Feature scaling and normalization continued...

We prepare the features to have dynamic range [-1, all

step 1: "Mean normalization": find the maximum likelihood ML mean for each feature and subtract it for all training instances ... Except the Bias!

$$x^{(k)}_{i,\text{normal}} = x^{(k)}_i - \hat{\mu}_{ML,i}$$

- k^{th} training instances
- feature "i"
- actual training point $x^{(k)}$ feature (i)

for instances $k = 1, \dots N$

for feature $i = 1, \dots d$ (not bias!)

Note: This makes the average value of all features equal to zero:

$$E_{x|w_i}\left[x^{(k)}_{i,\text{normal}}\right] = E_{x|w_i}\left[x^{(k)}_i - \hat{\mu}_{ML,i}\right]$$

$$= E_{x|w_i}\left[x^{(k)}_i\right] - \underbrace{\hat{\mu}_{ML,i}}_{= E_{x|w_i}\left[x^{(k)}_i\right]}$$

since ML mean is unbiased

$$= 0$$

Step 2: "Feature Scaling": find the ML variance, take its square root to get the standard deviation and divide the features.

$$x^{(k)}_{i,\text{scaled, new}} = \frac{x^{(k)}_{i,\text{normal}}}{\sqrt{\hat{\sigma}^2_{ML,i}}} = \frac{x^{(k)}_i - \hat{\mu}_{ML,i}}{\sqrt{\hat{\sigma}^2_{ML,i}}}$$

for instances $k = 1, \dots, N$
for features $i = 1, \dots d$
(not Bias!)

mean = 0
variance = 1

new scaled normalized features
all fall more or less in interval $[-1, 1]$

NOTE: If you have enough training data, then what should happen is that weights for useless attribute (like pupil diameter) will be zero or near zero.

↳ we can ditch useless features if their weights are zero

What if N, my number of training data, is small?

And if d is small?

↳ then $\frac{dJ}{dw}$, set that equal to zero, then solve for $\vec{w}_{MMSE}$

$\Rightarrow \vec{w}_{MMSE} = (X^T X)^{-1} X^T \vec{p}$

you get

matrix of training feature

vector of training labels

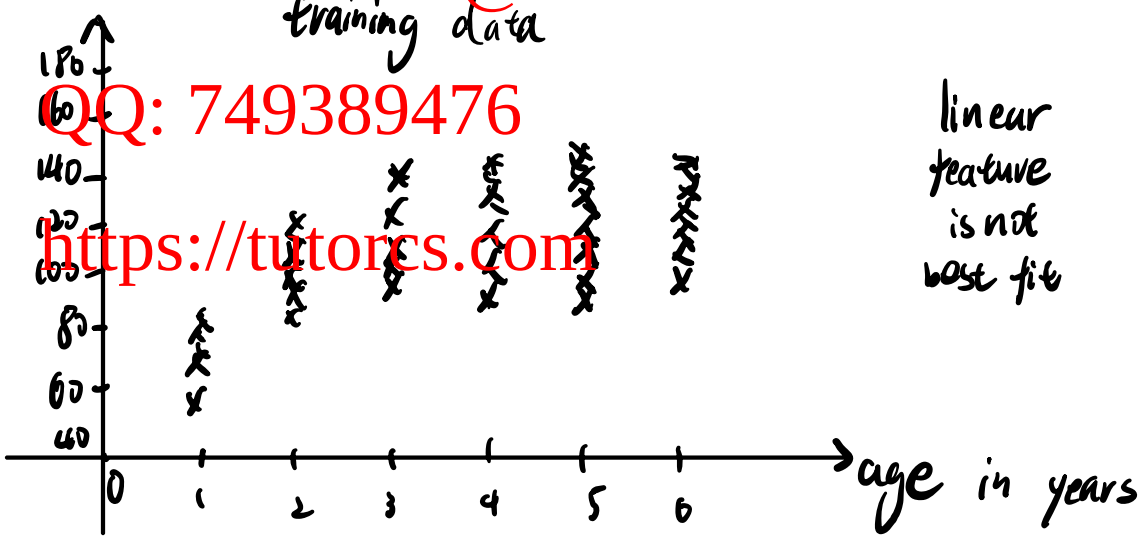
$X^T X$ is big matrix

$\Rightarrow$ inverting it is expensive

Non-Linear Polynomial Regression

Ex] Say we have 1 feature (age of the fish in years) to predict its length (in mm)

