Lecture 8: 2 February, 2023

Madhavan Mukund

https://www.cmi.ac.in/~madhavan

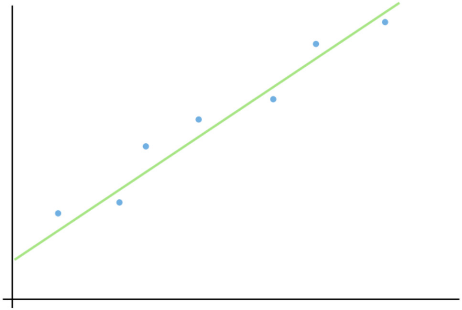
Data Mining and Machine Learning
January–April 2023

Linear regression

- Training input is $\{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$
  - Each input x_i is a vector $(x_i^1, \dots, x_i^k)$
  - Add $x_i^0 = 1$ by convention
  - y_i is actual output
- How far away is our prediction $h_\theta(x_i)$ from the true answer y_i?
- Define a cost (loss) function

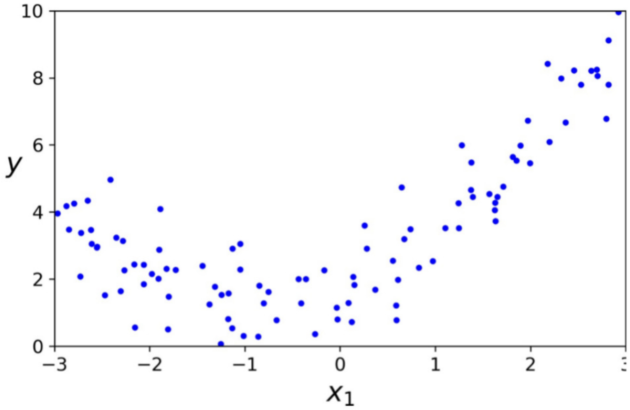
$$J(\theta) = \frac{1}{2}\sum_{i=1}^{n}(h_\theta(x_i) - y_i)^2$$

- Essentially, the sum squared error (SSE)
- Divide by n, mean squared error (MSE)



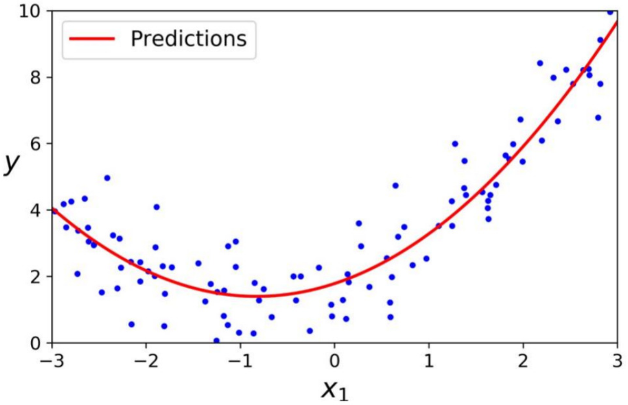
The non-linear case

- What if the relationship is not linear?



The non-linear case

- What if the relationship is not linear?
- Here the best possible explanation seems to be a quadratic
- Non-linear : cross dependencies
- Input $x_i : (x_{i_1}, x_{i_2})$
- Quadratic dependencies:
  $y = \theta_0 + \theta_1 x_{i_1} + \theta_2 x_{i_2} + \theta_{11} x_{i_1}^2 + \theta_{22} x_{i_2}^2 + \theta_{12} x_{i_1} x_{i_2}$



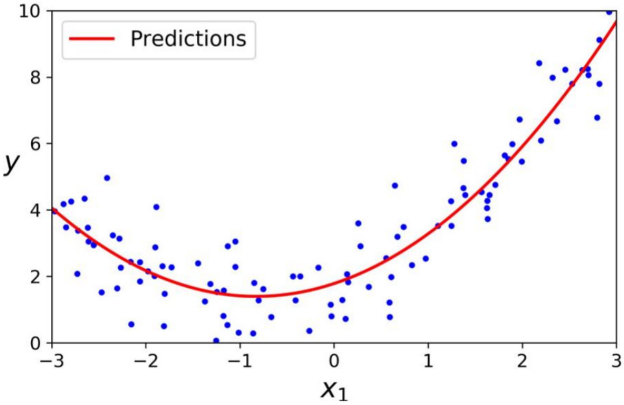
The non-linear case

- Recall how we fit a line

$$\begin{bmatrix} 1 & x_{i_1} \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

