Lecture 6: 10 February, 2022

Madhavan Mukund

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

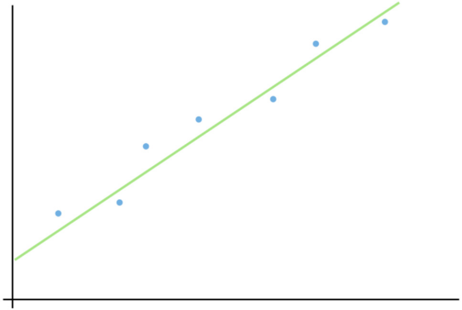
Data Mining and Machine Learning
January–May 2022

Finding the best fit line

- Training input is $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
  - Each input x_i is a vector $(x_i^1, \ldots, x_i^k)$
  - 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)

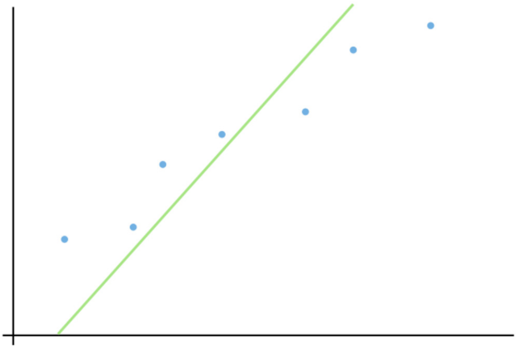


Minimizing SSE

- Write x_i as row vector $\left[\begin{array}{cccc} 1 & x_i^1 & \cdots & x_i^k \end{array}\right]$
- $X = \left[\begin{array}{cccc} 1 & x_1^1 & \cdots & x_1^k \\ 1 & x_2^1 & \cdots & x_2^k \\ & \cdots & & \\ 1 & x_i^1 & \cdots & x_i^k \\ & \cdots & & \\ 1 & x_n^1 & \cdots & x_n^k \end{array}\right]$, $y = \left[\begin{array}{c} y_1 \\ y_2 \\ \cdots \\ y_i \\ \cdots \\ y_n \end{array}\right]$
- Write θ as column vector, $\theta^T = \left[\begin{array}{cccc} \theta_0 & \theta_1 & \cdots & \theta_k \end{array}\right]$
- $J(\theta) = \dfrac{1}{2}\sum_{i=1}^{n}(h_\theta(x_i) - y_i)^2 = \dfrac{1}{2}(X\theta - y)^T(X\theta - y)$
- Minimize $J(\theta)$ — set $\nabla_\theta\, J(\theta) = 0$

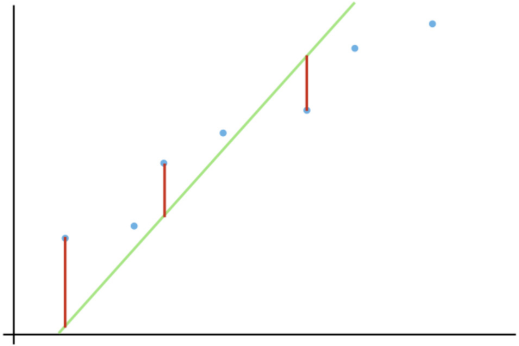
Minimizing SSE iteratively

- Normal equation $\theta = (X^T X)^{-1} X^T y$ is a closed form solution
- Computational challenges
  - Slow if n large, say $n > 10^4$
  - Matrix inversion $(X^T X)^{-1}$ is expensive, also need invertibility
- Iterative approach, make an initial guess



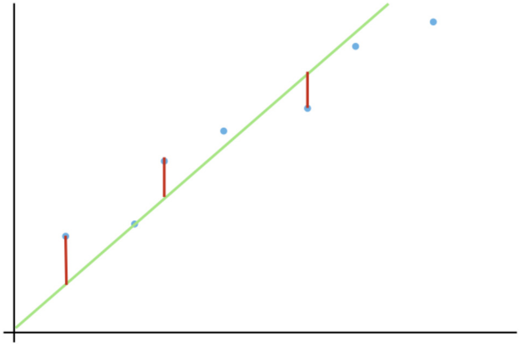
Minimizing SSE iteratively

- Normal equation $\theta = (X^T X)^{-1} X^T y$ is a closed form solution
- Computational challenges
  - Slow if n large, say $n > 10^4$
  - Matrix inversion $(X^T X)^{-1}$ is expensive, also need invertibility
- Iterative approach, make an initial guess
- Keep adjusting the line to reduce SSE



