Introduction to Machine Learning

Lecture 2 - Linear and Polynomial Regression

Amilcar Soares

amilcar.soares@lnu.se

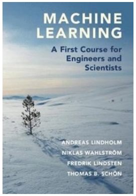
Slides and used datasets are available in Moodle

A big thanks to Dr. Jonas Lundberg for providing most of the slides for this Lecture.

4 april 2024

Agenda - Linear Regression

Reading Instructions



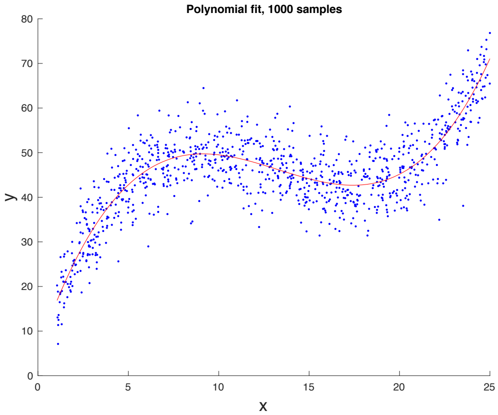
- Lindholm, A., Wahlstrom, N., Lindsten, F., & Schon, T. B. (2022). Machine learning: a first course for engineers and scientists. Cambridge University Press.
  - **Chapter 3:** Basic Parametric Models and a Statistical Perspective on Learning. **Pages 37 to 45**.
- This lecture and slides focus on mathematics (high-level), concepts, and understanding of concepts.

Datasets: `house_prices.csv`, `girls_height.csv`, `polynomial.csv`

Regression – Introduction

Regression

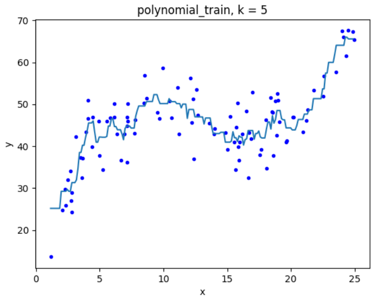
- Find a function $y = f(x)$ that fits a given dataset $(x_1, y_1), (x_2, y_2), \ldots$
- Once $f(x)$ found, make a prediction $y_i = f(x_i)$ for an unknow input x_i
- $(x_1, y_1), (x_2, y_2), \ldots$ is our training set
- $y = f(x)$ is our model



In general

A regression problem aims to find a function $y = f(X)$ that fits a given dataset (X, y) where X is a $n \times p$ matrix and y is a $n \times 1$ vector, where n = number of samples, p = number of features.

From the previous lecture: k-NN regression



- The figure above shows a $k = 5$ fit to a given polynomial dataset (x_i, y_i)
- It looks ugly, but it serves its purpose: to compute y for an arbitrary X.
- To build the plot:
  - Divide x-axis interval $[1, 25]$ into e.g., 200 equidistant points X_j
  - For each X_j: find the 5 data points in the dataset closest to X_j
  - Compute the average for the corresponding $y - value$ for the 5 selected data points $\Rightarrow Y_j$
  - Create the the Plot X_j, Y_j

Regression Example: House Prices in Oregon

Dataset

- X: House area in square feet
- y: Price in dollars
- Samples: 200

Q: What is the price for a 3500 square feet house in Oregon?

Solution: Find a function

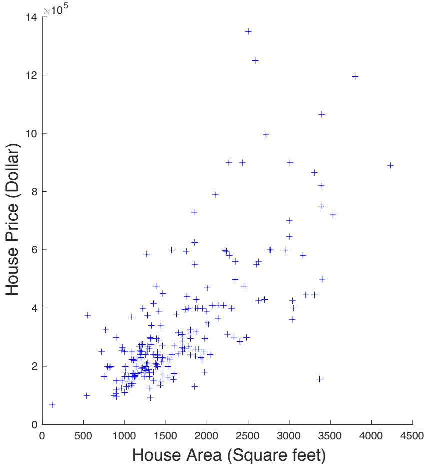
$$y = f(X)$$

that fits the data and then compute $f(3500)$ to find the predicted price.

Linear (1D) Regression: Find a function

$$y = \beta_1 + \beta_2 X$$

that fits the data.



Linear (1D) Regression - Introduction

- **Assumption**: $y = \beta_1 + \beta_2 x$ (our model)
- **Goal**: Find β_1, β_2 making $y = \beta_1 + \beta_2 x$ the best possible fit

The Least Square Method
The squared vertical distance between (x_i, y_i) and assumption $y = \beta_1 + \beta_2 x$ is

$$(y - y_i)^2 = ((\beta_1 + \beta_2 x_i) - y_i)^2.$$

The mean squared distance for the entire training set is

$$J(\beta_1, \beta_2) = \frac{1}{n} \sum_{i=1}^{n} ((\beta_1 + \beta_2 x_i) - y_i)^2.$$

Remaining Task: Find β_1 and β_2 that minimises the **cost function** $J(\beta_1, \beta_2)$

Basic Calculus - Find minimum values

From your basic calculus course

Problem: Find x that minimises a given function $f(x)$

Solution: Differentiate $f(x)$ with respect x and set it to zero:

$$\frac{df}{dx} = 0$$

and solve resulting equation for $x \Rightarrow$ extreme values for $f(x)$.

In our case

Problem: Find β_1 and β_2 that minimises the **cost function** $J(\beta_1, \beta_2)$

Solution: Differentiate $J(\beta_1, \beta_2)$ with respect to β_1 and β_2 and set them to zero:

$$\begin{aligned} \frac{\partial J}{\partial \beta_1} &= 0 \\ \frac{\partial J}{\partial \beta_2} &= 0 \end{aligned}$$

and solve resulting system of equations for β_1 and β_2.