- For quadratic, add new coefficients and expand parameters

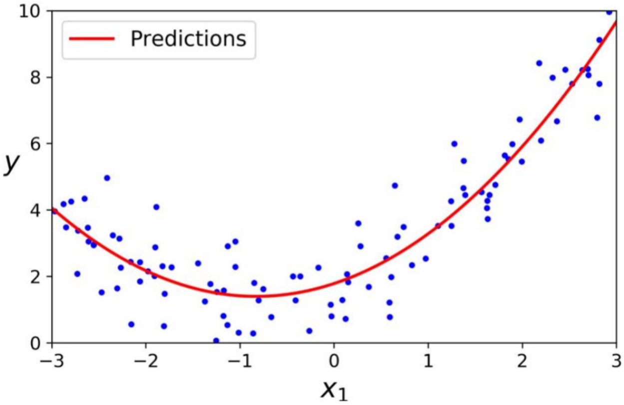
$$\begin{bmatrix} 1 & x_{i_1} & x_{i_1}^2 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$



The non-linear case

- Input (x_{i_1}, x_{i_2})
- For the general quadratic case, we are adding new derived "features"

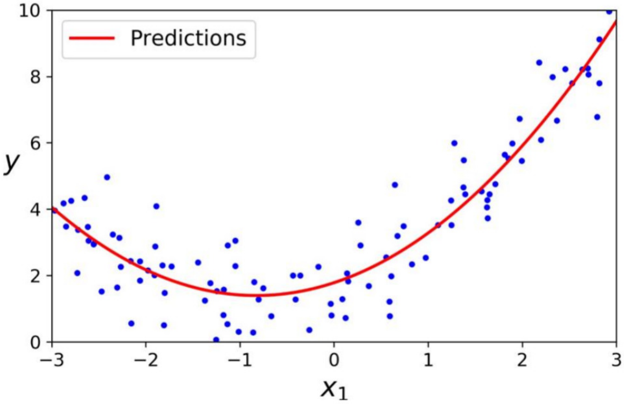
$$
\begin{aligned}
x_{i_3} &= x_{i_1}^2 \\
x_{i_4} &= x_{i_2}^2 \\
x_{i_5} &= x_{i_1} x_{i_2}
\end{aligned}
$$



The non-linear case

- Original input matrix

$$\begin{bmatrix} 1 & x_{1_1} & x_{1_2} \\ 1 & x_{2_1} & x_{2_2} \\ & \cdots & \\ 1 & x_{i_1} & x_{i_2} \\ & \cdots & \\ 1 & x_{n_1} & x_{n_2} \end{bmatrix}$$

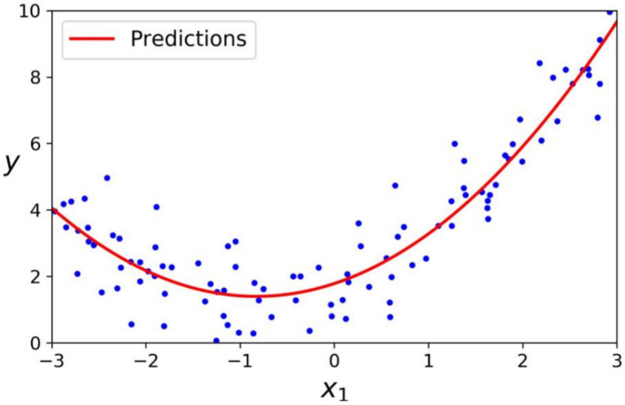


The non-linear case

- Expanded input matrix

$$\begin{bmatrix} 1 & x_{1_1} & x_{1_2} & x_{1_1}^2 & x_{1_2}^2 & x_{1_1}x_{1_2} \\ 1 & x_{2_1} & x_{2_2} & x_{2_1}^2 & x_{2_2}^2 & x_{2_1}x_{2_2} \\ & \cdots \\ 1 & x_{i_1} & x_{i_2} & x_{i_1}^2 & x_{i_2}^2 & x_{i_1}x_{i_2} \\ & \cdots \\ 1 & x_{n_1} & x_{n_2} & x_{n_1}^2 & x_{n_2}^2 & x_{n_1}x_{n_2} \end{bmatrix}$$