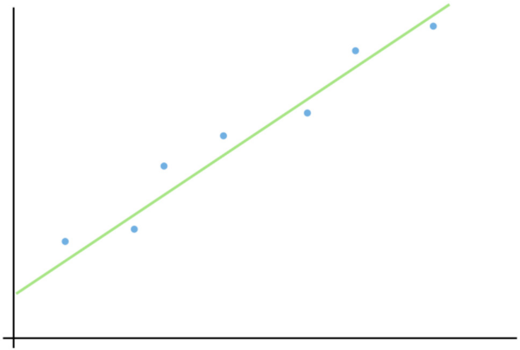
Minimizing SSE iteratively

- Normal equation $\theta = (X^T X)^{-1} X^T y$ is a closed form solution
- Computational challenges
  - Slow if n large, say $n > 10^4$
  - Matrix inversion $(X^T X)^{-1}$ is expensive, also need invertibility
- Iterative approach, make an initial guess
- Keep adjusting the line to reduce SSE
- Stop when we find the best fit line



Minimizing SSE iteratively

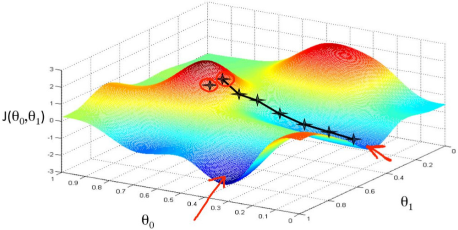
- Normal equation $\theta = (X^T X)^{-1} X^T y$ is a closed form solution
- Computational challenges
  - Slow if n large, say $n > 10^4$
  - Matrix inversion $(X^T X)^{-1}$ is expensive, also need invertibility
- Iterative approach, make an initial guess
- Keep adjusting the line to reduce SSE
- Stop when we find the best fit line
- How do we adjust the line?



Gradient descent

- How does cost vary with parameters $\theta = (\theta_0, \theta_1, \ldots, \theta_k)$?
  - Gradients $\dfrac{\partial}{\partial \theta_i} J(\theta)$
- Adjust each parameter against gradient
  - $\theta_i = \theta_i - \alpha \dfrac{\partial}{\partial \theta_i} J(\theta)$
- For a single training sample (x, y)

$$
\begin{aligned}
\frac{\partial}{\partial \theta_i} J(\theta) &= \frac{\partial}{\partial \theta_i} \frac{1}{2} (h_\theta(x) - y)^2 \\
&= 2 \cdot \frac{1}{2} (h_\theta(x) - y) \frac{\partial}{\partial \theta_i} (h_\theta(x) - y) \\
&= (h_\theta(x) - y) \frac{\partial}{\partial \theta_i} \left[\left(\sum_{j=0}^{k} \theta_j x_j \right) - y \right] \quad = \quad (h_\theta(x) - y) \cdot x_i
\end{aligned}
$$



Gradient descent

- For a single training sample (x, y), $\frac{\partial}{\partial \theta_i} J(\theta) = (h_\theta(x) - y) \cdot x_i$
- Over the entire training set, $\frac{\partial}{\partial \theta_i} J(\theta) = \sum_{j=1}^{n} (h_\theta(x_j) - y_j) \cdot x_j^i$

Batch gradient descent

- Compute $h_\theta(x_j)$ for entire training set $\{(x_1, y_1), \ldots, (x_n, y_n)\}$
- Adjust each parameter

$$\begin{aligned} \theta_i &= \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta) \\ &= \theta_i - \alpha \cdot \sum_{j=1}^{n} (h_\theta(x_j) - y_j) \cdot x_j^i \end{aligned}$$

- Repeat until convergence

Stochastic gradient descent

- For each input x_j, compute $h_\theta(x_j)$
- Adjust each parameter — $\theta_i = \theta_i - \alpha \cdot (h_\theta(x_j) - y) \cdot x_j^i$