Linear (1D) Regression - Finding β

$$J(\beta_1, \beta_2) = \frac{1}{n}\sum_{i=1}^{n}((\beta_1 + \beta_2 x_i) - y_i)^2.$$

We differentiate $J(\beta_1, \beta_2)$ with respect to β_1 and β_2 and set them to zero:

$$\begin{aligned}
\frac{\partial J}{\partial \beta_1} &= \frac{2}{n}\sum_{i=1}^{n}(\beta_1 + \beta_2 x_i - y_i) = 0 \\
\frac{\partial J}{\partial \beta_2} &= \frac{2}{n}\sum_{i=1}^{n} x_i(\beta_1 + \beta_2 x_i - y_i) = 0
\end{aligned}$$

Simplifying

$$\begin{aligned}
\beta_1 \sum_{i=1}^{n} 1 + \beta_2 \sum_{i=1}^{n} x_i &= \sum_{i=1}^{n} y_i \\
\beta_1 \sum_{i=1}^{n} x_i + \beta_2 \sum_{i=1}^{n} x_i^2 &= \sum_{i=1}^{n} y_i x_i
\end{aligned}$$

Solving for β_1, β_2 gives

$$\beta_1 = \frac{S_{xx}S_y - S_x S_{xy}}{nS_{xx} - S_x S_x}, \quad \beta_2 = \frac{nS_{xy} - S_x S_y}{nS_{xx} - S_x S_x}$$

where

$$S_x = \sum_{i=1}^{n} x_i, \quad S_y = \sum_{i=1}^{n} y_i, \quad S_{xx} = \sum_{i=1}^{n} x_i^2, \quad S_{xy} = \sum_{i=1}^{n} x_i y_i$$

Example: House Prices in Oregon

Q: What is the price for a 3500 square feet house in Oregon?

Solution

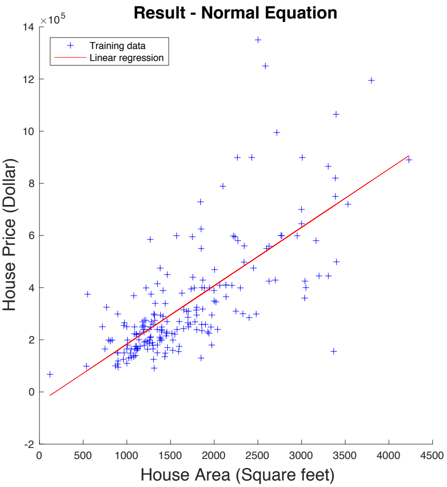
- Linear fit on the training set
- price $= \beta_1 + \beta_2 \times$area

Computing β_1 and β_2 as in previous slide

- $\beta_1 = -40259$, $\beta_2 = 223.77$

Answer
The price for a 3500 sqft house in Oregon is $742,946

The exact solution to the minimisation problem presented in the previous slide is called the **Normal Equation**.



Linear (Degree 1) Regression – Summary

- A dataset (x_i, y_i) with n samples
- Model (or hypothesis): $y = \beta_1 + \beta_2 x$ (Polynomial of degree 1)
- Problem: Find β_1 and β_2 that minimises the cost function

$$J(\beta_1, \beta_2) = \frac{1}{n}\sum_{i=1}^{n}((\beta_1 + \beta_2 x_i) - y_i)^2 \qquad (= MSE)$$

- The Normal Equation Solution

$$\beta_1 = \frac{S_{xx}S_y - S_x S_{xy}}{nS_{xx} - S_x S_x}, \quad \beta_2 = \frac{nS_{xy} - S_x S_y}{nS_{xx} - S_x S_x}$$

where

$$S_x = \sum_{i=1}^{n} x_i, \quad S_y = \sum_{i=1}^{n} y_i, \quad S_{xx} = \sum_{i=1}^{n} x_i^2, \quad S_{xy} = \sum_{i=1}^{n} x_i y_i$$

Linear regression is a parametric approach since it boils down to finding a few parameters β_1 and β_2.

Linear (Degree 1) Regression – Vectorised

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad X_{ext} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix}$$

- y is a $n \times 1$ vector, X_{ext} is a $n \times 2$ matrix and β is a 2×1 vector
- **Model:** $y = X_{ext}\beta$ (corresponds to $y = \beta_1 + \beta_2 x$)
- **Problem:** Find β that minimises the cost function

$$J(\beta) = \frac{1}{n}(X_{ext}\beta - y)^T(X_{ext}\beta - y)$$

- **Solution (Normal Equation)**

$$\beta = (X_{ext}^T X_{ext})^{-1} X_{ext}^T y$$

  An easy-to-read reference for deriving the Normal Equation: https://dzone.com/articles/derivation-normal-equation

- Computing β is $O(p^3)$ $\Rightarrow$ might be problematic for large number of features p.

Vectorization in Python

Cost Function and Normal Equation from the previous slide, respectively:

$$J(\beta) = \frac{1}{n}(X_{ext}\beta - y)^T(X_{ext}\beta - y)$$

$$\beta = (X_{ext}^T X_{ext})^{-1} X_{ext}^T y$$

Implement your solution with Numpy

- Assume X, y are Numpy (np) arrays
- **How to extend X?**
  `Xe = np.c_[np.ones((n,1)),X]`
- **How to run the model?**
  `np.dot(Xe,beta)`
- **How to implement the cost function?**
  `J = (j.T.dot(j))/n`, where `j = np.dot(Xe,beta)-y`
- **How to implement the normal Equation?**
  `beta = np.linalg.inv(Xe.T.dot(Xe)).dot(Xe.T).dot(y)`

The Numpy syntax is a bit strange at first so take your time to learn and use it

Gradient Descent - Motivation

We will often face the following minimization problem:

Find β that minimises a given cost function $J(\beta)$.