(Not classification, but gradient) training data



we could try $\vec{x} = \begin{bmatrix} 1 \\ x_1 \\ x_1^2 \end{bmatrix} \Rightarrow$ prediction $= w^T \vec{x} = w_0 + w_1 x_1 + w_2 x_1^2$

or not polynomial: $\vec{x} = \begin{bmatrix} 1 \\ x_1 \\ \sqrt{x_1} \end{bmatrix}$

NOTE: when you use higher order polynomial, the dynamic range gets more pronounced

Ex: if $x_1 = 50 \Rightarrow x_1^2 = 50^2$ bigger

if $x_1 = 0.1 \Rightarrow x_1^2 = 0.1^2$ smaller

$\Rightarrow$ normalizing and scaling is important

How do we know model to use?

To test your performance:

use 60% - 70% of data to train with your cost J_{train} to get $\vec{w}_{use}$

use 20% - 15% of data to test your model and test your design performance

why? Because in crossvalidation you may overfit your model to that crossvalidation data.

You will see that $J_{use,crossvalidation}$ is very small, but $J_{use,test}$ is big

this is the one we most care about

use 20% - 15% for cross-validation to test model

$J_{use,crossvalidation}$ crossvalidation to test model

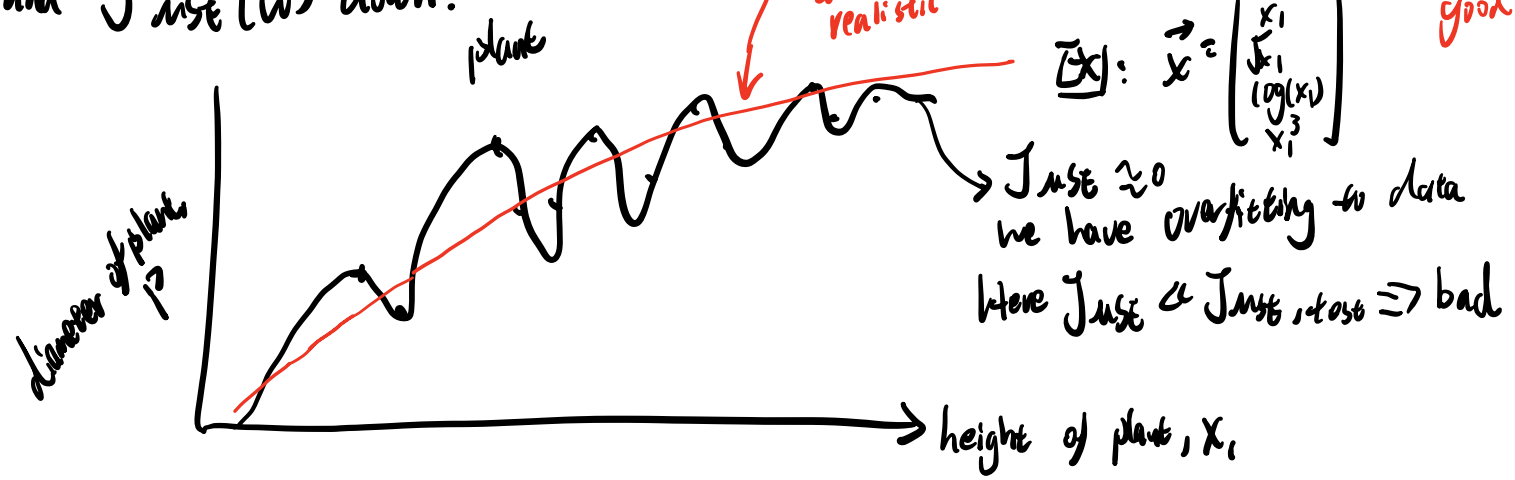
use 20% - 15% to test data to set it design worked

$J_{use,test}$ test data to set it design worked

Avoid overfiting: Regularization

If your $J_{use}(\vec{w})$ is high, you can invest a lot of time experimenting with high order polynomials that bring $J_{use,crossvalidation}$ and $J_{use}(\vec{w})$ down.

