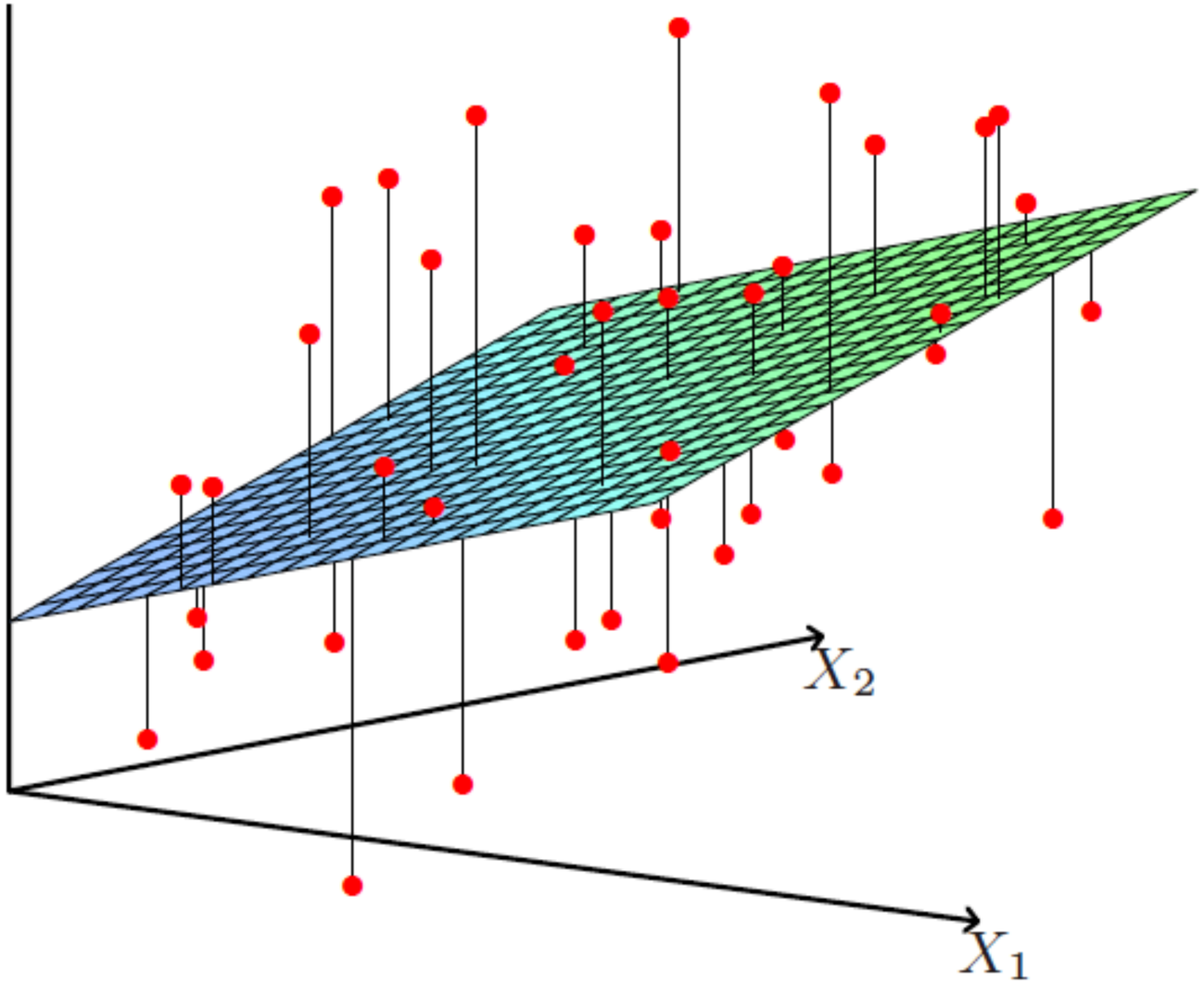
syde675 pattern recognition

Linear regression

J.Zelek 2022

Linear regression



regression

- What is regression?
- How does regression fit into the ML framework?

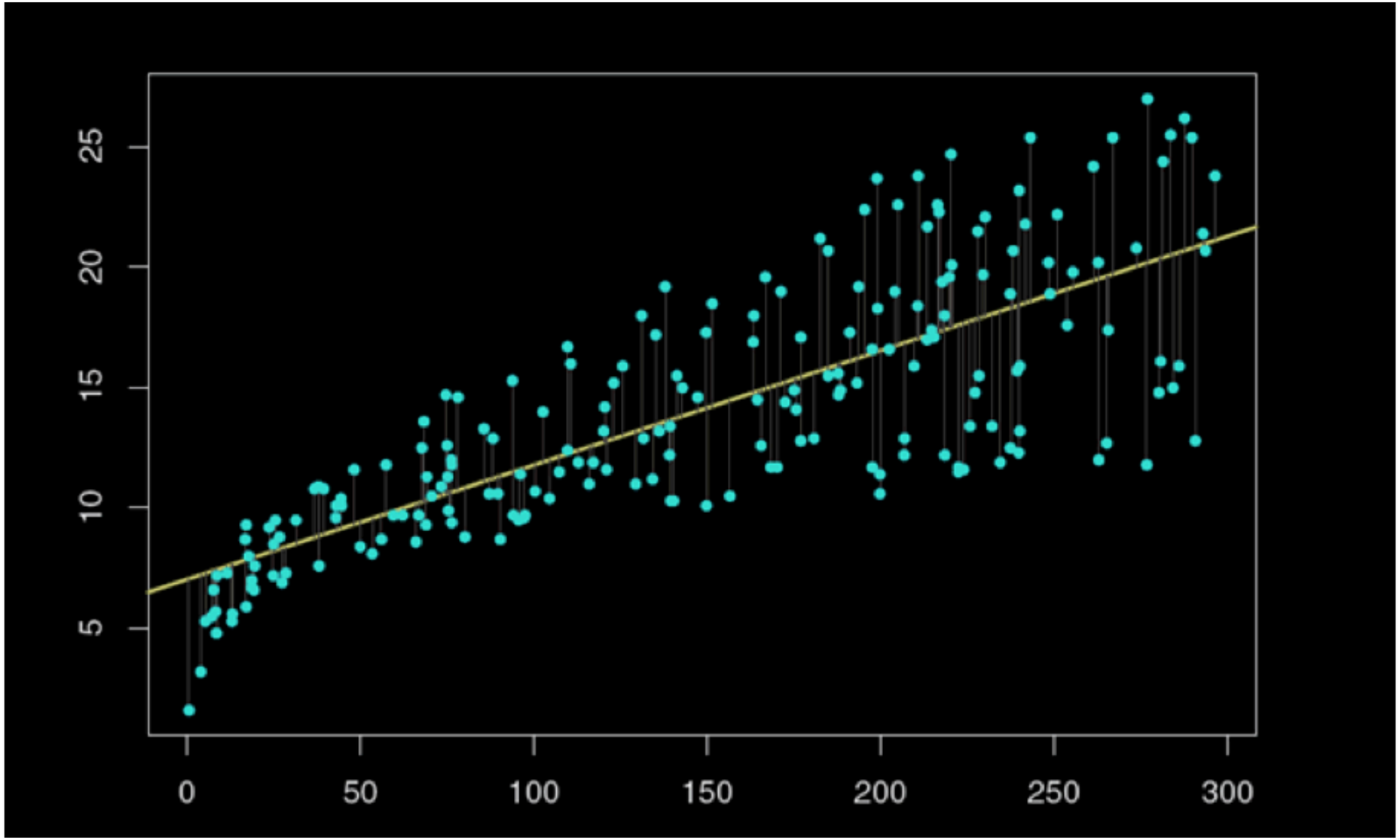
regression vs classification

- Classification: given point x, predict class (often binary)
- Regression: given point x, predict a numerical value
- Classification gives a discrete prediction, whereas regression gives us a quantitative prediction, usually on a continuous scale

Feature engineering

- What is x ?
- What is y ?
- $\phi(x)$ can be treated like x

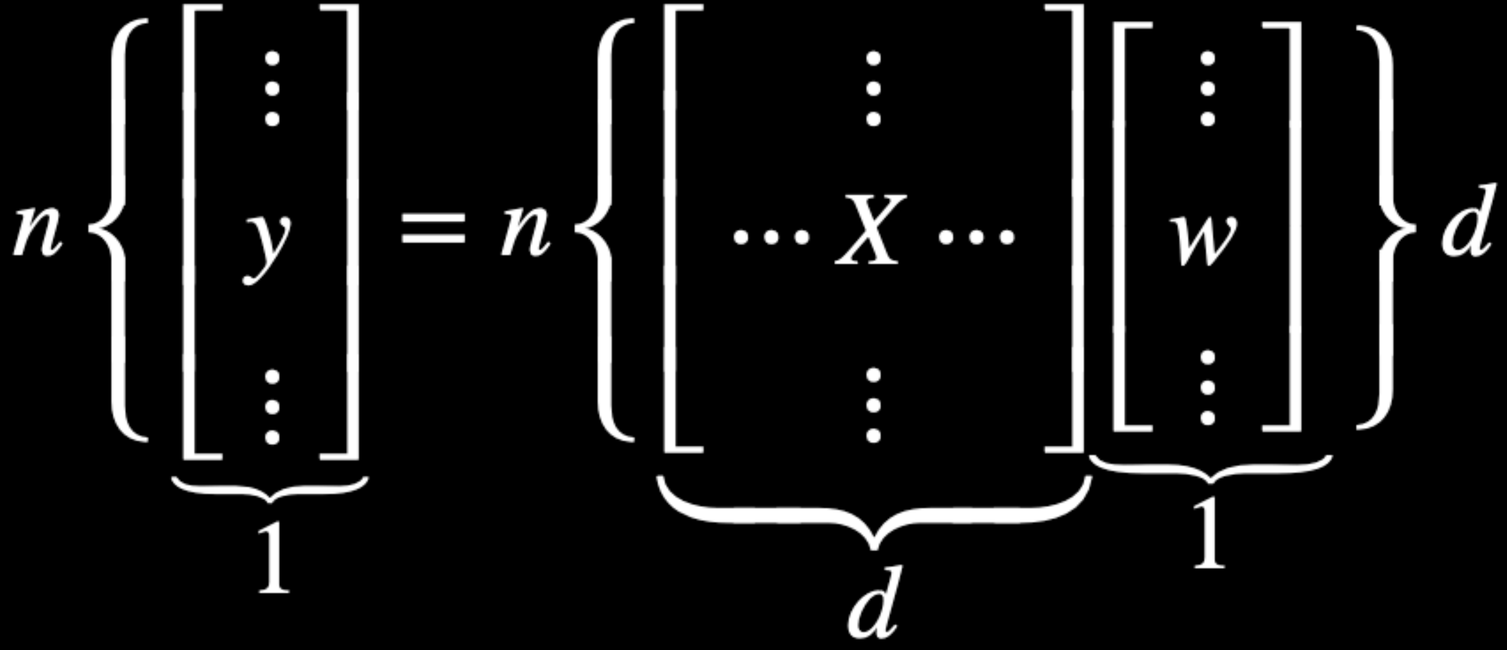
Linear regression intuition



Linear regression setup

n = number of data points

d = dimension of each data point



Linear regression

- Goal in machine learning is to “extract a relationship from data”
- Relationship is a function: $y = f(x)$ where $y \in \mathbb{R}$ is some quantity predicted from an input $x \in \mathbb{R}^d$
- The true relationship f is unknown to us & our aim is to recover it as well as we can from the data
- Our end product is a function $\hat{y} = h(x)$ is called the “hypothesis” that should approximate f
- We assume that we have access to a dataset $\mathbb{D} = \{(x_i, y_i)\}_{i=1}^n$ where each (x_i, y_i) is an example of the input-output mapping to be learned
- Since learning arbitrary functions is intractable, we restrict ourselves to some hypothesis class $\mathbb{H}$ of allowable functions

Linear regression

- In more detail, we typically employ a “parametric model” meaning that there is some finite-dimensional vector $w \in \mathbb{R}^d$, the elements of which are known as parameters or weights that control the behaviour of the function
- $h_w(x) = g(x, w)$ for some other function g
- The hypothesis class is then the set of all functions induced by the possible choices of the parameters w , $\mathbb{H} = \{h_w \,|\, w \in \mathbb{R}^d\}$
- After designating a cost function L , which measures how poorly the predictions $\hat{y}$ of the hypothesis match the true output y , we can proceed to search for the parameters that best fit the data by minimizing this function
- $w^* = \arg_w \min L(w)$

Ordinary Least Squares (1)

- OLS (Ordinary Least Squares) is one of the simplest regression problems but it is well understood & practically useful
- It is a linear regression problem, which means that we take h_w to be of the form $h_w = x^T w$
- We want: $y_i \approx \hat{y}_i = h_w(x_i) = x_i^T w$ for each $i = 1,\ldots,n$
- The set of equations can be written in matrix form as:

$$\underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}}_{\mathbf{y}} \approx \underbrace{\begin{bmatrix} \mathbf{x}_1{}^\top \\ \vdots \\ \mathbf{x}_n{}^\top \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}}_{\mathbf{w}}$$

Ordinary Least Squares (2)

- Matrix $X \in \mathbb{R}^{nxd}$ has the input datapoint x_i as its i'th row. This matrix is called the "design matrix". Usually $n \geq d$, meaning that there are more datapoints than measurements.
- There is no exact solution to the equation $y = Xw$
- Approximate solution found by minimizing the sum (the mean) of squared errors: $L(w) = \sum_{i=1}^{n} (x_i^T w - y_i)^2 = \min_W ||Xw - y||_2^2$
- Now we have an optimization problem

$$\underbrace{\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}}_{\mathbf{y}} \approx \underbrace{\begin{bmatrix} \mathbf{x}_1{}^\top \\ \vdots \\ \mathbf{x}_n{}^\top \end{bmatrix}}_{\mathbf{X}} \underbrace{\begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}}_{\mathbf{w}}$$

Vector calculus

- If $L : \mathbb{R}^d \to \mathbb{R}$ is continuously differentiable, then any local optimum w^* satisfies $\nabla L(w^*) = 0$, in the OLS case,

$$
\begin{aligned}
L(\mathbf{w}) &= \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 \\
&= (\mathbf{X}\mathbf{w} - \mathbf{y})^\top (\mathbf{X}\mathbf{w} - \mathbf{y}) \\
&= (\mathbf{X}\mathbf{w})^\top \mathbf{X}\mathbf{w} - (\mathbf{X}\mathbf{w})^\top \mathbf{y} - \mathbf{y}^\top \mathbf{X}\mathbf{w} + \mathbf{y}^\top \mathbf{y} \\
&= \mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} - 2\mathbf{w}^\top \mathbf{X}^\top \mathbf{y} + \mathbf{y}^\top \mathbf{y}
\end{aligned}
$$

- Using the following results from matrix calculus

$$
\begin{aligned}
\nabla_{\mathbf{x}}(\mathbf{a}^\top \mathbf{x}) &= \mathbf{a} \\
\nabla_{\mathbf{x}}(\mathbf{x}^\top \mathbf{A}\mathbf{x}) &= (\mathbf{A} + \mathbf{A}^\top)\mathbf{x}
\end{aligned}
$$

Vector calculus (2)

- The gradient of L is easily seen to be

$$\begin{aligned}\nabla L(\mathbf{w}) &= \nabla_{\mathbf{w}}(\mathbf{w}^{\mathsf{T}}\mathbf{X}^{\mathsf{T}}\mathbf{X}\mathbf{w} - 2\mathbf{w}^{\mathsf{T}}\mathbf{X}^{\mathsf{T}}\mathbf{y} + \mathbf{y}^{\mathsf{T}}\mathbf{y}) \\ &= \nabla_{\mathbf{w}}(\mathbf{w}^{\mathsf{T}}\mathbf{X}^{\mathsf{T}}\mathbf{X}\mathbf{w}) - 2\nabla_{\mathbf{w}}(\mathbf{w}^{\mathsf{T}}\mathbf{X}^{\mathsf{T}}\mathbf{y}) + \underbrace{\nabla_{\mathbf{w}}(\mathbf{y}^{\mathsf{T}}\mathbf{y})}_{\mathbf{0}} \\ &= 2\mathbf{X}^{\mathsf{T}}\mathbf{X}\mathbf{w} - 2\mathbf{X}^{\mathsf{T}}\mathbf{y}\end{aligned}$$

- Where in the last line we have used the symmetry of $X^T X$ to simplify $X^T X + (X^T X)T = 2X^T X$. Setting the gradient to 0, we conclude that any optimum w^*_{OLS} satisfies $X^T X w^*_{OLS} = X^T y$
- If X is full rank, then $X^T X$ is as well (assuming $n \geq d$) so we can solve for the unique solution: $w^*_{OLS} = (X^T X)^{-1} X^T y$

Vector calculus (3)

Note: Although we write $(\mathbf{X}^\top\mathbf{X})^{-1}$, in practice one would not actually compute the inverse; it is more numerically stable to solve the linear system of equations above (e.g. with Gaussian elimination).

