Simple Regression

——predicting house price

data: house id, houses square feet(x) and their prices(y)

output: price of a house

model: expected relationship between square feet and price

yi = f(xi) + ei

E(ei) = 0 equally likely to be above or below 0

*‘Essentially all models are wrong, but some are useful.’*

**Step 1- which model f(x)**

constant? linear? quadratic? polynomial?...

**Step 2- for a given model f(x) estimate function f-hat(x) from data**

find the best function

(1) simple linear regression

cost of using a given line: RSS = ∑(yi – y-hat )2

search over all possible lines and to try one resulting in lowest RSS (see in (2))

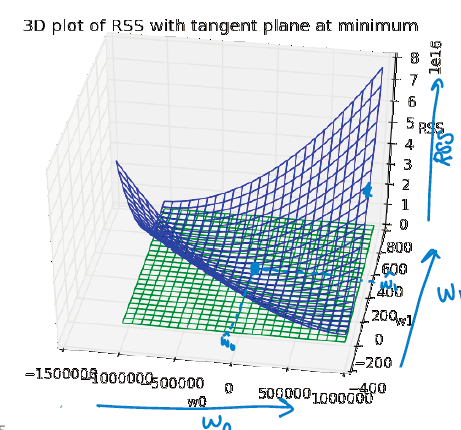
get: yi = w0 + w1x + e

predict: y-hat = w0-hat + w1\*square\_feet

interpret:

w0-hat: predicted value of house with 0 square feet

w1-hat: predicted change of price per square feet

(2) algorithms for fitting the model

search over all possible w0 and w1 to find best line

How?

plot RSS(w0, w1) and find the specific w0 and w1 that minimize RSS

i.e.

this is a convex function and we can sue the gradient descent algorithm to find the minimum (algorithm see (3))

let gradient = 0:

from first line, (average y minus average x \* w1-hat)

from second line,

plug in formulation of ,

in other way, prediction of w1(slope) is:

numerator = (sum of X\*Y) - (1/N)\*((sum of X) \* (sum of Y))

denominator = (sum of X^2) - (1/N)\*((sum of X) \* (sum of X))

slope = numerator / denominator

intercept = (mean of Y) - slope \* (mean of X)

however, gradient descent is always a more efficient way than solving gradient = 0

(3) to find maxima or minima:

convex/concave function

choose any a and any b, if the line connected g(a) and g(b) below g(w) everywhere- concave function

vice verse

but there are neither concave nor convex function

to find the minimum/maximum for a concave/convex function

at the point where the derivative is 0 (no rate of change of this function)

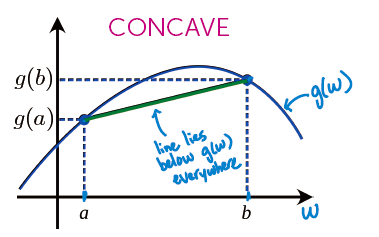
for function neither concave/convex: at multiple/no points, derivative is zero

find max/min via hill climbing/descent

move by a stepsize, and calculate derivative, converge to find the point derivative is 0

commonly, we decreasing the stepsize as the number of iterations increase, such as alpha/t or alpha/square(t)

in practice, stop when derivative < threshold set



when there are more than one variables (just like in the case RSS RSS(w0, w1)), instead of calculating derivative, we calculate gradients

w is a vector of variables like [w0, w1,…,wp]

: partial derivative, calculate derivative while treating other variables as constants

then converge until magnitude of gradient || lower than threshold, or just set gradient as 0

(4) influence of high leverage points

high leverage points:

It is at an extreme x value where there are not other observations. As a result, recalling the closed-form solution for simple regression, this point has the potential to dramatically change the least squares line since the center of x mass is heavily influenced by this one point and the least squares line will try to fit close to that outlying (in x) point. If a high leverage point follows the trend of the other data, this might not have much effect. On the other hand, if this point somehow differs, it can be strongly influential in the resulting fit.

influential observations:

An influential observation is one where the removal of the point significantly changes the fit. As discussed above, high leverage points are good candidates for being influential observations, but need not be. Other observations that are not leverage points can also be influential observations (e.g., strongly outlying in y even if x is a typical value).

figured as in ‘PhillyCrime.ipynb’

remove the sample “center city” which is strongly deviated from the common trend

High leverage points may not be influential observation.

While influential observation can exist in typical observations.

esp. high-end outlier points

high-end observations are more likely to be influential observations

figured as in ‘PhillyCrime.ipynb’

take regions with house value higher than 350000 as high-end outlier neighbors

(5) asymmetric cost function

other than RSS, there are other measures of error.

what if there might not be symmetric cost to those errors (E(e) != 0)

→ asymmetric error

we need to consider what if cost is too high and what if too low to establish a function

e.g. if house price is set too high, no offer at all; if too low, cost is much fewer- prefer to underestimate value than over

Multiple Regression

(1) Introduction

**Multiple regression**:

linear regression when you have multiple features

Each feature is some function of either single input or multiple inputs

**Polynomial regression**:

yi = w0 + w1xi + w2xi2 + … + wpxip + εi

we treat w0, w1xi, w2xi2, …, wpxip all as features

feature1 = 1 (constant feature) parameter1 = w0

feature2 = x parameter2 = w1

feature3 = x2 parameter3 = w2

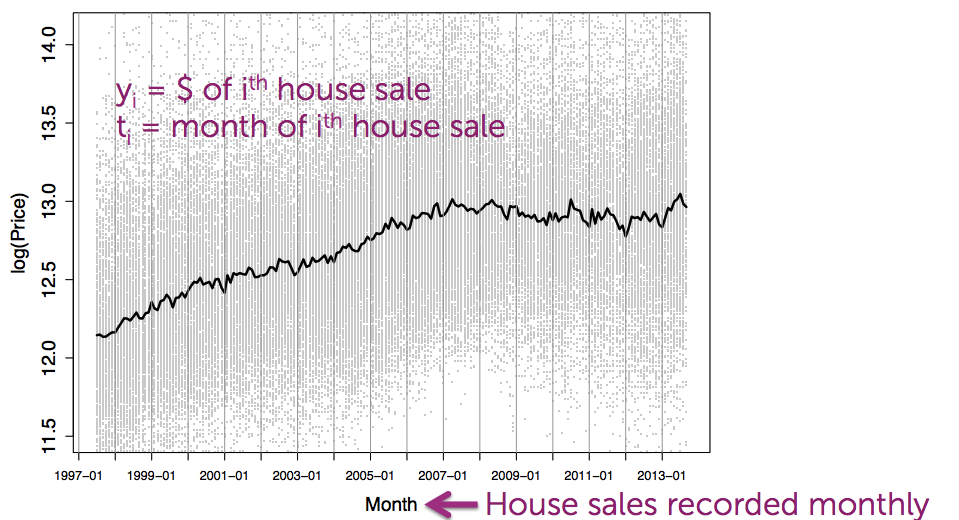
feature(p+1) = xp parameter(p+1) = wp

(2) application: detrending time series (seasonality)

we can see from the chart: overtime, house price rises

most houses listed in summer and good houses sell quickly; few homes listed in other time and transactions often leftover inventory or special circumstances

→ higher in summer, lower in off-month



model:

yi = w0 + w1ti + w2 soin(2πti/12 – Φ) + εi

w1ti : increasing trend

w2 sum(2πti/12 – Φ): seasonal component (reset annually)