The **gradient descent** is a numerical method to solve minimization problems.

In general

- Analytical solutions like the Normal Equation are not always possible.
  - In high-dimensional datasets, the matrix $X^T X$ can become extremely large, making the computation of its inverse computationally intensive and memory-intensive.
  - Additionally, the inversion of such a large matrix may not even be possible due to numerical instability or limited computational resources.
- **Therefore, we need to use numerical methods to solve the problem**
- In this lecture:
  - Simplest possible algorithm - (Batch) Gradient Descent
  - Feature normalization $\Rightarrow$ a method to speed up the gradient descent procedure
- Later on:
  - Other variants of Gradient Descent
  - Available support in Python

Gradient Descent - Introduction

Problem: Find x that minimises $f(x)$
Solution

1. Select a start value x^0 and (small) learning rate γ
2. Apply repeatedly $x^{j+1} = x^j - \gamma \frac{df}{dx}$
3. Stop when $|x^{j+1} - x^j| < \varepsilon$ or after a fix number of iterations

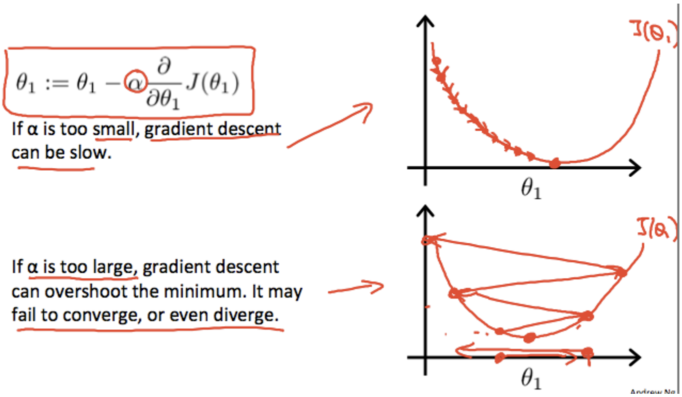
Notice

- Will, in general, find a local min
- Will find global min if $f(x)$ convex
- **Pros:** Simple and fast for strongly convex problems
- **Cons:** Slow $\Rightarrow$ requires many iterations and a small γ in many realistic cases

$f(x)$ is convex if the line segment between any two points on the function's graph lies above the graph.

Selecting learning rate

Our choice of learnig rate influence the convergence rate



$$\boxed{\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} J(\theta_1)}$$

If α is too small, gradient descent can be slow.

If α is too large, gradient descent can overshoot the minimum. It may fail to converge, or even diverge.

- learning rate too small $\Rightarrow$ slow convergence $\Rightarrow$ many iterations
- learning rate too large $\Rightarrow$ no convergence!

Gradient Descent for $J(\beta_1, \beta_2)$

The cost function as a function of β_1, β_2

$$\begin{aligned} J(\beta_1, \beta_2) &= \frac{1}{n}\sum_{i=1}^{n}((\beta_1 + \beta_2 x_i) - y_i)^2 \\ &= a + b\beta_1 + c\beta_2 + d\beta_1^2 + e\beta_1\beta_2 + f\beta_2^2. \end{aligned}$$

for some constants a, b, c, d, e, f that depends on x_i, y_i.
Also, $d = 1$ and $f = \frac{1}{n}\sum_{i=1}^{n} x_i^2$ will both be positive.

Thus

- $J(\beta_1, \beta_2)$ is an upward facing parabolic "bowl" (i.e. convex)
- ... $\Rightarrow$ will have a unique min $(\beta_1^{min}, \beta_2^{min})$
- ... $\Rightarrow$ we can take any starting point (β_1^0, β_2^0) in gradient descent

2D Gradient Descent

$$\begin{aligned} \beta_1^{j+1} = \beta_1^j - \gamma\frac{\partial J}{\partial \beta_1} &= \beta_1^j - \frac{2\gamma}{n}\sum_{i=1}^{n}(\beta_1^j + \beta_2^j x_i - y_i) \\ \beta_2^{j+1} = \beta_2^j - \gamma\frac{\partial J}{\partial \beta_2} &= \beta_2^j - \frac{2\gamma}{n}\sum_{i=1}^{n}x_i(\beta_1^j + \beta_2^j x_i - y_i) \end{aligned}$$

Gradient Descent for $J(\beta)$ – Vectorised

$$X = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \\ \vdots & \vdots \\ 1 & x_n \end{bmatrix}, \; y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \; \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \end{bmatrix}$$

Cost function: $J(\beta) = \frac{1}{n}(X\beta - y)^T(X\beta - y)$

2D Gradient Descent

$$\beta^{j+1} = \beta^j - \gamma \nabla J(\beta) = \beta^j - \frac{2\gamma}{n} X^T(X\beta^j - y)$$

where the gradient $\nabla J(\beta)$ is defined as

$$\nabla J(\beta) = \begin{bmatrix} \frac{\partial J}{\partial \beta_1} \\ \\ \frac{\partial J}{\partial \beta_2} \end{bmatrix}$$