In this derivation we have used the condition $\nabla L(\mathbf{w}^*) = \mathbf{0}$, which is a *necessary* but not *sufficient* condition for optimality. We found a critical point, but in general such a point could be a local minimum, a local maximum, or a saddle point. Fortunately, in this case the objective function is **convex**, which implies that any critical point is indeed a global minimum. To show that L is convex, it suffices to compute the **Hessian** of L, which in this case is

$$\nabla^2 L(\mathbf{w}) = 2\mathbf{X}^\top\mathbf{X}$$

and show that this is positive semi-definite:

$$\forall \mathbf{w},\ \mathbf{w}^\top(2\mathbf{X}^\top\mathbf{X})\mathbf{w} = 2(\mathbf{X}\mathbf{w})^\top\mathbf{X}\mathbf{w} = 2\|\mathbf{X}\mathbf{w}\|_2^2 \geq 0$$

Approach 2: Orthogonal projection (1)

There is also a linear algebraic way to arrive at the same solution: orthogonal projections.

Recall that if V is an inner product space and S a subspace of V, then any $\mathbf{v} \in V$ can be decomposed uniquely in the form

$$\mathbf{v} = \mathbf{v}_S + \mathbf{v}_\perp$$

where $\mathbf{v}_S \in S$ and $\mathbf{v}_\perp \in S^\perp$. Here $S^\perp$ is the orthogonal complement of S, i.e. the set of vectors that are perpendicular to every vector in S.

The **orthogonal projection** onto S, denoted P_S, is the linear operator that maps $\mathbf{v}$ to $\mathbf{v}_S$ in the decomposition above. An important property of the orthogonal projection is that

$$\|\mathbf{v} - P_S\mathbf{v}\| \leq \|\mathbf{v} - \mathbf{s}\|$$

for all $\mathbf{s} \in S$, with equality if and only if $\mathbf{s} = P_s\mathbf{v}$. That is,

$$P_S\mathbf{v} = \arg\min_{\mathbf{s}\in S} \|\mathbf{v} - \mathbf{s}\|$$

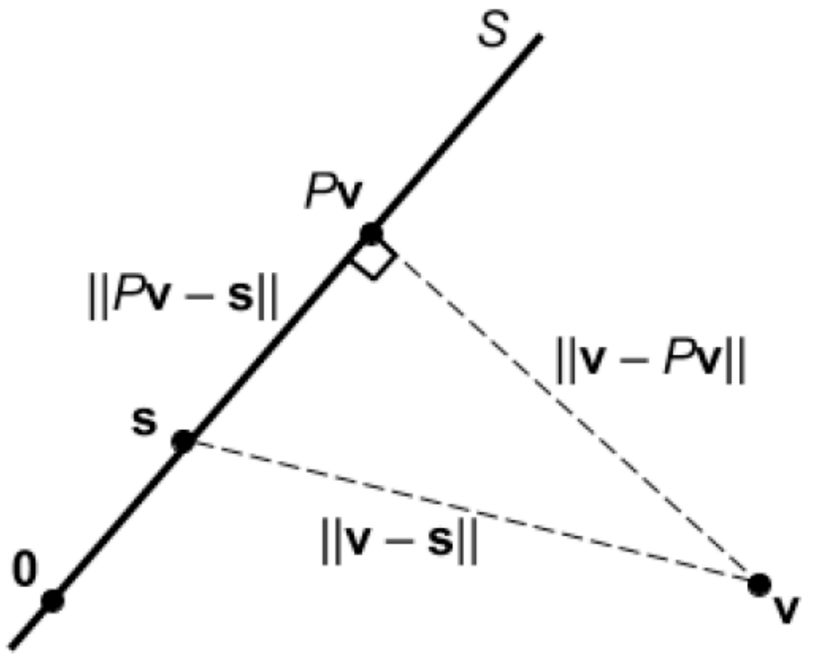
Approach 2: Orthogonal projection (2)

Proof. By the Pythagorean theorem,

$$\|\mathbf{v}-\mathbf{s}\|^2 = \|\underbrace{\mathbf{v}-P_S\mathbf{v}}_{\in S^\perp} + \underbrace{P_S\mathbf{v}-\mathbf{s}}_{\in S}\|^2 = \|\mathbf{v}-P_S\mathbf{v}\|^2 + \|P_S\mathbf{v}-\mathbf{s}\|^2 \geq \|\mathbf{v}-P_S\mathbf{v}\|^2$$

with equality holding if and only if $\|P_S\mathbf{v}-\mathbf{s}\|^2 = 0$, i.e. $\mathbf{s} = P_S\mathbf{v}$. Taking square roots on both sides gives $\|\mathbf{v}-\mathbf{s}\| \geq \|\mathbf{v}-P_S\mathbf{v}\|$ as claimed (since norms are nonnegative). □

Here is a visual representation of the argument above:



Approach 2: Orthogonal projection (3)

In the OLS case,

$$\mathbf{w}^*_{\text{OLS}} = \arg\min_{\mathbf{w}} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

But observe that the set of vectors that can be written $\mathbf{X}\mathbf{w}$ for some $\mathbf{w} \in \mathbb{R}^d$ is precisely the range of $\mathbf{X}$, which we know to be a subspace of $\mathbb{R}^n$, so

$$\min_{\mathbf{z} \in \text{range}(\mathbf{X})} \|\mathbf{z} - \mathbf{y}\|_2^2 = \min_{\mathbf{w} \in \mathbb{R}^d} \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2$$

By pattern matching with the earlier optimality statement about P_S, we observe that $P_{\text{range}(\mathbf{X})}\mathbf{y} = \mathbf{X}\mathbf{w}^*_{\text{OLS}}$, where $\mathbf{w}^*_{\text{OLS}}$ is any optimum for the right-hand side. The projected point $\mathbf{X}\mathbf{w}^*_{\text{OLS}}$ is always unique, but if $\mathbf{X}$ is full rank (again assuming $n \geq d$), then the optimum $\mathbf{w}^*_{\text{OLS}}$ is also unique (as expected). This is because $\mathbf{X}$ being full rank means that the columns of $\mathbf{X}$ are linearly independent, in which case there is a one-to-one correspondence between $\mathbf{w}$ and $\mathbf{X}\mathbf{w}$.

To solve for $\mathbf{w}^*_{\text{OLS}}$, we need the following fact[1]:

$$\text{null}(\mathbf{X}^\top) = \text{range}(\mathbf{X})^\perp$$

Since we are projecting onto range($\mathbf{X}$), the orthogonality condition for optimality is that $\mathbf{y} - P\mathbf{y} \perp$ range($\mathbf{X}$), i.e. $\mathbf{y} - \mathbf{X}\mathbf{w}^*_{\text{OLS}} \in \text{null}(\mathbf{X}^\top)$. This leads to the equation

$$\mathbf{X}^\top(\mathbf{y} - \mathbf{X}\mathbf{w}^*_{\text{OLS}}) = \mathbf{0}$$

which is equivalent to

$$\mathbf{X}^\top\mathbf{X}\mathbf{w}^*_{\text{OLS}} = \mathbf{X}^\top\mathbf{y}$$

as before.

OLS revisited: data & application

$$y \approx Xw$$

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} \approx \begin{bmatrix} \vec{x}_1^T \\ \vdots \\ \vec{x}_n^T \end{bmatrix} \cdot \begin{bmatrix} w_1 \\ \vdots \\ w_d \end{bmatrix}$$

Goal: find the best w

OLS revisited: model

$$y \approx Xw$$

$$\hat{y} = Xw$$

OLS revisited: optimization problem

$$y \approx Xw$$

$$\min_{w} ||Xw - y||_2^2$$

Goal: find the best w

OLS revisited: optimization algorithm

$$\min_{w} ||Xw - y||_2^2$$

$$\hat{w} = \left(X^T X\right)^{-1} X^T y$$

OLS revisited: optimization algorithm

$$\min_{w} ||Xw - y||_2^2$$

$$X^T X \hat{w} = X^T y$$

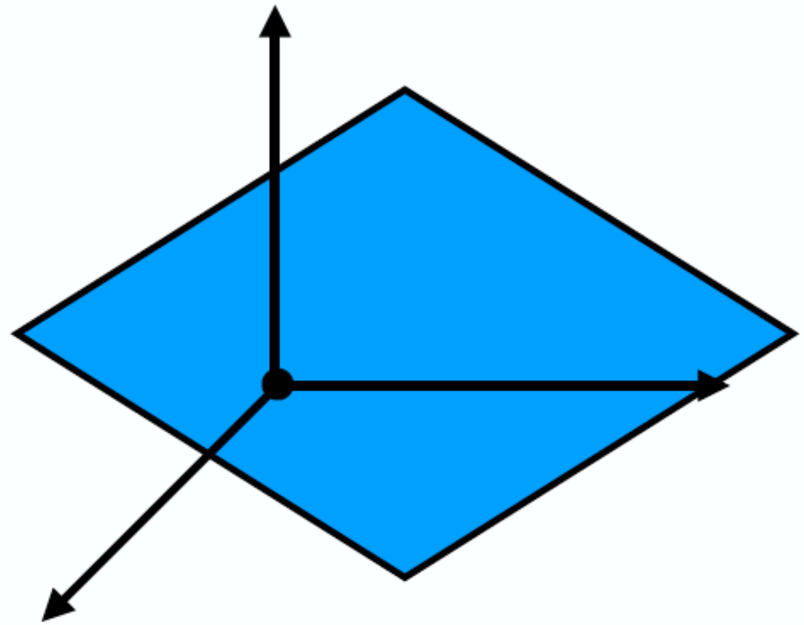
OLS revisited: why are normal eqns the soln?

$$X = \begin{bmatrix} \vec{x}_1^T \\ \vdots \\ \vec{x}_n^T \end{bmatrix} = \begin{bmatrix} \vec{x}^1 & \cdots & \vec{x}^d \end{bmatrix} = \begin{bmatrix} x_1^1 & \cdots & x_1^d \\ \\ \vdots & & \\ \\ \vdots & & \\ \\ x_n^1 & \cdots & x_n^d \end{bmatrix} \qquad n >> d$$

$$\hat{w} = \left(X^T X\right)^{-1} X^T y$$

OLS revisited: why are normal eqns the soln?

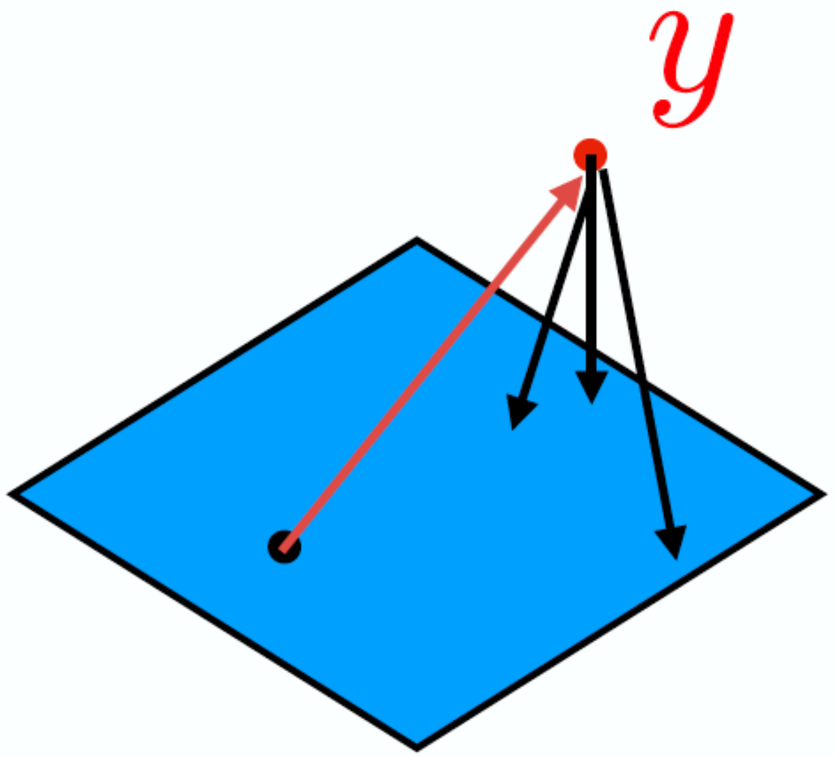
$$Xw \in \mathrm{col}(X) \subset \mathbb{R}^n$$



$$\hat{w} = \left(X^T X\right)^{-1} X^T y$$

OLS revisited: why are normal eqns the soln?