Φ: unknown phase or shift, also a parameter

since sin(a-b) = sin(a)cos(b) – cos(a)sin(b)

→

yi = w0 + w1ti + w2 sin(2πti/12) +w3cos(2πti/12) + εi (Φ put inside parameters)

other examples:

weather modeling, flu monitoring, demand forecasting, motion capture data

(3) generic basis expansion

model:

yi = w0h0(xi) + w1h1(xi) … + wDhD(xi) + εi = ∑wjhj(xi) +εi

feature1 = h0(x) often 1(constant)

feature2 = h1(x) e.g. x

feature3 = h2(x) e.g. x2 or sin(πx/12)

(4) incorporating multiple inputs

currently, we only have one input x

e.g. not only square feet matters, but also number of bathrooms

**a simple hyperplan:**

yi = w0 + w1**x**i[1] + … + wd**x**j[d] +εi

feature1 = 1

feature2 = **x**[1] e.g. square feet

feature3 = **x**[2] e.g. number of bathrooms

feature(d+1) =**x**[d] e.g. lot size

**D-dimensional curve:**

yi = w0h0(**xi**) + w1h1(**x**i) … + wDhD(**x**i) + εi = ∑wjhj(xi) +εi

feature1 = h0(**x**) e.g. 1

feature2 = h1(**x**) e.g. **x**[1] = square feet

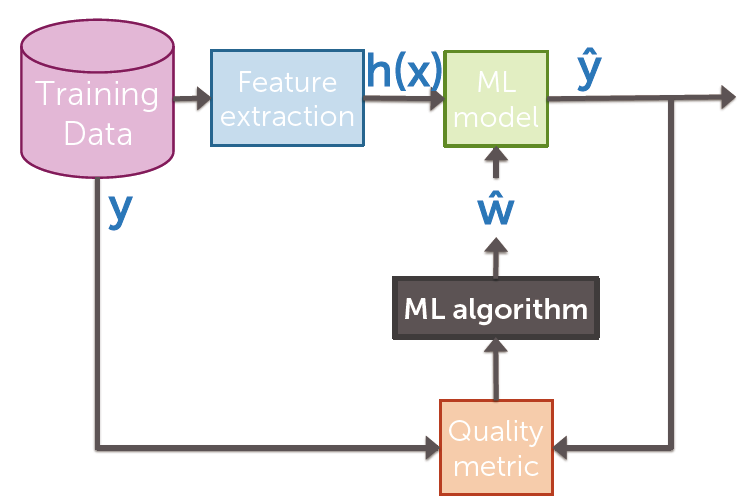
feature3 = h2(**x**) e.g. log(**x**[7])**x**[2] = log(#bed) \* #bath

interpreting the coefficients:

yi = w0-hat + w1-hat **x**[1] + … + wd-hat **x**[d] +εi

for w1-hat, where **x**[1] represents square feet, what is the predicted change of value of house if change 1 square feet (suppose all the other variables fixed)

so every time look at a coefficient, we need to consider all the other context at that time



(5) fitting D-dimensional curves

①single observation

rewrite in matrix notation:

(inner product)

where W = (w0, w1, …, wD)T, **h**(xi) = (h0(xi), h1(xi),…,hD(xi))T

②all observations together

(6) cost of fit for multiple regression

where

gradient of RSS:

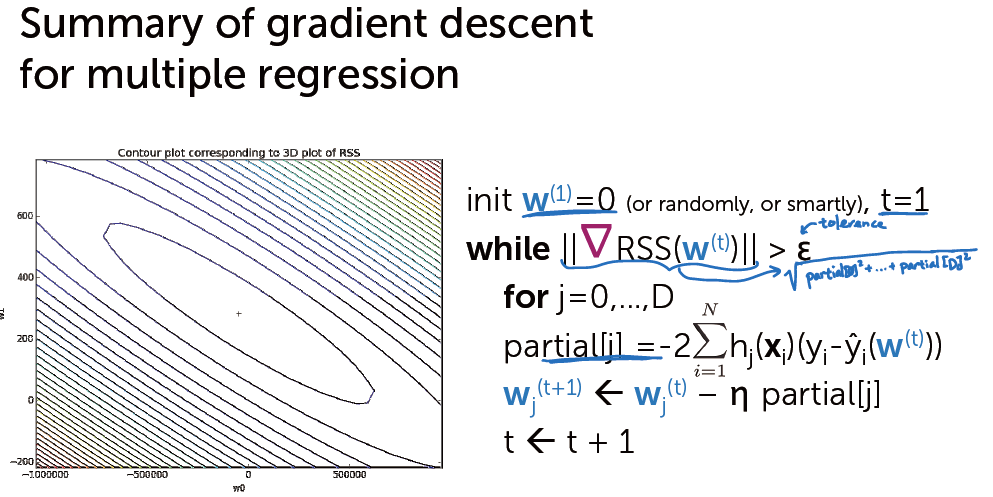
set gradient = 0: (alternatively, we can also use gradient decent algorithm)

H：num of features(D) X num of observations(N)

HT \* H：D X D matrix

it is invertible in most cases (N>D i.e. number of linearly independent observations is greater than number of features)

complexity of the inverse is **O**(D3) (the number of operations we have to do to invert this matrix scales cubically with the number of features of our model)



Assess performance

(1) Measuring loss

how much am I losing?  
what do we mean by ‘loss’?

e.g. in the predicting house price case

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?

→**Loss function**

y: true observation

w-hat: estimated parameters

examples for loss function:

①assume loss for under-predicting = over-predicting

——absolute error

——squared error

*‘Remember that all models are wrong; the practical question is how wrong do they have to be not be useful.’*

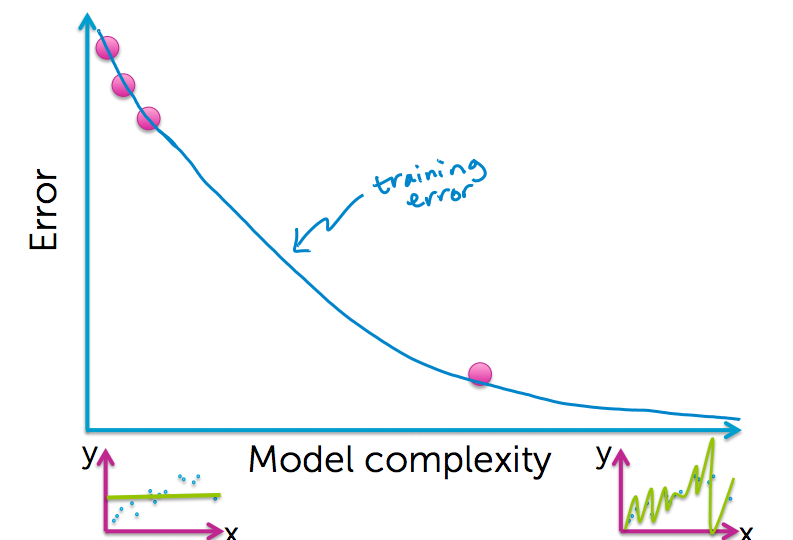
(2) Training error

define a loss function

training error = average loss on houses in training set =

if we define the loss function as squared error:

RMSE (root mean squared error) =

**training error v.s. model complexity:**

for a constant model (simplest model):

significant training error

for a linear model:

training error goes down

for a higher-order polynomial model:

very low training error

Is training error a good measure of predictive performance?

of course not

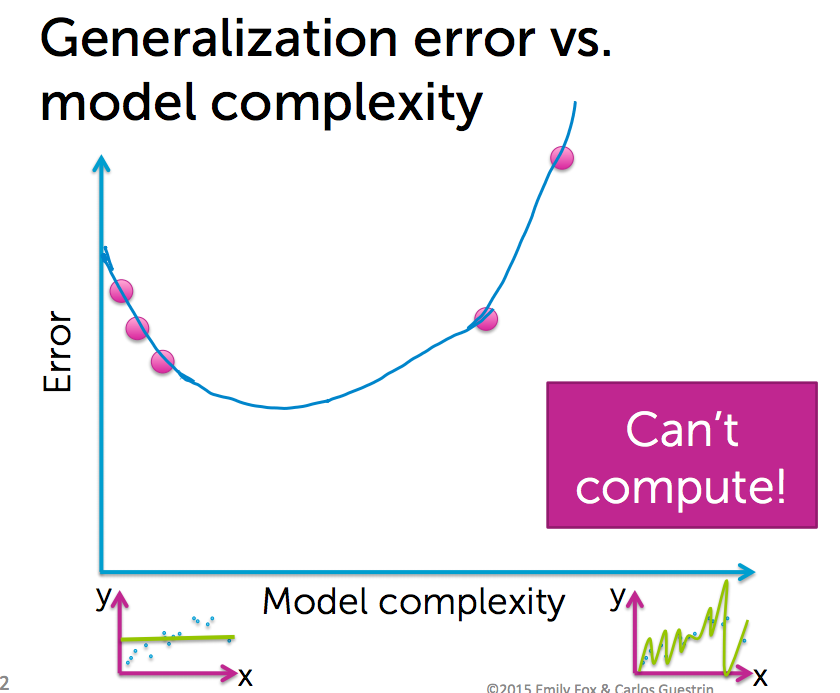
training error is overly optimistic

(3) Generalization error (true error)

want estimate of loss over all possible (house, price) pairs, that is to include lots of houses in neighborhood but not in dataset

distribution over houses- in our neighborhood, house of what square feet are we likely to see

distribution over sales prices- with a given square feet, what house prices are we likely to see

 average over all possible (**x**,y) pairs weighted by how likely each is

**generalization error v.s. model complexity**

for the blue & white area on the right:

close to blue: less likely to appear

close to white: very likely to be true

for a constant model:

only the middle part performs well

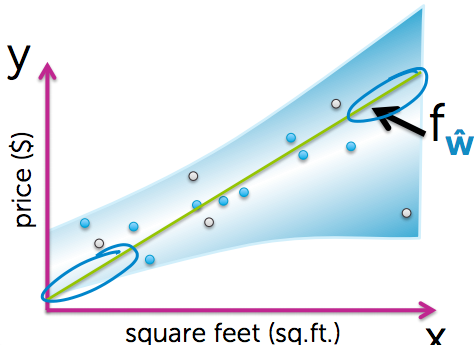
for a linear model:

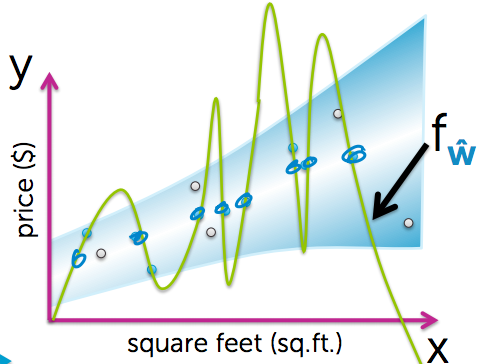
the head and tail part performs badly

for a higher-order polynomial model:

some part behaves really horribly

**we can’t compute generalization error.**





(4) Test error

→approximating generalization error:

approximate by looking at houses not in training set

hold out some (house, price) pairs that are not used for fitting the model i.e. test houses

Test error = average loss on houses in test set =

pay attention that fw-hat(x) is fit using training dataset

Since test set will not include all the houses not included in the train set, it will appear as a noisy type of the true error v.s. model complexity curve.

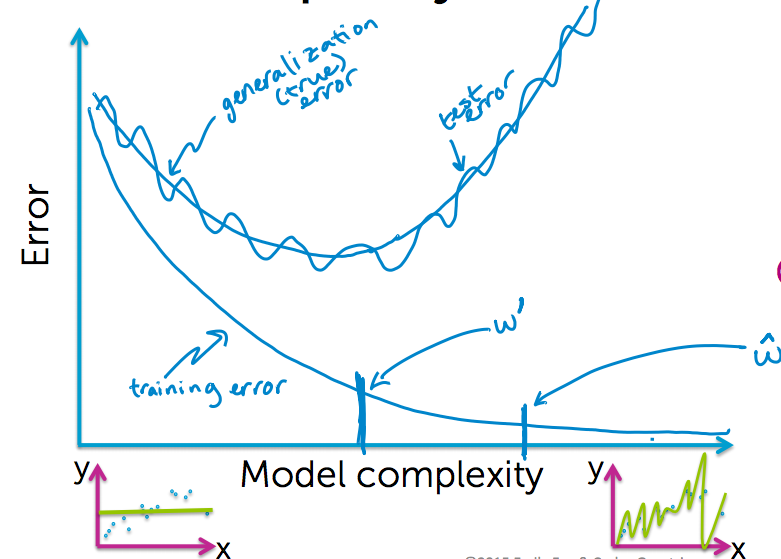
Test error is what we can compute and true error is what we really want.

(5) Overfitting

a model (w-hat) is overfit if there exist a model with estimated parameters(w’) such that :

training error of w-hat < training error of w’

true error of w-hat > true error of w’



(6) Training/test split

how many training set vs. how many test set?

if too few training set → poorly estimated

if too few test set → test error is bad approximation of generalization error

typically, we need just enough test points to form a reasonable estimate of generalization error

if this leaves too few for training, we can employ other methods like cross validation

(7) three sources of error

In forming predictions, there are 3 sources of errors:

explained technically in (10)

**①noise**

date inherently noisy

true relationship between x and y: yi = fw(true)(**x**i) +εi

we used to suppose E(εi)=0, but this is not true

this is irreducible error, we can do nothing

**②bias**

different choice of training dataset lead to different models

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

Bias is the different between the average fit and the true function:

higher complexity model tends to have low bias.

**③variance**

how much do specific fits vary from the expected fit?

