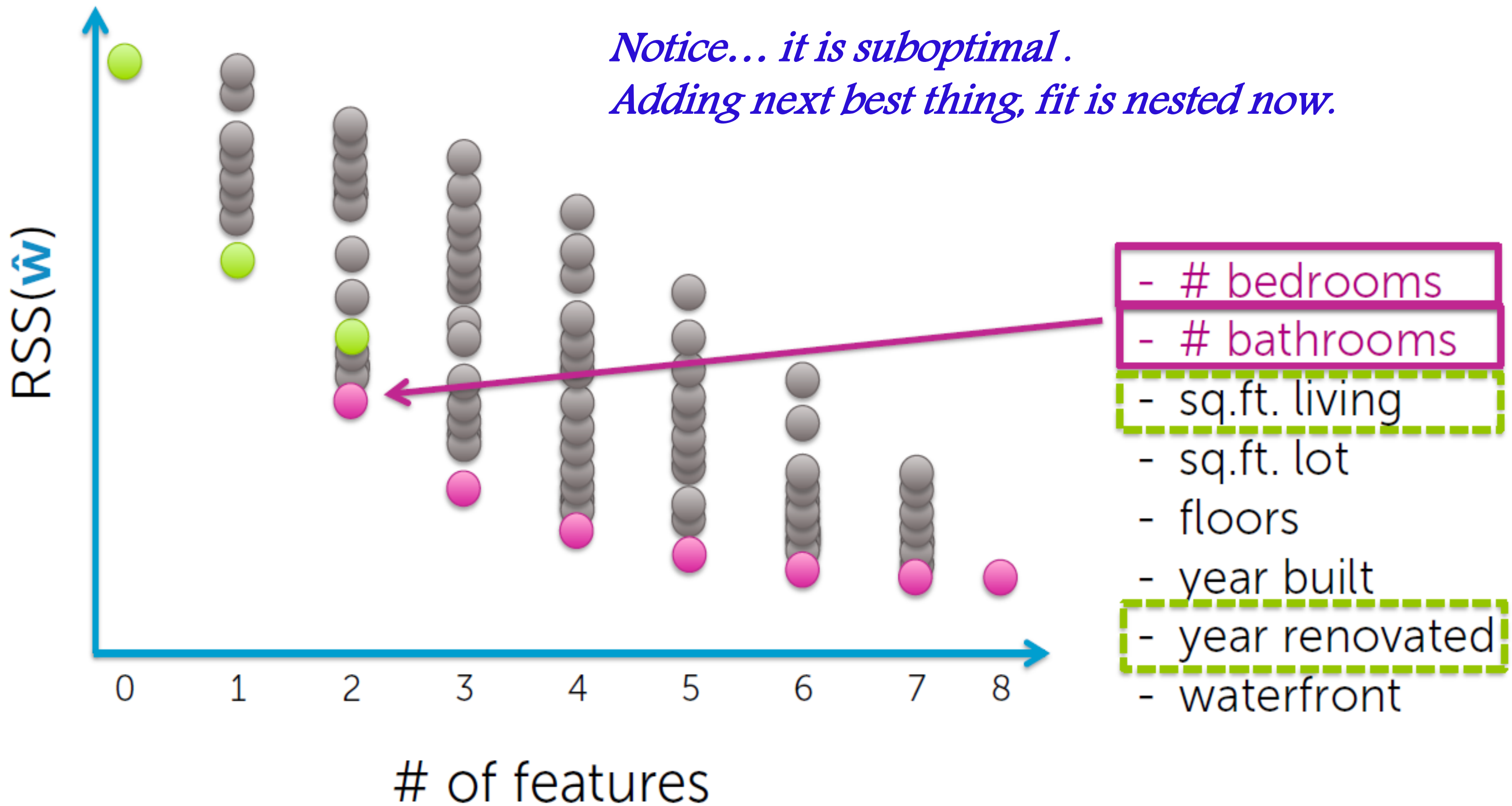
- # bedrooms
- # bathrooms
- sq.ft. living
- sq.ft. lot
- floors
- year built
- year renovated
- waterfront

Visualizing greedy algorithm



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

Visualizing greedy algorithm



Visualizing greedy algorithm

Complexity of forward stepwise

How many models were evaluated?

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

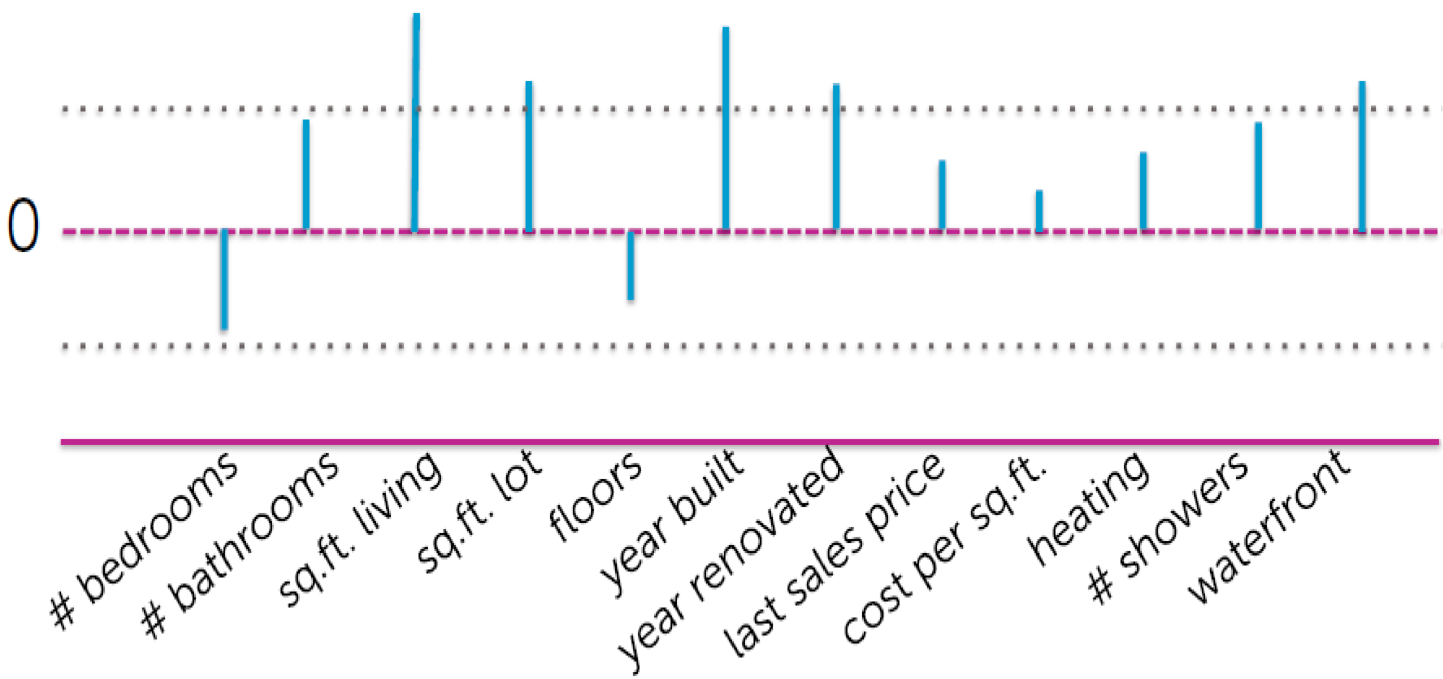
Using regularisation for features selection

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

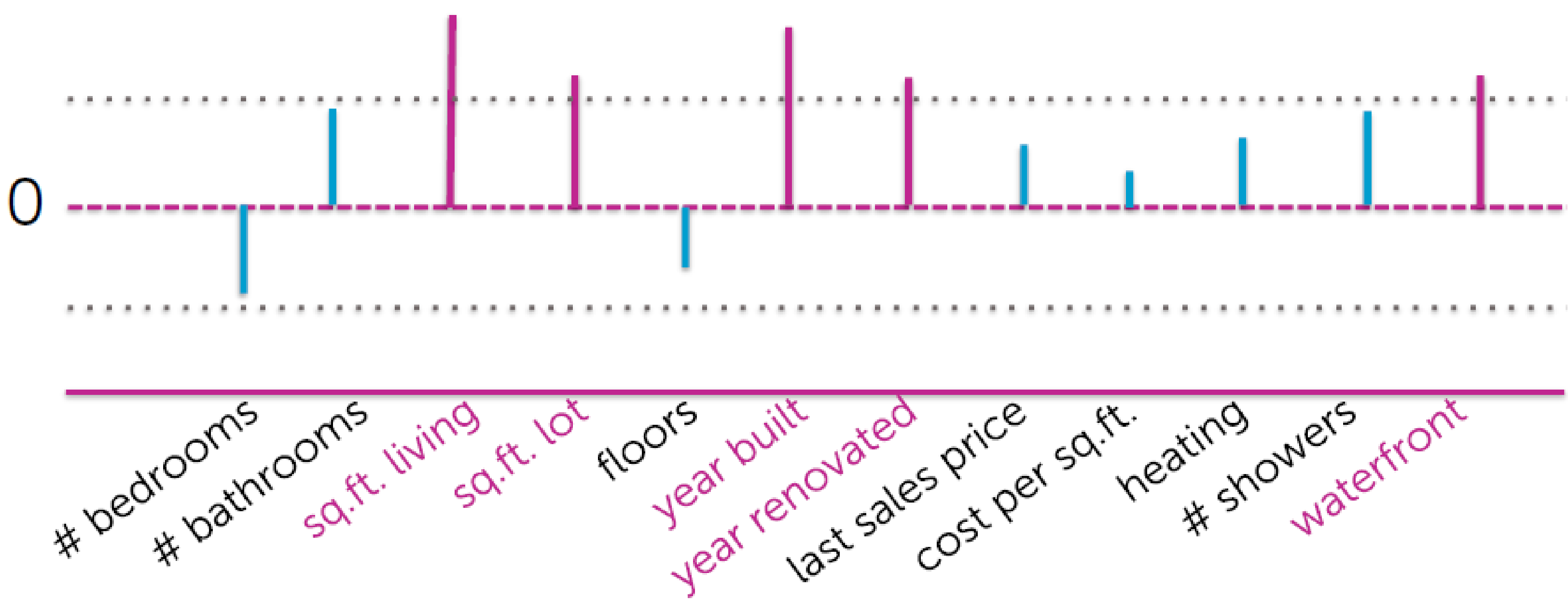
Thresholding ridge coefficients?