Gradient Descent for $J(\beta)$

Dataset:

$$x = [1, 2, 3]$$
$$y = [3, 4, 6]$$

Parameters:

$$\beta_1 = 0$$
$$\beta_2 = 0$$

Cost function:

$$J(\beta) = \frac{1}{3}\sum_{i=1}^{3}(\beta_1 + \beta_2 x_i - y_i)^2$$

Gradient Descent update rule:

$$\beta_1^{j+1} = \beta_1^j - \gamma \frac{\partial J(\beta)}{\partial \beta_1}$$

$$\beta_2^{j+1} = \beta_2^j - \gamma \frac{\partial J(\beta)}{\partial \beta_2}$$

Gradient Descent for $J(\beta)$ (continued)

Gradient Descent iteration 1:

$$\begin{aligned}
\frac{\partial J}{\partial \beta_1} &= \frac{2}{3}\sum_{i=1}^{3}(\beta_1 + \beta_2 x_i - y_i) \\
&= \frac{2}{3}(0 + 0 + 0 - 3 - 4 - 6) \\
&= \frac{2}{3}(-13) = -\frac{26}{3} = -8.67 \\
\frac{\partial J}{\partial \beta_2} &= \frac{2}{3}\sum_{i=1}^{3}x_i(\beta_1 + \beta_2 x_i - y_i) \\
&= \frac{2}{3}(1(0 + 0 - 3) + 2(0 + 0 - 4) + 3(0 + 0 - 6)) \\
&= \frac{2}{3}(-3 - 8 - 18) \\
&= -\frac{58}{3} = -19.33 \\
\beta_1^1 &= 0 - 0.02 \times -8.67 = 0.17 \\
\beta_2^1 &= 0 - 0.02 \times -19.33 = 0.39
\end{aligned}$$

Gradient Descent for $J(\beta)$ (continued)

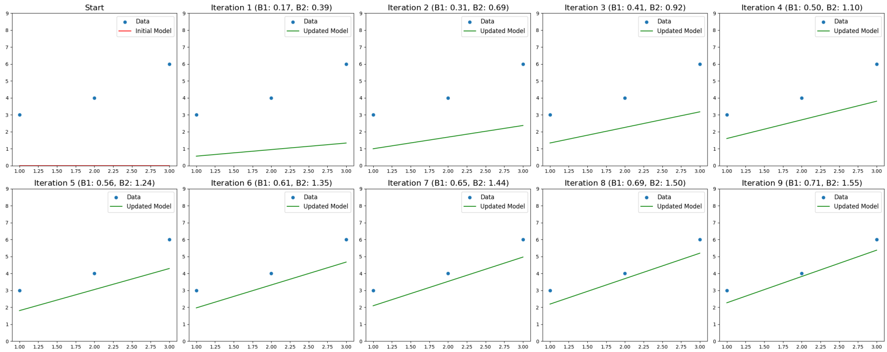
Gradient Descent iteration 2:

$$\begin{aligned}
\frac{\partial J}{\partial \beta_1} &= \frac{2}{3}\sum_{i=1}^{3}(\beta_1^1 + \beta_2^1 x_i - y_i) \\
&= \frac{2}{3}(0.17 + 0.39(1) + 0.39(2) + 0.39(3) - 3 - 4 - 6) \\
&= \frac{2}{3}(0.17 + 0.39 + 0.78 + 1.17 - 13) = \frac{2}{3}(-10.89) = -7.26 \\
\frac{\partial J}{\partial \beta_2} &= \frac{2}{3}\sum_{i=1}^{3} x_i(\beta_1^1 + \beta_2^1 x_i - y_i) \\
&= \frac{2}{3}(1(0.17 + 0.39 - 3) + 2(0.17 + 0.39(2) - 4) + 3(0.17 + 0.39(3) - 6)) \\
&= \frac{2}{3}(-2.44 - 3.22 - 4.00) = -\frac{29.72}{3} = -9.91 \\
\beta_1^2 &= \beta_1^1 - 0.02 \times (-7.26) = 0.17 - 0.02 \times (-7.26) \\
&= 0.17 + 0.1452 = 0.3152 \\
\beta_2^2 &= \beta_2^1 - 0.02 \times (-9.91) = 0.39 - 0.02 \times (-9.91) \\
&= 0.39 + 0.1982 = 0.5882
\end{aligned}$$

Gradient Descent for $J(\beta)$ (continued)

Gradient Descent for 9 iterations...

The small differences in the β values are due to the floating point implementation in the code (more precise when compared to the equation solving in the previous slides).



Gradient Descent in Practice

Initial Steps

1. Number of iterations $N = 10$, $\alpha = 0.00001$, $\beta^0 = (0, 0)$
2. Repeat $\beta^{j+1} = \beta^j - \alpha X^T(X\beta^j - y)$
3. Print/plot $J(\beta)$ vs N to make sure it is decreasing for each iteration

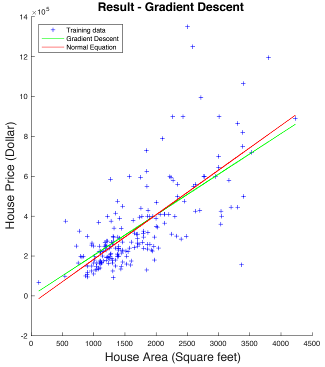
That is, we select a small $\alpha = 2\gamma/n$ and make a few iterations.
- $J(\beta)$ steadily decreasing $\Rightarrow$ α small enough (maybe too small)
- $J(\beta)$ fluctuating or increasing $\Rightarrow$ must decrease α

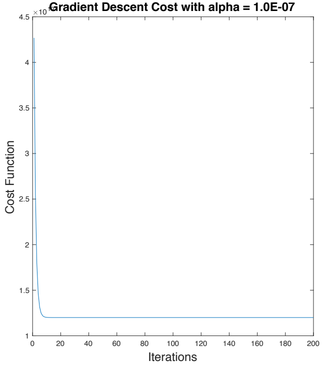
Fine Tuning (Try and error)

1. Modify N and α such that $J(\beta)$ rapidly decreases
2. ... and finally stabilizes at a certain minimum value
3. Stable $J(\beta)$ $\Rightarrow$ You have found β that minimises $J(\beta)$

A plot $J(\beta)$ vs N is a good way to manually see if $J(\beta)$ has stabilized.