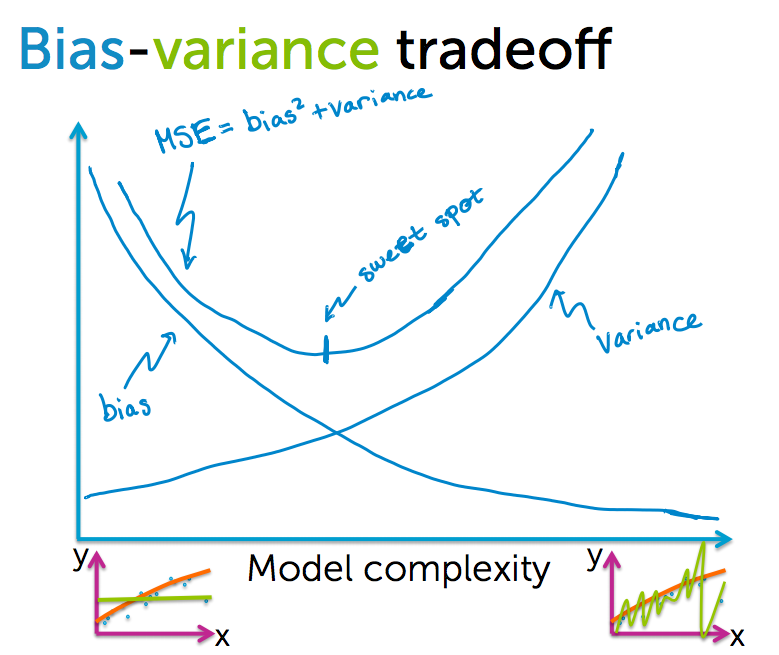
low complexity model tends to have low variance.

(8) Bias-variance tradeoff

MSE (mean squared error) = bias2 + variance

→try to lower MSE

but just like the generalization error, we cannot compute bias and variance.



(9) error v.s. amount of data

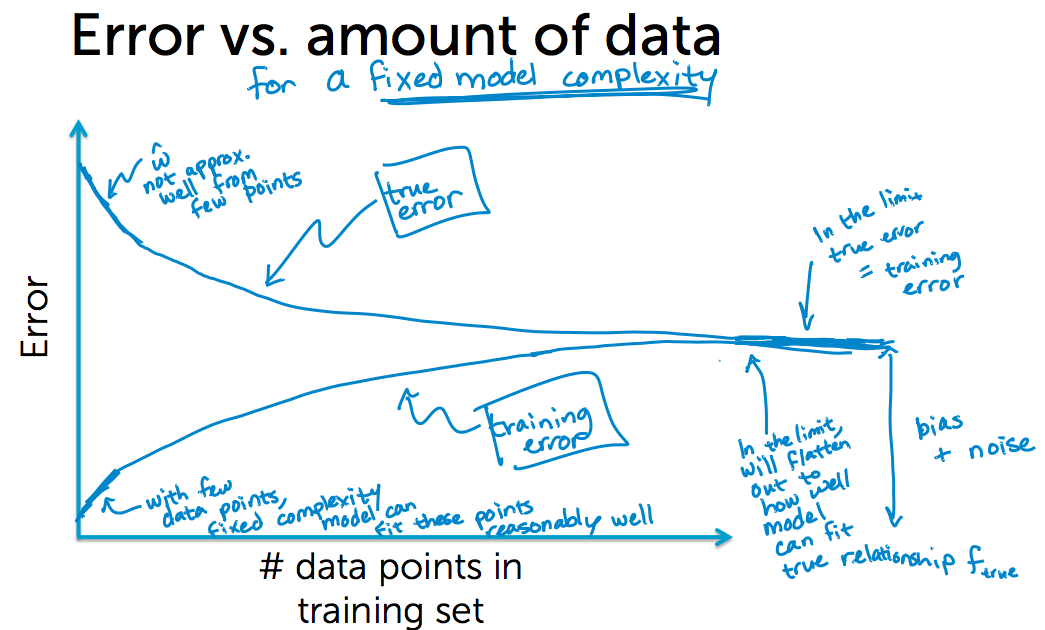
consider for a fixed model complexity

**true error** goes down when we have more data points in training set, but converge to a certain limit, which equals bias + noise.

**training error** goes up when we have more data points in training set, but also stop at a limit, which equals bias + noise, too. (imagine we have the whole dataset in the world as the training dataset, then training error is exactly the same as true error.)

In the limit, the curve will flatten out to how well model can fit true relationship ftrue.

In the limit, our true error = our training error.



(10) Technical explanation of the three sources of errors

Expected prediction error =

the expected value of our generalization error, over different training data sets

prediction error at target input

start by considering loss at target **x**t using squared error loss

average prediction error at **x**t =

① —— noise

suppose true relationship between square feet and house value is:

y = f**w**(true)(**x**) + ε relationship exist out there in the world

but this does not fully capture what we think about the value of a house, there are other factors that play. and all those factors there in the world are captured by noise term epsilon.

is the variance (spread) of this noise epsilon.

This is irreducible error.

② —— bias

bias- how well our model could on average fit the true relationship between x and y.

when we choose different training dataset, we will get different estimated parameters.

we look over all possible dataset of certain size N that we might have gotten.

where

expectation of the fit we got on specific training data set

tell us over all those possible datasets, this average function just can never capture anything close to their true relationship between x and y.

③ —— variance

over all possible fits we might see, how much do they deviate from the expected fit (average fit defined in bias)?

variance of some random variable is simply looking at the expected value of that random variable minus its main squared.

defines how much deviation of specific fit from expected fit at **X**t

tell us how much my specific function can deviate from my expected fit over all datasets.

expected prediction error at **x**t =

①:

y-f=ε, this is exactly ε2

②

since y-f =ε, which is independent wih f – fhat, this equals

③:

both f and f-bar do nothing with training dataset, so

by definition

→

expected prediction error =

(11) summary

During the regression/ML workflow:

①model selection

need to choose tuning parameters λ controlling model complexity

for each considered model complexity λ:

estimate parameters w on training data

assess performance of w on test data

choose λ\* to be λ with lowest test error

②model assessment

having selected a model, assess the generalization error

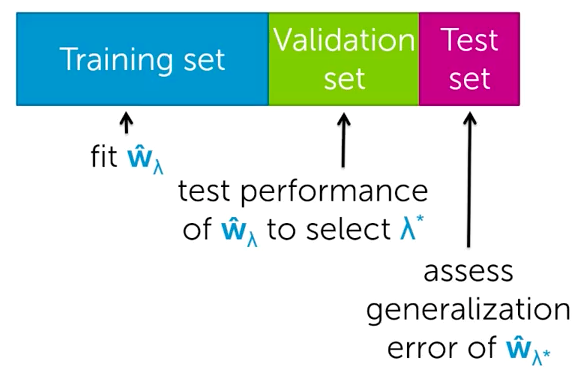
compute test error of wλ\* (fitted model for selected complexity) to approx. generalization error.

→But this process is overly optimistic.

we can modify by creating two test sets: split whole dataset into training set, validation set and test set (typically 80%, 10%, 10%).

select λ\* such that wλ\* minimizes error on validation set

approximate generalization error of wλ\* using test set



Ridge Regression

automatically balance between bias and variance

(1) Overfitting for polynomial regression

refer to the iPython Notebook

the higher the degree, the larger the coefficients tend to be

similar symptom when overfitting for more general multiple regression models (when we have more inputs)

→ridge regression: quantify the overfitting through measure of the magnitude of the coefficients

(2) Overfitting & number of observations & number of inputs

①observation numbers

few observations:

rapidly overfits as model complexity increases (easy to fit all of the observations perfectly but wildly)

many observations:

harder to overfit (we have more dense plotted observations)

②input numbers

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

data must include representative examples of all possible (sq.ft., $) pairs to avoid overfitting (quite hard)

many inputs:

data must include examples of all possible inputs and $ combinations to avoid overfitting (even more harder)

(3) Balancing fit and magnitude of coefficients

want to balance:

how well function fits data

magnitude of coefficients

total cost = measure of fit + measure of magnitude of coefficients

small measure of fit = good fit to training data

small magnitude of coefficients = not overfit

①measure of fit to training data

②measure of magnitude of coefficients

how to measure the overall magnitude of all the coefficient?