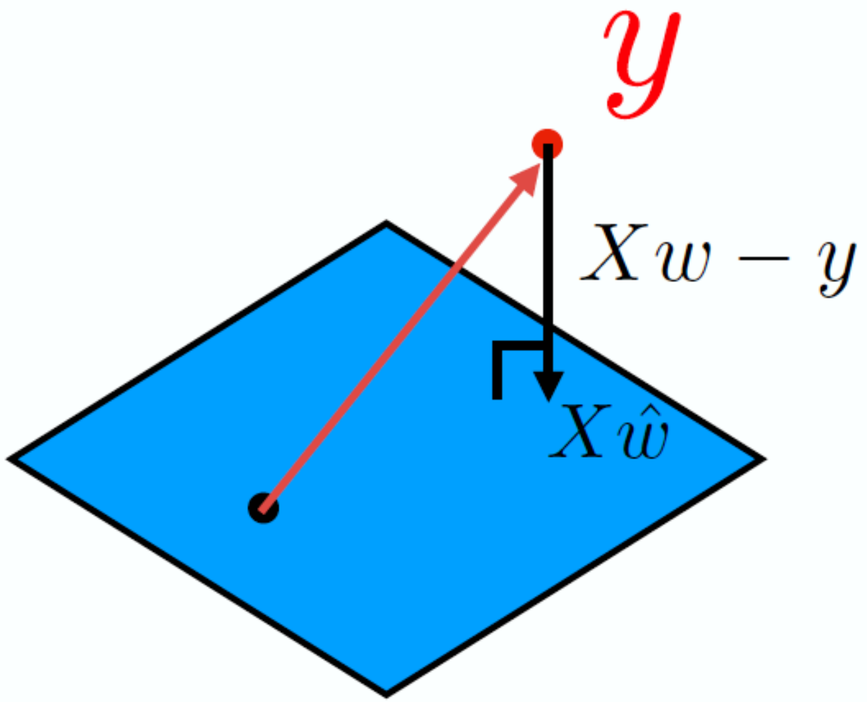
$$Xw - y$$



$$\hat{w} = \left(X^T X\right)^{-1} X^T y$$

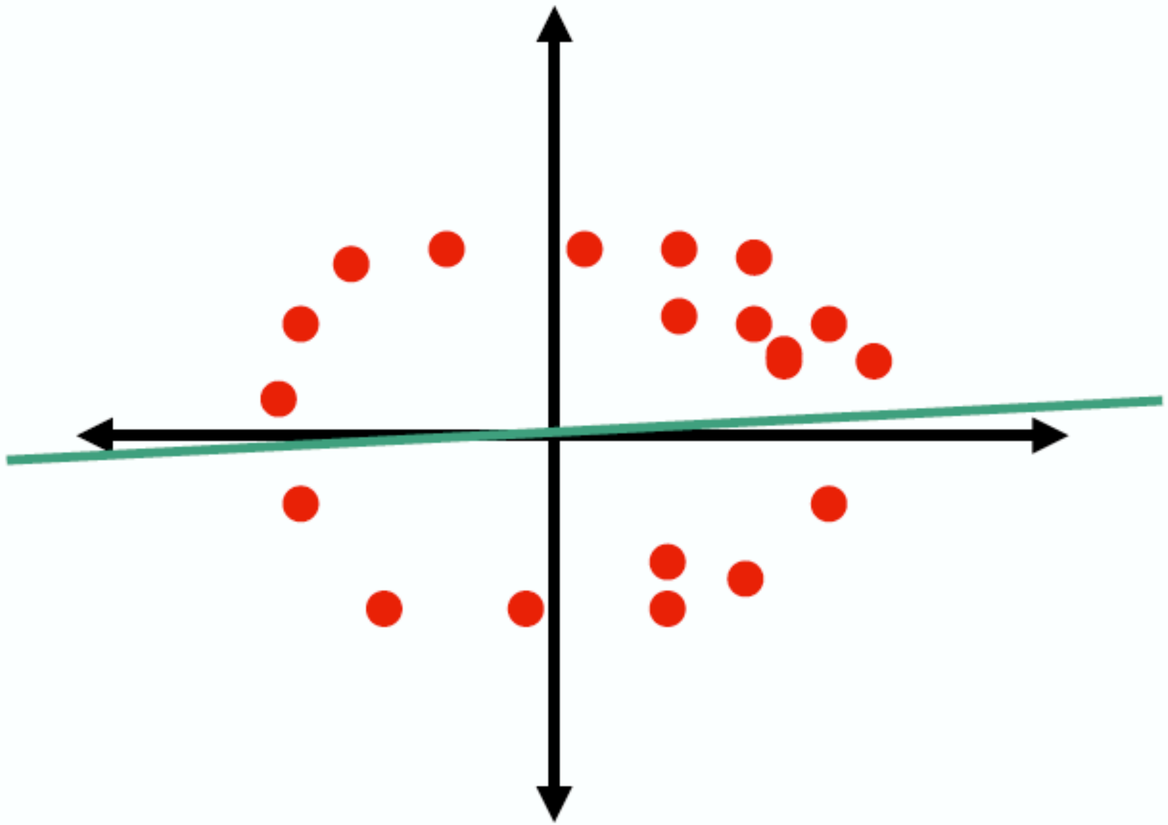
OLS revisited: why are normal eqns the soln?

$$\min_w ||Xw - y||_2^2$$



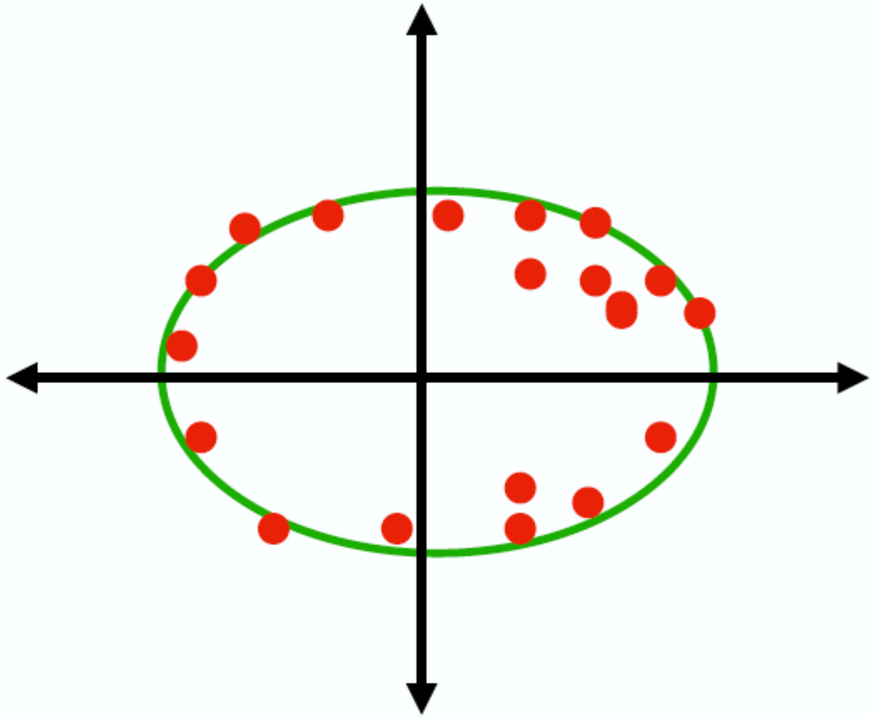
$$\hat{w} = \left(X^T X\right)^{-1} X^T y$$

Can we model this linearly?



$$(x_1, y_1)$$
$$(x_2, y_2)$$
$$\vdots$$
$$(x_n, y_n)$$

Can we model this linearly?



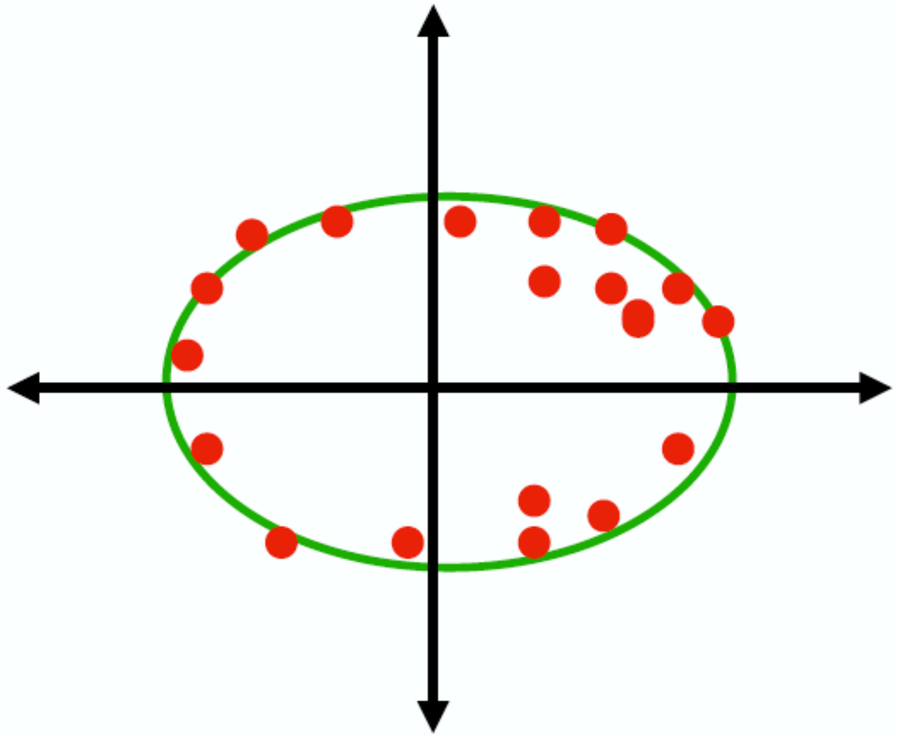
$$w_1 x_i^2 + w_2 y_i^2 + w_3 x_i y_i + w_4 x + w_5 y + w_6 = 0$$

$$\{x^2, y^2, xy, x, y, 1\}$$

$$\bar{X} = \begin{bmatrix} x_1^2 & y_1^2 & x_1 y_1 & x_1 & y_1 & 1 \\ \vdots & & & & & \vdots \\ x_n^2 & y_n^2 & x_n y_n & x_n & y_n & 1 \end{bmatrix}$$

$$\min_w ||Xw - 0||^2$$

Can we model this linearly?



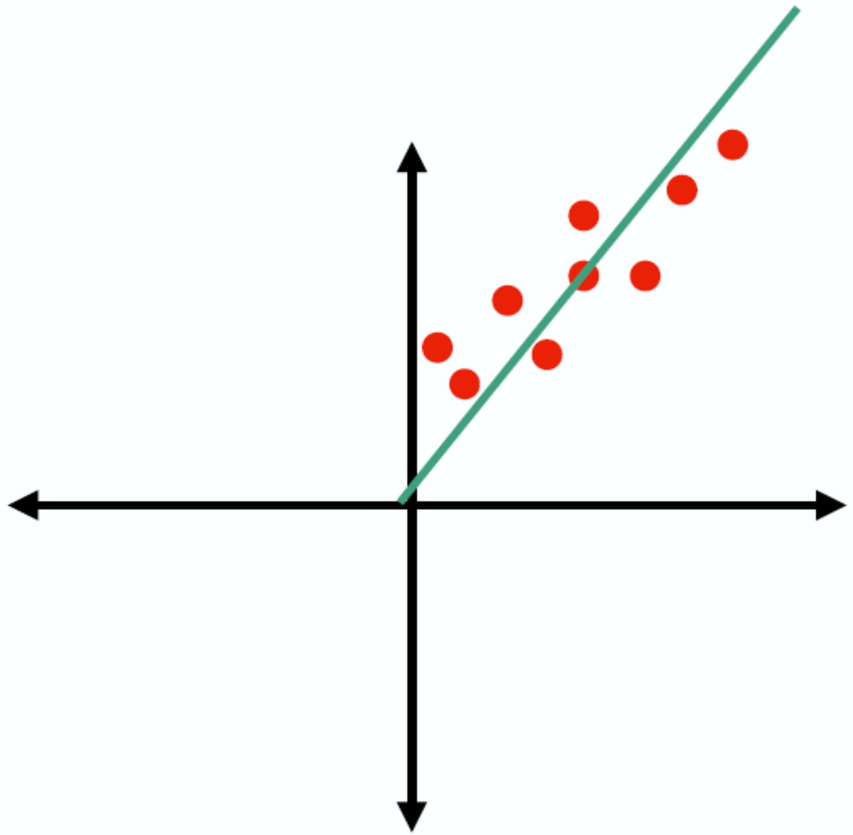
$$w_1 x_i^2 + w_2 y_i^2 + w_3 x_i y_i + w_4 x_i + w_5 y_i = 1$$

$$\{x_1^2, y_1^2, x_1 y_1, x_1, y_1\}$$

$$\bar{X} = \begin{bmatrix} x_1^2 & y_1^2 & x_1 y_1 & x_1 & y_1 \\ \vdots & & & & \vdots \\ x_n^2 & y_n^2 & x_n y_n & x_n & y_n \end{bmatrix}$$

$$\min_w ||\bar{X} w - \vec{1}||^2$$

Another example



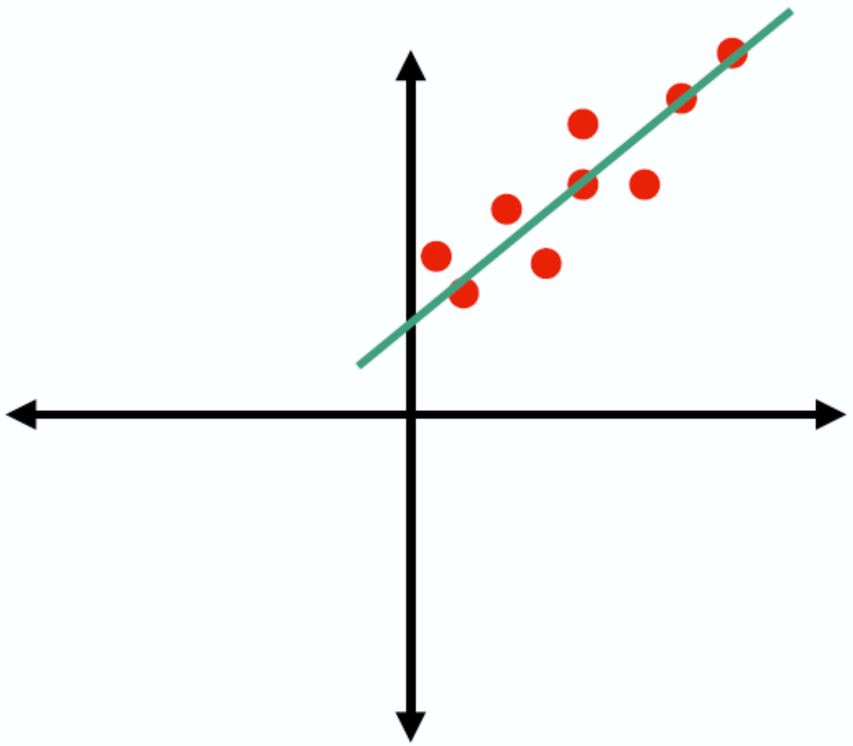
Model

$$\hat{y}_i = w_1 x_i$$

Features

$$\{x\}$$

Linear Affine model



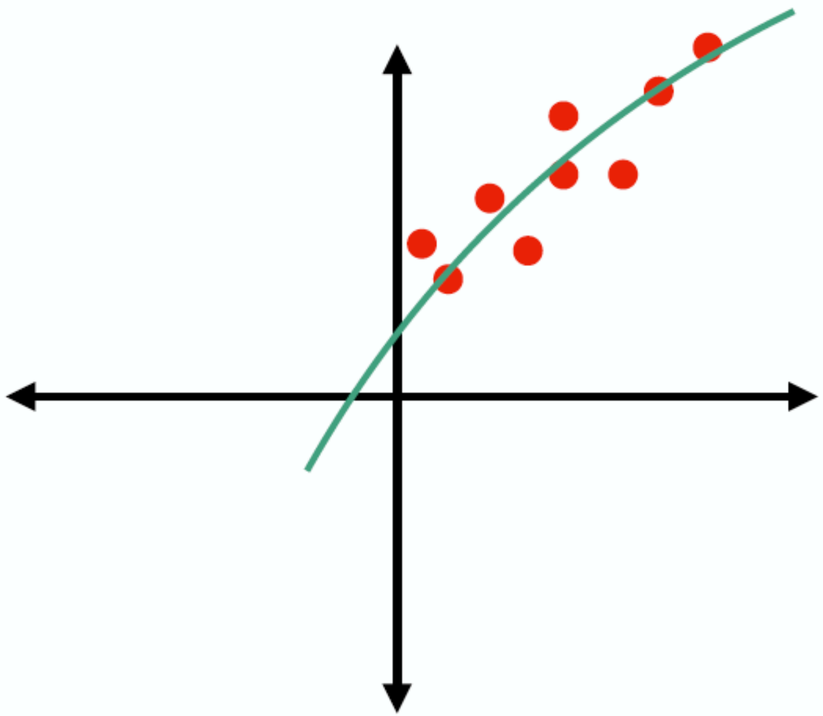
Model

$$\hat{y}_i = w_1 x_i + w_0$$

Features

$$\{1, x\}$$

Quadratic model



Model

$$\hat{y}_i = w_2 x_i^2 + w_1 x_i + w_0$$

Features

$$\{1, x, x^2\}$$

Polynomial features

Original data

$$X = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix}$$

Model

$$\hat{y}_i = \sum_{j=0}^{p} w_j x_i^j$$

Features

$$\{1, x, x^2, \cdots x^p\}$$

Polynomial features

Original data

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & & \vdots \\ x_{n1} & \cdots & x_{nd} \end{bmatrix}$$