House Prices in Oregon – Gradient Descent





$\beta_1, \beta_2 = -0.01996, 203.54$ (compared with -40259, 223.77)
Price for a 3500 sq-ft house: \$712404 (compared with \$742946)

Increase number of iterations $\Rightarrow$ we will get closer to Normal Equation result

Skip first 10-20 iterations in the $J(\beta)$ vs N plot to better capture asymptotic details.

Gradient Descent Alternatives

We used *Batch Gradient Descent*

$$\beta^{j+1} = \beta^j - \alpha X^T(X\beta^j - y)$$

that makes use of the entire dataset X, y in each iteration and $\alpha = \frac{2\gamma}{n}$.

Alternative Optimization Approaches

- *Mini-batch gradient descent* uses only a sample of the dataset to speed up the computations
- *Stochastic gradient descent* uses a randomly chosen single observation to speed up the computations
- *Adaptive methods* varies the step length γ depending on the gradient
- .. and many more fancy approaches that are designed to handle special cases

Python also comes with several predefined optimization methods. A few of these will presented later on.

10 minute break

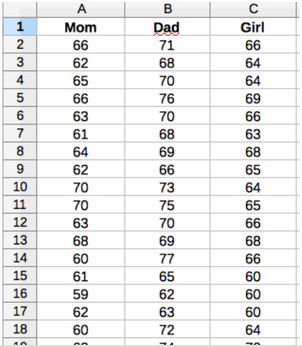
Coffee Break ...

Multivariate Linear Regression – Introduction

Previously: y (price) as a function of x (area, one feature)
Model: $y = \beta_1 + \beta_2 x$

Now: y as a function of multiple features $x_1, x_2, ...x_p$
Model: $y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p$ (multiple features, still linear)

Example: A girls height as a function of mom and dad height (two features)
Dataset: 214 observations of mom, dad, and girl heights

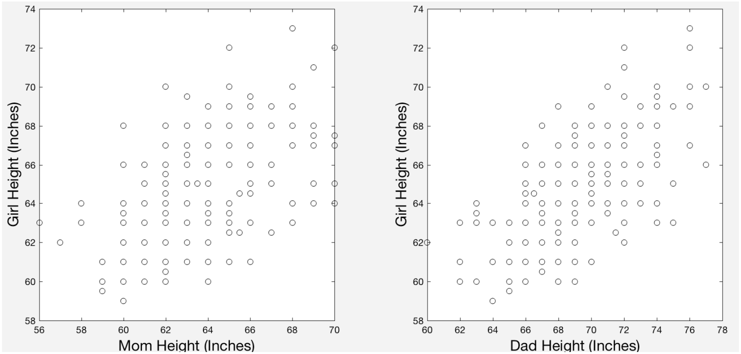


	A	B	C
1	Mom	Dad	Girl
2	66	71	66
3	62	68	64
4	65	70	64
5	66	76	69
6	63	70	66
7	61	68	63
8	64	69	68
9	62	66	65
10	70	73	64
11	70	75	65
12	63	70	66
13	68	69	68
14	60	77	66
15	61	65	60
16	59	62	60
17	62	63	60
18	60	72	64

Girls Height – Dataset

- Dataset: `girls_height.csv`, Samples: 214 girls
- Q: Predicted height for a girl who's mom is 65 inches and dad is 70 inches?

Dataset Plot



Multivariate Linear Regression – Setup

Assumption: Height $= a + b \times$MomHeight$+ c \times$DadHeight
Model: $y = \beta_1 + \beta_2 X_1 + \beta_3 X_2$

Vectorised Approach: Model $y = X\beta$

$$y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \quad X = \begin{bmatrix} 1 & x_1^1 & x_1^2 \\ 1 & x_2^1 & x_2^2 \\ \vdots & \vdots & \vdots \\ 1 & x_n^1 & x_n^2 \end{bmatrix}, \quad \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \end{bmatrix}$$

Problem Find β minimising cost function: $J(\beta) = \frac{1}{n}(X\beta - y)^T(X\beta - y)$

Exact Solution $\beta = (X^T X)^{-1} X^T y$ (Normal Equation)

Approximative Solution (Gradient Descent)

$$\beta^{j+1} = \beta^j - \gamma \nabla J(\beta) = \beta^j - \frac{2\gamma}{n} X^T (X\beta^j - y)$$

Notice

- The vectorized Cost Function, Normal Equation, and Gradient Descent are identical to the case $y = \beta_1 + \beta_2 x \Rightarrow$ A proper Python solution can be reused

Girls Height – Result

Normal Equation

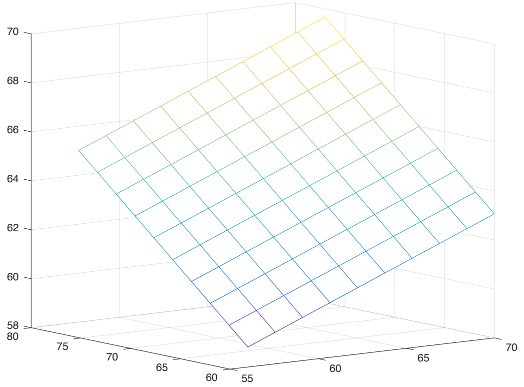
- $\beta = 18.50, 0.303, 0.388$

Height of girl with parents (65,70): 65.425

Gradient Descent

- $\alpha = 0.0002$, $N_{iter} = 20000000$
- $\beta = 18.48, 0.304, 0.388$

Height of girl with parents (65,70): 65.426



Notice: Similar results for gradient descent required **20 million iterations**!
$\Rightarrow$ a few minutes to compute!