→

sum?- if w0 is positive and w1 is negative on similar magnitude, its improper

sum of absolute value?-

**L1 norm**

sum of squares?

**L2 norm squared**

→if we choose L2 norm squared:

we need to minimize RSS(**w**) + λ||**w**||22 **Ridge regression (L2 regularization)**

add λ as a **tuning parameter**= balance of fit and magnitude

if λ=0: just minimize RSS(w) (old solution, lead to w-hatLS (LS for least squares)) (low bias, high variance)

if λ=infinity: for solutions will w-hat does not equal 0, the total cost will be infinity; only minimize when w-hat=0 (high bias, low variance)

if λ in between: 0<||**w-hat**||22<||**w-hatLS**||22

so what do we want for λ?

→λ controls model complexity and bias-variance trade-off

(4) Exploring tuning parameter λ

refer to the iPython Notebook

refer λ as l2\_penalty

small λ: model similar to least squares solution

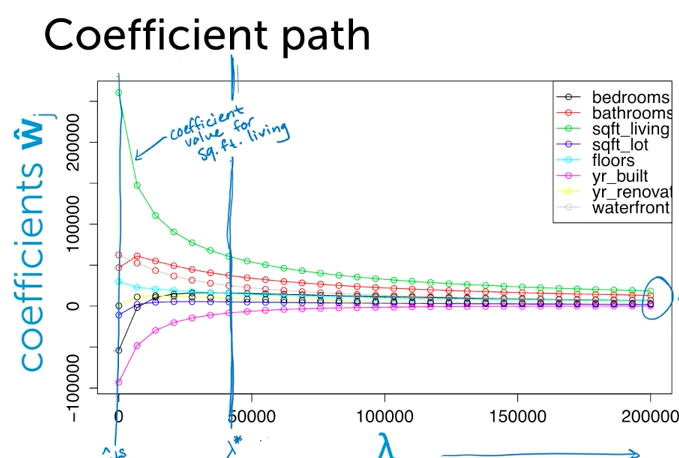
large λ: quite low coefficient with smooth and flat plot

but which one should be selected?

→**leave one out(LOO) cross validation**

approximates minimizing the average mean squared error

(5) Coefficient path with different λ



(6) fitting the ridge regression model (for given λ)

(refer to Multiple Regression(6))

we can set gradient to 0, or use the gradient descent algorithm.

set

interpreting ridge closed-form solution:

if λ = 0:

(H invertible if N>D, complexity of inverse = O(D3))

if λ = infinity:

this is invertible if λ >0, even if N<D.

complexity of inverse = O(D3) (this will be extremely large for large D (lots of features))

(7) how to choose tuning parameter λ

①if sufficient amount of data available:

fit wλ using training set

test performance of wλ to selected λ\* using validation set

assess the generalization error of wλ

refer to Assess performance (11)

②if start with smallish dataset

split off our test dataset

to choose validation set from the rest data:

if only choose a subset, it will not be reasonable for assessing performance

so we choose all subsets and average performance over all subsets

→**K-fold cross validation**

preprocessing- randomly assign data to K groups

for each one of the k subsets, estimate wλ on the training subsets(remaining part) and compute error on the validation subset(this subset)

compute average error

then repeat this procedure for each choice of λ to minimize the average error

what value of K?

formally, the best approximation occurs for validation sets of size1 (K=N (number of observations)) ——**leave-one-out cross validation**

but this is computationally intensive- requires computing N fits of model per λ, so we typically choose K=5 or 10

(8) how to deal with the intercept term

there is always a constant feature w0 – the first column of **H** matrix will be 1 entirely

to minimize cost, it also encourage intercept term w0 to be small

λ can be seen as a penalty for large w0.

if we don’t penalize intercept:

(Imod is a modified identity matrix where the first entity is 0)

if we center data first:

if data are first centered about 0, then favoring small intercept not so worried

step1: transform y to have 0 mean

step2: run ridge regression as normal

Lasso Regression

(1) feature selection

why select features?

for efficiency

for interpretability

the blow three options are algorithms to select features.

(2) option 1: all subsets algorithm

try all subsets of the features and compare their performances

no features:

yi = εi

very high training error(RSS)

one features:

lower training error;

try each one feature to fit a model, find the one with lowest training error

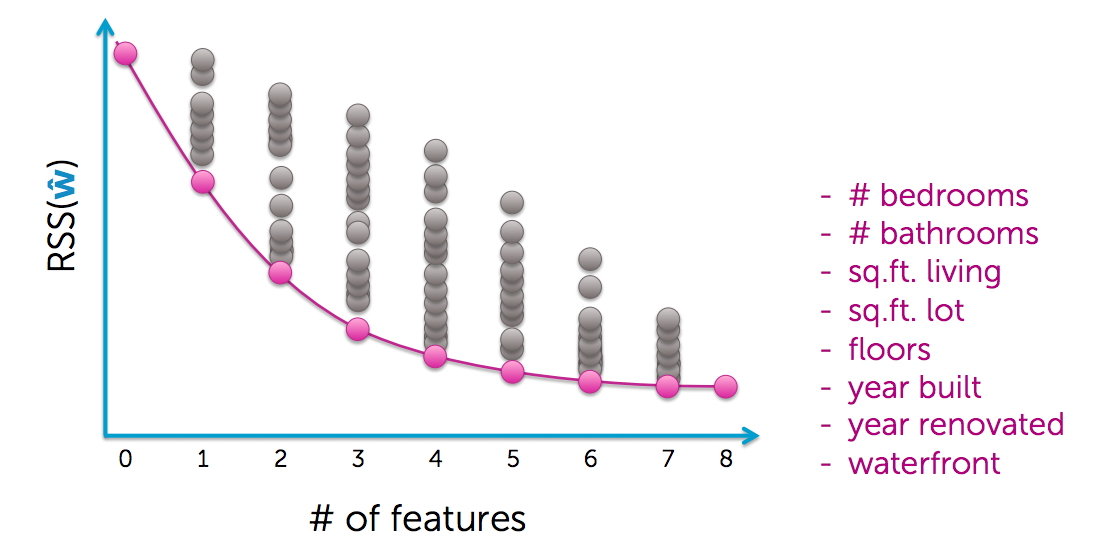
two features:

search over all combinations of two features and find the best combination

…

until fitting the model with all the features

connect points representing all best possible models for each number of features



to choose a model complexity:

option1: assess on validation set

option2: cross validation

option3: other metrics for penalizing model complexity like BIC

…

complexity of ‘all subsets’ algorithm

2D models have to be searched over

typically computationally infeasible

(3) option2: greedy algorithm

forward step algorithm:

pick a dictionary of features {h0(**x**), …, hD(**x**)}

greedy heuristic:

start with empty set of features F0 = ∅(or simple set like just h0(**x**)=1)

fit model using current feature set Ft to get w-hatt

select next best feature hj\*(**x**)

e.g. hj(**x**) resulting in lowest training error when learning with Ft + {hj(**x**)}

set Ft+1 <- Ft + {hj\*(x)}

recurs

therefore error never increase

compare with all-subset algorithm:

the best model when F only contain one feature is the same as all-subset algorithm when searching over one-feature subsets

the best model when all features have been included (the last time in the loop) is the same as the all-subset algorithm, since the only choice is including all features

when do we stop greedy procedure:

asses on validation set or use cross validation

complexity of forward step wise:

1st step: D model

2nd step: D-1 model

…

how many steps? it depends on when we chose to stop.

at most: O(D2) << 2D for large D

other greedy algorithms:

backward stepwise

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

combining forward and backward stepwise

in forward algorithm, insert steps to remove features no longer important

(4) option3: regularize

use regularized regression to implicitly perform feature selection for us

→ **LASSO**

in ridge regression, we use L2 regularized regression

total cost = RSS(**w**) + λ||**w**||22

this encourage coefficients to be small, but not 0

since the coefficients shrinking and shrinking towards 0 when we increase λ

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?

set a threshold and set all ridge coefficients under that to 0 (assume those features not in the model)

flaw: when we have both features # bathrooms and # showers, which are quite close, their ridge coefficients tend to be close too, but lower than only including one of them in the features. This will lead to excluding of # bathrooms and # showers features, while one of them should have been included. →when we have a set of strongly correlated features, then ridge regression is gonna prefer a solution that places a bunch of smaller weights on all the features, rather than one large weight on one of the features.

better solution→

we use **L1 norm** this time

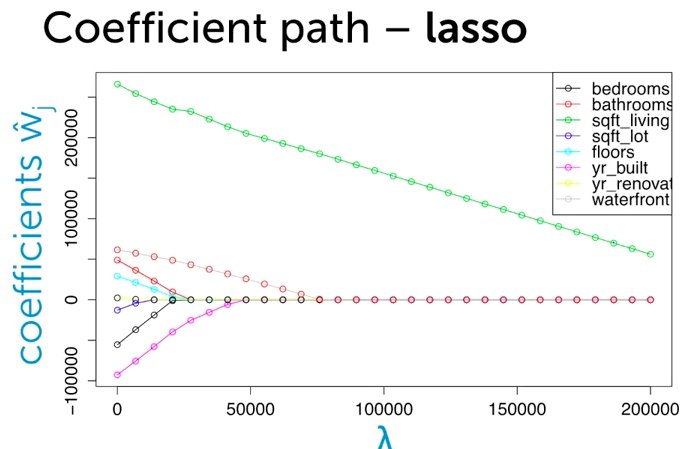
total cost = RSS(**w**) + λ||**w**||1 , this will leads to **sparse** solution

**LASSO Regression** a.k.a. **L1 regularized regression**

searching over the tuning parameter λ = balance of fit and sparsity

if λ=0: old least squares solution

if λ=infinity: w-hat = all 0



(5) visualizing ridge cost

suppose there are two features and visualize the ridge cost in 2D.

RSS(**w**) + λ||**w**||22 =

only look at RSS(w) part, which can be expressed as a2w02 + b2w12  = constant,

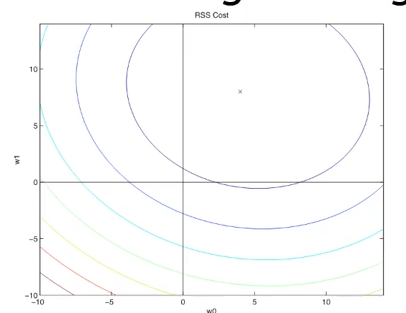
so this is an ellipse as is shown in the left picture below.

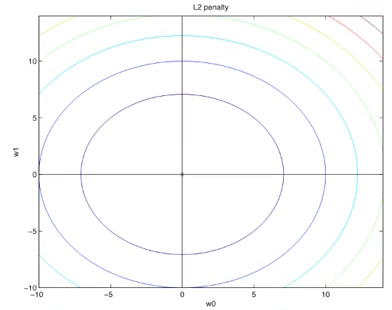
For each ellipse on the left picture, it represents RSS(w0,w1) equals certain constant.

the x is the least squares solution.

then look at the L2 penalty part, which is obviously an circle as is shown in the right picture below. each circle represents L2 penalty equals a certain constant.

the 0 point is the solution for minimum L2 penalty.





then we add the two contour plots together and get the plot for total costs.

when we increase λ, w0 and w1 shrink together to 0

for a specific λ value, some balance between RSS and L2 penalty

(6) visualizing lasso cost

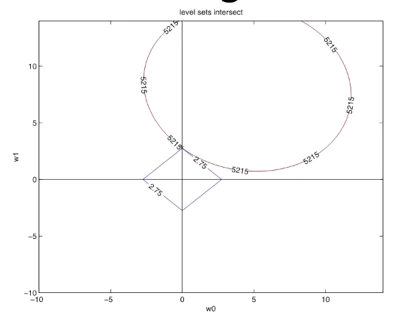
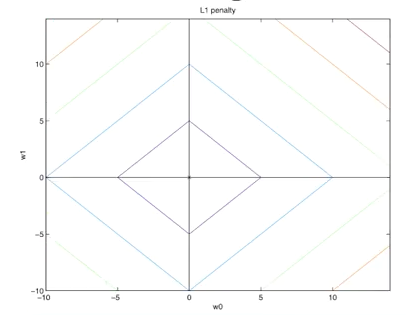
RSS(**w**) + λ||**w**||1 =

RSS contour or lasso is exactly the same for ridge

L1 penalty contour looks like diamond.

to minimize it, w will be set to 0.

then we add the two contours, when we increase λ, w0 first shrink to 0, and then w1, since only at the corner, can we get optimal solution.



(7) Lasso Regression

Refer to the iPython Notebook Overfitting\_Demo\_Ridge\_Lasso in week 4

solver = ‘fista’ - a fancy version of gradient descent

when the penalty strength is really small, we don’t get any sparsity at all (17 non-zero-weight features)

when the penalty gets higher, the solution gets more sparse and the plot gets smoother

(8) how to optimize Lasso solution

In the past, to solve for w-hat, previously took gradient of total cost objective and either:

derived closed-form solution

used in gradient descent algorithm

optimizing the lasso objective:

Lasso total cost = RSS(**w**) + λ||**w**||1

the derivative of |wj| is +1/-1 in different parts, and there is no derivative in the at point 0

→use subgradients instead

but even if we could compute derivative, no closed-form solution, we have to use gradient descent algorithm

→**coordinate descent**

(8) coordinate descent

goal: minimize some function g

often, hard to find minimum for all coordinates, but easy for each coordinate, when keep others fixed

initialize w-hat = 0 (or smartly…)

why not converged:

pick a coordinate j

w-hatj <- find the ω that minimize g(w0, w1, …,ω, wj+1, …, wD)

where the w0, w1, …, comes from previous iteration

then pick another coordinate … loop…

how do we pick next coordinate?

at random(‘random’ or ‘stochatic’), round robin(follow the order), …

differences from gradient descent:

no stepsize to choose

(9) normalizing our features

scale training columns as

this will put each one of our features into the same numeric range

attention: you have to normalize the test dataset as well

(10) coordinate descent for least squares regression (for normalized features):

fix all coordinates w-j and take partial w.r.t. wj

by definition

thus

let :

so for least squares regression:

initialize w-hat = 0

while not converged

for j=0,1,…,D

compute ρj and set w-hatj=ρj

is exactly the measure of correlation between our future j and this residual between the predicted value for all features excluding j.