J_{use} is bigger but more realistic $\Rightarrow$ better predictor for test data $\Rightarrow J_{use} \approx J_{use,test}$ good



Ex: $\vec{x} = \begin{bmatrix} x_1 \\ \sqrt{x_1} \\ \log(x_1) \\ x_1^3 \end{bmatrix}$

$J_{use} \approx 0$ we have overfitting to data

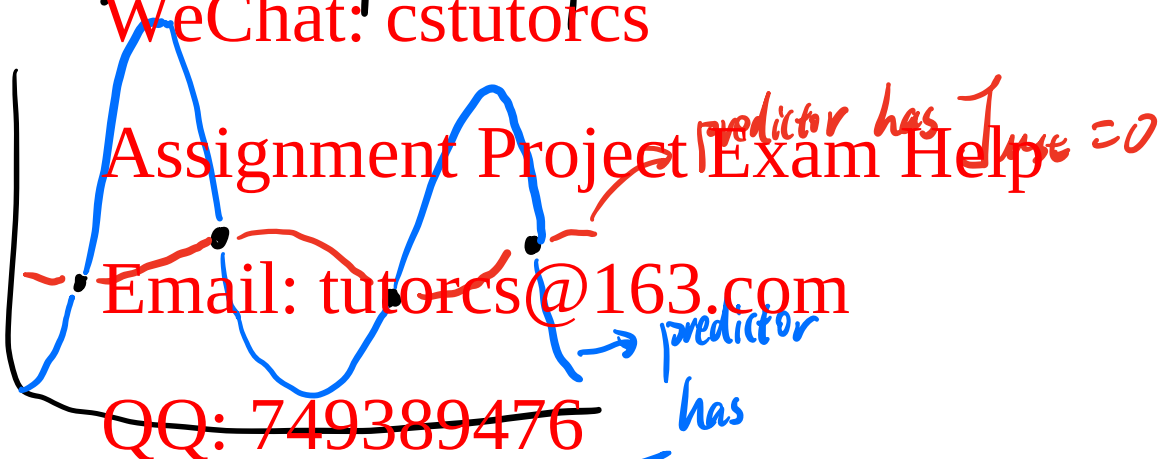
Here $J_{use} \ll J_{use,test} \Rightarrow$ bad

When you have many feature, it is difficult to know how to simplify the model to reduce overfiting.

Step1: discord feature that are redundant.

step 2: "REGU" the weighe vector", $\vec{w}$

avoid



Both are 4th order polynomials but the blue one has bigger coefficients (in normal)

Regularization means "punishing" choice of $\vec{w}$ that has large weighe values.

We do this by including it in the cost function.

$$J_{MSE-W-regwlization}(\vec{w}) = \frac{1}{N}\left\{\sum_{k=1}^{N}\left(\underbrace{h_{\vec{w}}(\vec{x}^{(k)})}_{\text{prediction } \vec{w}^T\vec{x}^{(k)}} - y_k\right)^2 + \lambda\sum_{j=1}^{d} w_j^2\right\}$$

$$= J_{MSE} + \frac{\lambda}{N} \| \overrightarrow{\text{weight without bias}} \|^2$$

Def: $\lambda \geq 0$ "Regularization parameter"

