# Lecture 10: Linear Regression

Madhavan Mukund

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

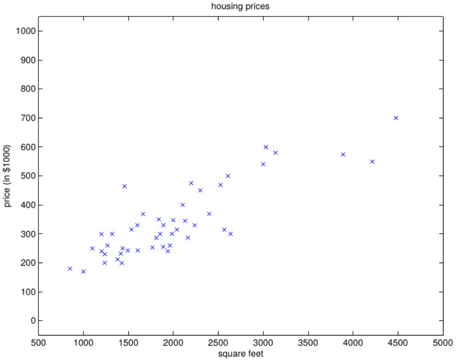
Data Mining and Machine Learning
August–December 2020

# Predicting numerical values

- Data about housing prices
- Predict house price from living area

| Living area ($feet^2$) | Price (1000$s) |
|---|---|
| 2104 | 400 |
| 1600 | 330 |
| 2400 | 369 |
| 1416 | 232 |
| 3000 | 540 |
| ⋮ | ⋮ |

- Scatterplot corresponding to the data
- Fit a function to the points



# Linear predictors

- A richer set of input data
- Simplest case: fit a linear function with parameters $\theta = (\theta_0, \theta_1, \theta_2)$

  $h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$
- Input $x$ may have $k$ features $(x_1, x_2, \ldots, x_k)$
- By convention, add a dummy feature $x_0 = 1$
- For $k$ input features

  $$h_\theta(x) = \sum_{i=0}^{k} \theta_i x_i$$

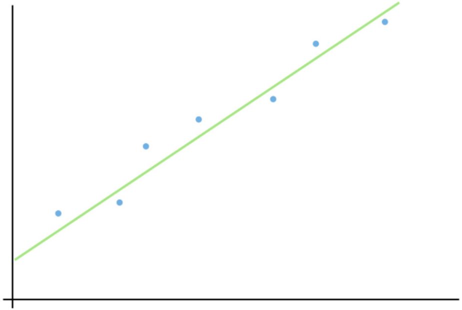
| Living area (feet$^2$) | #bedrooms | Price (1000$s) |
|---|---|---|
| 2104 | 3 | 400 |
| 1600 | 3 | 330 |
| 2400 | 3 | 369 |
| 1416 | 2 | 232 |
| 3000 | 4 | 540 |
| ⋮ | ⋮ | ⋮ |

# 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

$$J(\theta) = \frac{1}{2} \sum_{i=0}^{n} (h_\theta(x_i) - y_i)^2$$

- 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 $\theta$ as column vector, $\theta^T = \left[\begin{array}{cccc} \theta_0 & \theta_1 & \cdots & \theta_k \end{array}\right]$

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

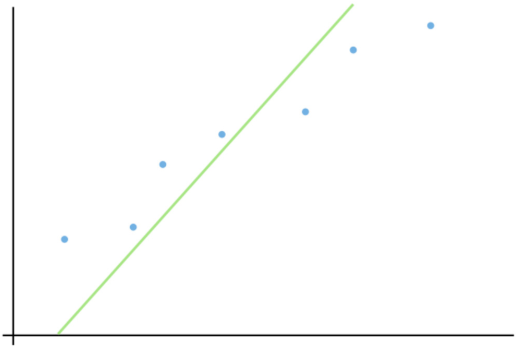
- Minimize $J(\theta)$ — set $\nabla_\theta\, J(\theta) = 0$

# Minimizing SSE

- $J(\theta) = \dfrac{1}{2}(X\theta - y)^T(X\theta - y)$
- $\nabla_\theta\ J(\theta) = \nabla_\theta\ \frac{1}{2}(X\theta - y)^T(X\theta - y)$
- To minimize, set $\nabla_\theta\ \frac{1}{2}(X\theta - y)^T(X\theta - y) = 0$
- Expand, $\frac{1}{2}\nabla_\theta\ (\theta^T X^T X\theta - y^T X\theta - \theta^T X^T y + y^T y) = 0$
  - Check that $y^T X\theta = \theta^T X^T y = \sum_{i=1}^{n} h_\theta(x_i) \cdot y_i$
- Combining terms, $\frac{1}{2}\nabla_\theta\ (\theta^T X^T X\theta - 2\theta^T X^T y + y^T y) = 0$
- After differentiating, $X^T X\theta - X^T y = 0$
- Solve to get normal equation, $\theta = (X^T X)^{-1} X^T y$