Feature Normalization

- Gradient descent required 20 million iterations to compute β in the Girls Height example.
- This is quite common on multivariate problems. Especially if the values in different features are vastly different. E.g. in range $[0, 1]$ in one feature, and in range $[1000, 5000]$ in another.
- Typical solutions
    - Normalize data $\Rightarrow$ all features of similar size
    - Replace Gradient Descent with more advanced optimization methods

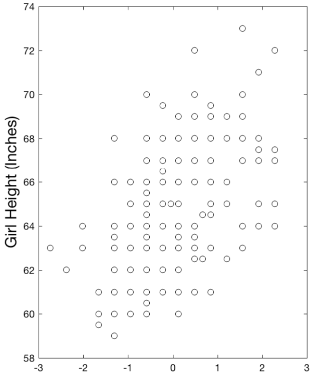
Feature Normalization For each feature X^i (but not for the one-column)

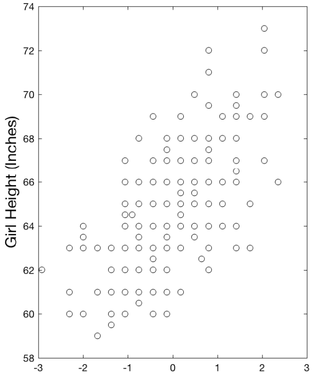
1. Compute mean μ_i
2. Compute standard deviation σ_i
3. Compute normalized X_n^i as $X_n^i = (X^i - \mu_i)/\sigma_i$
4. Build extended matrix $X_{ne} = [\mathbf{1}, X_n]$ and continue

After normalization each feature X_n^i will have a mean value of 0, and a standard deviation of 1.

Girls Height – Feature Normalization

After normalization with $\mu = [63.63, 69.41]$ and $\sigma = [2.79, 3.22]$

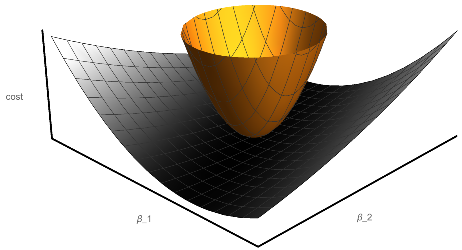




Notice

- All feature values centered around 0
- All features have the same spread (standard deviation 1)

Feature Normalization



- Feature normalization turns every "bowl" into a uniform one (the yellow) that is strongly convex $\Rightarrow$ the gradient descent iteration converges rapidly
- The girls height "bowl" looks a bit like the black one. Rather strongly convex for β_2, β_3 and very flat for the intercept coefficient β_1
- Strongly convex for $\beta_2, \beta_3 \Rightarrow$ must use a small learning rate (step size) α
- Small $\alpha \Rightarrow$ very slow convergence for $\beta_1 \Rightarrow$ We need 20 million iterations

Normalized Girls Height – Result

Normal Equation

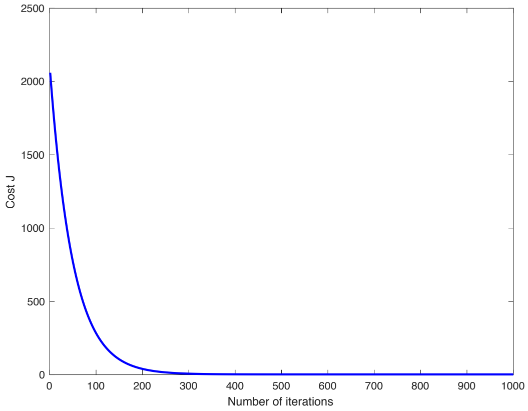
- $\beta = [64.8, 0.845, 1.26]$
- $J = 4.048$

Height of girl with parents (65,70): 65.425

Gradient Descent

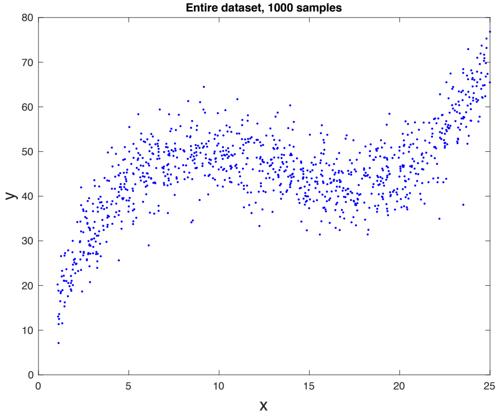
- $\alpha = 0.01,\ N_{iter} = 1000$
- $\beta = [64.8, 0.845, 1.26]$
- $J = 4.048$

Height of girl with parents (65,70): 65.422



Notice Gradient descent requires just 1000 iterations! $\Rightarrow$ less than a second. Also, parents (65,70) must be normalized to (0.4898, 0.1821) before computing the height using our new β.