To find best $\vec{w}$ 2cd case:

$$\frac{\partial J_{MSE\text{-}w\text{-}reg}}{\partial w_i} = \frac{2}{N}\sum_{k=1}^{N} (w^T x^{(k)} - p_k) \cdot x_i^{(k)} + \frac{\lambda}{N} 2 w_i \quad i=1,\dots d$$

so $\vec{w}$ that minimizes $J_{MSE\text{-}w\text{-}regularization}$ (with regularization) is

$$\vec{w}_{k+1} = \vec{w}_k - \eta \begin{bmatrix} \frac{2}{N}\sum_{j=1}^{N} (w_k^T x^{(j)} - p_j) \\ \frac{2}{N}\sum_{j=1}^{N} (w_k^T x^{(j)} - p_j) \cdot x_1^{(j)} + \frac{2\lambda}{N} w_1 \\ \frac{2}{N}\sum_{j=1}^{N} (w_k^T x^{(j)} - p_j) \cdot x_d^{(j)} + \frac{2\lambda}{N} w_d \end{bmatrix} \leftarrow \text{bias term}$$

$$= \begin{bmatrix} w_0(1-0) \\ w_1(1-\frac{2\lambda\eta}{N}) \\ \vdots \\ w_d(1-\frac{2\lambda\eta}{N}) \end{bmatrix} - \eta \begin{bmatrix} \frac{2}{N}\sum_{j=1}^{N} (w_k^T x^{(j)} - p_j) \\ \frac{2}{N}\sum_{j=1}^{N} (w_k^T x^{(j)} - p_j) x_1^{(k)} \\ \frac{2}{N}\sum_{j=1}^{N} (w_k^T x^{(j)} - p_j) x_d^{(k)} \end{bmatrix}$$

If we wanted a closed form solution with Regularization, (for small d), then

because:

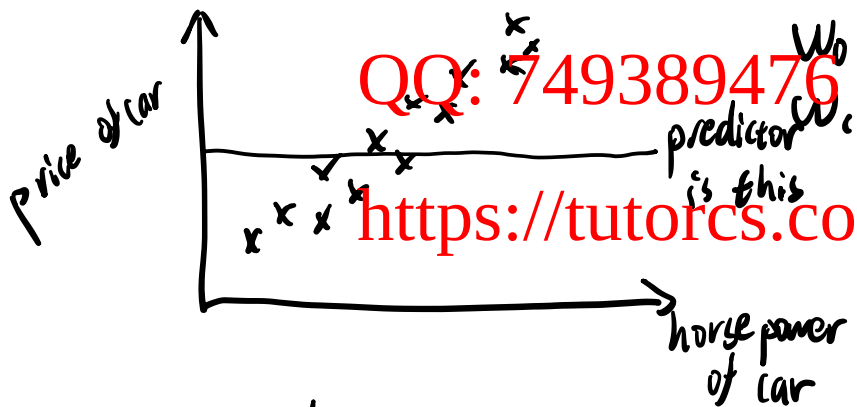
$$\vec{w}_{MSE\text{-}w\text{-}regularization} = \left(X^T X + \begin{bmatrix} 0 & 0 & \cdots & 0 \\ 0 & \lambda & & \\ \vdots & & \ddots & \\ 0 & & & \lambda \end{bmatrix}\right)^{-1} X^T \vec{p}$$

HIGH BIAS VS HIGH VARIANCE TRADE-OFF

Ex] car price prediction → say after mean normalization, scaling, and tuning λ (regularization parameter), your J_{MSE} remains high even after your gradient descent converge

Typically: what is that you are either underfitting or overfitting

Def] A model suffers from HIGH Bias if it underfits the data:



$w_0 = 8.5$ $w_1 = 0$ $\vec{w} = \begin{bmatrix} 8.5 \\ 0 \end{bmatrix}$

$\vec{w}^T \vec{x}$

This model underfits yielding bad J_{MSE}, train and J_{MSE}, test

This example ignores horsepower

NOTE: collecting more data will not solve a high bias problem

How do we solve high bias (overfitting)?