Model

$$\hat{y}_i = \sum_{j=0}^{p} w_{j0} x_{i1}^p + \cdots + w_{jd} x_{id}^p$$

$$+ w_{j,d+1} x_{i1}^{p-1} x_{i2} + \cdots$$

Features

$$\{ \quad \begin{matrix} 0, & x_1, & \cdots, & x_d, \\ x_1^2, & x_2^2, & \cdots, & x_d^2, \\ x_1 x_2, & x_1 x_3, & \cdots, & x_1 x_d, \\ & & \cdots, & \\ x_1^p, & x_2^p, & \cdots, & x_d^p \end{matrix} \quad \}$$

Polynomial features

Original data

$$X = \begin{bmatrix} x_{11} & \cdots & x_{1d} \\ \vdots & & \vdots \\ x_{n1} & \cdots & x_{nd} \end{bmatrix}$$

Model

$$\hat{y}_i = \sum_{j=0}^{p} w_{j0} x_{i1}^p + \cdots + w_{jd} x_{id}^p$$

$$+ w_{j,d+1} x_{i1}^{p-1} x_{i2} + \cdots$$

Features

$$\{ \quad \begin{matrix} 0, & x_1, & \cdots, & x_d, \\ x_1^2, & x_2^2, & \cdots, & x_d^2, \\ x_1 x_2, & x_1 x_3, & \cdots, & x_1 x_d, \\ & & \cdots, & \\ x_1^p, & x_2^p, & \cdots, & x_d^p \end{matrix} \quad \}$$

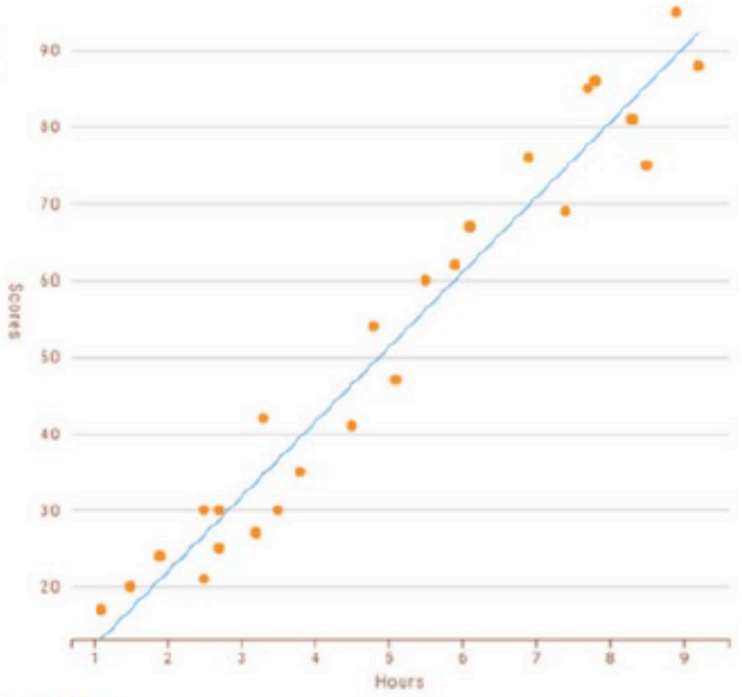
Polynomial features

- Universal function approximations
- Number of params grows superlinearly in polynomial degree
- Later we will talk about how to avoid a lot of this computation

SIMPLE LINEAR REGRESSION

Predicting a response using a single feature.

It is a method to predict dependent variable (Y) based on values of independent variables (X). It is assumed that the two variables are linearly related. Hence, we try to find a linear function that predicts the response value(y) as accurately as possible as a function of the feature or independent variable(x).



Dependent Variable

$$y = b_0 + b_1 x_1$$

Independent Variable

How to find the best fit line?

In this regression model, we are trying to minimize the errors in prediction by finding the "line of best fit" – the regression line from the errors would be minimal. We are trying to minimize the length between the observed value (Yi) and the predicted value from our model (Yp).

Predicted Value

$y_i \quad y_p$

observed Value

$$\min \{\text{SUM}(y_i - y_p)^2\}$$

In this regression task, we will predict the percentage of marks that a student is expected to score based upon the number of hours they studied.

Slope

$$Score = b_0 + b_1 * hours$$

Y - intercept

https://www.theinsaneapp.com/2021/11/machine-learning-algorithms-for-beginners.html

STEP 1: PREPROCESS THE DATA

We will follow the same steps as in my previous infographic of Data Preprocessing.

- Import the Libraries.
- Import the DataSet.
- Check for Missing Data.
- Split the DataSet.
- Feature Scaling will be taken care by the Library we will use for Simple Linear Regression Model.

STEP 2: FITTING SIMPLE LINEAR REGRESSION MODEL TO THE TRAINING SET

To fit the dataset into the model we will use LinearRegression class from sklearn.linear_model library. Then we make an object regressor of LinearRegression Class. Now we will fit the regressor object into our dataset using fit() method of LinearRegression Class.

STEP 3: PREDICTING THE RESULT

Now we will predict the observations from our test set. We will save the output in a vector Y_pred. To predict the result we use predict method of LinearRegression Class on the regressor we trained in the previous step.

STEP 4: VISUALIZATION

The final step is to visualize our results. We will use matplotlib.pyplot library to make Scatter Plots of our Training set results and Test set results to see how close our model predicted the Values

https://www.theinsaneapp.com/2021/11/machine-learning-algorithms-for-beginners.html

Choosing a restaurant

- In everyday life we need to make decisions by taking into account lots of factors
- The question is what weight we put on each of these factors (how important are they with respect to the others).
- Assume we would like to build a recommender system for ***ranking*** potential restaurants based on an individuals' preferences
- If we have many observations we may be able to recover the weights

Reviews (out of 5 stars)	$	Distance	Cuisine (out of 10)	score
4	30	21	7	8.5
2	15	12	8	7.8
5	27	53	9	6.7
3	20	5	6	5.4

?

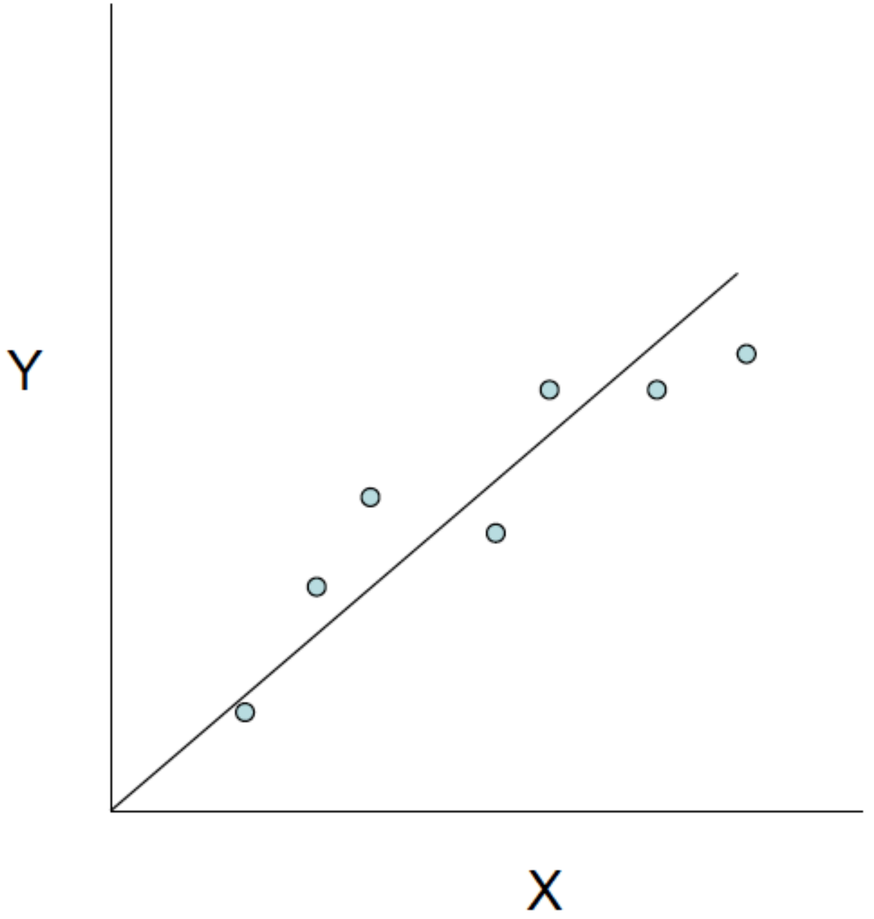
Linear Regression

- Given an input x we would like to compute an output y
- For example:

  - Predict height from age

  - Predict Google's price from Yahoo's price

  - Predict distance from wall using sensor readings



Note that now Y can be **continuous**

Linear Regression

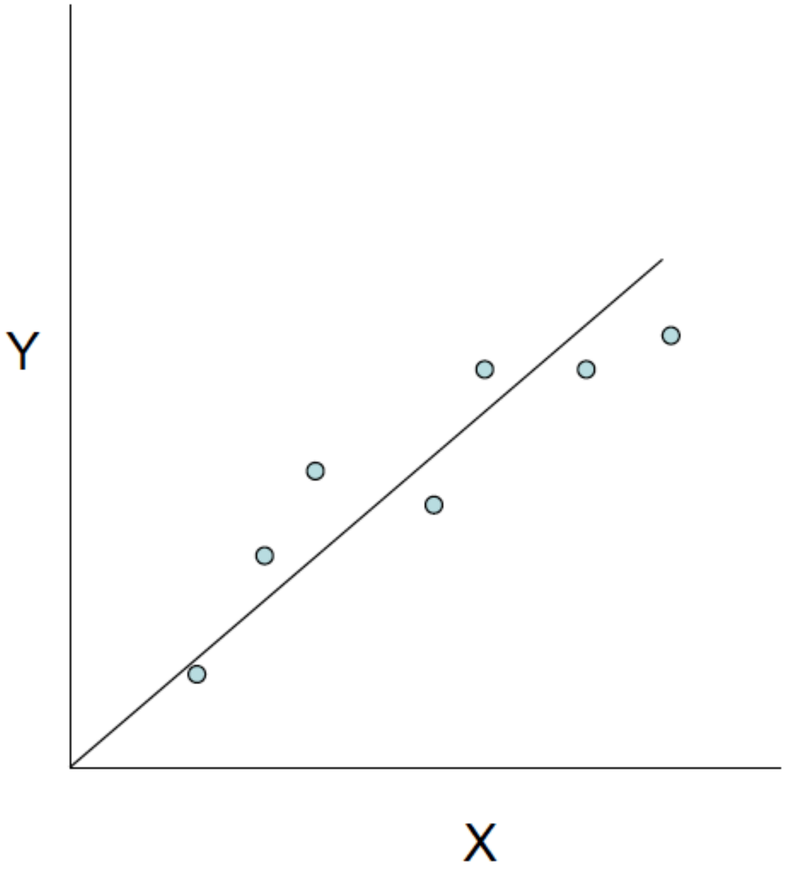
- Given an input x we would like to compute an output y
- In linear regression we assume that y and x are related with the following equation:

Observed values

What we are trying to predict

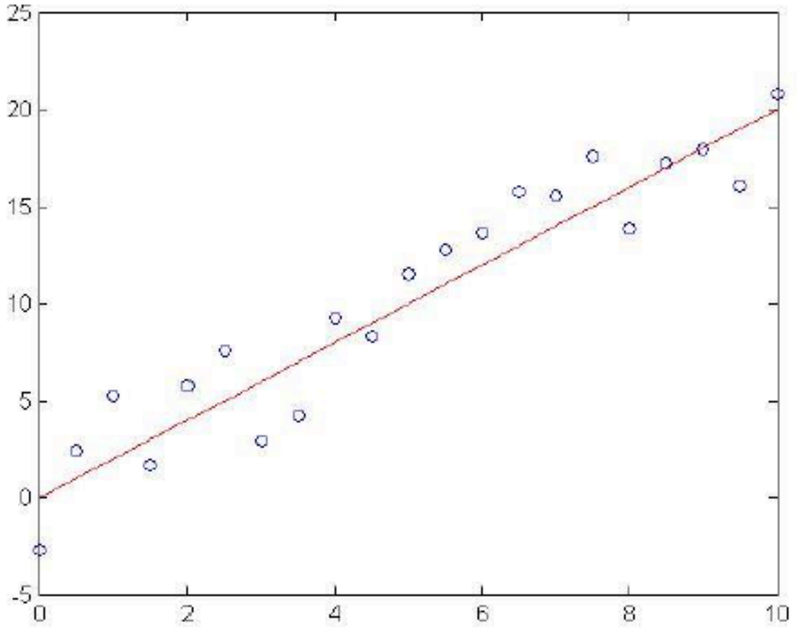
$$y = wx+\varepsilon$$

where w is a parameter and ε represents measurement or other noise



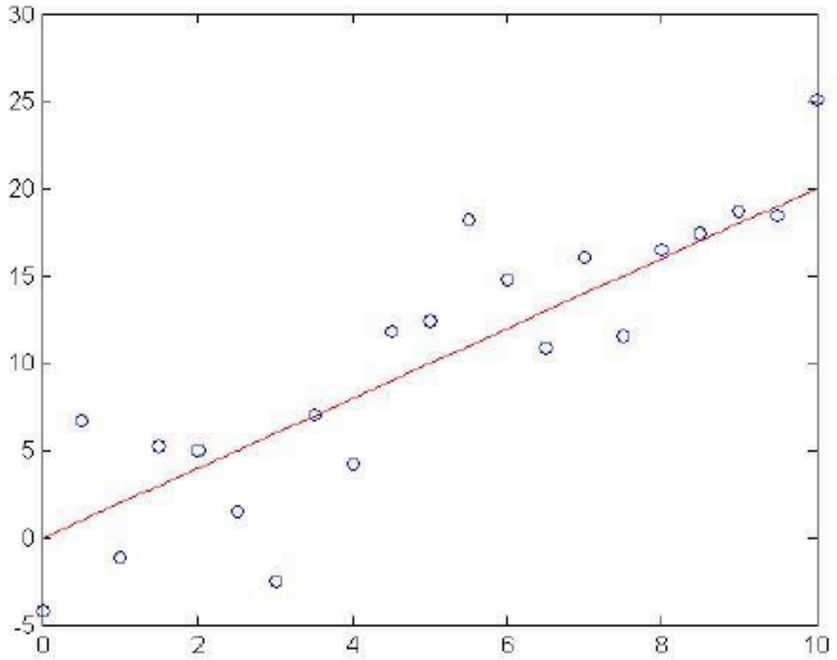
Regression Example

- Generated: w=2
- Recovered: w=2.05
- Noise: std=2



Regression Example

- Generated: w=2
- Recovered: w=2.08
- Noise: std=4

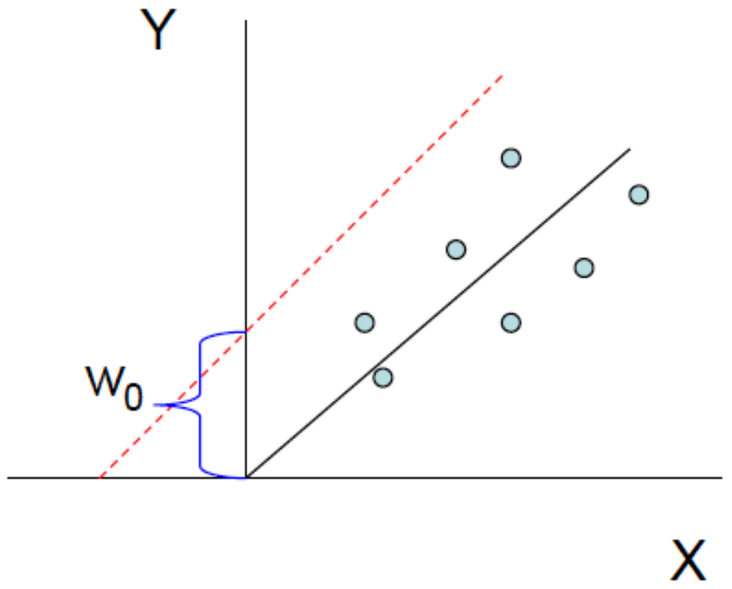


Bias Term

- So far we assumed that the line passes through the origin
- What if the line does not?
- No problem, simply change the model to