Polynomial – Dataset



- Dataset: `polynomial.csv` with 1000 observations
- Artificial dataset generated by Jonas Lundberg
- Q: What is a suitable model?

Polynomial Regression – Setup

Observation: A polynomial of degree 3 could handle the up-down-up scenario
Model: $y = \beta_1 + \beta_2 X + \beta_3 X^2 + \beta_4 X^3$

Vectorised Approach Model $y = X\beta$

$$X = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ \vdots & \vdots & \vdots & \vdots \\ 1 & x_n & x_2^2 & x_n^3 \end{bmatrix}, \; y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{bmatrix}, \; \beta = \begin{bmatrix} \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix}$$

In Python X is replaced by `X = np.c_[np.ones((n,1)),X,X**2,X**3]`

Problem Find β minimising cost function: $J(\beta) = \frac{1}{n}(X\beta - y)^T(X\beta - y)$

Exact Solution $\beta = (X^TX)^{-1}X^Ty$ (Normal Equation)

Approximative Solution (Gradient Descent)

$$\beta^{j+1} = \beta^j - \gamma\nabla J(\beta) = \beta^j - \frac{2\gamma}{n}X^T(X\beta^j - y)$$

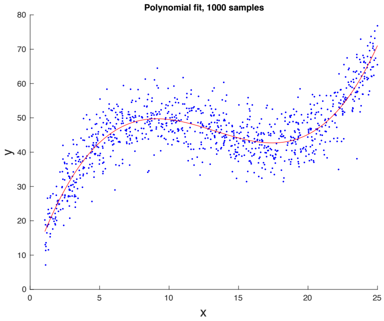
Notice

- Once again, no change in the vectorised versions
- $\Rightarrow$ A proper Python solution can be reused

Polynomial – Result

Algorithm

1. Read data $\Rightarrow$ vectors X and y
2. Extend $X \Rightarrow$ $X = [1, X, X^2, X^3]$
3. Compute β as $\beta = (X^T X)^{-1} X^T y$
4. Plot $x\beta$ vs x

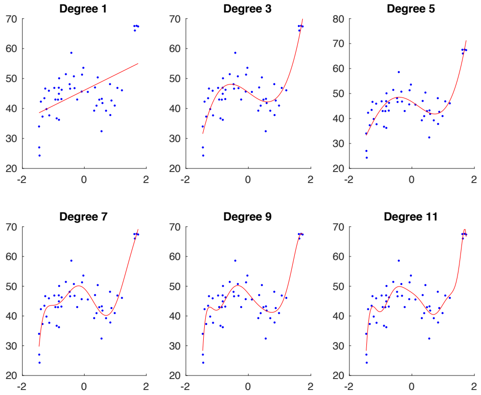


The dataset was generated from:

$$y = f(x) = 5 + 12x - x^2 + 0.025x^3 + \underbrace{normrnd(0, 5)}_{= \varepsilon \text{ (Noise)}}$$

Polynomial fit: $\beta = [5.50, 11.7, -0.981, 0.0246]$ **OK or ...?**

Which one is the best fit?



A small **feature normalized** subsample of 50 observations from `polynomial.csv`

- **Question:** Which one is the best fit?
- **Question:** How do we compare different models?

Mean Square Error

- In regression we use a training set X,y to compute a β that makes $y = X\beta$ a good fit to the training data.
- $y = X\beta$ is our model, the result of the training phase
- The mean square error (MSE) for a given β is defined as

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(y_i - f(x_i))^2 \quad \text{where} \quad f(x_i) = x_i\beta$$

or vectorized

$$MSE = \frac{1}{n}(X\beta - y)^T(X\beta - y)$$

- That is, the mean of the vertical squared distances
- Notice that MSE is the same as the cost function $J(\beta)$ for linear regression.
- Obviously, low *MSE* is a good fit and high *MSE* is a bad fit

Train and Test Error

Assume a training set X_{train}, y_{train} and a test set X_{test}, y_{test}

- **Training phase:** X_{train}, y_{train} $\Rightarrow$ Estimate β $\Rightarrow$ create a model $y = X\beta$
- **Training error:** Compute $MSE = \frac{1}{n}(X\beta - y)^T(X\beta - y)$ with X_{train},y_{train} using β from the training phase.
- **Test error:** Compute $MSE = \frac{1}{n}(X\beta - y)^T(X\beta - y)$ with X_{test},y_{test} using β from the training phase.

Training vs Test

- The Mean Squared Error (MSE) computed using the training set is referred to as the *Training MSE* estimation.
- The MSE computed using a separate test set is the *Test MSE* estimation.
- A test set resembles the training set but is not utilized in model construction, remaining unseen by the model during training.
- **A high-performing model exhibits a low Test MSE, indicating strong performance on unseen data.**

Training vs Test – polynomial.csv

- We used a small sample of 50 points to train various polynomial models
- We use the remaining 950 points to evaluate the model

Degree	MSE_{train}	MSE_{test}
1	55.7	69.1
2	53.4	75.4
3	22.3	**27.6**
4	21.4	30.6
5	21.3	31.3
6	18.5	39.4
7	17.7	67.6
8	16.7	197.7
9	16.6	284.5
10	16.5	558.7

Conclusions

- MSE_{train} is steadily decreasing $\Rightarrow$ higher order polynomials can always better adapt to the training data
- MSE_{test} has a minimum at degree 3 $\Rightarrow$ best to handle unseen data
  $\Rightarrow$ **Degree 3 gives the best fit!**