(11) coordinate descent for lasso (for normalized features):

how to set correlation measure ρj depend on the value of λ:

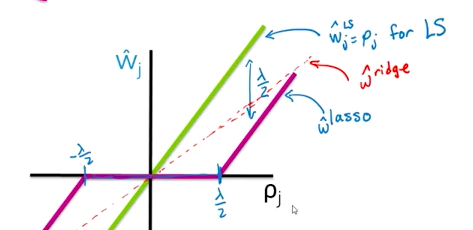
initialize w-hat = 0

while not converged

for j=0,1,…,D

compute

set



(12) assess convergence

when to stop?

we know that for convex problems, we will take smaller and smaller steps

so we can sweep over all coordinates, if the max step we take in the entire cycle is less than a certain threshold, it is considered as converged

(13) other lasso solvers

classically: least angle regress (LARS)

then: coordinate descent algorithm

now:

parallel CD(coordinate descent)

other parallel learning approaches for linear models

parallel stochastic gradient descent (SGD)

parallel independent solutions then averaging

alternating directions method of multipliers (ADMM)

(14) coordinate descent for lasso (unnormalized features)

precompute:

initialize w-hat = 0

while not converged

for j=0,1,…,D

compute

set

(15) how to choose λ

the same as in ridge regression

if sufficient data: separate a validation set

if not sufficient: make cross validation

but both tend to favor less sparse solutions, which is a bit smaller than optimal choice for feature selection

(16) debiasing lasso

Lasso shrinks coefficients relative to LS solution

→ more bias, less variance

can reduce bias as follows:

reduce lasso to select features

then run least squares regression with only selected features

(17) other issues with standard lasso objective

①with group of highly correlated features, lasso tends to select amongst them arbitrarily

often prefer to select all together in practice

②often, empirically ridge has better predictive performance than lasso, but lasso leads to sparser solution

→**elastic net**

aims to address these issues

hybrid between lasso and ridge regression

use both L1 and L2 penalties

(18) interpretation

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

Nearest Neighbors & Kernel Regression

(1) Nonparametric regression

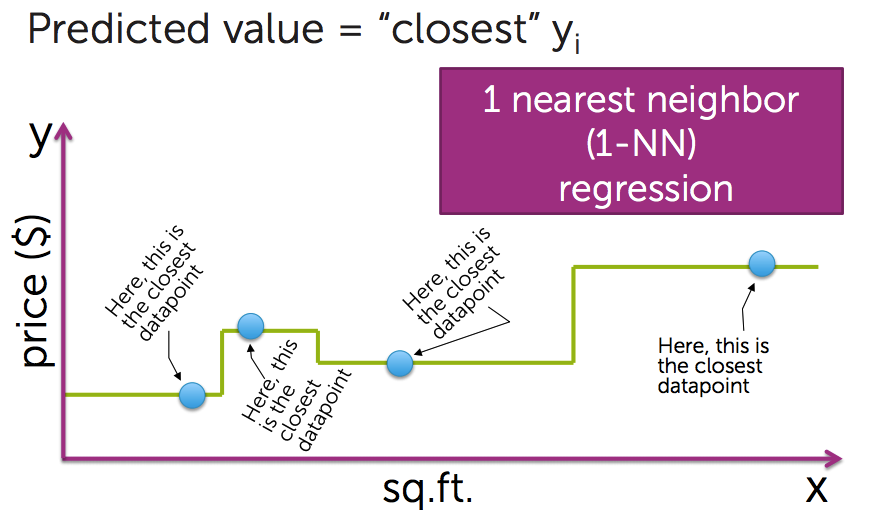
Limitations of parametric regression

we fit the model to all inputs data, but normally a model may fit perfectly for a certain scope of the inputs, but not all

→fitting our function locally to different regions of the input spaces, but meanwhile we don't want to infer ‘structural breaks’

(2) Nearest neighbor regression

for each input, look for the closest observation and its outputted value, and predict your value is exactly equivalent to that value.



procedure:

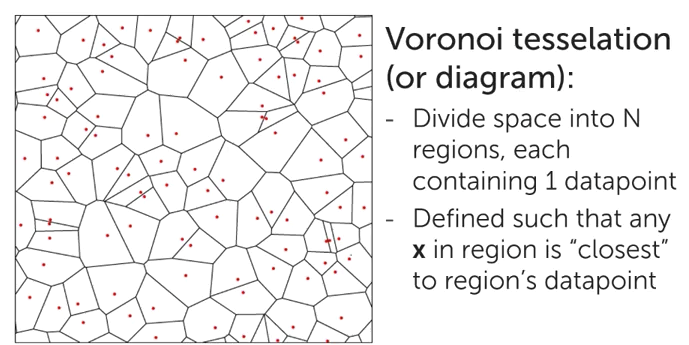
dataset of (house, price) pairs (**x**1, y1), (**x**2, y2), …, (**x**N, yN)

query point: **x**q

find closest **x**i in dataset (key: distance matric refer to (3) below)

predict

In high dimension: Voronoi tesselation



(3) Distance metrics

In 1D, distance(xj, xq) = | xj - xq |

in multiple dimensions:

①weight different dimensions differently

→weighting housing inputs

some inputs are more relevant than others

e.g. bedrooms, bathrooms, sq.rt.living

a1-ad: weight on each inputs, defining relative importance

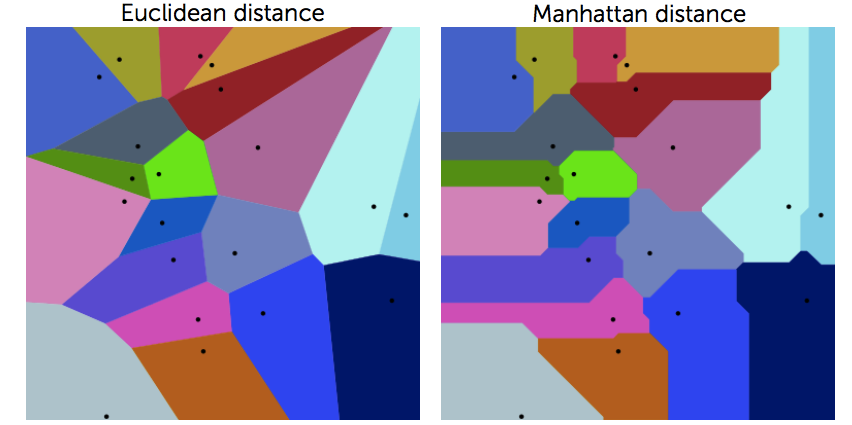
②other distance metrics

each distance metrics leads to different predictive surfaces

Euclidean distance: all weights are 1

Manhattan distance: measuring distance along axis-aligned directions

distance along x direction + distance along y direction



(4) 1-Nearest neighbor algorithm

initialize Dist2NN = infinity, closest house = None

For i=1, 2, …, N

compute dist = distance(i house, query house)

if dist < Dist2NN

set closest house = i house

set Dist2NN = dist

return most similar house i house

this works pretty good when we have dense input space, but sensitive to regions with little data, really sensitive to noise in the data.