$$y = w_0 + w_1 x + \varepsilon$$

- Can use least squares to determine w_0 , w_1



$$w_0 = \frac{\sum_i y_i - w_1 x_i}{n}$$

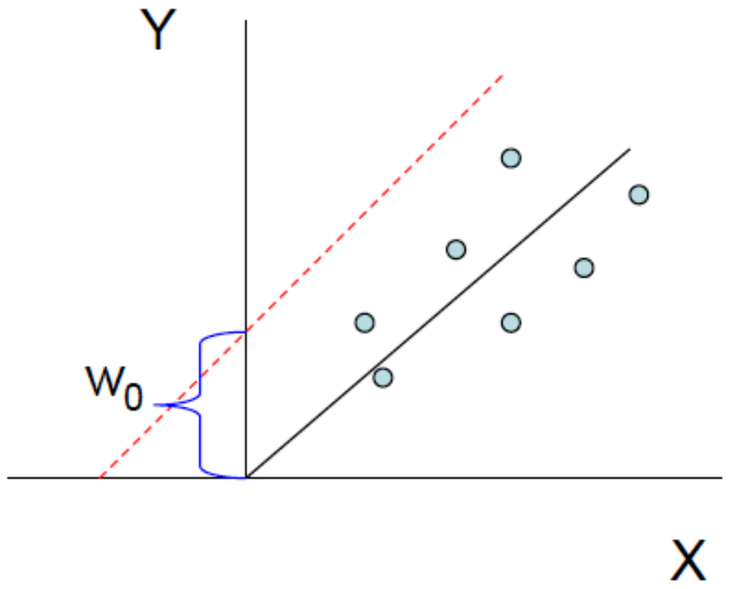
$$w_1 = \frac{\sum_i x_i (y_i - w_0)}{\sum_i x_i^2}$$

Bias Term

- So far we assumed that the line passes through the origin
- What if the line does not?
- No problem, simply change the model to

$$y = w_0 + w_1 x + \varepsilon$$

- Can use least squares to determine w_0 , w_1



$$w_0 = \frac{\sum_i y_i - w_1 x_i}{n}$$

$$w_1 = \frac{\sum_i x_i (y_i - w_0)}{\sum_i x_i^2}$$

Multivariate regression

- What if we have several inputs?

  - Stock prices for Yahoo, Microsoft and Ebay for the Google prediction task

- This becomes a multivariate linear regression problem
- Again, its easy to model:

$$y = w_0 + w_1 x_1 + \ldots + w_k x_k + \varepsilon$$

Google's stock price

Yahoo's stock price

Microsoft's stock price

Multivariate regression

$$y=10+3x_1^2-2x_2^2+\varepsilon$$

In some cases we would like to use polynomial or other terms based on the input data, are these still linear regression problems?

Yes. As long as the coefficients are linear the equation is still a linear regression problem!

Non linear basis function

- So far we only used the observed values
- However, linear regression can be applied in the same way to functions of these values
- As long as these functions can be directly computed from the observed values the parameters are still linear in the data and the problem remains a linear regression problem

$$y = w_0 + w_1 x_1^2 + \ldots + w_k x_k^2 + \varepsilon$$

Non linear basis function

- What type of functions can we use?
- A few common examples:

- Polynomial: $\phi_j(x) = x^j$ for j=0 … n

- Gaussian: $\phi_j(x) = \frac{(x - \mu_j)}{2\sigma_j^2}$

- Sigmoid: $\phi_j(x) = \frac{1}{1 + \exp(-s_j x)}$

Any function of the input values can be used. The solution for the parameters of the regression remains the same.

LMS for the general linear regression problem

$$y = \sum_{j=0}^{n} w_j \phi_j(x)$$

Our goal is to minimize the following loss function:

$$J(\mathrm{w}) = \sum_i (y^i - \sum_j w_j \phi_j(x^i))^2$$

w – vector of dimension k+1
$\phi(x^i)$ – vector of dimension k+1
y^i – a scaler

Moving to vector notations we get:

$$J(\mathrm{w}) = \sum_i (y^i - \mathrm{w}^{\mathrm{T}} \phi(x^i))^2$$

We take the derivative w.r.t **w**

$$\frac{\partial}{\partial w} \sum_i (y^i - \mathrm{w}^{\mathrm{T}} \phi(x^i))^2 = 2 \sum_i (y^i - \mathrm{w}^{\mathrm{T}} \phi(x^i)) \phi(x^i)^{\mathrm{T}}$$

Equating to 0 we get $2 \sum_i (y^i - \mathrm{w}^{\mathrm{T}} \phi(x^i)) \phi(x^i)^{\mathrm{T}} = 0 \Rightarrow$

$$\sum_i y^i \phi(x^i)^{\mathrm{T}} = \mathrm{w}^{\mathrm{T}} \left[\sum_i \phi(x^i) \phi(x^i)^{\mathrm{T}} \right]$$

LMS for the general linear regression problem

$$J(\mathbf{w}) = \sum_i (y^i - \mathbf{w}^{\mathrm{T}} \phi(x^i))^2$$

We take the derivative w.r.t **w**

$$\frac{\partial}{\partial w} \sum_i (y^i - \mathbf{w}^{\mathrm{T}} \phi(x^i))^2 = 2 \sum_i (y^i - \mathbf{w}^{\mathrm{T}} \phi(x^i)) \phi(x^i)^{\mathrm{T}}$$

Equating to 0 we get

$$2 \sum_i (y^i - \mathbf{w}^{\mathrm{T}} \phi(x^i)) \phi(x^i)^{\mathrm{T}} = 0 \Rightarrow$$

$$\sum_i y^i \phi(x^i)^{\mathrm{T}} = \mathbf{w}^{\mathrm{T}} \left[\sum_i \phi(x^i) \phi(x^i)^{\mathrm{T}} \right]$$

Define:

$$\Phi = \begin{pmatrix} \phi_0(x^1) & \phi_1(x^1) & \cdots & \phi_m(x^1) \\ \phi_0(x^2) & \phi_1(x^2) & \cdots & \phi_m(x^2) \\ \vdots & \vdots & \cdots & \vdots \\ \phi_0(x^n) & \phi_1(x^n) & \cdots & \phi_m(x^n) \end{pmatrix}$$

Then deriving w we get:

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

LMS for the general linear regression problem

$$J(\mathbf{w}) = \sum_i (y^i - \mathbf{w}^T \phi(x^i))^2$$

Deriving w we get: $\mathbf{w} = (\Phi^T \Phi)^{-1} \Phi^T \mathbf{y}$

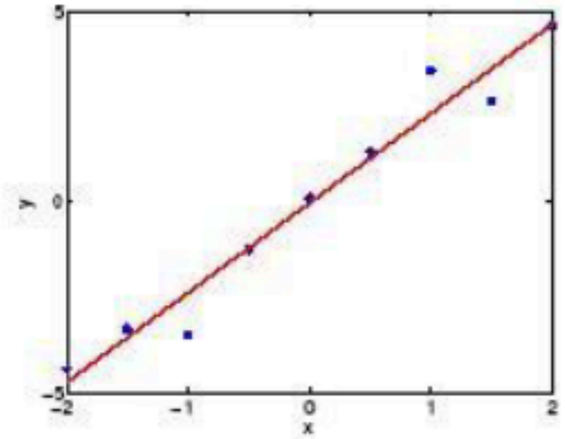
k+1 entries vector

n by k+1 matrix

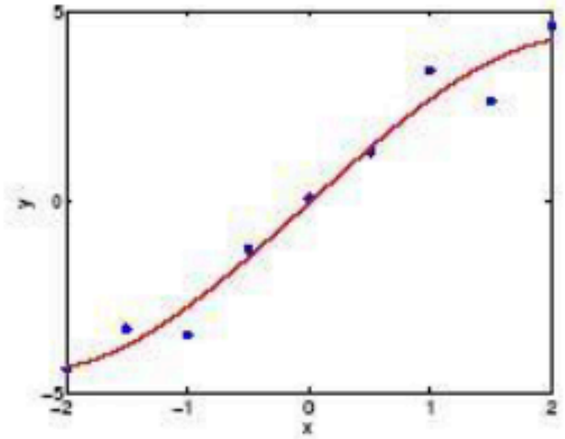
n entries vector

This solution is also known as 'psuedo inverse'

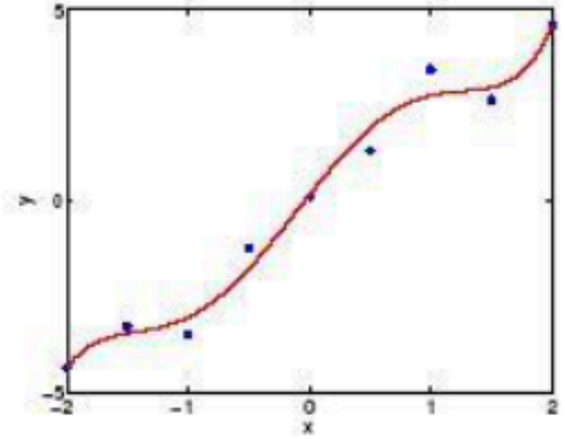
Example - polynomial regression



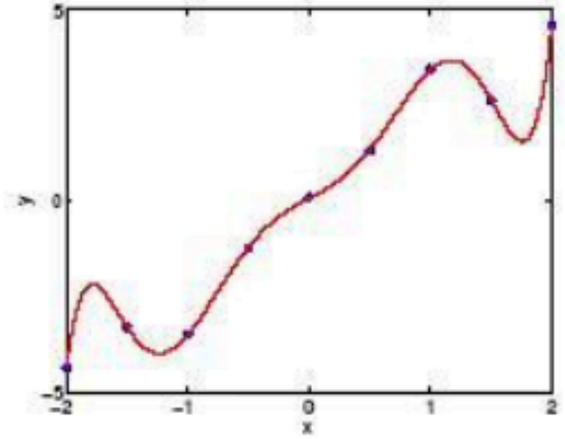
degree = 1, CV = 0.6



degree = 3, CV = 1.5



degree = 5, CV = 6.0



degree = 7, CV = 15.6

A probabilistic interpretation

Our least squares minimization solution can also be motivated by a probabilistic in interpretation of the regression problem: $y = \mathbf{w}^{\mathrm{T}} \phi(x) + \varepsilon$

The MLE for w in this model is the same as the solution we derived for least squares criteria:

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

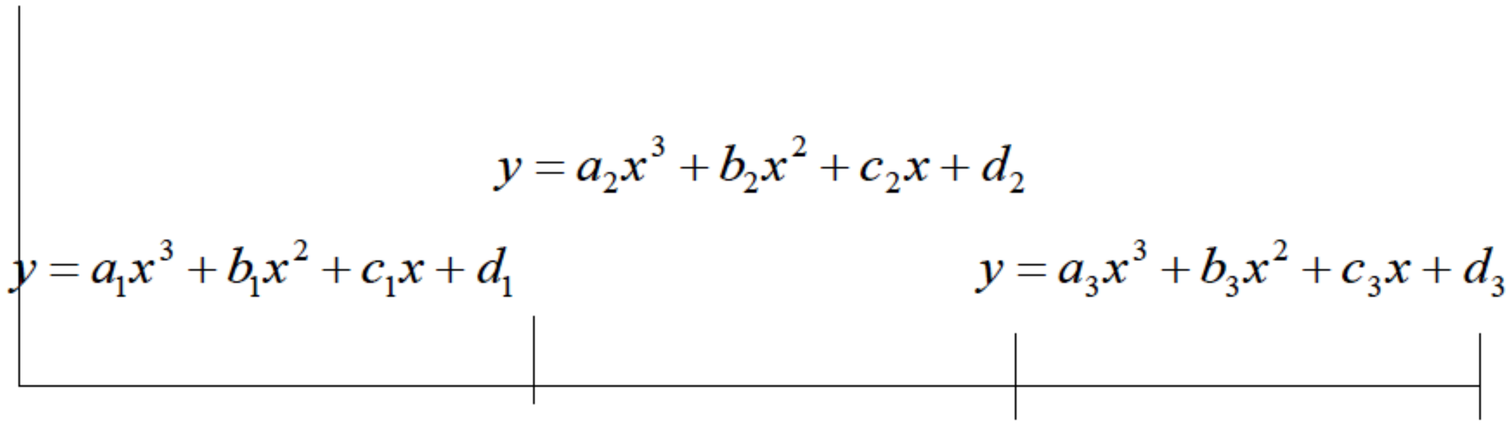
MLE = Maximum Likelihood Estimation

Other types of linear regression

- Linear regression is a useful model for many problems
- However, the parameters we learn for this model are **global**; they are the same regardless of the value of the input x
- Extension to linear regression adjust their parameters based on the region of the input we are dealing with

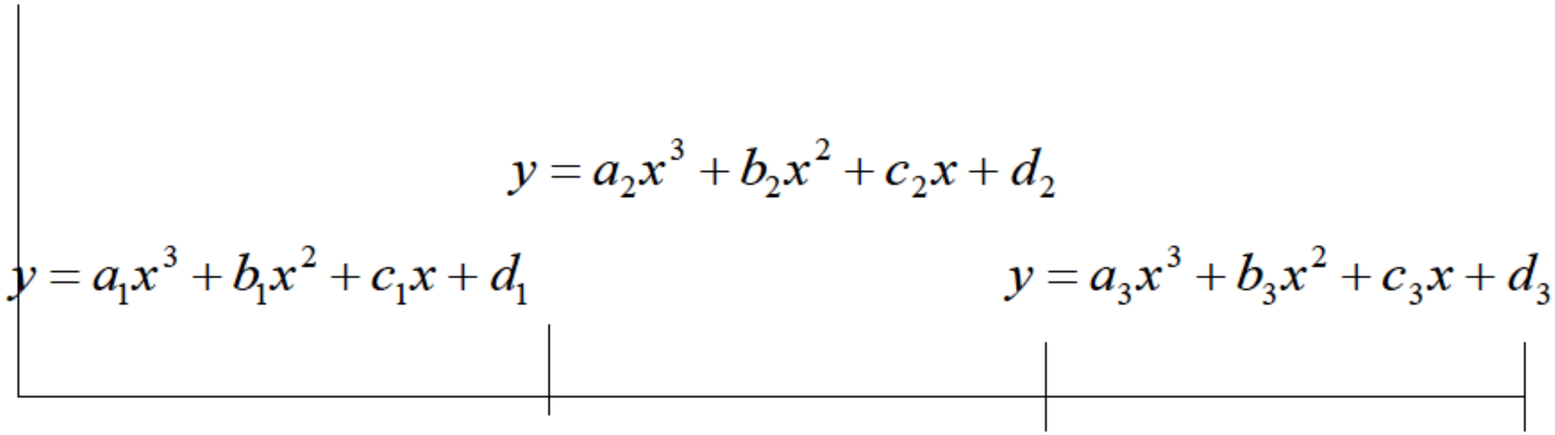
Splines

- Instead of fitting one function for the entire region, fit a set of piecewise (usually cubic) polynomials satisfying continuity and smoothness constraints.
- Results in smooth and flexible functions without too many parameters
- Need to define the regions in advance (usually uniform)