Pros and cons

- Faster progress for large batch size
- May oscillate indefinitely

Regression and SSE loss

- Training input is $\{(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)\}$
  - Noisy outputs from a linear function
  - $y_i = \theta^T x_i + \epsilon$
  - $\epsilon \sim \mathcal{N}(0, \sigma^2)$: Gaussian noise, mean 0, fixed variance σ^2
  - $y_i \sim \mathcal{N}(\mu_i, \sigma^2)$, $\mu_i = \theta^T x_i$
- Model gives us an estimate for θ, so regression learns μ_i for each x_i
- Want Maximum Likelihood Estimator (MLE) — maximize

$$\mathcal{L}(\theta) = \prod_{i=1}^{n} P(y_i \mid x_i; \theta)$$

- Instead, maximize log likelihood

$$\ell(\theta) = \log\left(\prod_{i=1}^{n} P(y_i \mid x_i; \theta)\right) = \sum_{i=1}^{n} \log(P(y_i \mid x_i; \theta))$$

Log likelihood and SSE loss

- $y_i = \mathcal{N}(\mu_i, \sigma^2)$, so $P(y_i \mid x_i; \theta) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\mu_i)^2}{2\sigma^2}} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\theta^T x_i)^2}{2\sigma^2}}$
- Log likelihood (assuming natural logarithm)

$$\ell(\theta) = \sum_{i=1}^{n} \log\left(\frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(y-\theta^T x_i)^2}{2\sigma^2}}\right) = n\log\left(\frac{1}{\sqrt{2\pi\sigma^2}}\right) - \sum_{i=1}^{n} \frac{(y-\theta^T x_i)^2}{2\sigma^2}$$

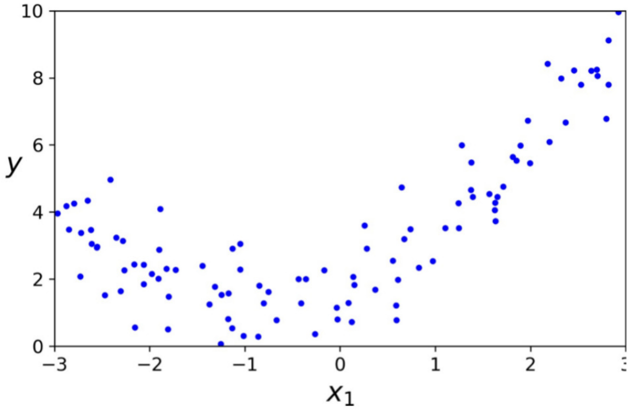
- To maximize $\ell(\theta)$ with respect to θ, ignore all terms that do not depend on θ
- Optimum value of θ is given by

$$\hat{\theta}_{\text{MSE}} = \arg\max_{\theta}\left[-\sum_{i=1}^{n}(y_i - \theta^T x_i)^2\right] = \arg\min_{\theta}\left[\sum_{i=1}^{n}(y_i - \theta^T x_i)^2\right]$$

- Assuming data points are generated by linear function and then perturbed by Gaussian noise, SSE is the “correct” loss function to maximize likelihood

The non-linear case

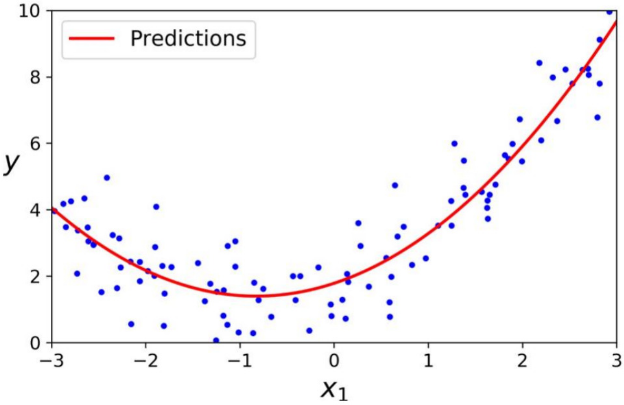
- What if the relationship is not linear?