- New columns are computed and filled in from original inputs



Exponential parameter blow-up

- Cubic derived features

$x_{i_1}^3,\ x_{i_2}^3,\ x_{i_3}^3,$

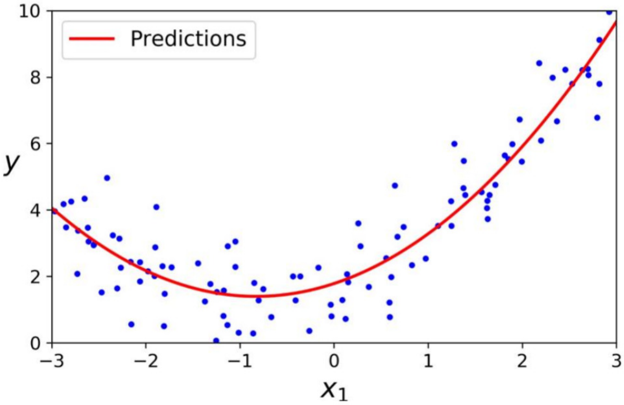
$x_{i_1}^2 x_{i_2},\ x_{i_1}^2 x_{i_3},$
$x_{i_2}^2 x_{i_1},\ x_{i_2}^2 x_{i_3},$
$x_{i_3}^2 x_{i_1},\ x_{i_3}^2 x_{i_2},$

$x_{i_1} x_{i_2} x_{i_3},$

$x_{i_1}^2,\ x_{i_2}^2,\ x_{i_3}^2,$

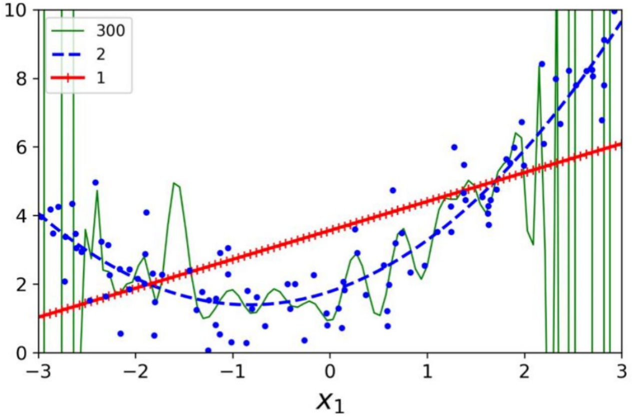
$x_{i_1} x_{i_2},\ x_{i_1} x_{i_3},\ x_{i_2} x_{i_3},$

$x_{i_1},\ x_{i_2},\ x_{i_3}.$



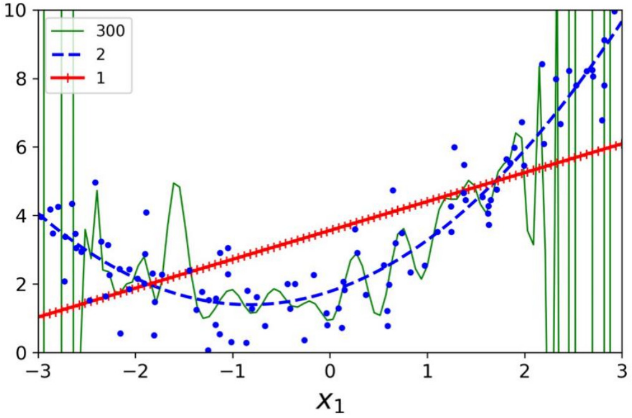
Higher degree polynomials

- How complex a polynomial should we try?
- Aim for degree that minimizes SSE
- As degree increases, features explode exponentially



Overfitting

- Need to be careful about adding higher degree terms
- For n training points,can always fit polynomial of degree $(n-1)$ exactly
- However, such a curve would not generalize well to new data points
- Overfitting — model fits training data well, performs poorly on unseen data