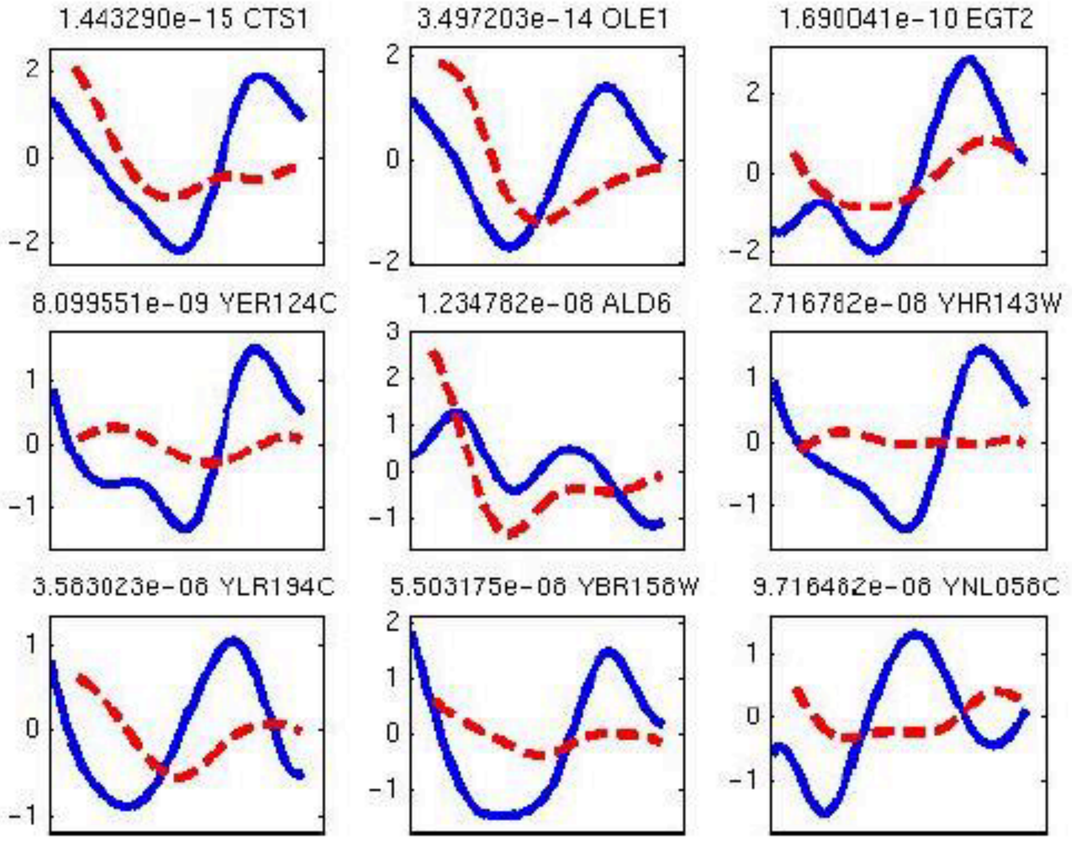
Splines

- The polynomials are not independent
- For cubic splines we require that they agree in the border point on the value, the values of the first derivative and the value of the second derivative
- How many free parameters do we actually have?



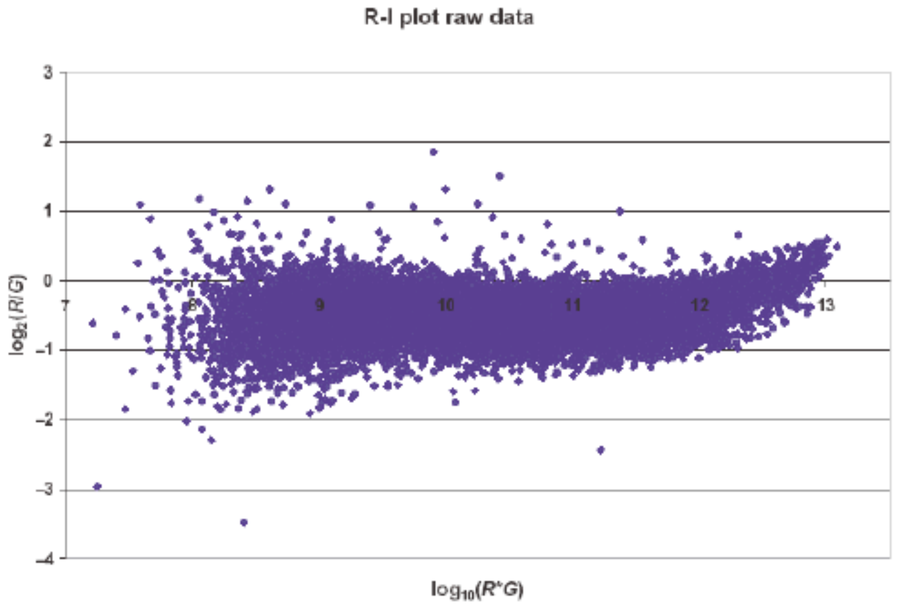
Splines

- Splines sometimes contain additional requirements for the first and last polynomial (for example, having them start at 0)
- Once Splines are fitted to the data they can be used to predict new values in the same way as regular linear regression, though they are limited to the support regions for which they have been defined
- Note the range of functions that can be displayed with relatively small number of polynomials (in the example I am using 5)



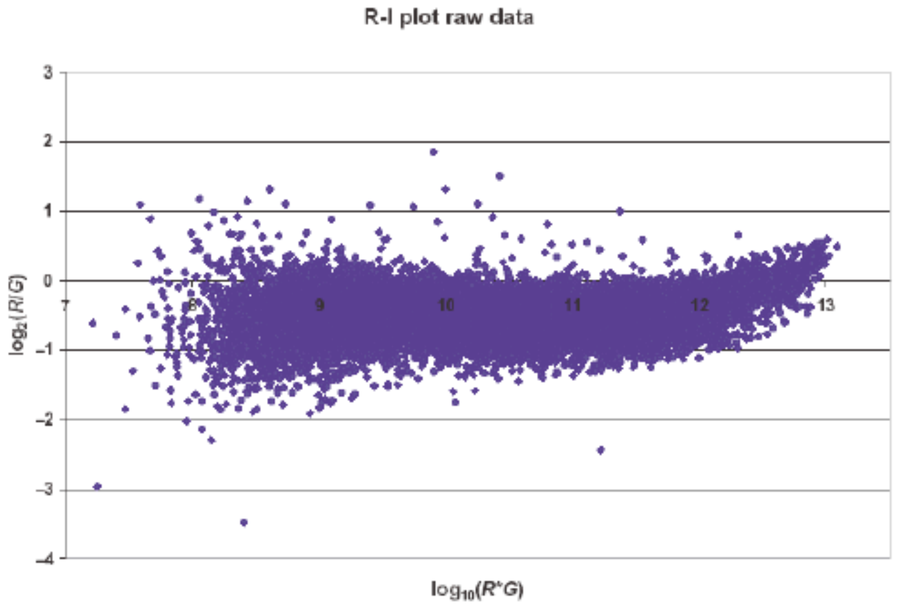
Locally weighted models

- Splines rely on a fixed region for each polynomial and the weight of all points within the region is the same.
- An alternative option is to set the region based on the density of the input data and have points closer to the point we are trying to estimate have a higher weight



Locally weighted models

- Splines rely on a fixed region for each polynomial and the weight of all points within the region is the same.
- An alternative option is to set the region based on the density of the input data and have points closer to the point we are trying to estimate have a higher weight

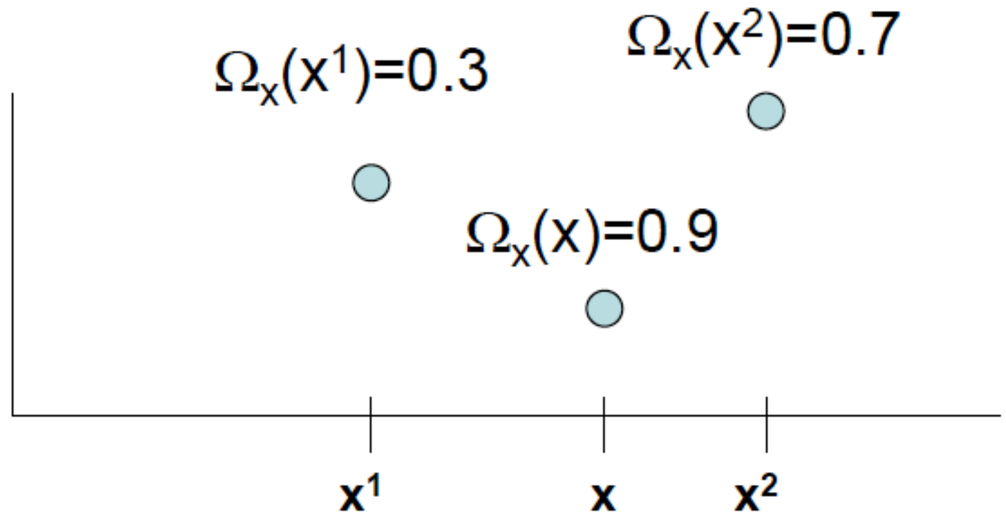


Weighted regression

- For a point x we use weight function Ω_x centered at x to assign weight to points in x's vicinity
- Next we solve the following *weighted* regression problem

$$\min_w \sum_i \Omega_x(x^i)(y^i - w^T \phi(x^i))^2$$

- The solution is the same as our general solution (the weight is given for every input)



Determining the weights

- There are a number of ways to determine the weights
- One options is to use a Gaussian centered at *x*, such that

$$\Omega_x(x^i) = \frac{1|}{\sqrt{2\pi\sigma}} e^{-\frac{(x-x^i)^2}{2\sigma^2}}$$

σ^2 is a parameter that should be selected by the user

More on these weights when we discuss kernels

Notes

- Linear regression
  - basic model
  - as a function of the input
- Solving linear regression
- Error in linear regression
- Advanced regression models

Ridge Regression

- While OLS can be used for solving linear least squares problems, it falls short due to numerical instability & generalization issues. Numerical instability arises when the features of the data are close to collinear (leading to linearly dependent feature columns) causing the input matrix X to lose its rank or have singular values that are very close to 0
- Why are small singular values bad? Let us illustrate this via the singular value decomposition (SVD) of X: $X = U\Sigma V^T$ where $U \in \mathbb{R}^{nxn}$, $\Sigma \in \mathbb{R}^{nxd}$, $V \in \mathbb{R}^{dxd}$. In the context of OLS, we must have that $X^T X$ is invertible or equivalently, rank($X^T X$)=rank(X^T)=rank(X)=d. Assuming that X^T and X are full column rank d, we can express the SVD of X as:

Ridge Regression

$$\mathbf{X} = \mathbf{U}\begin{bmatrix}\boldsymbol{\Sigma}_d \\ \mathbf{0}\end{bmatrix}\mathbf{V}^\top$$

where $\boldsymbol{\Sigma}_d \in \mathbb{R}^{d\times d}$ is a diagonal matrix with strictly positive entries. Now let's try to expand the $(\mathbf{X}^\top\mathbf{X})^{-1}$ term in OLS using the SVD of $\mathbf{X}$:

$$\begin{aligned}(\mathbf{X}^\top\mathbf{X})^{-1} &= (\mathbf{V}\begin{bmatrix}\boldsymbol{\Sigma}_d & \mathbf{0}\end{bmatrix}\mathbf{U}^\top\mathbf{U}\begin{bmatrix}\boldsymbol{\Sigma}_d \\ \mathbf{0}\end{bmatrix}\mathbf{V}^\top)^{-1} \\ &= (\mathbf{V}\begin{bmatrix}\boldsymbol{\Sigma}_d & \mathbf{0}\end{bmatrix}\mathbf{I}\begin{bmatrix}\boldsymbol{\Sigma}_d \\ \mathbf{0}\end{bmatrix}\mathbf{V}^\top)^{-1} \\ &= (\mathbf{V}\boldsymbol{\Sigma}_d^2\mathbf{V}^\top)^{-1} = (\mathbf{V}^\top)^{-1}(\boldsymbol{\Sigma}_d^2)^{-1}\mathbf{V}^{-1} = \mathbf{V}\boldsymbol{\Sigma}_d^{-2}\mathbf{V}^\top\end{aligned}$$

This means that $(\mathbf{X}^\top\mathbf{X})^{-1}$ will have singular values that are the squared inverse of the singular values of $\mathbf{X}$, potentially leading to extremely large singular values when the singular value of $\mathbf{X}$ are close to 0. Such excessively large singular values can be very problematic for numerical stability purposes. In addition, abnormally high values to the optimal $\mathbf{w}$ solution would prevent OLS from generalizing to unseen data.

Ridge Regression

There is a very simple solution to these issues: penalize the entries of $\mathbf{w}$ from becoming too large. We can do this by adding a penalty term constraining the norm of $\mathbf{w}$. For a fixed, small scalar $\lambda > 0$, we now have:

$$\min_{\mathbf{w}} \left\|\mathbf{X}\mathbf{w} - \mathbf{y}\right\|_2^2 + \lambda \|\mathbf{w}\|_2^2$$

Note that the λ in our objective function is a **hyperparameter** that measures the sensitivity to the values in $\mathbf{w}$. Just like the degree in polynomial features, λ is a value that we must choose arbitrarily through validation. Let's expand the terms of the objective function:

$$\begin{aligned} L(\mathbf{w}) &= \|\mathbf{X}\mathbf{w} - \mathbf{y}\|_2^2 + \lambda \|\mathbf{w}\|_2^2 \\ &= \mathbf{w}^\top \mathbf{X}^\top \mathbf{X}\mathbf{w} - 2\mathbf{w}^\top \mathbf{X}^\top \mathbf{y} + \mathbf{y}^\top \mathbf{y} + \lambda \mathbf{w}^\top \mathbf{w} \end{aligned}$$

Finally take the gradient of the objective and find the value of $\mathbf{w}$ that achieves $\mathbf{0}$ for the gradient:

$$\begin{aligned} \nabla_{\mathbf{w}} L(\mathbf{w}) &= \mathbf{0} \\ 2\mathbf{X}^\top \mathbf{X}\mathbf{w} - 2\mathbf{X}^\top \mathbf{y} + 2\lambda \mathbf{w} &= \mathbf{0} \\ (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})\mathbf{w} &= \mathbf{X}^\top \mathbf{y} \\ \mathbf{w}^*_{\text{RIDGE}} &= (\mathbf{X}^\top \mathbf{X} + \lambda \mathbf{I})^{-1} \mathbf{X}^\top \mathbf{y} \end{aligned}$$

Ridge Regression

This value is guaranteed to achieve the (unique) global minimum, because the objective function is **strongly convex**. To show that f is strongly convex, it suffices to compute the Hessian of f, which in this case is

$$\nabla^2 L(\mathbf{w}) = 2\mathbf{X}^\top\mathbf{X} + 2\lambda\mathbf{I}$$

and show that this is **positive definite (PD)**:

$$\forall \mathbf{w} \neq \mathbf{0},\ \mathbf{w}^\top(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})\mathbf{w} = (\mathbf{X}\mathbf{w})^\top\mathbf{X}\mathbf{w} + \lambda\mathbf{w}^\top\mathbf{w} = \|\mathbf{X}\mathbf{w}\|_2^2 + \lambda\|\mathbf{w}\|_2^2 > 0$$

Since the Hessian is positive definite, we can equivalently say that the eigenvalues of the Hessian are strictly positive and that the objective function is strongly convex. A useful property of strongly convex functions is that they have a unique optimum point, so the solution to ridge regression is unique. We cannot make such guarantees about ordinary least squares, because the corresponding Hessian could have eigenvalues that are 0. Let us explore the case in OLS when the Hessian has a 0 eigenvalue. In this context, the term $\mathbf{X}^\top\mathbf{X}$ is not invertible, but this does *not* imply that no solution exists! In OLS, there always exists a solution, and when the Hessian is PD that solution is unique; when the Hessian is PSD, there are infinitely many solutions. (There always exists a solution to the expression $\mathbf{X}^\top\mathbf{X}\mathbf{w} = \mathbf{X}^\top\mathbf{y}$, because the range of $\mathbf{X}^\top\mathbf{X}$ and the range space of $\mathbf{X}^\top$ are equivalent; since $\mathbf{X}^\top\mathbf{y}$ lies in the range of $\mathbf{X}^\top$, it must equivalently lie in the range of $\mathbf{X}^\top\mathbf{X}$ and therefore there always exists a $\mathbf{w}$ that satisfies the equation $\mathbf{X}^\top\mathbf{X}\mathbf{w} = \mathbf{X}^\top\mathbf{y}$.)