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:

  $y = \theta_0 + \theta_1 x_{i_1} + \theta_2 x_{i_2} + \theta_{11} x_{i_1}^2 + \theta_{22} x_{i_2}^2 + \theta_{12} x_{i_1} x_{i_2}$



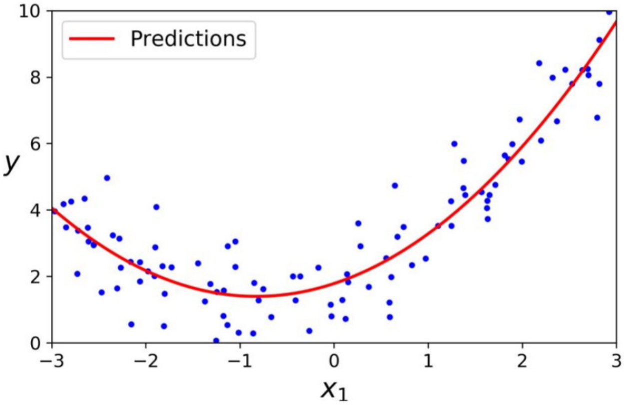
The non-linear case

- Recall how we fit a line

$$\begin{bmatrix} 1 & x_i \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \end{bmatrix}$$

- For quadratic, add new coefficients and expand parameters

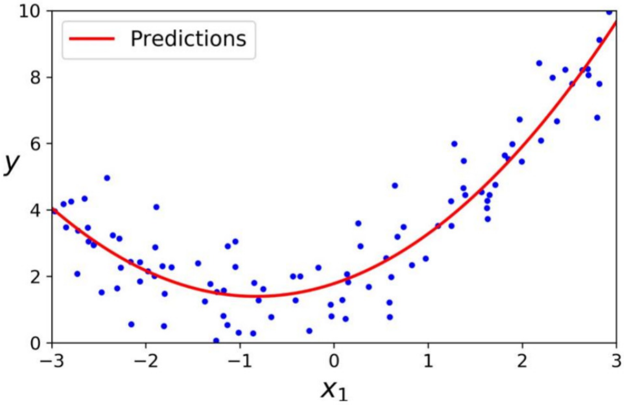
$$\begin{bmatrix} 1 & x_i & x_i^2 \end{bmatrix} \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \end{bmatrix}$$



The non-linear case

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

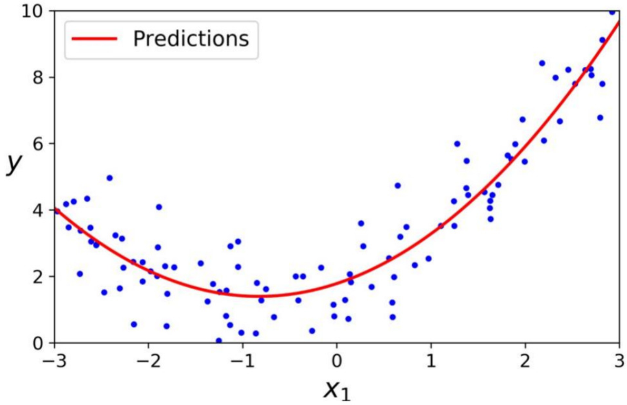
$$
\begin{aligned}
x_{i_3} &= x_{i_1}^2 \\
x_{i_4} &= x_{i_2}^2 \\
x_{i_5} &= x_{i_1} x_{i_2}
\end{aligned}
$$



The non-linear case

- Original input matrix

$$\begin{bmatrix} 1 & x_{1_1} & x_{1_2} \\ 1 & x_{2_1} & x_{2_2} \\ & \cdots & \\ 1 & x_{i_1} & x_{i_2} \\ & \cdots & \\ 1 & x_{n_1} & x_2 \end{bmatrix}$$