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



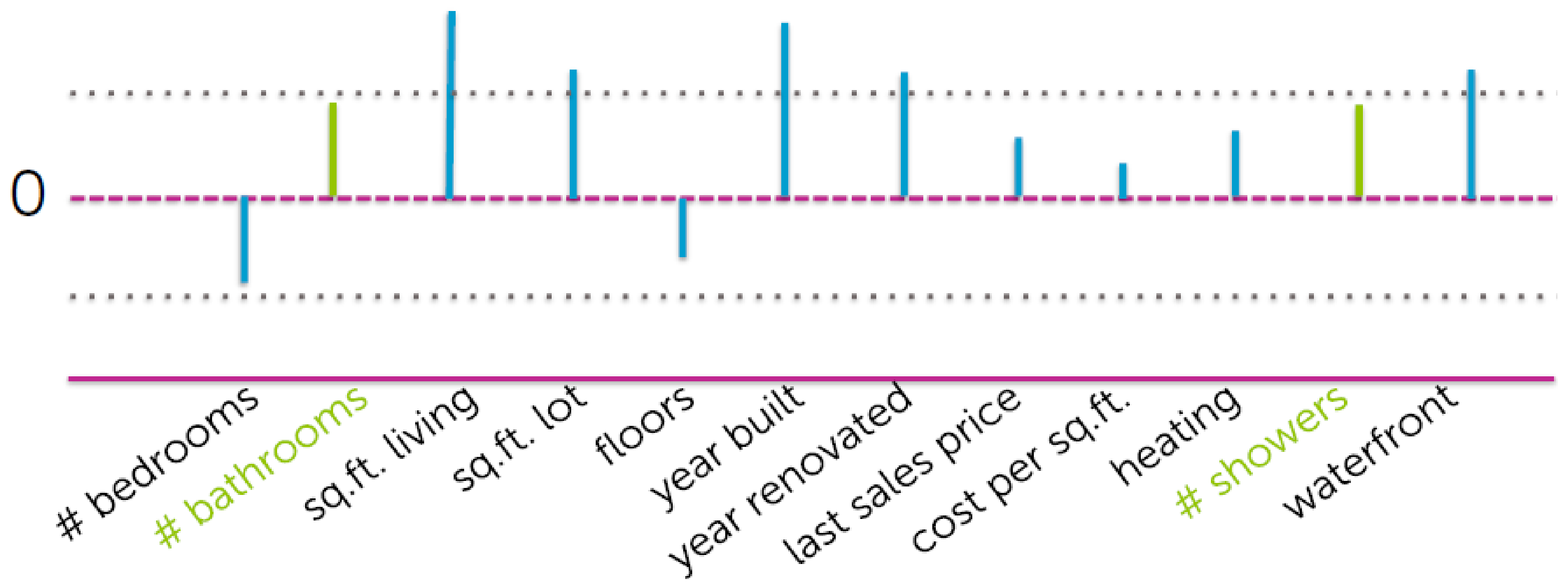
Thresholding ridge coefficients?

Selected features for a given threshold value



Thresholding ridge coefficients?

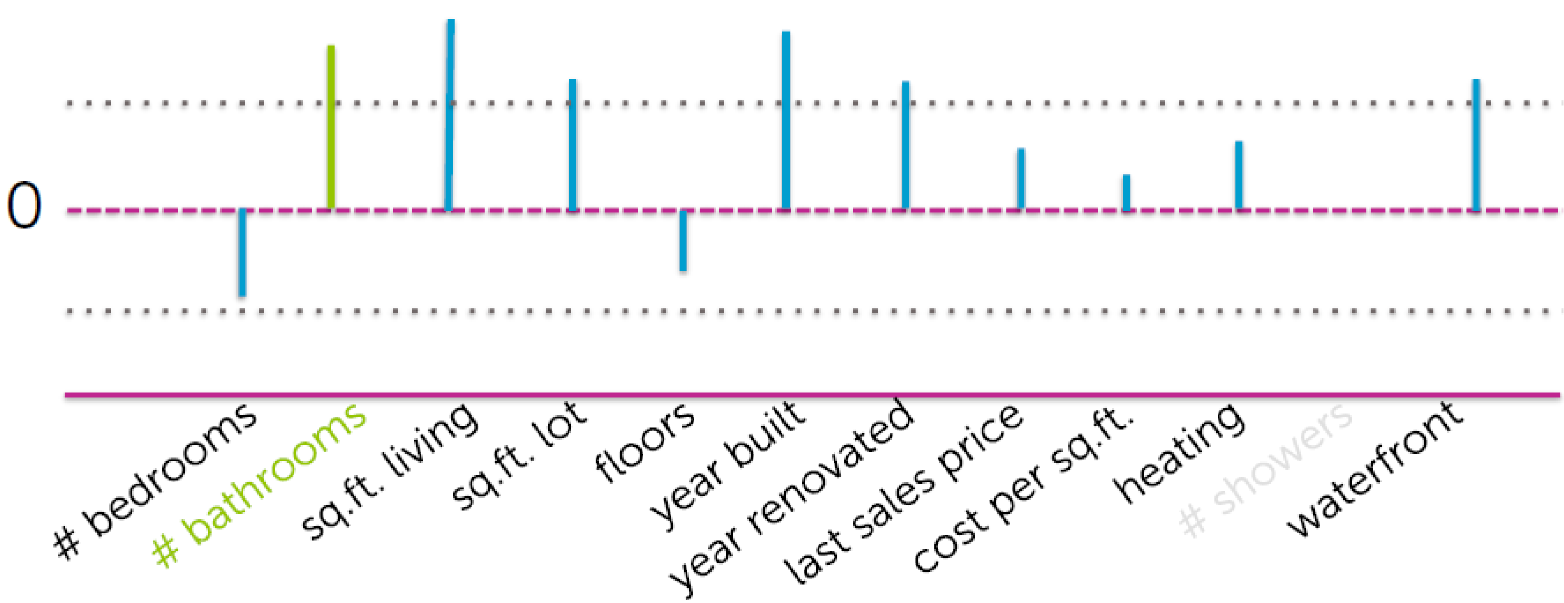
Let's look at two related features…



Nothing measuring bathrooms was included!

Thresholding ridge coefficients?

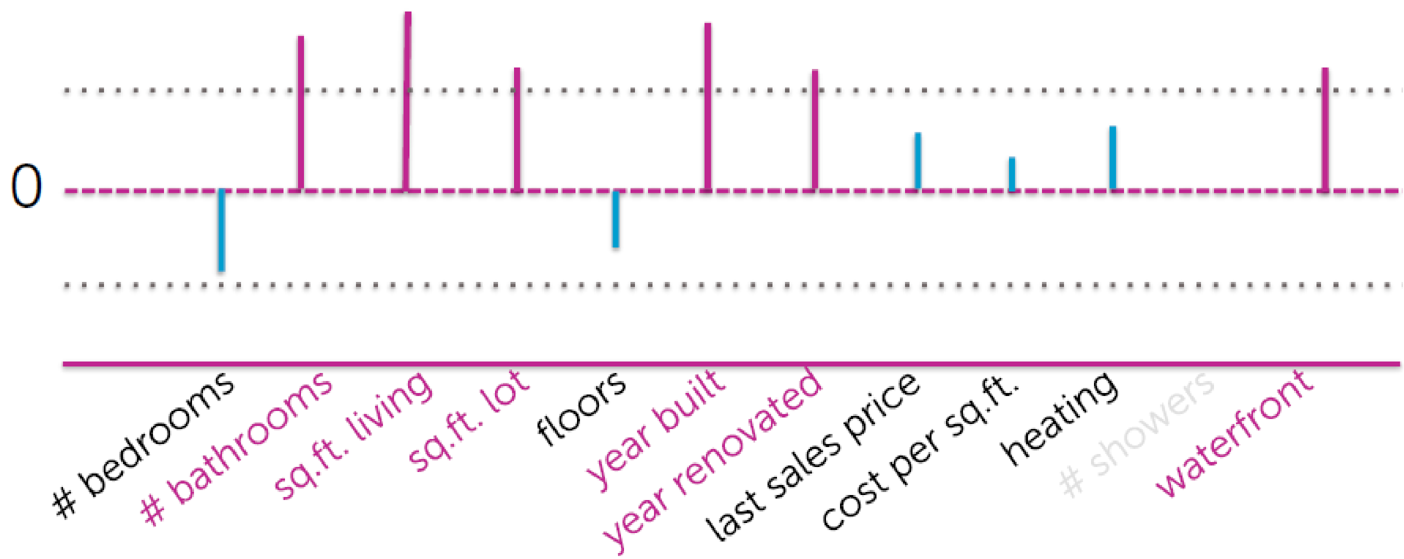
If only one of the features had been included…



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 =

$$\underbrace{\text{measure of fit}}_{\text{RSS}(\mathbf{w})} + \lambda \underbrace{\text{measure of magnitude of coefficients}}_{||\mathbf{w}||_1 = |w_0| + \ldots + |w_D|}$$