Ridge Regression

The technique we just described is known as **ridge regression**. Note that now the expression $\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}$ is invertible, regardless of rank of $\mathbf{X}$. Let's find $(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1}$ through SVD:

$$
\begin{aligned}
(\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I})^{-1} &= \left(\mathbf{V}\begin{bmatrix}\boldsymbol{\Sigma}_r & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\mathbf{U}^\top\mathbf{U}\begin{bmatrix}\boldsymbol{\Sigma}_r & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\mathbf{V}^\top + \lambda\mathbf{I}\right)^{-1} \\
&= \left(\mathbf{V}\begin{bmatrix}\boldsymbol{\Sigma}_r^2 & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\mathbf{V}^\top + \lambda\mathbf{I}\right)^{-1} \\
&= \left(\mathbf{V}\begin{bmatrix}\boldsymbol{\Sigma}_r^2 & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix}\mathbf{V}^\top + \mathbf{V}(\lambda\mathbf{I})\mathbf{V}^\top\right)^{-1} \\
&= \left(\mathbf{V}\left(\begin{bmatrix}\boldsymbol{\Sigma}_r^2 & \mathbf{0}\\ \mathbf{0} & \mathbf{0}\end{bmatrix} + \lambda\mathbf{I}\right)\mathbf{V}^\top\right)^{-1} \\
&= \left(\mathbf{V}\begin{bmatrix}\boldsymbol{\Sigma}_r^2 + \lambda\mathbf{I} & \mathbf{0}\\ \mathbf{0} & \lambda\mathbf{I}\end{bmatrix}\mathbf{V}^\top\right)^{-1} \\
&= (\mathbf{V}^\top)^{-1}\begin{bmatrix}\boldsymbol{\Sigma}_r^2 + \lambda\mathbf{I} & \mathbf{0}\\ \mathbf{0} & \lambda\mathbf{I}\end{bmatrix}^{-1}\mathbf{V}^{-1} \\
&= \mathbf{V}\begin{bmatrix}(\boldsymbol{\Sigma}_r^2 + \lambda\mathbf{I})^{-1} & \mathbf{0}\\ \mathbf{0} & \frac{1}{\lambda}\mathbf{I}\end{bmatrix}\mathbf{V}^\top
\end{aligned}
$$

Ridge Regression

Now with our slight tweak, the matrix $\mathbf{X}^\top\mathbf{X} + \lambda\mathbf{I}$ has become full rank and thus invertible. The singular values have become $\frac{1}{\sigma^2+\lambda}$ and $\frac{1}{\lambda}$, meaning that the singular values are guaranteed to be at most $\frac{1}{\lambda}$, solving our numerical instability issues. Furthermore, we have partially solved the overfitting issue. By penalizing the norm of $\mathbf{x}$, we encourage the weights corresponding to relevant features that capture the main structure of the true model, and penalize the weights corresponding to complex features that only serve to fine tune the model and fit noise in the data.

Least Squares Polynomial Regression

Replace each X_i with feature vector $\Phi(X_i)$ with all terms of degree $0 \ldots p$

e.g., $\Phi(X_i) = [X_{i1}^2 \quad X_{i1}X_{i2} \quad X_{i2}^2 \quad X_{i1} \quad X_{i2} \quad 1]^\top$

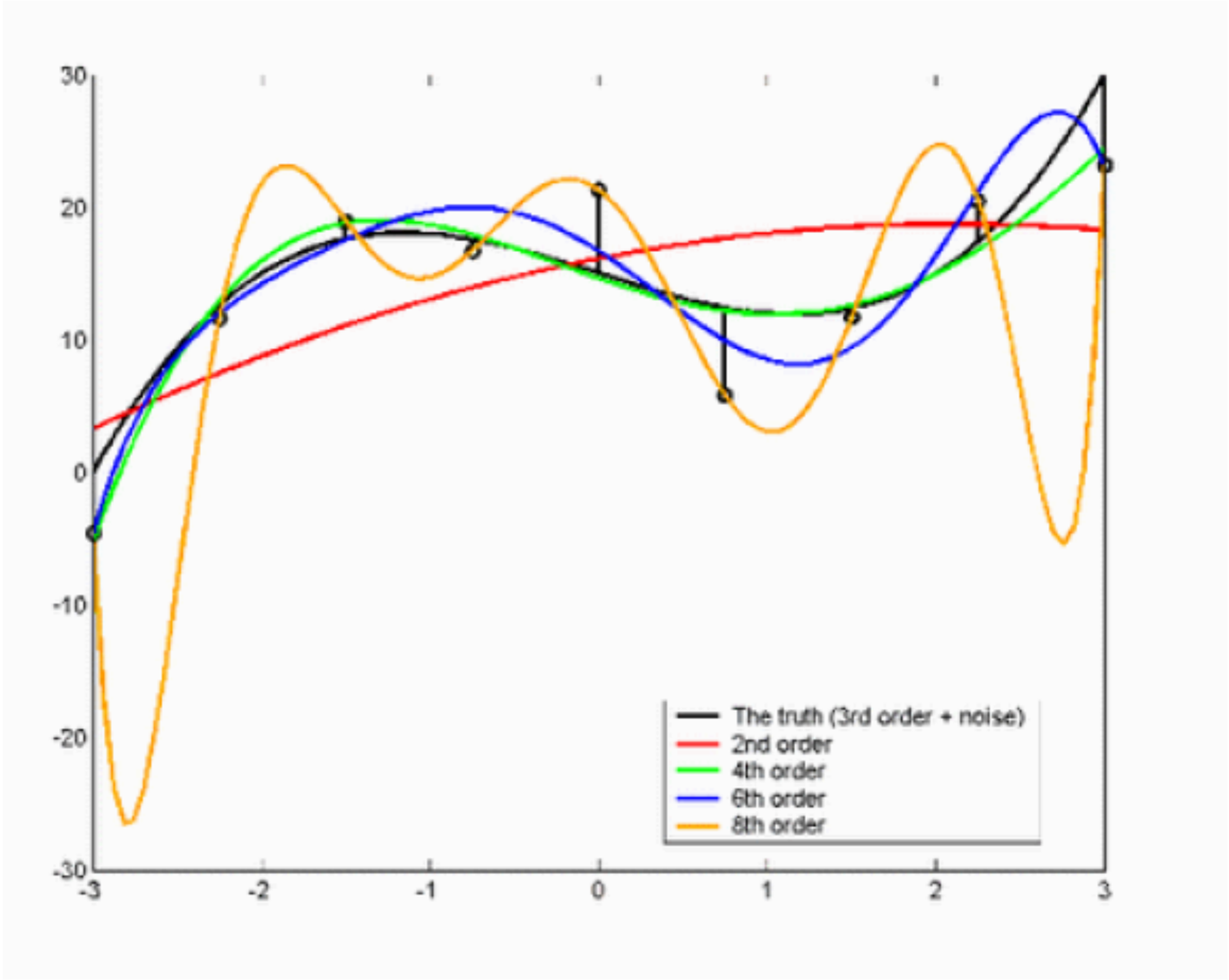
[Notice that we've added the fictitious dimension "1" here, so we don't need to add it again to do linear or logistic regression. This basis covers all polynomials quadratic in X_{i1} and X_{i2}.]

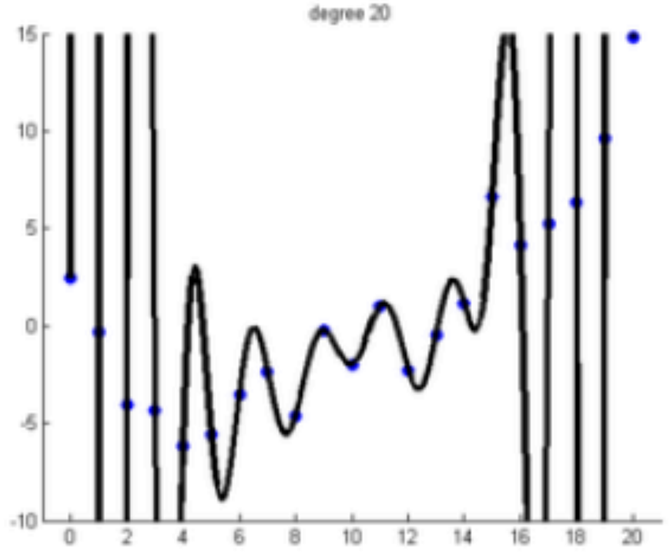
Can also use non-polynomial features (e.g., edge detectors).
Otherwise just like linear or logistic regression.

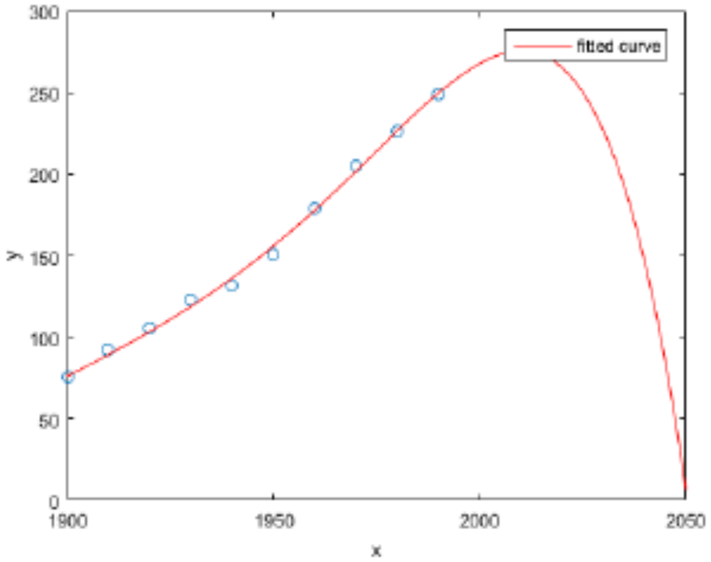
Log. reg. + quadratic features = same form of posteriors as QDA.

Very easy to overfit!

Least Squares Polynomial Regression







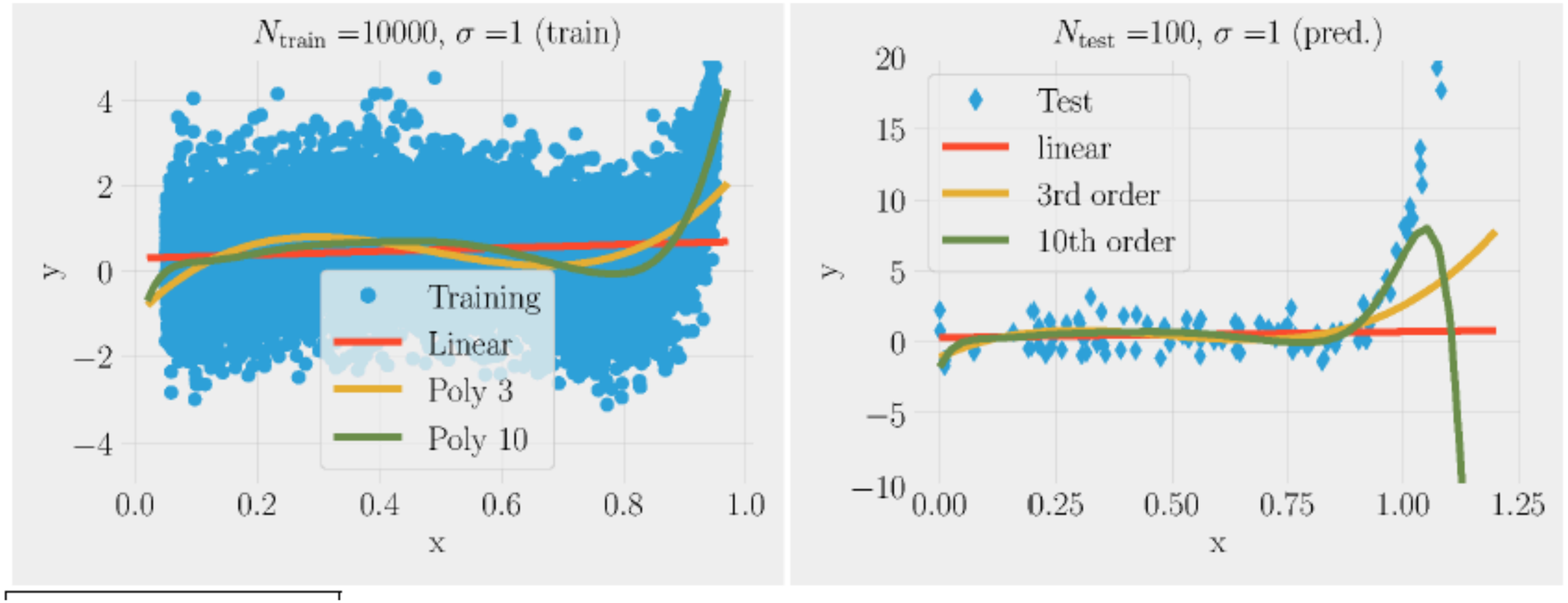
Least Squares Polynomial Regression

[Here are some examples of polynomial overfitting, to show the importance of choosing the polynomial degree very carefully. At left, we have sampled points from a degree-3 curve (black) with added noise. We show best-fit polynomials of degrees 2, 4, 6, and 8 found by regression of the black points. The degree-4 curve (green) fits the true curve (black) well, whereas the degree-2 curve (red) underfits and the degree-6 and 8 curves (blue, yellow) overfit the noise and oscillate. The oscillations in the yellow degree-8 curve are a characteristic problem of polynomial interpolation.]

[At upper right, a degree-20 curve shows just how insane high-degree polynomial oscillations can get. It takes a great deal of densely spaced data to tame the oscillations in a high degree curve, and there isn't nearly enough data here.]

[At lower right, somebody has regressed a degree-4 curve to U.S. census population numbers. The curve doesn't oscillate, but can you nevertheless see a flaw? This shows the difficulty of *extrapolation* outside the range of the data. As a general rule, extrapolation is much harder than interpolation. The *k*-nearest neighbor classifier is one of the few that does extrapolation decently without occasionally returning crazy values.]

Least Squares Polynomial Regression



[This example shows that a fitted degree-10 polynomial (green) can be tamed by using a very large amount of training data (left), even if the training data is noisy. The training data was generated from a different degree-10 polynomial, with noise added. On the right, we see that it even does decent extrapolation for a short distance, albeit only because the original data was also from a degree-10 polynomial.]

Weighted Least Squares Regression

Linear regression fn (1) + squared loss fn (A) + cost fn (c).