(5) K-Nearest neighbors regression

get more ‘comps’

procedure:

dataset of (house, price) pairs (**x**1, y1), (**x**2, y2), …, (**x**N, yN)

query point: **x**q

find k closest **x**i in dataset: (**x**NN1, **x**NN2, …, **x**NNk) (**x**NNk is the farthest)

such that for any xi not in nearest neighbor set,

the distance (**x**i, **x**q) >= distance(**x**NNk, **x**q)

predicted value is the average value of the k nearest neighbors

algorithm:

initialize Dist2kNN = sort(dist1, …, distk), closest house = (house1, …, housek) (first k)

For i=k+1, …, N

compute dist = distance(house i, query house)

if dist < Dist2kNN[k]

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

remove furthest house and shift queue:

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

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

set Dist2kNN [j] = dist

set closest house[j] = house i

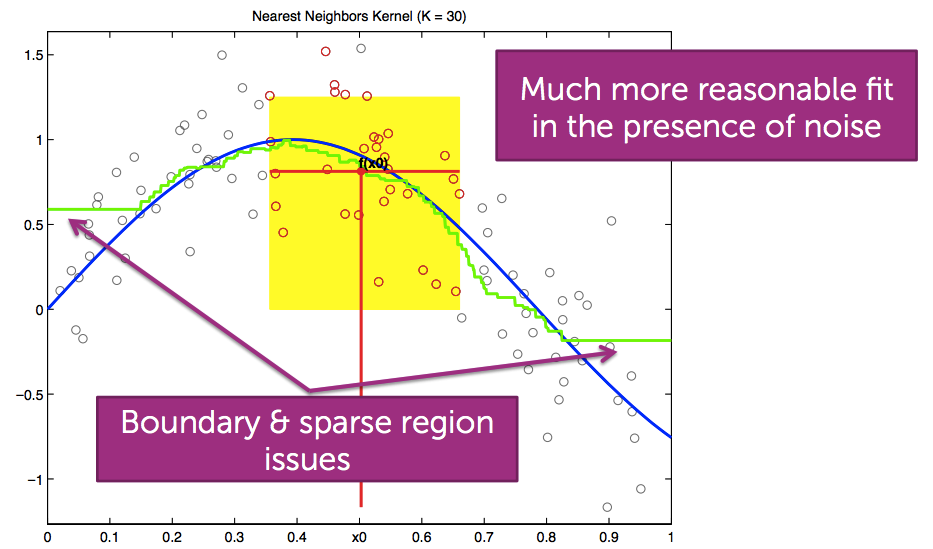
return k most similar houses, and compute the mean

k-NN in practice works more reasonable than 1-NN method in the presence of noise

but at boundary, it will be a straight line

another issue is that k-NN result is discontinuous

there will be a jump in the predicted value of house differs slightly



(6) Weighted K-nearest neighbors

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

predict:

CqNNj is the weights

when distance(**x**NNj, **x**q) is large, we want CqNNj to be small, vice versa

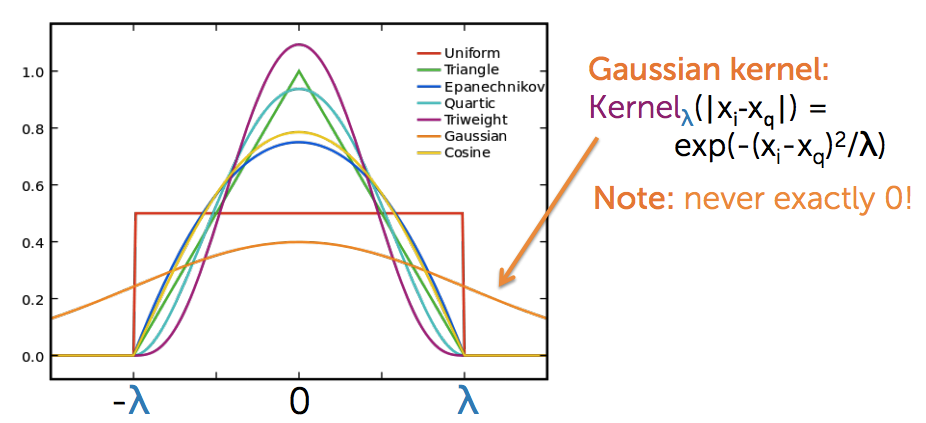
one choice to weights:

CqNNj = 1 / distance((**x**j, **x**q)

another choice: Kernel weights for d=1

define CqNNj = Kernelλ(|xNNj-xq|)

the kernel is defining how the weights are gonna decay, if at all, as a function of the distance between a given point and a query point



(7) Kernel regression

Instead of just weighting NN (selected K nearest neighbors), weight all points,

predict

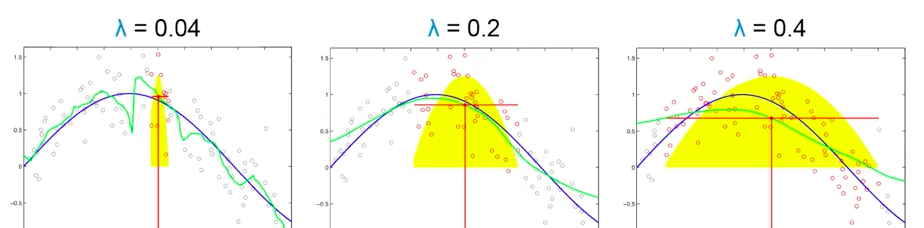
Nadaraya-Warson Kernel weighted average

the predicted output appears a lot smoother than our standard K-NN’s fit

choice of bandwidth

often, choice kernel matters much less than choice of λ

(choice of the kernel changes the pattern of the yellow area, but λ changes size)



λ = 0.04: too small, overfitting (low bias, high variance)

λ = 0.2: just suitable

λ = 0.4: too large, get over smoothing

choosing λ (or k in k-NN):

use cross validation

(8) global fits v.s. local fits

contrasting with global average:

a globally constant fit weights all points equally:

but for kernel regression, we slowly add in some points and let other gradually die off

local fit:

fit a constant function locally at each point- locally weighted averages

fit a line or polynomial locally at each point- locally weighted linear regression

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

(9) nonparametric

K-NN and Kernel regression are both examples of nonparametric regression

general goals of nonparametric:

flexibility

make few assumptions about f(**x**)

complexity can grow with the number of observations N

other choices

splines, trees, locally weighted structured regression models

(10) performance of NN as amount of data grows

limiting behavior of NN:

①Noiseless setting (εi=0):

In the limit of getting an infinite amount of noiseless data, the MES of 1-NN fit goes to 0, since the true error goes to 0 with increase data amount

but for quadratic fit, MSE never gets to 0, even if have infinite data, since true error never goes to 0

②Noisy data setting:

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

but quadratic fit will always have bias

(11) NN and kernel methods for large d or small N

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

the most dimension d you have, the more points N you need to cover the space

need N=O(exp(d)) data points for good performance

so for large d or small N, parametric model become useful

(12) complexity of NN search

Naïve approach: Brute force search

given a query point **x**q

scan through each point **x**1, **x**2, .., **x**N

O(N) distance computations per 1-NN query

O(Nlogk) per k-NN query

what if N is huge: really intensive to do