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

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

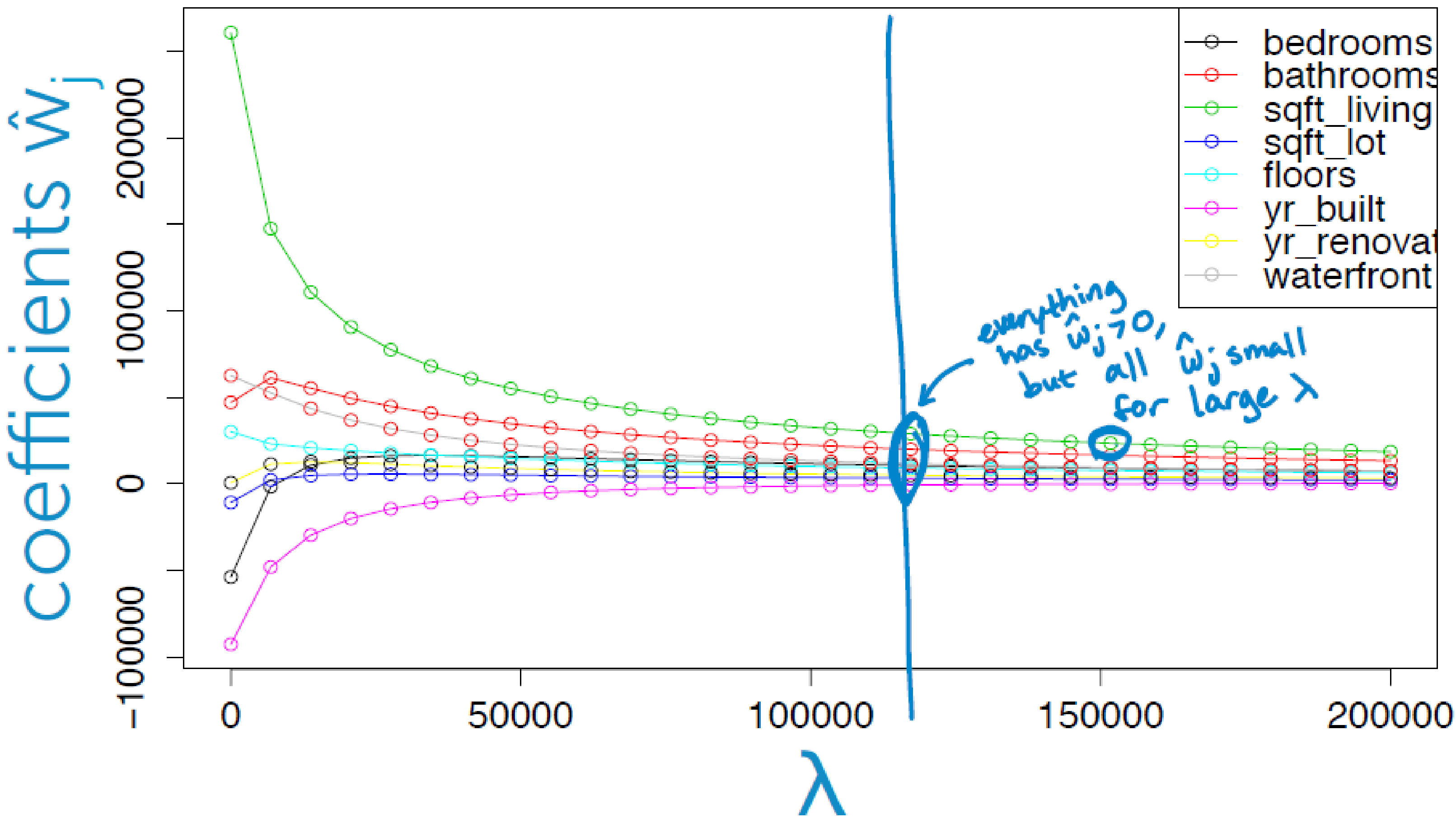
tuning parameter = balance of fit and sparsity

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

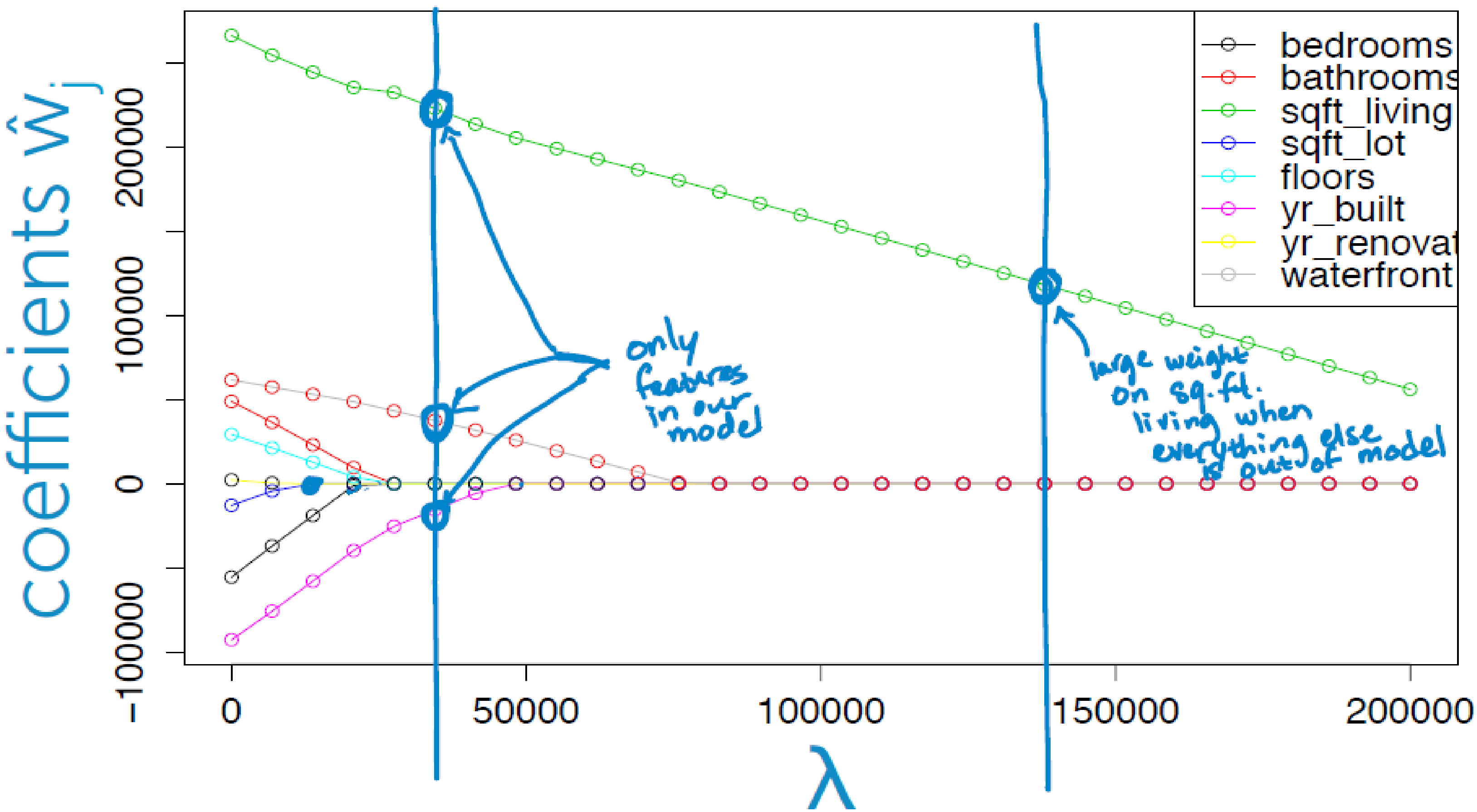
If $\lambda=\infty$: $\hat{w}^{lasso} = 0$

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

Coefficient path: ridge



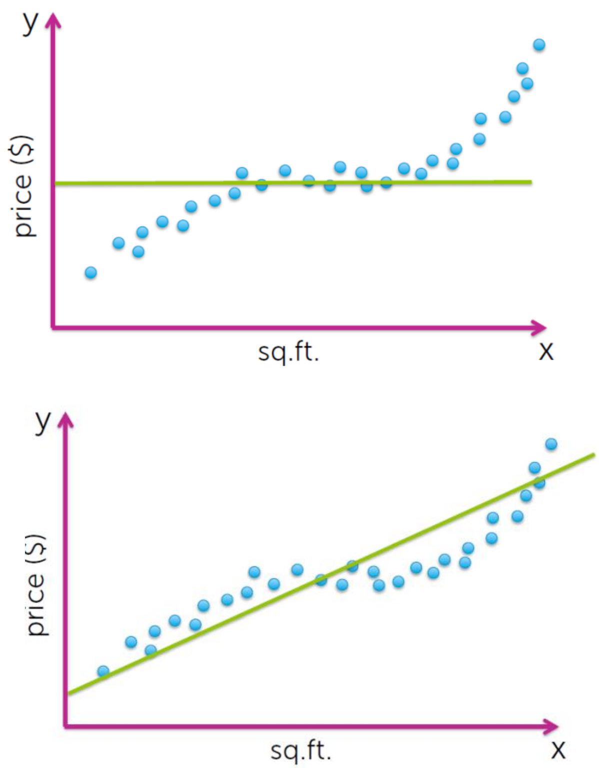
Coefficient path: lasso



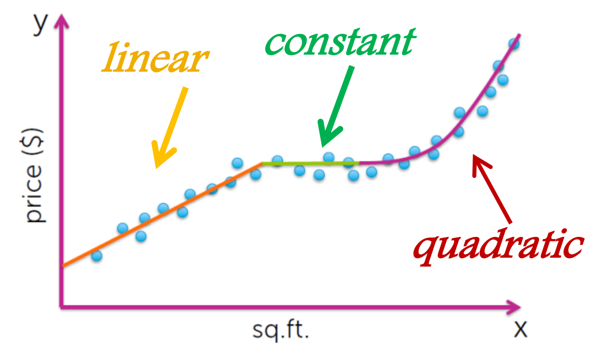
NONPARAMETRIC REGRESSION

Fit globaly vs fit locally

Parametric models



Below …
f(x) is not really
a polynomial function



What alternative do we have?

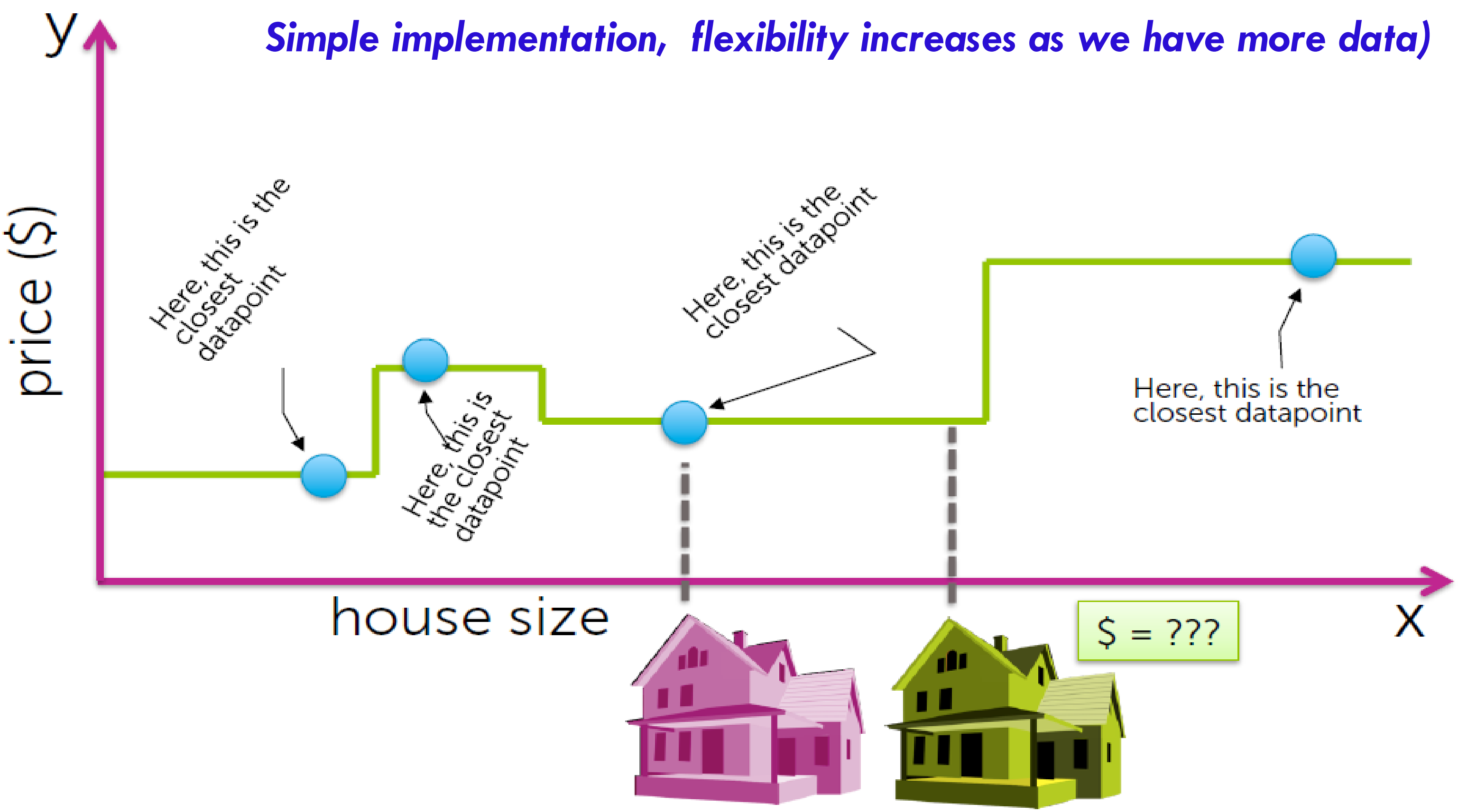
If we:

- Want to allow flexibility in f(**x**) having local structure
- Don't want to infer "structural breaks"

What's a simple option we have?

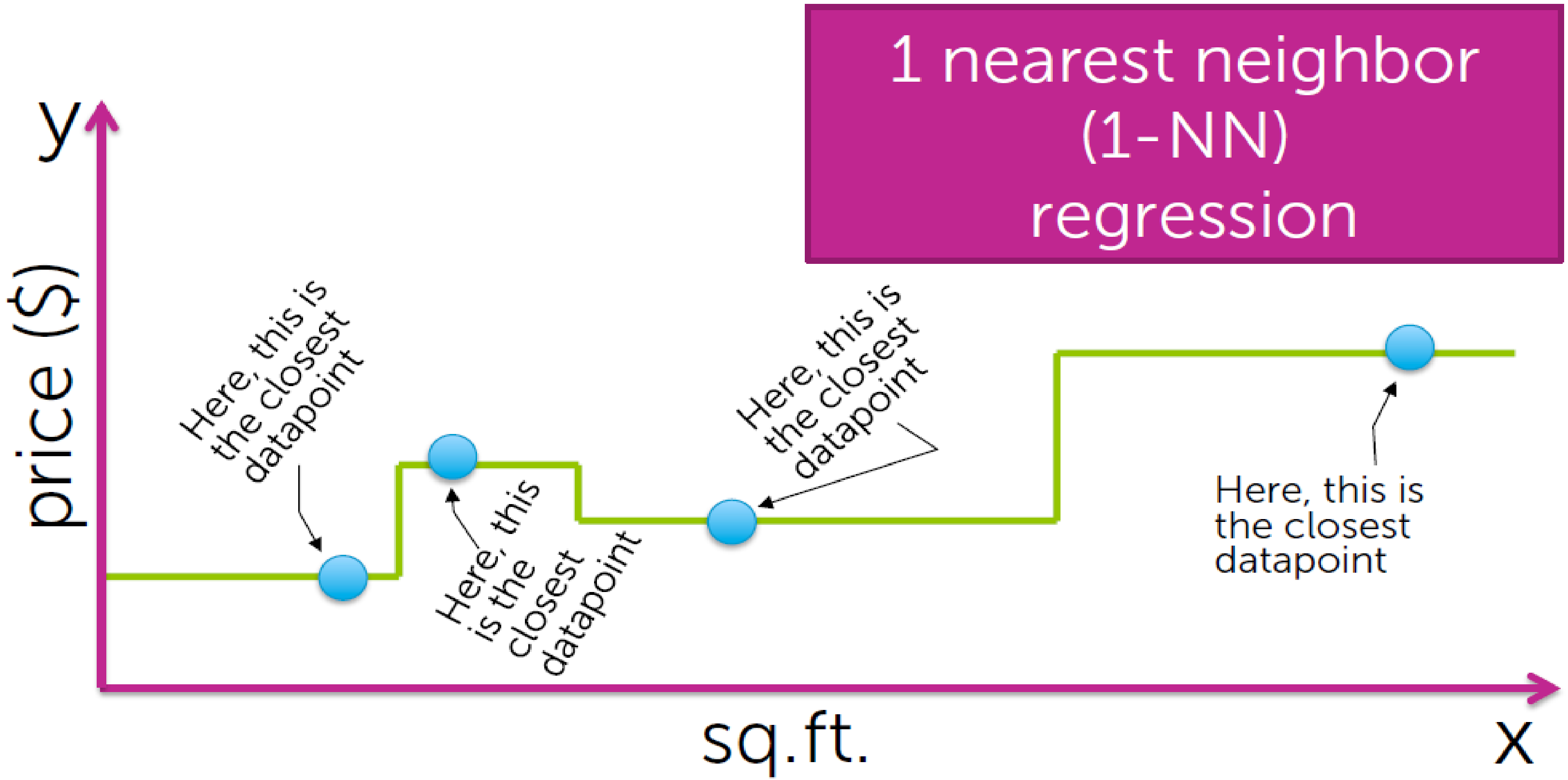
- Assuming we have plenty of data...

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

$ = ???

$ = 850k

1-NN regression more formally

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

Query point: $\mathbf{x}_q$ ← \$?

big lime green house

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

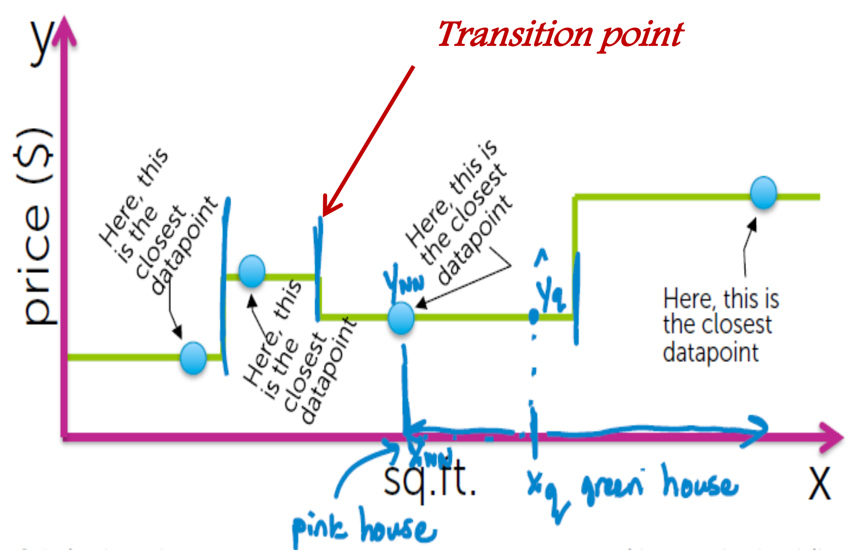
$X_{NN} \leftarrow \min_i \text{distance}(x_i, x_q)$

big pink house

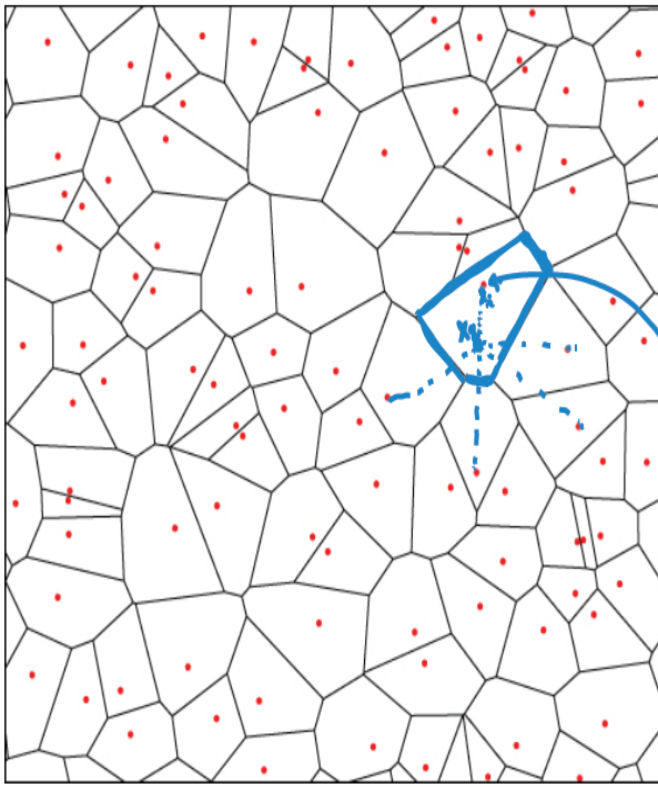
2. Predict

$\hat{y}_q = y_{NN}$

sales price of big pink house



Visualizing 1-NN in multiple dimensions



Voronoi tesselation (or diagram):

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

X_q 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:

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

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

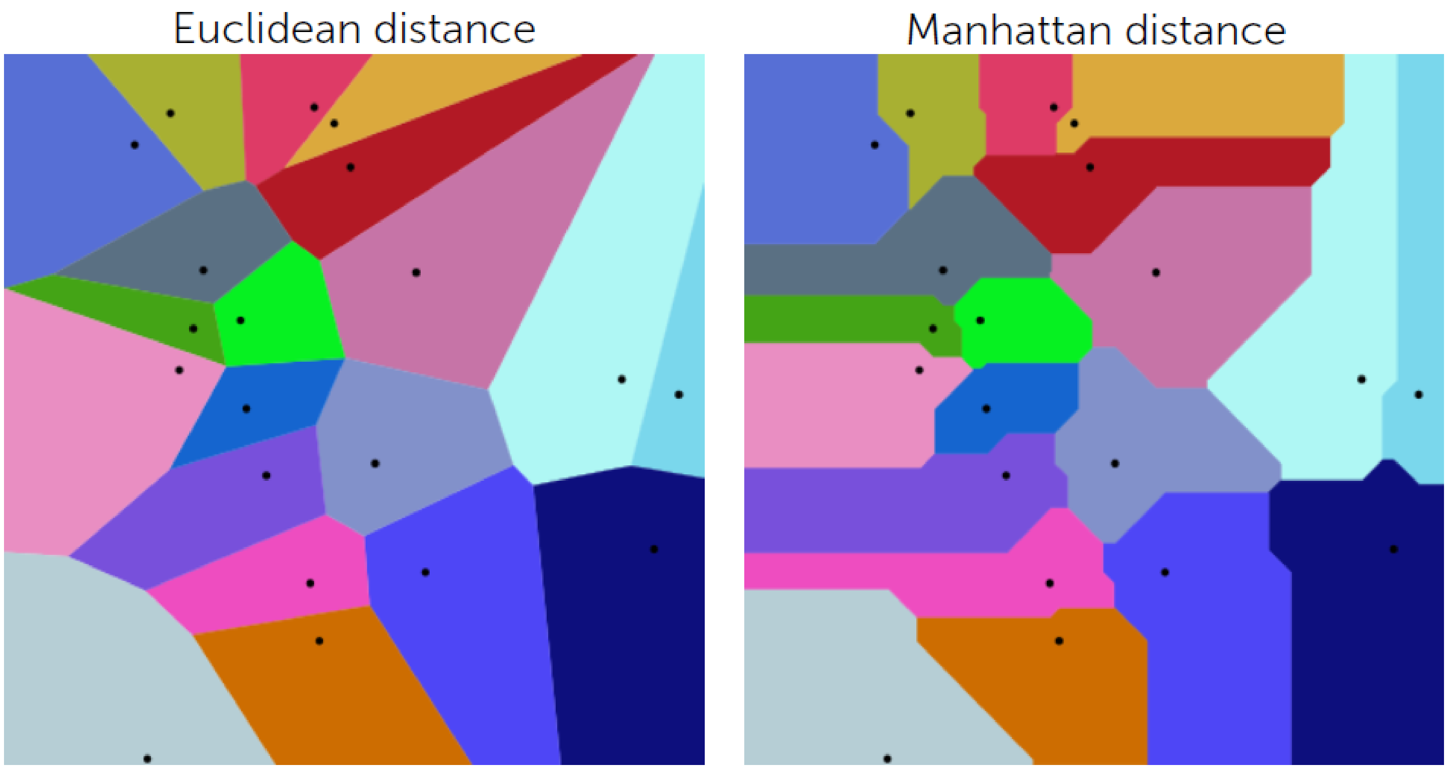
$$\text{distance}(\mathbf{x}_j, \mathbf{x}_q) = \sqrt{a_1(\mathbf{x}_j[1]-\mathbf{x}_q[1])^2 + \ldots + a_d(\mathbf{x}_j[d]-\mathbf{x}_q[d])^2}$$

weight on each input
(defining relative importance)

Other example distance metrics:

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

Different distance metrics



Performing 1-NN search

- Query house:

- Dataset:

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

1-NN algorithm

closest house

Initialize **Dist2NN** = ∞,  = Ø

For i=1,2,…,N

query house

Compute: **δ = distance(**  $_i$ **,**  $_q$ **)**

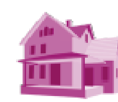
If **δ < Dist2NN**

set   $_i$

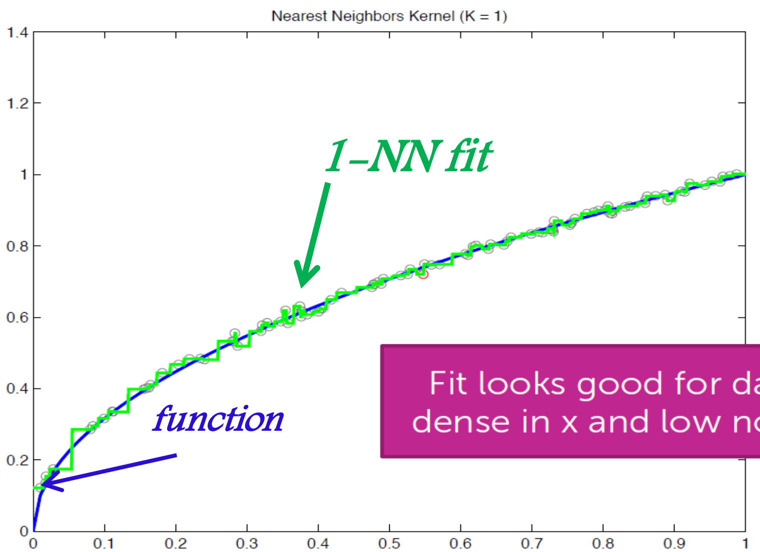
set **Dist2NN = δ**

Return most similar house 

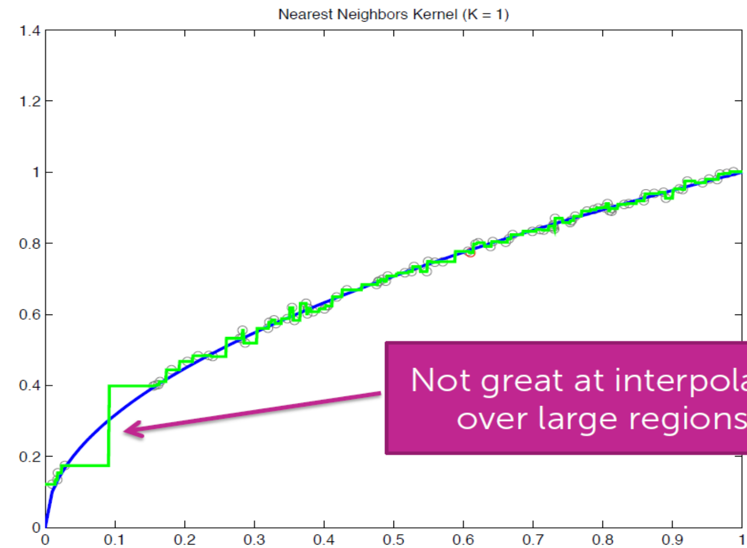
closest house to query house



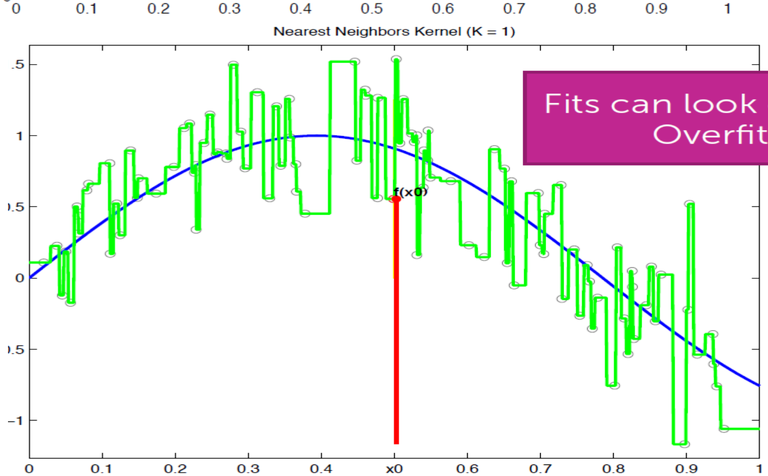
1-NN in practice



Fit looks good for data dense in x and low noise



Not great at interpolating over large regions…

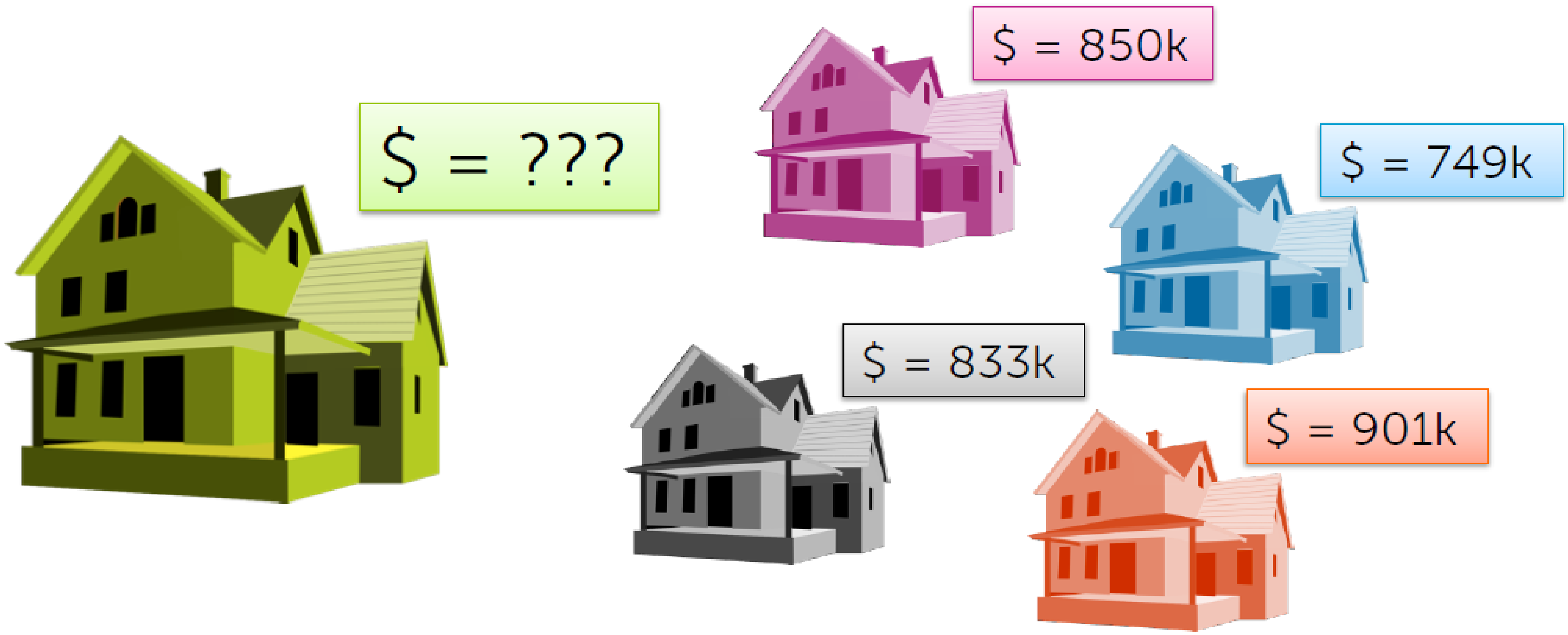


Fits can look quite wild… Overfitting?