- add more features ($x_1^2, x_1^3, \sqrt{x_1}, x_1 \cdot x_2, x_1 \cdot x_2 \cdot x_3$, etc...)

- see if you add more attributes
- if you are underfitting, then you may be regularizing too much, so you don't need to be doing that

to select λ

Step1: select a λ that scales over orders of magnitude

$\lambda \in \{0, 0.001, 0.1, 1\}$

Def: A model suffers from "High Variance" if it overfits the data

To detect high variance, you'll see $J_{MSE, train}$ is small and $J_{MSE, test}$ is big

How to solve high Variance (overfitting)?

- collect more training data (make N bigger).
  (Do NOT use your test data).
- ditch some non-linear attributes (x_1^2, x_1^3, etc)

- if you are overfitting, that you are not regularizing enough, so make λ bigger

BIAS - VARIANCE TRADE-OFF

Def] "generalization" = "expected test error"
= measure of how accurately an algorithm is able to predict outcomes for the test data (not used in training)

- $D = \{(\vec{x}, y_1), (\vec{x}_2, y_2), \ldots, (\vec{x}_n, y_n)\}$

= set of training instances with features and labels

note: for any feature vector, we may have more than one label

Ex] $\vec{x}_i$ has label $P_i = 7$
$\vec{x}_j$ has label $P_j = 7$
} there is an underline conditional PDF $f(y|x)$

Ex] two plants may have exactly the same height but different widths.

The most likely or expected width is $E_{y|x}[y]$

Hence: there is a joint distribution $f(x, y)$

- $h_D(x)$ = our imperfect prediction for y

  = prediction hypothesis (say for linear regression or for logistic regression), as a function of vector x, and h was trained from all x's in our limited set D.

  $\Rightarrow$ h's must also be random

- $\bar{h}(x) = E_D[h_D(x)] = \int h_D(x) f(D)\, dD$

  = expected classifier or predictor using same imperfect algorithm but with all the data in the world

Def We can define the generalized error of our imperfect algorithm:

$$E_{x,y,D\text{-trained}}[(h_D(x)-y)^2] = \int_{-\infty}^{\infty}\int\int (h_D(x)-y)^2 f(x,y,D)\, dy\, d\vec{x}\, dD$$

- $\vec{y}(x)$ = MAP estimate of label for x, which is the very best estimate in the world, but it requires us to know the underlying pdf of y given x: $f(y|x)$

$$= E_{y|x}[y] = \int_{-\infty}^{\infty} y f(y|x)\, dy = \text{MAP estimator} = \text{expected label}$$

FACT: you can show that the generalized error has three components that shows trade-off between bias and variance

Track #1: add and subtract expected classifier

$$E_{x,y,D\text{-trained}}[\{y-h_D(x)\}^2] = E_{x,y,D\text{-trained}}[(\{h_D(x)-\bar{h}(x)\}+\{\bar{h}(x)-y\})^2]$$

= algebra, probabilities, cross-term is zero

$$= E_{x,D\text{-trained}}[(h_D(x)-\bar{h}(x))^2] + E_{x,y}[(\bar{h}(x)-y)^2]$$

exactly definition of variance of $h_D(x)$

The classifier or predictors variance shows how sensitive your classifier/predictor is to changing from one set D to one other set D'

How much are you overfitting your h to your predictor D?

How far off are we from the average classifier that uses all the data in the world and not our limited D?

Next: Trick #2: add and subtract the MAP estimate $\bar{y}(x)$ of label for feature x:

$\mathbb{E}_{x,y,D\text{-train}}[(h_D(x)-y)^2] =$ Var of $h_D(x)$

$\{\bar{h}(x)-\bar{y}(x)\}+\{\bar{y}(x)-y\})^2]$

... algebra, probability, cross term goes to zero

$= \text{Var of } h_D(x) + \mathbb{E}_x[(\bar{h}(x)-\bar{y}(x))^2] + \mathbb{E}_{x,y}[(\bar{y}(x)-y)^2]$

"bias squared of estimator"

"noise of estimator"