Reducible and Irreducible Errors

We generated data from

$$y = f(x) = 5 + 12x - x^2 + 0.025x^3 + \underbrace{normrnd(0,5)}_{= \varepsilon \text{ (Noise)}}$$

Our polynomial fit gave the model.

$$\hat{y} = \hat{f}(x) = 5.5 + 11.7x - 0.98x^2 + 0.0246x^3$$

- The error can be divided into two parts

$$E(y - \hat{y})^2 = [f(X) - \hat{f}(X)]^2 + Var(\varepsilon)$$

  - $E(y - \hat{y})^2$ is the total error of our regression
  - $[f(X) - \hat{f}(X)]^2$ is the error due to our model $\hat{f}(X)$ (reducible)
  - $Var(\varepsilon)$ is the error due to the noise (irreducible)
- A better model can reduce the error; a worse one (e.g., $\beta_1 + \beta_2 x$) may increase it.
- We can never eliminate the irreducible error $Var(\varepsilon)$ due to noise.

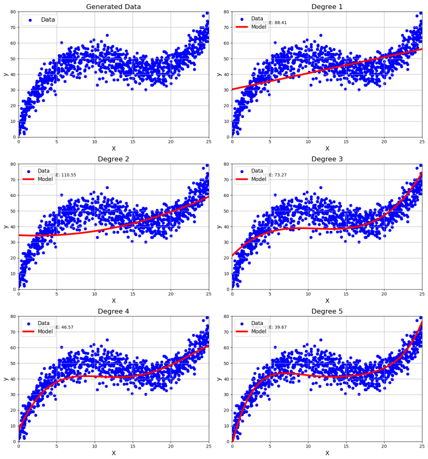
Variance and Bias, Over- and Underfitting

The reducible error for model $\hat{f}(x)$ can further be divided into two parts

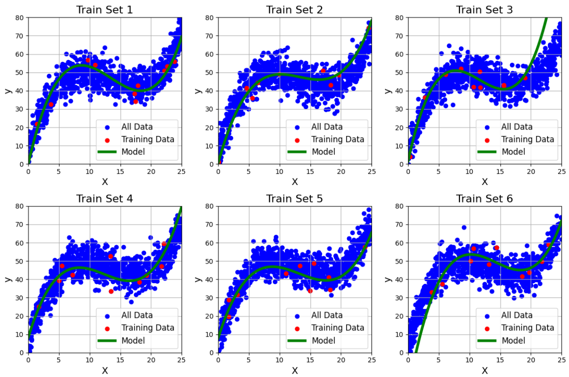
$$Reducible\ Error = Bias + Variance$$

- **Bias** refers to the error introduced by approximating a real-world problem with a simplified model.
  - It measures how far off the model's predictions are from the true values.
  - A model with **high bias** pays little attention to the training data and oversimplifies the problem, leading to **underfitting**.
- **Variance** refers to the amount the model's prediction would change if we trained it on a different dataset.
  - It measures the model's sensitivity to the fluctuations in the training data.
  - A model with **high variance** fits the training data too closely and captures noise and underlying patterns, leading to **overfitting**.
- Variance and Bias can not be avoided for realistic datasets
- Achieving a balance between Variance and Bias is essential to minimize errors, known as **the Bias-Variance Trade-off**.

Bias



- The models cannot capture the true underlying relationship between the features (X) and the target variable (y).



- The models exhibit high variability in their predictions when trained on different subsets of the data.
- Despite using the same degree of polynomial features, the models show significant differences in their shapes and predictions across different training datasets.
- This variability arises from the sensitivity of the models to the specific samples in the training data, leading to different learned patterns and resulting in a wide range of predictions.
- Consequently, the models demonstrate high sensitivity to variations in the training data, indicating high variance.
- In other words, the models tend to overfit the training data, capturing noise and idiosyncrasies specific to each training set rather than the true underlying relationship between the features and the target variable.

Regression Summary

All scenarios (linear, multivariate, polynomial) are treated the same way

- Dataset: $[X, y]$
- Extend X to fit scenario $X_{ext} = [1, X, ...]$
- Model: $y = X_{ext}\beta$
- Problem: Find β that minimises the cost function

$$J(\beta) = \frac{1}{n}(X_{ext}\beta - y)^T(X_{ext}\beta - y)$$

- Exact Solution (Normal Equation)

$$\beta = (X_{ext}^T X_{ext})^{-1} X_{ext}^T y$$

- Approximative Solution (Gradient Descent)

$$\beta^{j+1} = \beta^j - \gamma\nabla J(\beta) = \beta^j - \frac{2\gamma}{n} X_{ext}^T(X_{ext}\beta^j - y)$$

- Apply feature normalization of X^i to speed up gradient descent
    1. Compute mean μ_i and standard deviation σ_i
    2. Compute normalized X^i as $X_n^i = (X^i - \mu_i)/\sigma_i$

Assignment Summary

Lecture 2 is the basis for an entire part of Assignment 2.

1. Part 1: Implementations of regression models using the normal equation and the gradient descent. An abstract class must be followed to implement those. All guidelines are given in the pdf file and the classes documentation.
2. Part 2: Put your implementations to use in a Multivariate Regression Model.
3. Part 3: Put your implementations to use with polynomials with degrees >1.

Feel free to exchange Python findings in forum. Do **not** exchange exercise solutions.

End of Lecture!

Next Lecture: Logistic Regression (**in both Vaxjo and Kalmar!**)
Logistic regression is (despite the name) a classification method

Start to work on Assignment 2! It is more time-consuming compared to Assignment 1