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: $(\mathbf{x}_1,y_1), (\mathbf{x}_2,y_2),\ldots,(\mathbf{x}_N,y_N)$

Query point: $\mathbf{x}_q$

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

   $(x_{NN_1}, x_{NN_2}, \ldots, x_{NN_k})$ such that for any x_i not in nearest neighbor set,

   $\text{distance}(x_i, x_q) \geq \text{distance}(x_{NN_k}, x_q)$

2. Predict

   $$\hat{y}_q = \frac{1}{k}(y_{NN_1} + y_{NN_2} + \ldots + y_{NN_k})$$

   $$= \frac{1}{k}\sum_{j=1}^{k} y_{NN_j}$$

K-NN more formally

- Query house: 
- Dataset: 
- **Specify:** Distance metric
- **Output:** Most similar houses

K-NN algorithm

sort first **k houses**
by distance to query house

Initialize **Dist2kNN** = sort(δ_1,…,δ_k) ← list of sorted distances

= sort(, …,)$_1$ $_k$ ← list of sorted houses

For i=k+1,…,N

query house

Compute: δ = **distance(** $_i$ **,** $_q$ **)**

If δ < **Dist2kNN**[k]

find j such that δ > **Dist2kNN**[j-1] but δ < **Dist2kNN**[j]

remove furthest house and shift queue:

[j+1:k] = [j:k-1]

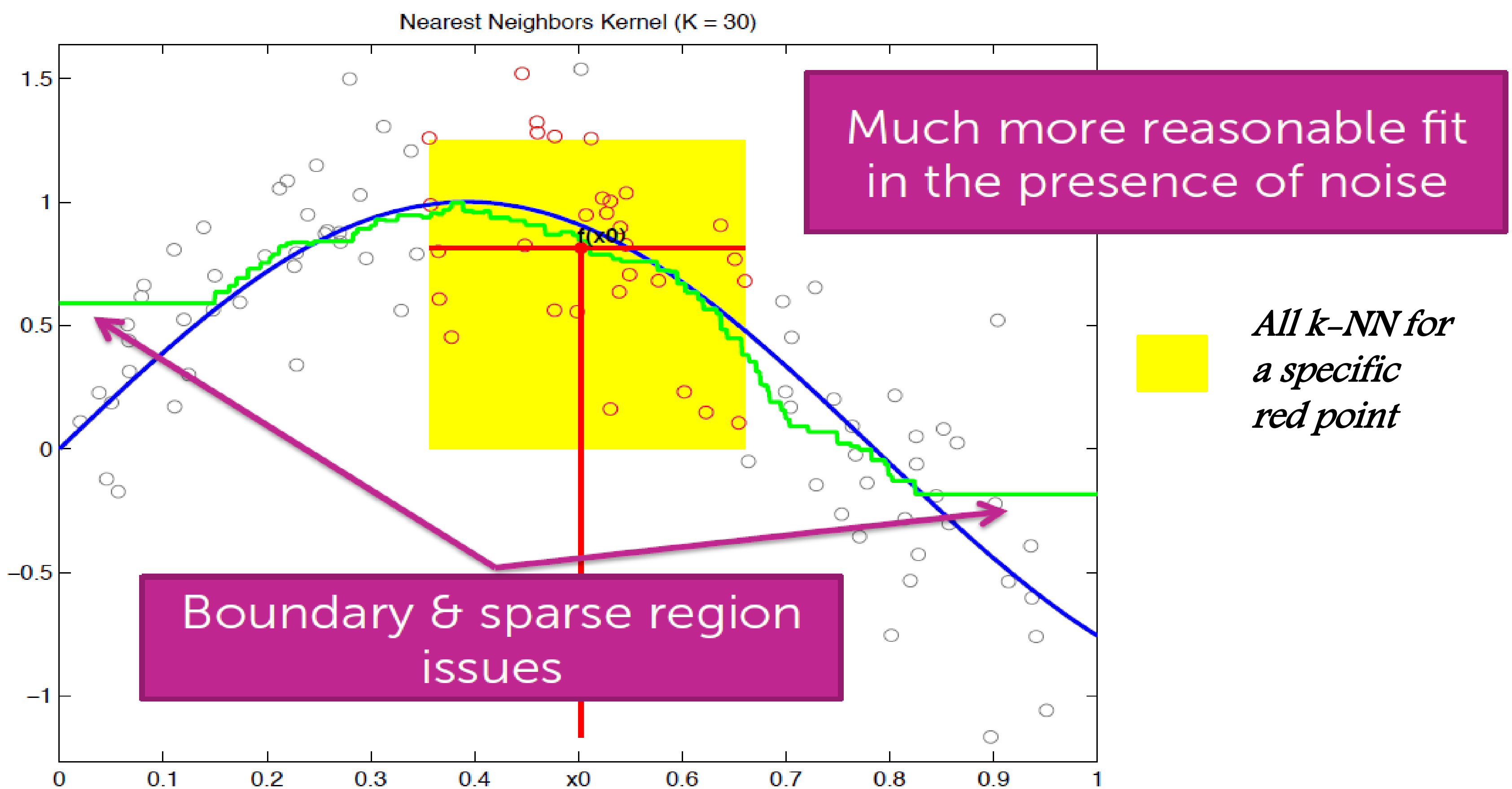
Dist2kNN[j+1:k] = **Dist2kNN**[j:k-1]

insert new NN

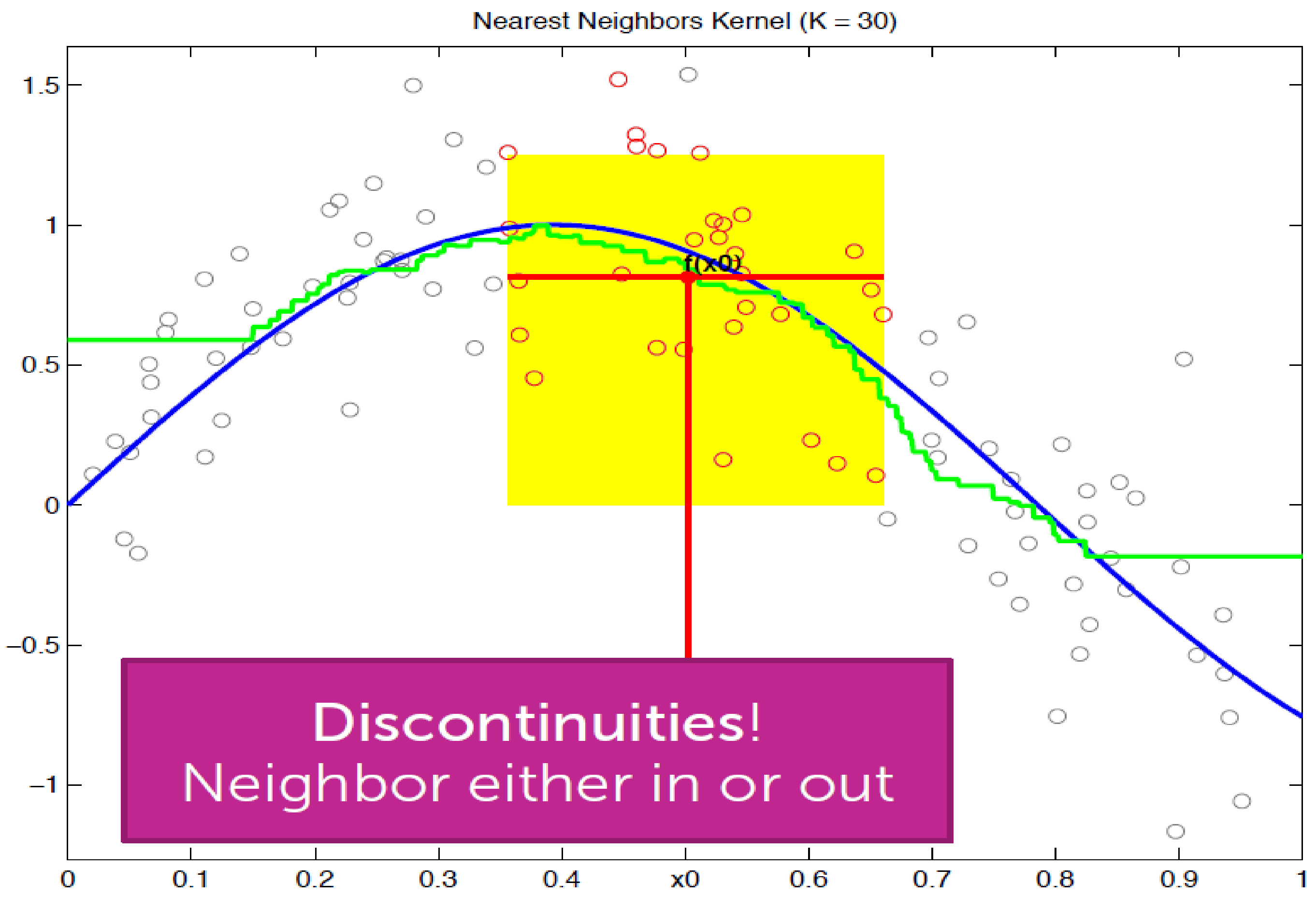
~~set~~ **Dist2kNN**[j] = δ and [j] = $_i$

Return k most similar houses ← 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

weights on NN

Predict:

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

How to define weights

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

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

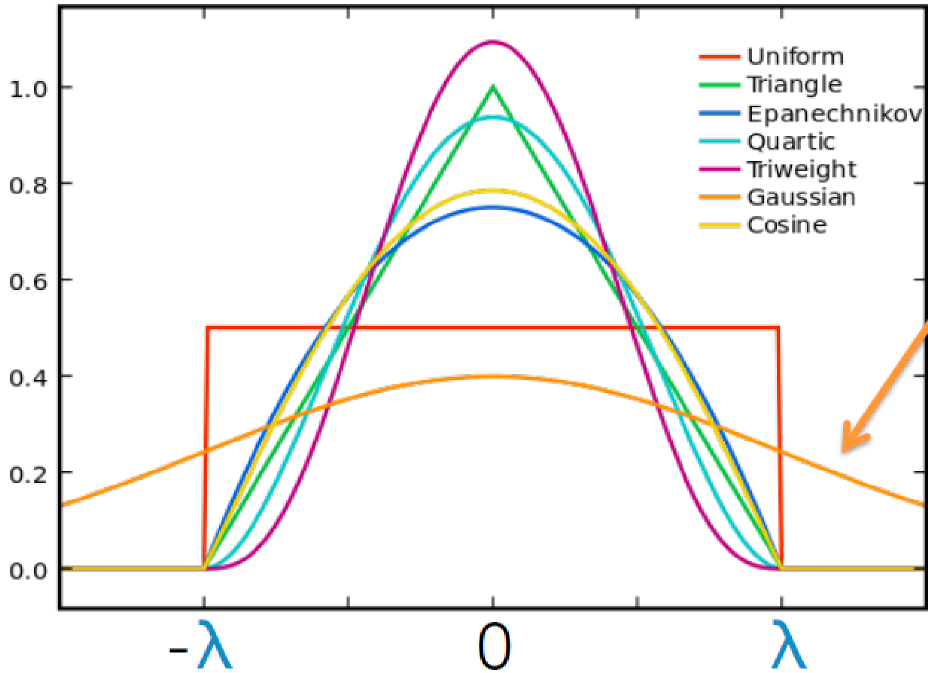
Simple method:

$$c_{qNN_j} = \frac{1}{\text{distance}(x_j, x_q)}$$

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

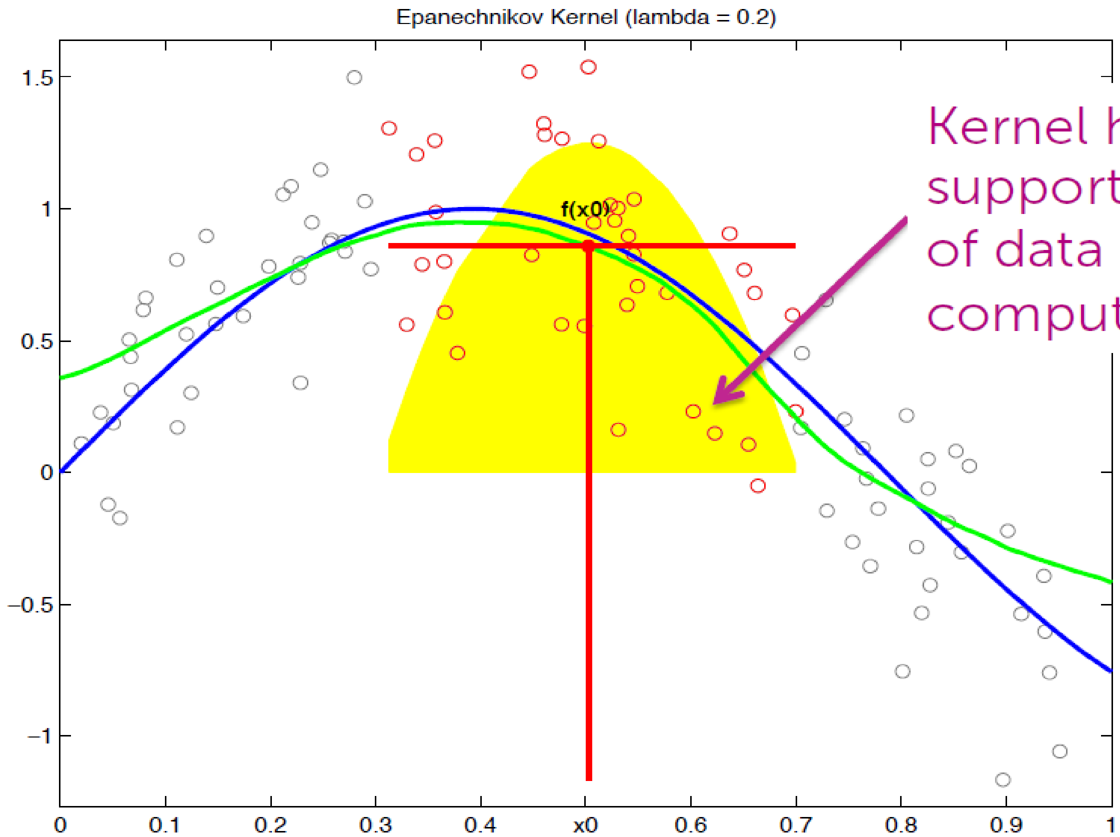
Instead of just weighting NN, weight *all* points

Predict:

weight on each datapoint

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

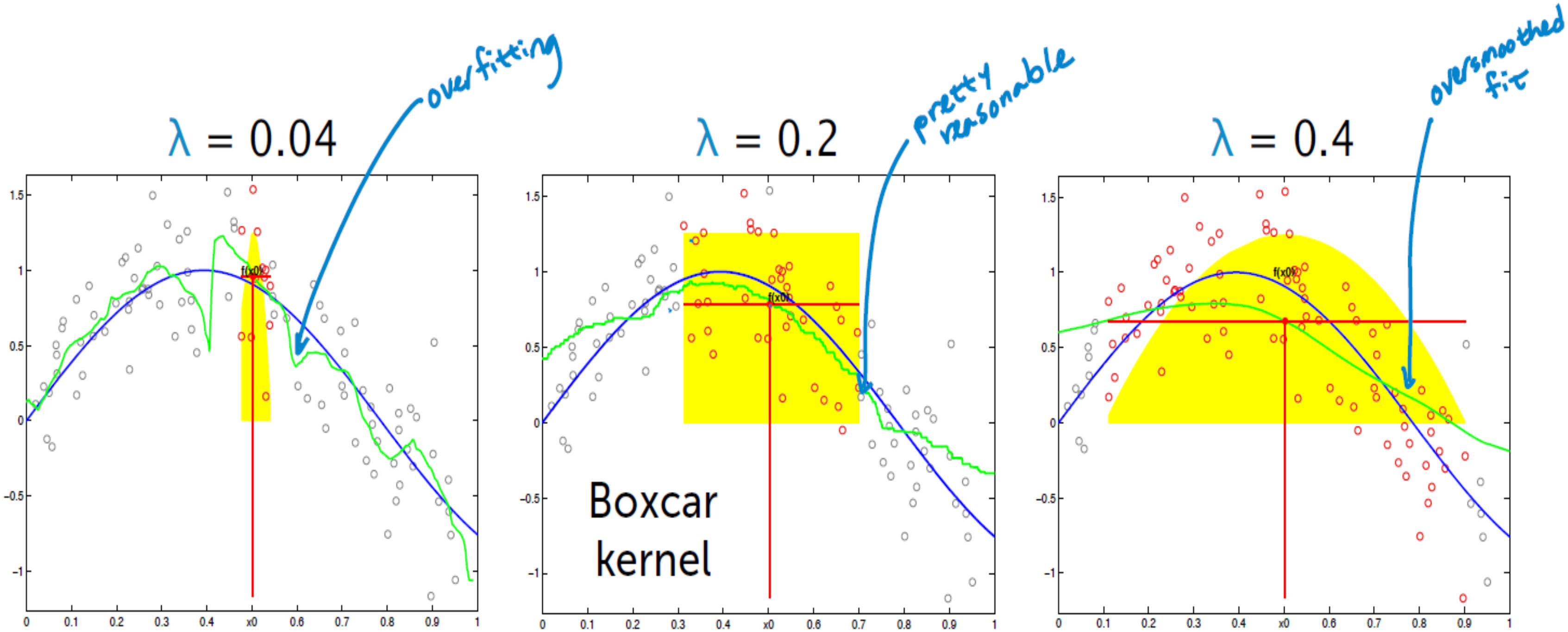
Kernel regression in practice



Kernel has bounded support…Only subset of data needed to compute local fit

Choice of bandwith 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...

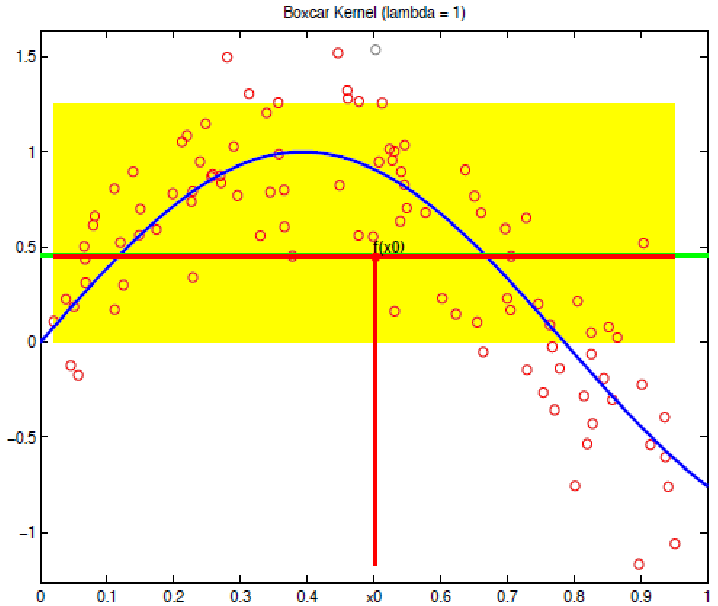
Cross Validation

Contrasting with global average

A globally constant fit weights all points equally

equal weight on each datapoint

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

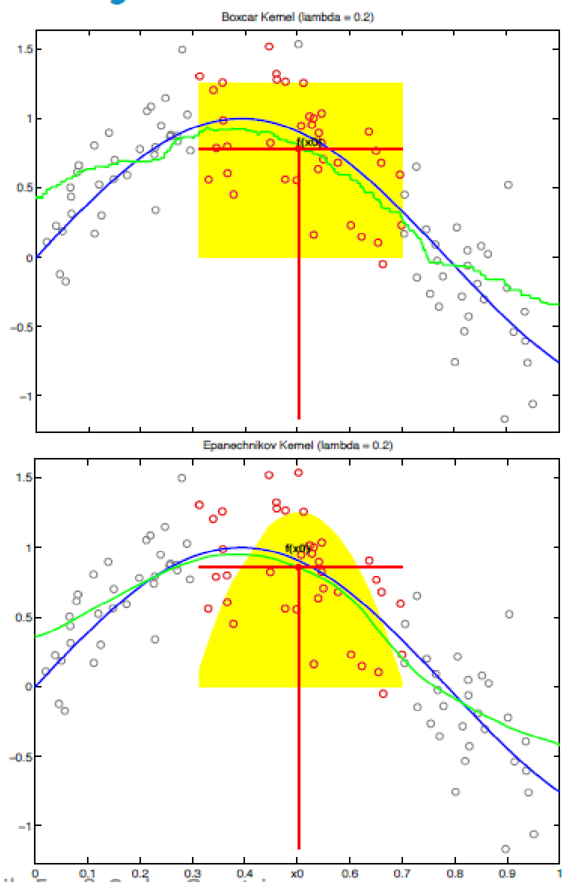


Contrasting with global average

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(**x**)
- Complexity can grow with the number of observations N