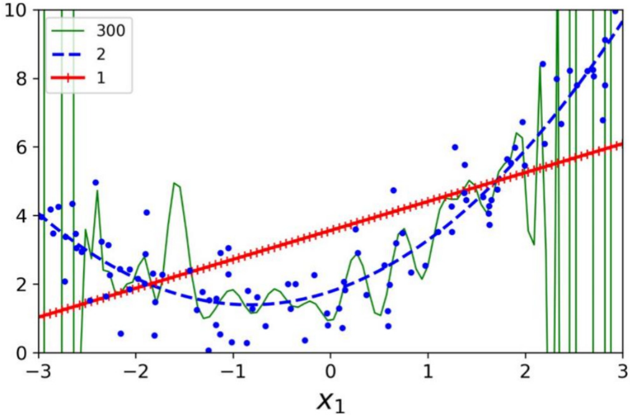


Regularization

- Need to trade off SSE against curve complexity
- So far, the only cost has been SSE
- Add a cost related to parameters $(\theta_0, \theta_1, \ldots, \theta_k)$
- Minimize, for instance

$$\frac{1}{2}\sum_{i=1}^{n}(z_i - y_i)^2 + \sum_{j=1}^{k}\theta_j^2$$

- Second term penalizes curve complexity



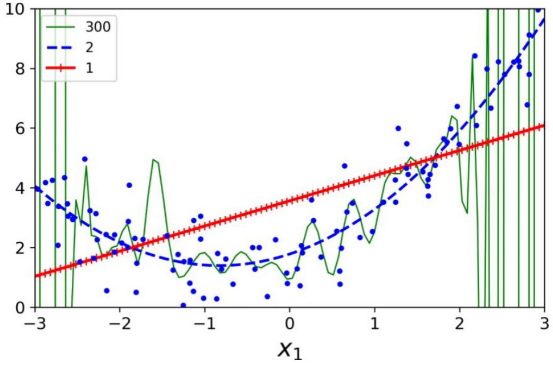
Regularization

- Variations on regularization
  - Change the contribution of coefficients to the loss function
- Ridge regression:

  Coefficients contribute $\sum_{j=1}^{k} \theta_j^2$
- LASSO regression:

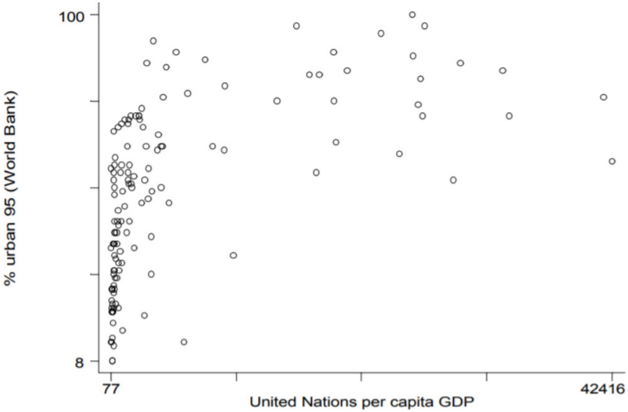
  Coefficients contribute $\sum_{j=1}^{k} |\theta_j|$
- Elastic net regression:

  Coefficients contribute $\sum_{j=1}^{k} \lambda_1|\theta_j| + \lambda_2\theta_j^2$



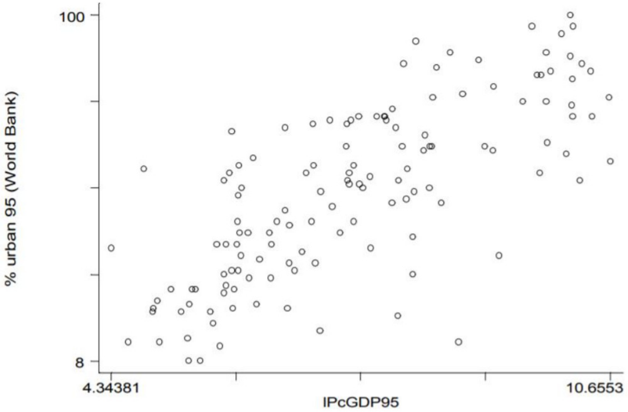
The non-polynomial case

- Percentage of urban population as a function of per capita GDP
- Not clear what polynomial would be reasonable



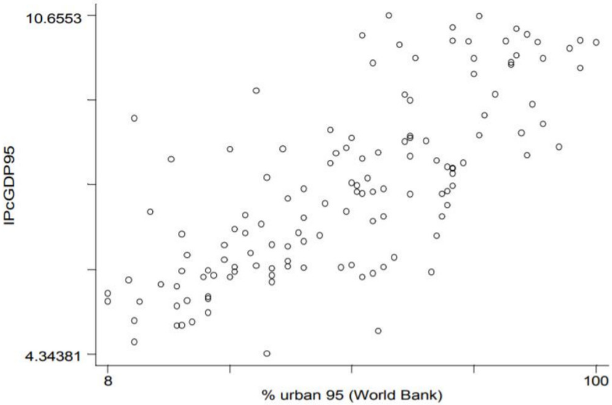
The non-polynomial case

- Percentage of urban population as a function of per capita GDP
- Not clear what polynomial would be reasonable
- Take log of GDP
- Regression we are computing is $y = \theta_0 + \theta_1 \log x_1$