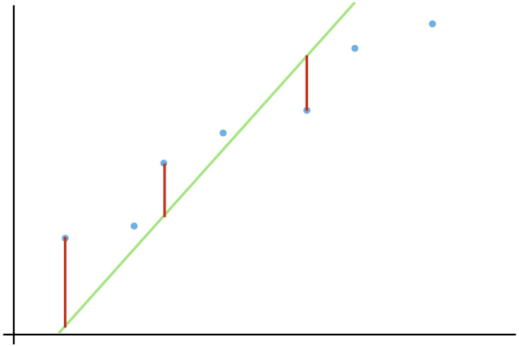
# Minimizing SSE iteratively

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



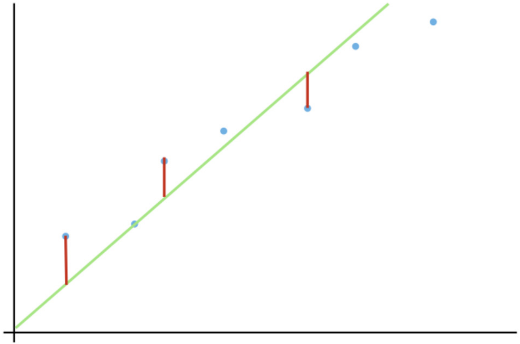
# Minimizing SSE iteratively

- 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



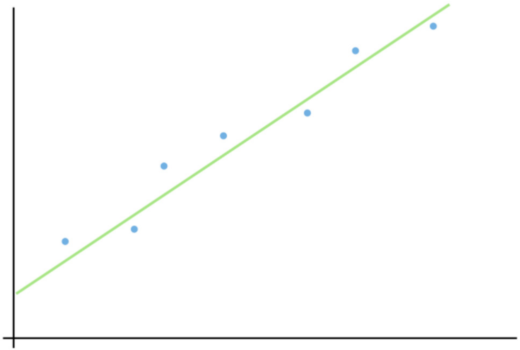
# Minimizing SSE iteratively

- 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



# Minimizing SSE iteratively

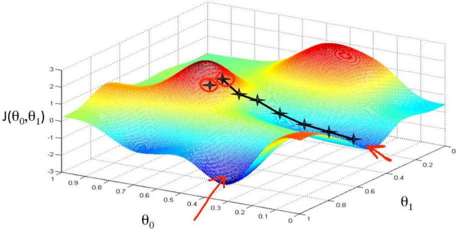
- Normal equation $\theta = (X^TX)^{-1}X^Ty$ is a closed form solution
- Computational challenges
  - Slow if $n$ large, say $n > 10^4$
  - Matrix inversion $(X^TX)^{-1}$ is expensive, also need invertibility
- Iterative approach, make an initial guess
- Keep adjusting the line to reduce SSE
- Stop when 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 $\frac{\partial}{\partial \theta_i} J(\theta)$
- Adjust each parameter against gradient
  - $\theta_i = \theta_i - \alpha \frac{\partial}{\partial \theta_i} J(\theta)$
- For a single training sample $(x, y)$

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



# 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 $\sigma^2$
  - $y_i \sim \mathcal{N}(\mu_i, \sigma^2)$, $\mu_i = \theta^T x_i$
- Model gives us an estimate for $\theta$, so regression learns $\mu_i$ 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 $\theta$, ignore all terms that do not depend on $\theta$
- Optimum value of $\theta$ 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

# Summary

- Regression finds the best fit line through a set of points
- Measure goodness of fit using sum squared error (SSE)
  - Justification: MLE assuming data points are linear points perturbed by Gaussian noise
- Normal equation gives a direct solution in terms of matrix operations
- Gradient descent provides an iterative solution
  - Batch gradient descent vs stochastic gradient descent