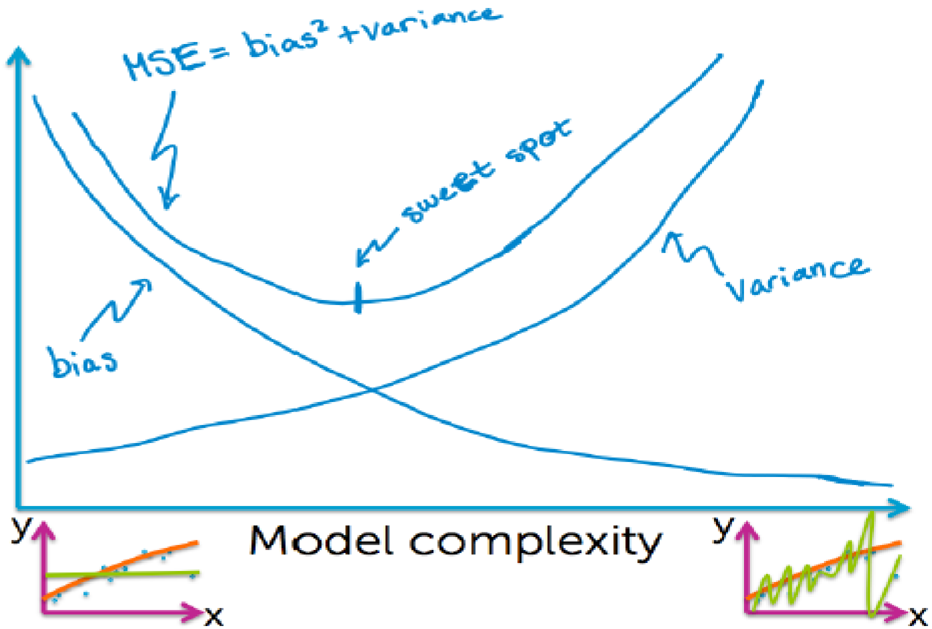
Lots of other choices:

- Splines, trees, locally weighted structured regression models…

Limiting behaviour of NN

Noiseless setting ($\varepsilon_i = 0$)

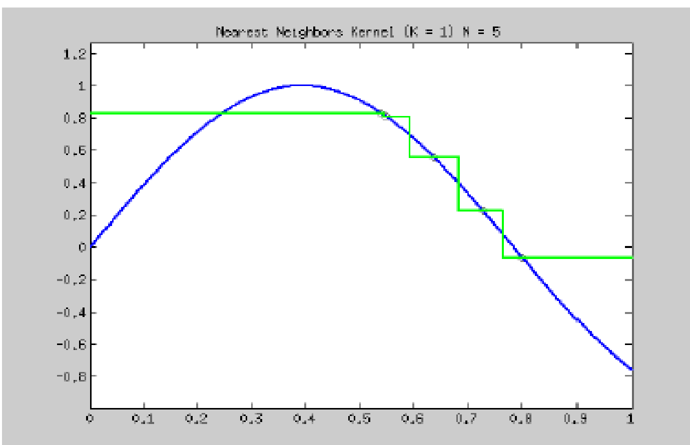
In the limit of getting an infinite amount of noiseless data, the MSE of 1-NN fit goes to 0



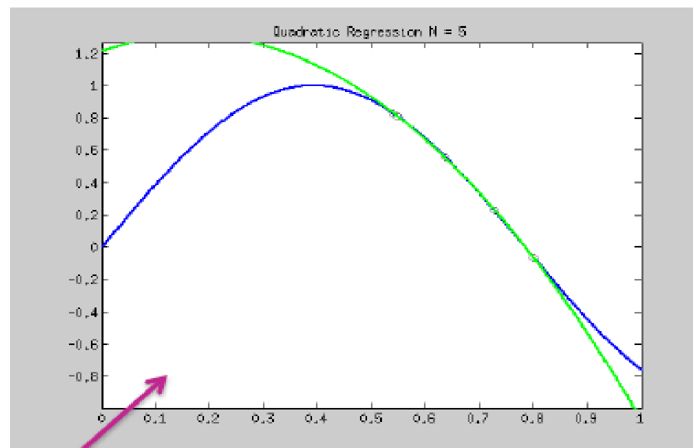
Limiting behaviour of NN

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



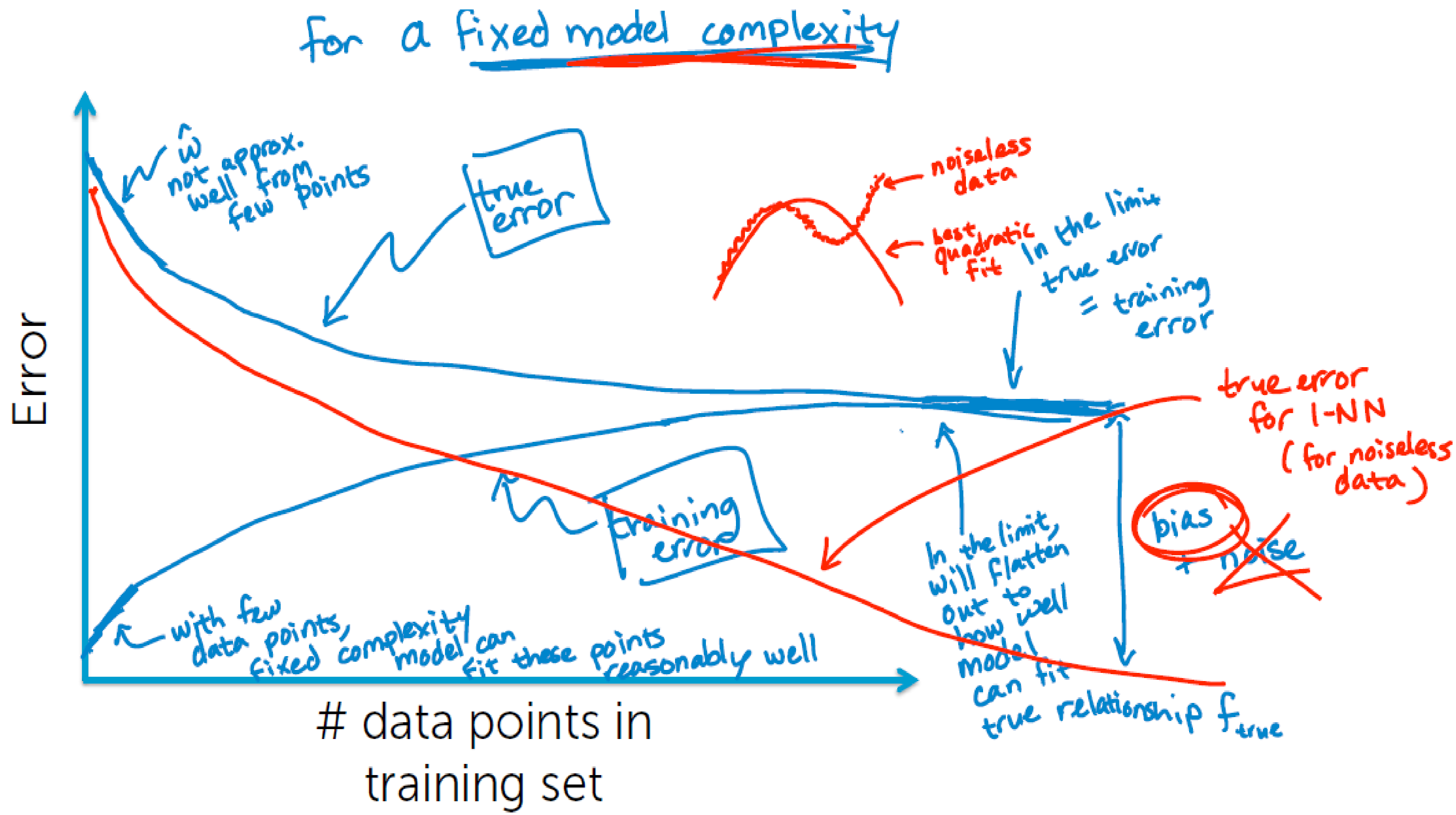
1-NN fit



Quadratic fit

Not true for parametric models!

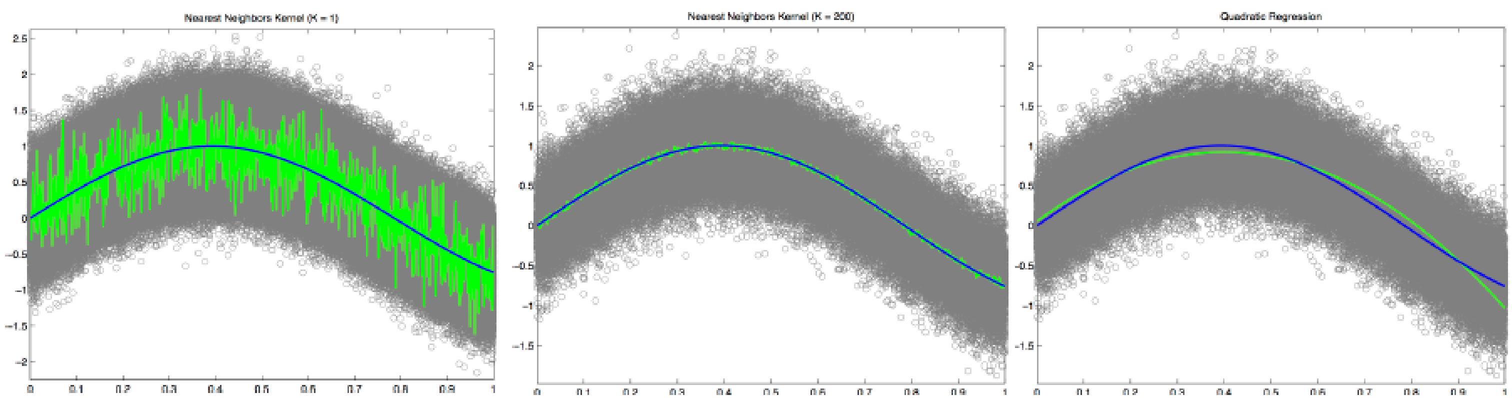
Error vs amount of data



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



1-NN fit 200-NN fit Quadratic fit

Issues: NN and kernel methods

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

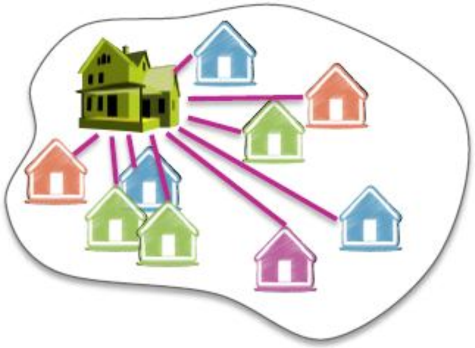
- the more dimensions d you have, the more points N you need to cover the space
- need $N = O(\exp(d))$ data points for good performance

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 $\mathbf{x}_1,\mathbf{x}_2,\ldots,\mathbf{x}_N$
- O(N) distance computations per 1-NN query!
- O(Nlogk) per k-NN query!



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

Will talk more about efficient methods in Clustering & Retrieval course

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