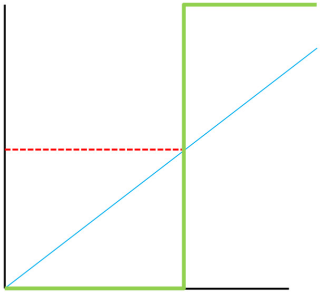
The non-polynomial case

- Reverse the relationship
- Plot per capita GDP in terms of percentage of urbanization
- Now we take log of the output variable
  $\log y = \theta_0 + \theta_1 x_1$
- Log-linear transformation
- Earlier was linear-log
- Can also use log-log



Regression for classification

- Regression line
- Set a threshold
- Classifier
  - Output below threshold : 0 (No)
  - Output above threshold : 1 (Yes)
- Classifier output is a step function



Smoothen the step

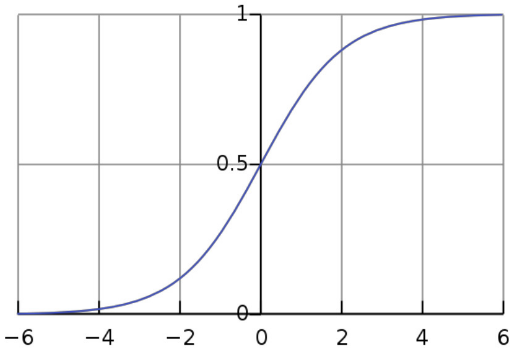
- Sigmoid function

$$\sigma(z) = \frac{1}{1+e^{-z}}$$

- Input z is output of our regression

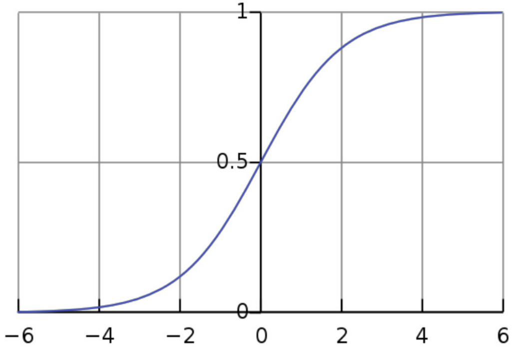
$$\sigma(z) = \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+\cdots+\theta_k x_k)}}$$

- Adjust parameters to fix horizontal position and steepness of step



Logistic regression

- Compute the coefficients?
- Solve by gradient descent
- Need derivatives to exist
  - Hence smooth sigmoid, not step function
  - Check that $\sigma'(z) = \sigma(z)(1 - \sigma(z))$
- Need a cost function to minimize



MSE for logistic regression and gradient descent

- Suppose we take mean squared error as the loss function.

$$C = \frac{1}{n}\sum_{i=1}^{n}(y_i - \sigma(z_i))^2, \text{ where } z_i = \theta_0 + \theta_1 x_{i_1} + \theta_2 x_{i_2}$$

- For gradient descent, we compute $\frac{\partial C}{\partial \theta_1}, \frac{\partial C}{\partial \theta_2}, \frac{\partial C}{\partial \theta_0}$
- Consider two inputs $x = (x_1, x_2)$
  - For $j = 1, 2$,

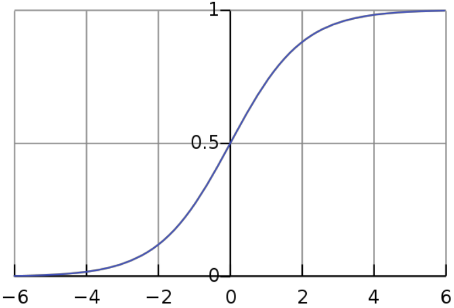
$$\frac{\partial C}{\partial \theta_j} = \frac{2}{n}\sum_{i=1}^{n}(y_i - \sigma(z_i)) \cdot -\frac{\partial \sigma(z_i)}{\partial \theta_j} = \frac{2}{n}\sum_{i=1}^{n} (\sigma(z_i) - y_i)\, \frac{\partial \sigma(z_i)}{\partial z_i}\frac{\partial z_i}{\partial \theta_j}$$

$$= \frac{2}{n}\sum_{i=1}^{n}(\sigma(z_i) - y_i)\sigma'(z_i)x_{i_j}$$

  - $\frac{\partial C}{\partial \theta_0} = \frac{2}{n}\sum_{i=1}^{n}(\sigma(z_i) - y_i)\frac{\partial \sigma(z_i)}{\partial z_i}\frac{\partial z_i}{\partial \theta_0} = \frac{2}{n}\sum_{i=1}^{n}(\sigma(z_i) - y_i)\sigma'(z_i)$