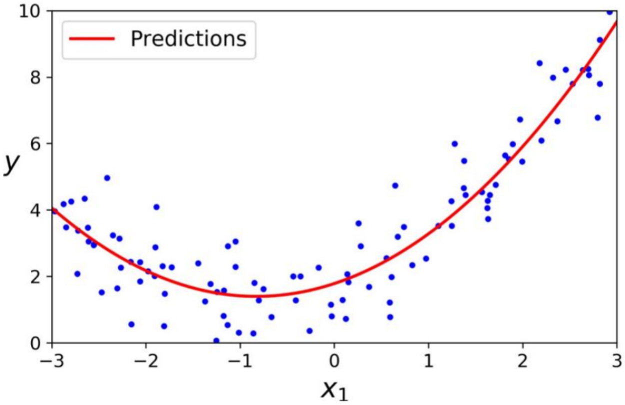


The non-linear case

- 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

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

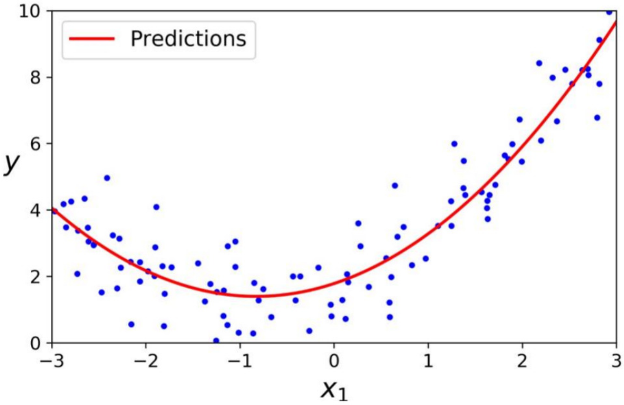
$x_{i_1}^2 x_{i_2},\ x_{i_1}^2 x_{i_3},$
$x_{i_2}^2 x_{i_1},\ x_{i_2}^2 x_{i_3},$
$x_{i_3}^2 x_{i_1},\ x_{i_3}^2 x_{i_2},$

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

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

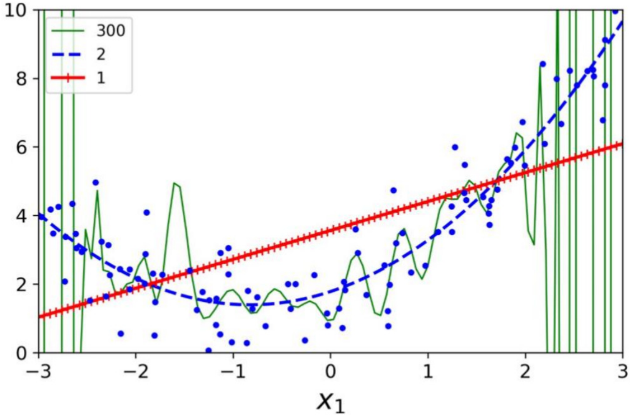
$x_{i_1} x_{i_2},\ x_{i_1} x_{i_3},\ x_{i_2} x_{i_3},$

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



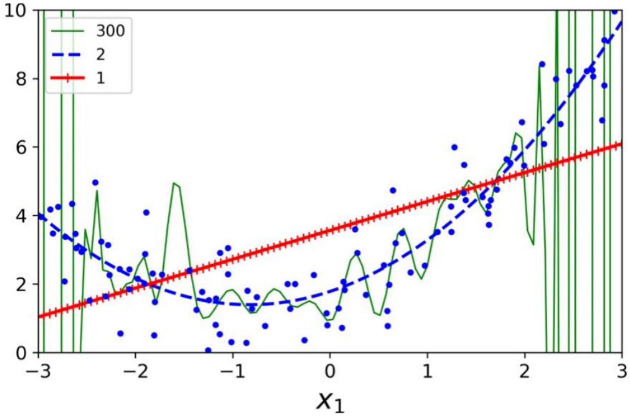
Higher degree polynomials

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



Overfitting

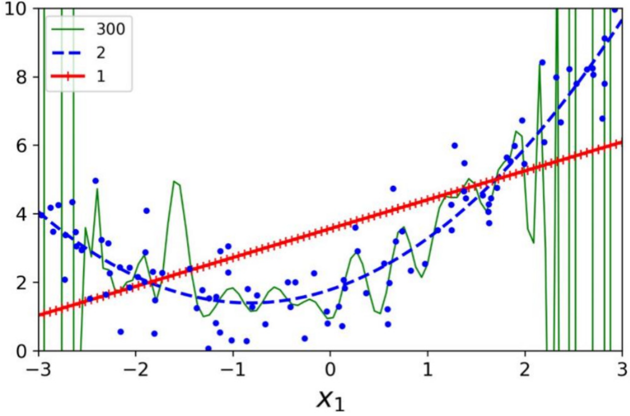
- Need to be careful about adding higher degree terms
- For n training points,can always fit polynomial of degree $(n-1)$ exactly
- However, such a curve would not generalize well to new data points
- Overfitting — model fits training data well, performs poorly on unseen data



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

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



Regularization

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

- Second term penalizes curve complexity
- Variations on regularatization
  - Ridge regression: $\sum_{j=1}^{k}\theta_j^2$
  - LASSO regression: $\sum_{j=1}^{k}|\theta_j|$
  - Elastic net regression: $\sum_{j=1}^{k}\lambda_1|\theta_j| + \lambda_2\theta_j^2$