[The idea of weighted least-squares is that some sample points might be more trusted than others, or there might be certain points you want to fit particularly well. So you assign those more trusted points a higher weight. If you suspect some points of being outliers, you can assign them a lower weight.]

Assign each sample pt a weight ω_i; collect ω_i's in $n \times n$ diagonal matrix Ω.

Greater $\omega_i \rightarrow$ work harder to minimize $|\hat{y}_i - y_i|^2$ recall: $\hat{y} = Xw$ [$\hat{y}_i$ is predicted label for X_i]

$$\boxed{\text{Find } w \text{ that minimizes } (Xw - y)^\top \Omega (Xw - y)} = \sum_{i=1}^{n} \omega_i \, (X_i \cdot w - y_i)^2$$

[As with ordinary least-squares regression, we find the minimum by setting the gradient to zero, which leads us to the normal equations.]

Solve for w in normal equations: $X^\top \Omega X w = X^\top \Omega y$

Newton's Method

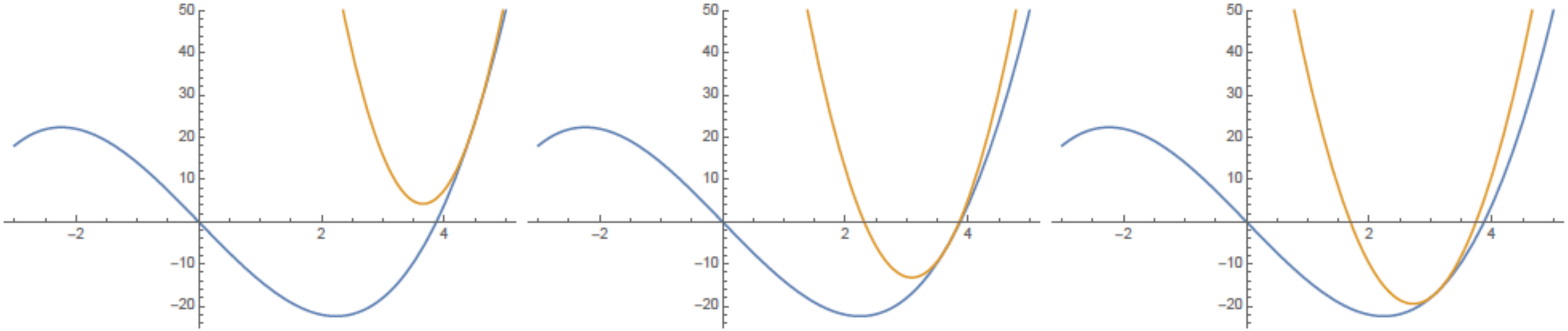
Iterative optimization method for smooth fn $J(w)$.
Often much faster than gradient descent.

Idea: You're at point v. Approximate $J(w)$ near v by quadratic fn.
Jump to its unique critical pt. Repeat until bored.

We say that $x = c$ is a critical point of the function $f(x)$ if $f(c)$ exists and if either of the following are true.

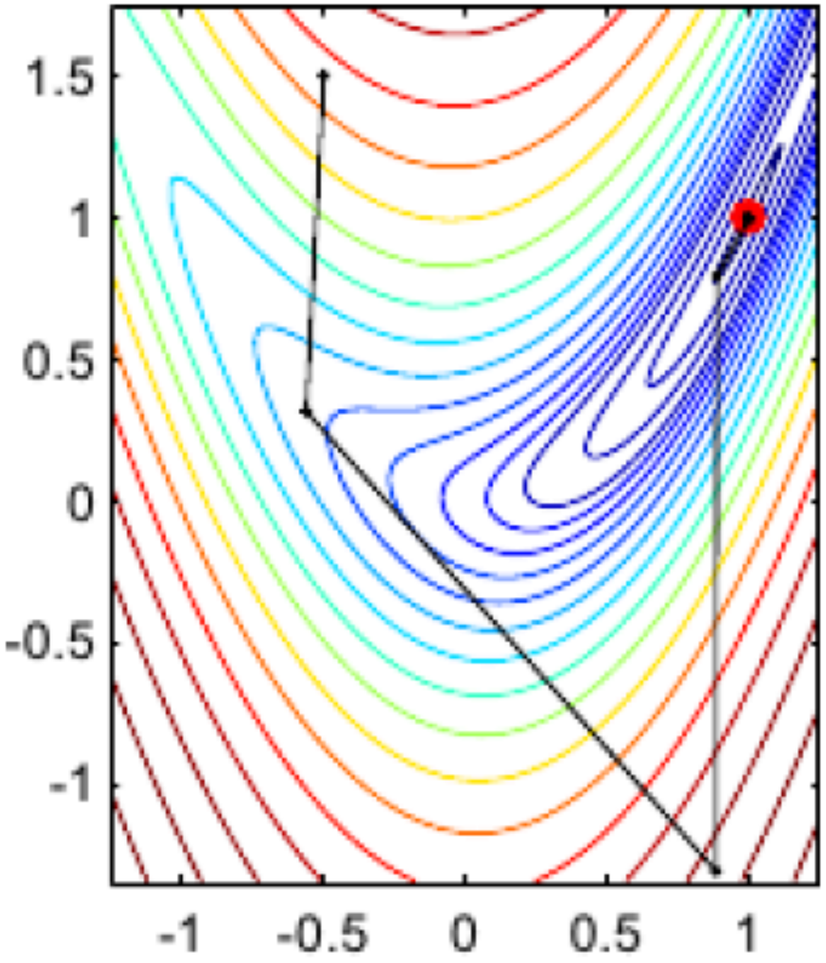
$f'(c) = 0$ OR $f'(c)$ doesn't exist

Newton's Method



newton1.pdf, newton2.pdf, newton3.pdf [Three iterations of Newton's method in one-dimensional space. We seek the minimum of the blue curve, J. Each brown curve is a local quadratic approximation to J. Each iteration, we jump to the bottom of the brown parabola.]

Newton's Method



newton2D.png [Steps taken by Newton's method in two-dimensional space.]

Newton's Method

Taylor series about v:

$$\nabla J(w) = \nabla J(v) + (\nabla^2 J(v))\,(w - v) + O(\|w - v\|^2)$$

where $\nabla^2 J(v)$ is the Hessian matrix of J at v.

Find critical pt w by setting $\nabla J(w) = 0$:

$$w = v - (\nabla^2 J(v))^{-1}\,\nabla J(v)$$

[This is an iterative update rule you can repeat until it converges to a solution. As usual, we probably don't want to compute a matrix inverse directly. It is faster to solve a linear system of equations, typically by Cholesky factorization or the conjugate gradient method.]

Newton's Method

Newton's method:

pick starting point w
repeat until convergence
$e \leftarrow$ solution to linear system $(\nabla^2 J(w))\, e = -\nabla J(w)$
$w \leftarrow w + e$

Warning: Doesn't know difference between minima, maxima, saddle pts.
Starting pt must be "close enough" to desired critical pt.

[If the objective function J is actually quadratic, Newton's method needs only one step to find the exact solution. The closer J is to quadratic, the faster Newton's method tends to converge.]

Newton's Method

[Newton's method is superior to blind gradient descent for some optimization problems for several reasons. First, it tries to find the right step length to reach the minimum, rather than just walking an arbitrary distance downhill. Second, rather than follow the direction of steepest descent, it tries to choose a better descent direction.]

[Nevertheless, it has some major disadvantages. The biggest one is that computing the Hessian can be quite expensive, and it has to be recomputed every iteration. It can work well for low-dimensional weight spaces, but you would never use it for a neural network, because there are too many weights. Newton's method also doesn't work for most nonsmooth functions. It particularly fails for the perceptron risk function, whose Hessian is zero, except where the Hessian is not even defined.]

Gradient Descent for Multivariate LR

Hypothesis: $h_\theta(x) = \theta^T x = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_n x_n$

Parameters: $\theta_0, \theta_1, \cdots, \theta_n$

Cost function:

$$J(\theta_0, \theta_1, \cdots, \theta_n) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^{(i)}) - y^{(i)})^2$$

Gradient descent:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{\partial}{\partial \theta_j} J(\theta_0, \theta_1, \cdots, \theta_n)$$

$\}$ (simultaneously update for every $j = 0, 1, \cdots, n$

Gradient Descent

Previously (n=1):

Repeat $\{$

$$\theta_0 := \theta_0 - \alpha \underbrace{\frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})}_{\frac{\partial}{\partial\theta_0}J(\theta)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x^{(i)}$$

(simultaneously update θ_0, θ_1)

$\}$

New algorithm $(n \geq 1)$:

Repeat $\{$

$$\theta_j := \theta_j - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_j^{(i)}$$

(simultaneously update θ_j for $j = 0, \ldots, n$)

$\}$

$$\theta_0 := \theta_0 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_0^{(i)}$$

$$\theta_1 := \theta_1 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_1^{(i)}$$

$$\theta_2 := \theta_2 - \alpha \frac{1}{m}\sum_{i=1}^{m}(h_\theta(x^{(i)}) - y^{(i)})x_2^{(i)}$$

$\ldots$

Appendix:

Appendix: Linear Algebra Basics

Transpose

- $(A^T)^T = A$
- $(AB)^T = B^T A^T$
- $(A+B)^T = A^T + B^T$

$$A = \begin{bmatrix} 0 & 1 \\ 2 & 4 \\ 1 & -1 \end{bmatrix} \Rightarrow A^T = \begin{bmatrix} 0 & 2 & 1 \\ 1 & 4 & -1 \end{bmatrix}$$

Symmetric Matrices

- A square matrix $A \in R^{n \times n}$ is symmetric if $A = A^T$

$$\begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 8 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 3 \\ 2 & 4 & 5 \\ 3 & 5 & 8 \end{bmatrix}^{\mathrm{T}}$$

Trace

- The trace of a square matrix $A \in R^{n \times n}$ is the sum of diagonal elements in the matrix:

$$\mathrm{tr}A = \sum_{i=1}^{n} A_{ii}$$

$$\mathrm{A} = \begin{bmatrix} 2 & 8 & 9 \\ 1 & 3 & 4 \\ 7 & 4 & 2 \end{bmatrix}$$

$$tr(\mathrm{A}) = 2+3+2 = 7$$

Appendix: Linear Algebra Basics

❑ **Inner product** (dot product or scalar product):

$$x^T y \in \mathbb{R} = \begin{bmatrix} x_1 & x_2 & \cdots & x_n \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix} = \sum_{i=1}^{n} x_i y_i$$

(note that always: $x^T y = y^T x$*)*

❑ **Outer product:**

$$xy^T \in \mathbb{R}^{m \times n} = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix} \begin{bmatrix} y_1 & y_2 & \cdots & y_n \end{bmatrix} = \begin{bmatrix} x_1 y_1 & x_1 y_2 & \cdots & x_1 y_n \\ x_2 y_1 & x_2 y_2 & \cdots & x_2 y_n \\ \vdots & \vdots & \ddots & \vdots \\ x_m y_1 & x_m y_2 & \cdots & x_m y_n \end{bmatrix}$$

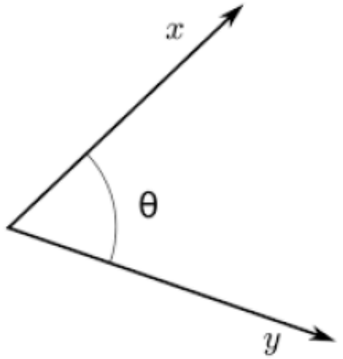
Appendix: Linear Algebra Basics

- **Euclidean norm**, or ℓ2-norm, of a vector x ∈ R^n:

$$\|x\|_2 = (x^T x)^{1/2} = (x_1^2 + \cdots + x_n^2)^{1/2}.$$

- **Angle** between nonzero vectors x, y ∈ R^n:

$$\angle(x, y) = \cos^{-1}\left(\frac{x^T y}{\|x\|_2 \|y\|_2}\right)$$



- Two vectors x, y ∈ R^n are said:
  - **Orthogonal:** $x^T y = 0$
  - **Orthonormal:** $x^T y = 0$ and $\|x\| = \|y\| = 1$

Appendix: Linear Algebra -linearly dependent

- **Linearly dependent:**
  - A set of vectors $\{x_1, x_2, \ldots x_n\} \subset R^m$ is said to be **linearly dependent** If there exists a set of coefficients $a_1, a_2, \ldots, a_n$ (at least one different than zero) such that:

$$a_1x_1 + a_2x_2 + \ldots, +a_nx_n = 0$$

  - Otherwise the vectors is said to be **linearly independent**

- For example the following examples are linearly dependent because $x_3 = -2x_1 + x_2$

$$x_1 = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \quad x_2 = \begin{bmatrix} 4 \\ 1 \\ 5 \end{bmatrix} \quad x_3 = \begin{bmatrix} 2 \\ -3 \\ -1 \end{bmatrix}$$

Appendix: Linear Algebra - inverse

Inverse

- The inverse of a square matrix $A \in R^{n \times n}$ is denoted A^{-1} such that

$$A^{-1}A = I = AA^{-1}$$

Non-Singular (invertible) and Singular (non-invertible)

- A^{-1} exists if and only if A is non-singular. Otherwise:

Pseudo Inverse

- The pseudo-inverse matrix $A^{\dagger}$ is typically used whenever A^{-1} does not exist (because A is not square or A is singular):

$$A^{\dagger} = [A^T A]^{-1} A^T \qquad (Note\ A^{\dagger}A = I)$$

  - Assuming $[A^T A]$ is non-singular

Appendix: Linear Algebra - inverse

Inverse

- $(A^{-1})^{-1} = A$
- $(AB)^{-1} = B^{-1}A^{-1}$
- $(A^{-1})^{\top} = (A^{\top})^{-1}$

Appendix: Matrix derivatives

- Suppose $f: R^{m \times n} \rightarrow R$ is a function that takes as input a matrix A of size $m \times n$ and **returns a real value (scalar)**.
- The derivation (gradient) of f (with respect to $\boldsymbol{A} \in R^{m \times n}$) is defined as:

$$\frac{\partial f(A)}{\partial A}_{\cdot} \in \mathbb{R}^{m \times n} = \begin{bmatrix} \frac{\partial f(A)}{\partial A_{11}} & \frac{\partial f(A)}{\partial A_{12}} & \cdots & \frac{\partial f(A)}{\partial A_{1n}} \\ \frac{\partial f(A)}{\partial A_{21}} & \frac{\partial f(A)}{\partial A_{22}} & \cdots & \frac{\partial f(A)}{\partial A_{2n}} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f(A)}{\partial A_{m1}} & \frac{\partial f(A)}{\partial A_{m2}} & \cdots & \frac{\partial f(A)}{\partial A_{mn}} \end{bmatrix}$$

- In particular, for a vector $\boldsymbol{x} \in R^{n \times 1}$, i.e. $f(\boldsymbol{x}): R^n \rightarrow R$:

$$\frac{\partial f(x)}{\partial x} = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \frac{\partial f(x)}{\partial x_2} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}$$

Appendix: Matrix derivatives

- $\frac{\partial (a^T x)}{\partial x} = \frac{\partial (x^T a)}{\partial x} = a$ a and x are a vector, i.e., $a, x \in R^n$
- $\frac{\partial (a^T X b)}{\partial X} = \frac{\partial (b^T X^T a)}{\partial X} = ab^T$ X *is* a Matrix
- $\frac{\partial (x^T x)}{\partial x} = 2x$
- $\frac{\partial (x^T A x)}{\partial x} = (A + A^T)x = 2Ax \ (if \ A \ is \ symetirc)$ A *is* a Matrix

Credits

- Previous syde675 course offerings
- Berkeley (Sahai, Tobin, Shewchuk)
- CMU (Paczos, Bar-Joseph)
- ..