MSE for logistic regression and gradient descent . . .

- For $j = 1, 2$, $\dfrac{\partial C}{\partial \theta_j} = \dfrac{2}{n}\sum_{i=1}^{n}(\sigma(z_i) - y_i)\sigma'(z_i)x_j^i$, and $\dfrac{\partial C}{\partial \theta_0} = \dfrac{2}{n}\sum_{i=1}^{n}(\sigma(z_i) - y_i)\sigma'(z_i)$
- Each term in $\dfrac{\partial C}{\partial \theta_1}, \dfrac{\partial C}{\partial \theta_2}, \dfrac{\partial C}{\partial \theta_0}$ is proportional to $\sigma'(z_i)$
- Ideally, gradient descent should take large steps when $\sigma(z) - y$ is large
- $\sigma(z)$ is flat at both extremes
- If $\sigma(z)$ is completely wrong, $\sigma(z) \approx (1 - y)$, we still have $\sigma'(z) \approx 0$
- Learning is slow even when current model is far from optimal



Loss function for logistic regression

- Goal is to maximize log likelihood
- Let $h_\theta(x_i) = \sigma(z_i)$. So, $P(y_i = 1 \mid x_i; \theta) = h_\theta(x_i)$,
  $P(y_i = 0 \mid x_i; \theta) = 1 - h_\theta(x_i)$
- Combine as $P(y_i \mid x_i; \theta) = h_\theta(x_i)^{y_i} \cdot (1 - h_\theta(x_i))^{1-y_i}$
- Likelihood: $\mathcal{L}(\theta) = \prod_{i=1}^{n} h_\theta(x_i)^{y_i} \cdot (1 - h_\theta(x_i))^{1-y_i}$
- Log-likelihood: $\ell(\theta) = \sum_{i=1}^{n} y_i \log h_\theta(x_i) + (1 - y_i) \log(1 - h_\theta(x_i))$
- Minimize cross entropy: $-\sum_{i=1}^{n} y_i \log h_\theta(x_i) + (1 - y_i) \log(1 - h_\theta(x_i))$

Cross entropy and gradient descent

- $C = -[y \ln(\sigma(z)) + (1-y)\ln(1-\sigma(z))]$

- $$\begin{aligned}\frac{\partial C}{\partial \theta_j} = \frac{\partial C}{\partial \sigma}\frac{\partial \sigma}{\partial \theta_j} &= -\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\frac{\partial \sigma}{\partial \theta_j} \\ &= -\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\frac{\partial \sigma}{\partial z}\frac{\partial z}{\partial \theta_j} \\ &= -\left[\frac{y}{\sigma(z)} - \frac{1-y}{1-\sigma(z)}\right]\sigma'(z)x_j \\ &= -\left[\frac{y(1-\sigma(z)) - (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right]\sigma'(z)x_j\end{aligned}$$

Cross entropy and gradient descent . . .

- $\dfrac{\partial C}{\partial \theta_j} = -\left[\dfrac{y(1-\sigma(z)) - (1-y)\sigma(z)}{\sigma(z)(1-\sigma(z))}\right]\sigma'(z)x_j$

- Recall that $\sigma'(z) = \sigma(z)(1-\sigma(z))$

- Therefore, $\begin{aligned}\dfrac{\partial C}{\partial \theta_j} &= -[y(1-\sigma(z)) - (1-y)\sigma(z)]x_j \\ &= -[y - y\sigma(z) - \sigma(z) + y\sigma(z)]x_j \\ &= (\sigma(z) - y)x_j\end{aligned}$

- Similarly, $\dfrac{\partial C}{\partial \theta_0} = (\sigma(z) - y)$

- Thus, as we wanted, the gradient is proportional to $\sigma(z) - y$

- The greater the error, the faster the learning rate